Jose Tellez

CS 320

Professor Fernando Nieto

October 17, 2025

Summary and Reflections Report

In Project One, I developed a mobile application that included Contact, Task, and Appointment services. A critical part of this project was writing unit tests to ensure that each feature functioned correctly and met the software requirements. This process allowed me to verify functionality, identify potential errors early, and explore different testing strategies to build reliable and maintainable software.

For the Contact Service, I focused on testing the creation, retrieval, and deletion of contacts. Each test confirmed that contact details—such as ID, first name, last name, phone number, and address—were accurately stored. For example, a test verified that when a new contact was created, the first name matched the expected value using assertEquals("John", contact.getFirstName()), and similar assertions checked the other fields. This approach ensured that the Contact Service met the requirement for complete and accurate contact information.

In the Task Service, I tested the creation, priority assignment, and completion of tasks. Boundary testing was applied to verify that tasks with the lowest and highest allowed priorities were handled correctly, while invalid priorities triggered exceptions. These tests ensured that the application adhered to the requirement for valid task priorities and could handle edge cases without errors.

The Appointment Service required careful validation of input rules and the handling of scheduling conflicts. For example, I verified that appointments with invalid IDs, past dates, or overly long descriptions were rejected using assertions such as:

**assertThrows(IllegalArgumentException.class, () -> {**

**new Appointment(null, futureDate, "Checkup");**

**});**

I also tested valid appointments with code like:

**Appointment appt = new Appointment("12345", futureDate, "Doctor visit");**

**assertEquals("12345", appt.getAppointmentId());**

**assertEquals("Doctor visit", appt.getDescription());**

**assertEquals(futureDate, appt.getAppointmentDate());**

These tests covered over 90% of the code, demonstrating comprehensive validation. Efficiency was achieved by creating reusable objects, such as futureDate, to avoid redundant setup in multiple tests.

Throughout this project, I employed several testing techniques. Unit testing allowed me to evaluate each feature independently, while boundary testing ensured that the system handled extreme input values correctly. Negative testing assessed how the application responded to invalid inputs, such as null or overly long values. I did not use integration testing, performance testing, or user acceptance testing, which are more applicable to larger projects, but these techniques remain valuable for ensuring overall system functionality, responsiveness, and user satisfaction.

My approach to testing was grounded in a cautious and thoughtful mindset. For instance, in the Appointment Service, I anticipated potential scheduling conflicts and designed tests to detect overlapping times. To limit bias, I included tests that were likely to fail, rather than only tests that I expected to pass, ensuring objectivity when evaluating my own code.

Discipline and commitment to quality were essential throughout the project. Skipping tests or taking shortcuts could have introduced technical debt, making the software harder to maintain or expand in the future. By adhering to a thorough and structured testing process, I ensured the reliability and maintainability of the application.

In conclusion, Project One highlighted the importance of careful, requirement-driven testing. Through unit tests, boundary checks, and negative testing, I validated each feature’s functionality, identified edge cases, and ensured adherence to software requirements. This experience reinforced that disciplined testing, attention to detail, and a commitment to quality are critical components of professional software engineering practice.