Jordan Krueger

Project 2

6/22/2025

Summary and Reflections Report

Summary

For Project One, I built and tested three features for a mobile app: Contact Service, Task Service, and Appointment Service. I used JUnit and wrote tests right after finishing each class. This helped me catch bugs early and make sure things were working as expected.

In Contact Service, I tested that IDs, phone numbers, and addresses weren’t null or too long. Task Service involved testing updates and deletions, making sure tasks couldn’t be duplicated. Appointment Service focused on preventing past dates, long descriptions, and invalid IDs.

For example, there was a rule that appointment dates couldn’t be in the past. So, I wrote this:

@Test(expected = IllegalArgumentException.class)

public void testPastDate() {

Calendar c = Calendar.getInstance();

c.add(Calendar.DATE, -1);

new Appointment("id1", c.getTime(), "Too late");

}

My tests weren’t just basic checks. I tested good inputs, bad inputs, and edge cases. For example:

assertEquals("Haircut", service.getAppointment("one").getNote());

This confirmed that the data saved correctly. At first, writing tests felt repetitive. But once I saw how much time I saved by catching problems early, It made sense on why we were using Junit instead of different types of testing. I also used @Before to keep my test setup clean and avoid repeating code.

Testing Techniques

I used unit testing, which means testing one piece of code at a time. It was fast and helped me stay focused. I also did some boundary testing, like checking if inputs right at the limit still worked. I didn’t use integration, system, or acceptance testing. Those are better when you’re testing how different parts work together or doing a final check before release. Unit tests are best early on. Integration tests are good when pieces start connecting. System and acceptance tests are for the big picture near the end.

Mindset

This project made me think more like a tester. I didn’t just assume things worked, I tried to break my code by giving it bad input. That helped me find bugs I wouldn’t have thought about otherwise. To avoid bias, I tried to be the annoying user who does everything wrong. I tested deleting things that didn’t exist and using descriptions that were too long. For example, I had a address that read to long ” The one ring to rule them all st” and “"1231231125" that hit me with IllegalArgumentException: Address must not be null and max 30 characters. This all together was more than 30 characters so I had to change the street name to fit the original argument. It’s easy to skip tests when you're tired, but that just causes problems later. I learned it’s better to slow down, test your stuff, and avoid future headaches.

In conclusion this project showed me that testing isn’t just a chore. It helps me code better and faster in the long run. If I were to just speed through without checking anything I would put myself in a position where I can or will be potentially behind. This can cause problems within a team and the company itself. Testing is important no matter the workload because it can defeat those pesky late-stage bugs and keeps everything running smoothly and providing you a headache free product.