**Project Two: Summary And Reflections Report**

Najjah Gray-Thompson

Southern New Hampshire University

CS 320

Angelo Luo

August 22, 2025

My contribution to the development of the mobile application consisted of creating three core features: contact service, task service and appointment service. In conjunction with creation of each service, unit tests were also constructed to ensure reliability. This report offers an overview of my unit testing method, my reflections on creating the JUnit tests, and an assessment of the testing methods I used during this project. In the process, I will demonstrate how my tests met the software requirements and how my experiences highlighted the value of testing in professional software engineering.

A screen shot of a computer code

AI-generated content may be incorrect.For the contact service, the test validated that contacts could be added, updated, and deleted correctly. I tested how tasks were created and updated for the task service. And the appointment service tests were aimed at ensuring that appointments could be scheduled without duplication. These unit tests were created in accordance with the software specifications. Each test also verified constraints on user input and validated the necessary fields. For example, the requirement that contact IDs must be unique was tested by asserting that adding a duplicate ID threw an exception. Similarly, in the task service, I tested that task identifiers were unique, that task names did not exceed 20 characters, and that descriptions did not exceed 50 characters. For every requirement description, a test was created to ensure the program reacts accordingly.

The effectiveness of my JUnit tests was strong, and I validated this using the coverage percentage. In all service classes, the test suite's line coverage was above 90%, meaning that almost all code paths were run during testing. Also, to ensure meaningful validation of outcomes, I emphasized assertion diversity with a variety of JUnit assertions. This diversity of assertions A screen shot of a computer

AI-generated content may be incorrect.ensured that my tests were validating correctness thoroughly.

A screenshot of a computer program

AI-generated content may be incorrect. To ensure my tests were technically sound, I included edge cases in addition to standard input cases. One way I implemented this was by testing null values in all constructors to make sure the application would reject incomplete or invalid data. In the figure above, the test throws an exception if the contact ID is null. It was also imperative to ensure efficiency. For example, use of the @BeforeEach method was an essential factor in maintaining efficiency. In doing this, I was able to prevent rewriting set up codes for the date in the appointment service test.

One of the biggest aspects of this project was the utilization of testing techniques. Equivalence partitioning, boundary value analysis, and exception testing were the most common techniques I utilized. Using equivalence partitioning, I classified inputs into valid and invalid cases, such as valid contact names under ten characters and invalid names that exceeded that limit. Boundary value analysis allowed me to compare inputs at the limits of constraints, such as a 10-character name (valid) vs an 11-character name (invalid). Exception testing was especially critical because the services were designed to throw exceptions for incorrect inputs like null fields or duplicate identifiers. By combining these techniques, I ensured that I tested both correct and incorrect entries.

While the previous testing techniques mentioned were useful, there is still a variety of testing techniques that were not applied. Integration testing is a method that validates the interaction between multiple services. This method was not used because my focus was to test the services individually. Another common testing technique is system testing. System testing is used to assess the overall application as a whole. Being that my portion of development consisted of providing deliverables, this technique was also not used. Both methods are excellent ways to test a program, however they did not fit the needs for this project.

I approached the project with caution, mindful that even little errors in coding could have serious consequences later on. For example, if I had neglected to properly validate appointment dates, users could have scheduled overlapping or invalid appointments, which defeats the purpose of the application overall. Recognizing the interrelationships of the code made me more cautious in my testing approach.

I set out to reduce bias when I was developing this product. It was critical to test not only the application's success, but also its failure. For example, I evaluated cases when phone numbers were missing digits and task descriptions exceeded the predefined limit. In doing so, I was able to not just avoid bias but also confront weakness rather than presuming correctness. Without this, it is very simple to miss minor errors. Essentially to reduce bias, it was best to include these negative tests.

Lastly, my commitment to quality was illustrated throughout this project. It was essential to maintain a disciplined approach to writing and running tests. This work is a representation of my skill and capabilities. I also value the clients and their final product. Cutting corners would be a disservice to both me and the client. With the client’s needs and my reputation as a priority, I can assure this will avoid technical debt in the future.