Jael Andre

CS 320

Professor Alesso

08/13/24

Summary and Reflections Report

\*\*Unit Testing Approach: \*\*

In developing the mobile application, I employed a structured unit testing approach for each of the three features: Contact, Task, and Appointment services. My primary focus was to ensure that each unit of the code—each class and method—functioned as intended. For the Contact class, the unit tests were designed to verify constraints like the uniqueness of the contact ID, and the non-nullability of fields like first name, last name, phone number, and address. Similarly, for the Task and Appointment services, the unit tests ensured compliance with the specific requirements, such as ID uniqueness, name and description constraints, and date validation for appointments.

My approach was tightly aligned with the software requirements. For instance, the Contact class required the phone number to be exactly 10 digits, a constraint that was rigorously tested with both valid and invalid inputs. I wrote test cases like `testInvalidPhone()` to confirm that the phone number met this requirement and threw an exception otherwise. This methodical approach ensured that all specified constraints were enforced, reflecting a direct alignment with the requirements

The effectiveness of the JUnit tests was validated by achieving over 80% test coverage, which indicates that the majority of the code was executed during testing. This high coverage was a result of comprehensive test cases that covered normal, edge, and error scenarios. For example, in the `TaskServiceTest`, the `testAddTask()` method was instrumental in verifying that tasks with duplicate IDs were not allowed, which was critical to maintaining the integrity of the data.

Writing these JUnit tests was both a learning experience and an exercise in meticulousness. I made sure the code was technically sound by thoroughly testing each functionality. For example, in the `testInvalidAppointmentDate()` method, I verified that the system correctly handled past dates, ensuring the `before(new Date())` check worked as expected. The efficiency of the code was also considered; for instance, using assertions like `assertThrows` to catch exceptions succinctly demonstrated both correctness and performance.

\*\*Testing Techniques:\*\*

The primary testing technique employed in this project was \*\*unit testing\*\*, characterized by testing individual components of the code in isolation. This technique is particularly useful for catching bugs early and ensuring that each part of the codebase behaves as expected independently. I focused on both positive and negative test cases to validate functionality and error handling.

Another technique that I considered but did not use was \*\*integration testing\*\*. Unlike unit testing, integration testing involves combining individual components and testing them as a group to ensure they work together. This approach would be practical in projects where the interaction between different modules or services is critical, and it would be useful in a more complex application involving databases or external APIs.

\*\*Mindset: \*\*

Throughout the project, I adopted a cautious mindset, recognizing the importance of precision in software testing. Appreciating the complexity and interrelationships of the code was crucial. For instance, when testing the Appointment service, I had to ensure that the date validation logic didn't inadvertently affect other parts of the service. This cautious approach helped prevent potential bugs that could have arisen from overlooking these relationships.

Limiting bias was another key aspect of my testing process. By writing tests that deliberately tried to "break" the code, I was able to approach the code from a user’s perspective rather than the developer’s. For example, in the `testInvalidFirstName()` method, I used invalid inputs to ensure that the system would reject them, helping to avoid the assumption that my code was flawless. This process underscored the importance of objective testing.

Lastly, the discipline to maintain high-quality code was essential. Cutting corners in testing could lead to technical debt, where issues accumulate over time, making the system harder to maintain and evolve. For example, by rigorously testing the constraints on ID fields across all services, I avoided potential future issues related to data integrity. Moving forward, I plan to maintain this discipline by consistently applying thorough testing and code reviews to mitigate risks and prevent the accumulation of technical debt.

In conclusion, this project not only enhanced my technical skills in unit testing but also reinforced the importance of a meticulous and unbiased approach to software development. By committing to these principles, I aim to ensure that my future projects are both durable and maintainable.