# **Summary**

For this project, I developed contact, task, and appointment services. After carefully reviewing each requirement and getting a deep understanding of the requirements, I would write a test code for each requirement. Then I would write code to fulfill the test code, and in-turn would be fulfilling that requirement. I did this for each requirement.

I used JUnit to test the Task constructor, which included checking that valid arguments were accepted, and invalid arguments threw an IllegalArgumentException. My approach was aligned with the given software requirements. In TaskServices, for the methods addTask (TaskServiceTest.java, lines 15-18), deleteTask (TaskServiceTest.java, lines 31-35), and updateTask (TaskServiceTest.java, lines 45-49), I carefully review the requirements of each. For testing the addTask, I correctly added a new task to the map, like wise with deleteTask, I correctly removed a task from the map. Each method was tested to ensure they threw the appropriate exception for invalid input.

I believe my JUnit tests were effective based on their coverage percentage. I aimed to test each method and constructor according to its expected behavior. I quickly learned that by writing JUnit test, it gave me a deeper understanding of the requirements. The code coverage for this project is 85.5%.

To ensure my code was technically sound, I intentionally wrote tests with invalid inputs. I then checked that these tests cases to ensure that they threw the appropriate exceptions. In Contact,java, lines 12-13, I wrote code indicating that the first name cannot be null. In ContactTest.java, lines 31-35, I tested to see what would happen if the input for the first name was null. As expected behavior for this test, an IllegalArgumentException was threw. In doing so, I developed the code with all possible scenarios and writing tests to test these scenarios to ensure that the code behaves as expected.

To ensure efficiency in my code, I aimed to use as few resources as possible and minimize unnecessary repetition. In TaskServiceTest.java, lines 21-28, I used a single test method to test both addTask and updateTask, since both methods relied on the same input and output conditions. This was useful in determining if boundaries were correctly set, as they had the same conditions for input and output.

# **Reflection**

The primary software testing techniques employed in this project were black-box testing and white-box testing. Black-box testing involved testing the behavior of the code without examining the internal implementation. Furthermore, white-box testing involved examining the codes internal implementation to ensure industry standards were followed throughout. To ensure all possible scenarios were covered I utilized both black-box testing and white-box testing.

Other software testing techniques that I did not use for this project include load testing and performance testing. Load testing is testing the software under heavy user loads. Performance testing is testing the speed and responsiveness of software. Neither of these were tested in this project, but are important, nevertheless. Before deploying such software, these tests should be performed to ensure high quality software is being delivered.

The mindset that I adopted while working on this project included curiosity, openness, and acceptance. Approaching this project with a curious, open, and acceptance mindset was essential for me. Staying curious as how to meet the requirements and develop test for the requirements. Being open to approaching a problem from more than one direction and allowing myself to think outside the box for solutions to roadblocks as they came up. Finally acceptance, accepting that I may not be able to think or come up with a solution to every scenario at this stage of my path of becoming a developer. This mindset allowed me to cautiously and carefully review tests to get the most coverage possible in all of the files. Relationships is important when developing test, when a method is called independently, it may behave as expected but when a method is called by another method, it may behave not as expected.

I did my best to not be biased during code review. Being biased makes us look for what we want to see, not what we’re not expecting. I aimed to approach the code review objectively and without preconceived notions or assumptions. I carefully reviewed the requirements and tested the code for its expected behavior without trying to sway its behavior. When doing a code review on ones own code, I find it important to take breaks often to come back with a fresh perspective after each break to attempt to mitigate bias.

Being disciplined in your commitment to quality as a software engineering professional, cutting corners can lead to technical debt, which is known to cause issues later on down the line. To avoid technical debt, prioritizing writing maintainable, modular code that follows industry standards. Providing high quality documentation and thoroughly tested code. This can aid in avoiding issues later on and lead to being able to deliver high quality code.

# **References**

Boni Garcia. (2017). *Mastering Software Testing with JUnit 5 : A Comprehensive, Hands-on*

*Guide on Unit Testing Framework for Java Programming Language*. Packt Publishing.

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019). *Software testing:*

*An istqb-bcs certified tester foundation guide - 4th edition*. BCS Learning &

Development Limited.