### Tanisha Wilson

CS 320

### **Summary and Reflections Report**

#### **Summary**

In this project, I focused on testing the mobile application to make sure that it worked as expected and adhered to the software requirements. I wrote tests for three main features: Appointment, Contact, and Task. Each feature had specific requirements, and my goal was to guarantee that the code met those requirements through detailed testing.

For the Appointment feature, I wrote tests that validated the creation, deletion, and management of appointments. This included verifying that the appointment ID matched the expected value, the date was set in the future, and the description was within the allowed character limit. For example, in the testValidAppointment() method, I ensured that the appointment ID was correct, the date was valid, and the description was consistent with the input.

For the Contact feature, I tested the creation, update, and deletion of contacts while ensuring that the system handled errors properly, such as duplicate IDs or invalid input. One important test was testAddContactWithDuplicateId(), where I checked that the system would throw an exception if a duplicate ID was added. This approach made sure the application responded correctly to invalid data and provided feedback to users.

For the Task feature, I focused on testing the task creation, modification, and deletion, making sure that task names and descriptions adhered to the defined length limits. In tests like testAddTaskWithDuplicateId(), I validated that the system would throw an exception when attempting to create a task with a duplicate ID. Throughout all the tests, I aimed to ensure that all features functioned properly, followed the software’s requirements, and handled edge cases correctly.

#### **Reflection**

While working on this project, I used various software testing techniques to confirm that the code was both solidly built and reliable. One of the key techniques I used was boundary testing, which focuses on testing the limits of input values. For example, in the testAppointmentDescriptionBoundary() method, I tested the description field to ensure that input beyond the allowed character limit would trigger an exception. I applied similar boundary tests to other features, like Contact IDs and Task descriptions, to ensure that the system adhered to the defined constraints.

Another important technique I used was exception testing, which ensures that the system correctly handles invalid inputs and edge cases. In methods like testAddContactWithDuplicateId() and testPhoneSetter(), I checked that the system would properly throw exceptions when provided with invalid contact IDs or phone numbers. This helped ensure the application failed gracefully when incorrect data was entered.

I also applied equivalence partitioning, which involves testing different input ranges by covering both valid and invalid values. For example, I tested various valid and invalid contact IDs, descriptions, and task names to ensure that the system responded appropriately to different inputs. This technique helped minimize the number of tests while ensuring that the most important cases were covered.

Throughout the project, I maintained a careful and thorough mindset. I recognized that even small issues in the code could lead to larger problems down the line. For example, in the Contact feature, I paid close attention to edge cases, such as testing for null values and duplicate IDs, to make sure the system didn't accept invalid data. This required careful consideration of how the different components of the system interacted with each other, and I made sure to test all possible scenarios.

I also remained aware of potential bias while testing my own code. As a developer, it can be easy to overlook certain edge cases or assume that the code is flawless. However, by focusing on the requirements and ensuring that I tested every possible situation, I was able to avoid missing important errors. For instance, in the testDeleteContactWithNonExistingId() test, I made sure that attempting to delete a non-existent contact resulted in an exception being thrown.

Lastly, I recognized the importance of maintaining discipline and not cutting corners when testing or writing code. While it might be tempting to skip certain tests or rush through the process, I made sure to thoroughly test each feature to ensure its quality. By focusing on writing clean, well-tested code, I was able to avoid creating long term issues that would make the system harder to maintain in the future.

#### **Mindset**

Overall, the unit testing approach I used helped make sure that the mobile application was functional, met the requirements, and handled all edge cases. The tests were designed to ensure that both valid and invalid inputs were properly managed, and the techniques used were well-suited for testing the Appointment, Contact, and Task features. By staying disciplined and carefully testing the system, I was able to produce high-quality code. Moving forward, I will continue to prioritize quality in my testing approach to ensure the code remains reliable and maintainable in the future.