Ian Coxon

Southern New Hampshire University

Dr. Karl Lewis

Date: April 15, 2025

CS 320 Project Two – Summary and Reflections

Summary

For Project One, I worked on building and testing the Contact, Task, and Appointment services. My approach was to use unit testing throughout the entire development cycle using JUnit 5. I didn’t just save testing for after the code was written—I tested as I went. That mindset helped me avoid a lot of problems before they grew into bigger issues.

Each of the three classes had different requirements. For example, the Contact class required checks for null values and maximum lengths for fields like first name, last name, phone number, and address. Dr. Lewis pointed out in my Module 3 submission that I had missed some of the null and length validations. I took that feedback seriously and went back in Module 6 to fix those issues. I added specific tests using assertThrows() to catch when someone tries to pass a null or overly long field.

For instance, in ContactTest.java, I added tests like:

assertThrows(IllegalArgumentException.class, () -> new Contact("123", null, "Smith", "1234567890", "123 Main St"));

For my Task and Appointment classes, I followed a similar pattern, verifying all required fields and ensuring length restrictions and null conditions were tested. My test methods were built around real-case thinking: What would actually go wrong in the wild?

To make sure the tests were effective, I ran JaCoCo and checked the code coverage in Eclipse. Most classes were over 80%, especially TaskService and AppointmentService. ContactService was lower initially, but after I added more tests for each field, the percentage improved. I took screenshots of the coverage reports for each service and included them in my submission folder.

I also made sure the code was technically sound by testing individual methods in isolation and keeping my logic clean. In ContactServiceTest.java, for example, I updated fields one at a time and tested the result using assertEquals() to make sure the updates worked. I didn’t try to shortcut with assumptions—I wanted clear pass/fail results.

Efficiency-wise, I used helper methods to avoid repeating date creation in the appointment tests and kept each test method focused and readable. The more readable the test, the easier it is to maintain.

Reflection

The main testing technique I used was unit testing. This involves testing small, isolated pieces of code—usually one class or method at a time. I used JUnit 5 to write tests for each class, checking not only the happy paths but also the edge cases. For example, I tested what happens if someone tries to input a null address or an appointment date in the past.

Other testing techniques I didn’t use in this project include integration testing, regression testing, and acceptance testing. Integration testing checks how different parts of an app work together—like if a user interface correctly talks to a backend. We didn’t need that here since this was a backend-only assignment. Regression testing is useful in bigger apps where you want to make sure your changes didn’t break anything that used to work. Acceptance testing is when the client signs off on the software before release.

Each technique has its place. Unit testing is great early on. Integration testing is key in larger systems. Regression testing helps when you’re managing ongoing releases. Acceptance testing is how you wrap up a project with confidence.

As I worked on this project, I found myself shifting into a tester’s mindset. I wasn’t just coding to make things work—I was looking for ways they might fail. That cautious approach helped me write more defensive code. I also learned how tricky it can be to test your own work without bias. It’s easy to assume something works just because you wrote it, so I forced myself to test unexpected inputs.

Being disciplined with quality matters. Shortcuts might save you five minutes now, but they can cost you hours later. I’ve seen that firsthand in my own work and even on the shop floor when something wasn’t double-checked. That lesson carries over to code. Going forward, I’ll continue to write tests alongside my code and run coverage reports so I can catch what I missed.

This project taught me that quality isn’t a phase—it’s a mindset.