Seema Chavesta

Feb 22, 2025

**Summary and Reflections Report**

**Summary**

As a software engineer, I recently completed the development of a mobile application for a customer, delivering the contact, task, and appointment services. Throughout this project, I focused on ensuring that my unit testing approach was comprehensive and aligned with the software requirements.

For each feature, I took a structured approach to unit testing. In the **Contact Service**, I ensured that all constraints, such as unique IDs, non-null values, and valid lengths, were properly tested. In the **Task Service**, I tested task creation, updates, and deletions to confirm that each function adhered to the requirements. Finally, in the **Appointment Service**, I initially used the Date class but later transitioned to LocalDate to align with modern Java standards. I also included tests to ensure that null dates were not allowed, following best practices.

My testing approach was strongly aligned with the software requirements. For instance, I validated constraints on IDs and descriptions across all services, ensuring that invalid inputs were properly rejected. The **effectiveness of my JUnit tests** was confirmed through a **coverage percentage of 87.8%**, which indicates that nearly all critical paths in the code were tested. This high coverage gives me confidence in the robustness of my implementation.

Writing the JUnit tests was an insightful experience. To ensure technical soundness, I incorporated assertions such as assertEquals, assertThrows, and assertNotNull, making sure that the expected behaviors were consistently met. For instance, in the AppointmentServiceTest, I validated that past dates were correctly rejected using:

assertThrows(IllegalArgumentException.class, () -> new Appointment("12345", pastDate, "Doctor Visit"));

To improve efficiency, I structured my tests to avoid redundant code. I used @BeforeEach to initialize common objects, reducing repetition. This helped keep the test suite maintainable and clear.

**Reflection**

**Testing Techniques**

In this project, I employed **unit testing** and **exception testing** to verify the correctness of my code. Unit testing allowed me to validate each individual function in isolation, while exception testing helped ensure that invalid inputs were properly handled. These techniques improved the reliability of the application and helped catch errors early.

Other testing techniques that I did not use in this project include **integration testing** and **system testing**. While unit tests focus on small, isolated components, integration testing verifies interactions between multiple components, and system testing evaluates the entire application. These could be beneficial for a more complex system where different services interact.

Each testing technique has its place in software development. **Unit testing** is crucial for catching errors at the function level, **integration testing** ensures that components work well together, and **system testing** provides a high-level verification before deployment.

**Mindset**

Throughout this project, I adopted a **cautious and detail-oriented mindset**. I recognized that even small changes in code could lead to unintended consequences. One specific example was ensuring that updates in the AppointmentService did not allow past dates, which could have caused logical errors in scheduling.

I also made a conscious effort to **limit bias** while reviewing my own code. Testing my own work required me to step back and think critically about potential failure points. I made sure to write negative test cases, such as attempting to create an appointment with a null description, to avoid assumptions about correctness.

Being disciplined in writing and testing code is essential in software engineering. Cutting corners can lead to long-term technical debt, making future maintenance difficult. To avoid this, I plan to continue emphasizing rigorous testing practices, code reviews, and refactoring when necessary.

This project reinforced the importance of thorough testing and maintaining high-quality code. Moving forward, I will continue refining my testing strategies to improve efficiency and reliability in future projects.