Britian Holcomb

CS 320

Project Two

For the mobile app’s contact, task, and appointment services, I used a straightforward unit testing approach to ensure the code worked as expected. For the Contact Service, I focused on testing basic functions like adding, updating, and deleting contacts. I also made sure contacts had unique IDs and handled invalid inputs. For the Task Service, I checked that tasks were created properly with valid descriptions and due dates, and tested limits like description length and valid date formats. The Appointment Service tests focused on scheduling accuracy, making sure appointments didn’t overlap and that invalid date inputs were handled correctly. My testing followed the project requirements closely. For example, the Contact Service required unique IDs, so I wrote a test to check for duplicates. In the Task Service, a rule was to limit task descriptions to 50 characters, which I tested by adding descriptions of different lengths. For the Appointment Service, I tested that overlapping appointments were flagged. I know my tests were effective because I achieved over 90% code coverage, which means almost all the code was tested.

Writing the JUnit tests was a good learning experience. I made sure my code was technically correct by including specific checks in the tests. For example, in the test for adding a contact, I verified the contact was added successfully and that the details were correct. I also made my tests efficient by reusing setup methods to avoid writing the same code repeatedly, which saved time and made the tests easier to manage.

I used a few key testing techniques. Boundary testing helped me check edge cases, like testing the longest allowed task description. I also did positive and negative testing to make sure valid inputs worked and invalid ones didn’t. I analyzed code coverage to ensure I tested all the important parts of the code. Techniques I didn’t use included integration testing, which checks how different parts of the app work together, and performance testing, which looks at how the app handles heavy use.

During this project, I was careful and paid attention to details, especially when dealing with connected features. For example, I made sure task and appointment services used the same rules for date validation. I also asked my tutor to review my tests to avoid missing any mistakes. This helped me spot an edge case I hadn’t thought of, where appointments on different days could still overlap.

It’s important to stay disciplined and not skip steps when testing. Skipping tests can lead to buggy code that’s harder to fix later. For example, by sticking to the process, I avoided errors in the Contact Service that could have caused duplicate entries. To prevent issues like this in the future, I plan to use tools like automated testing and make sure all code gets reviewed carefully before it’s finished.

This project showed me how important unit testing is for making sure software works correctly. By following a clear process and being thorough, I was able to deliver reliable, high-quality code. I also learned where I can improve, like doing more integration and performance testing in future projects.