### **Summary and Reflections Report**

#### **Summary**

**Unit Testing Approach**During the development of the contact, task, and appointment services for the mobile application, I employed a thorough unit testing approach aimed at ensuring each service met the defined requirements and behaved correctly in various scenarios. Here's a brief overview of the approach for each service:

* **Contact Service**: I created unit tests to validate constraints on the contact ID, first name, last name, phone number, and address. The tests focused on operations like adding, updating, and deleting contacts, ensuring that all constraints were respected.
* **Task Service**: Unit tests for the Task class focused on validating the ID, name, and description fields, ensuring they adhered to the length constraints. The tests also verified that tasks were added, deleted, and updated correctly, maintaining data integrity.
* **Appointment Service**: For the Appointment class, I wrote tests to validate the appointment ID, date, and description. These tests ensured that dates were not set in the past and that all constraints were met. The AppointmentService tests managed appointments, ensuring that updates and deletions were handled correctly.

**Alignment with Software Requirements**My unit testing approach was closely aligned with the software requirements. Each test was designed to validate specific requirements, such as the contact ID length constraint, which I tested using assertions. For instance:

assertThrows(IllegalArgumentException.class, () -> {

new Contact("12345678901", "John", "Doe", "1234567890", "123 Elm St");

});

This methodical testing ensured that the application met the specified requirements and functioned as intended.

**Overall Quality of JUnit Tests**The overall quality of my JUnit tests is reflected in the coverage percentage, which was over 80% for all services. High coverage indicates that the majority of the code paths were tested, allowing me to identify and address potential issues early, reducing the likelihood of bugs in the final product.

**Experience Writing JUnit Tests**Writing JUnit tests was a valuable learning experience, pushing me to consider edge cases and the robustness of my code. I ensured that my code was technically sound by using assertions to validate expected behavior. For instance:

assertFalse(service.addTask(task2)); // Ensures duplicate task IDs are not allowed

I also focused on efficiency by avoiding redundant tests and keeping the code concise and targeted.

#### **Reflection**

**Testing Techniques**I primarily used unit testing and boundary testing. Unit testing allowed me to test individual components in isolation, while boundary testing ensured that the system handled edge cases effectively. These techniques were practical for this project, providing confidence in the code's correctness before integration.

Other testing techniques I did not use include integration testing and system testing. Integration testing focuses on interactions between integrated components, and system testing evaluates the system as a whole. These techniques are typically used later in the development process, once unit tests have validated individual components.

**Mindset**Throughout this project, I maintained a cautious mindset, recognizing the complexity of the code and its interrelationships. This was especially important in the AppointmentService, where correct date handling was critical. For example, I tested that past dates were not allowed:

assertThrows(IllegalArgumentException.class, () -> {

new Appointment("12345", pastDate, "Doctor's appointment");

});

This attention to detail was crucial in ensuring the system's correct behavior.

To limit bias in my review, I approached testing from an end user's perspective and sought peer feedback. Developers may overlook certain errors if they assume specific inputs will never occur, leading to inadequate testing. Being disciplined in my commitment to quality is essential in avoiding technical debt. I plan to adhere to best practices like comprehensive testing, regular code reviews, and refactoring to  
  
**How can I ensure that my code, program, or software is functional and secure?**

To ensure functionality and security, I follow a disciplined approach to testing and secure coding practices. Unit tests validate individual components, while boundary tests ensure the system handles edge cases effectively. I also incorporate validation checks for inputs, enforce secure coding practices (such as parameter sanitization and exception handling), and perform code reviews. Security considerations are integrated into my development process, ensuring that common vulnerabilities (like input validation errors) are addressed early.

#### **How do I interpret user needs and incorporate them into a program?**

To interpret user needs, I focus on clear communication with stakeholders and gathering detailed requirements. I translate these needs into functional specifications, ensuring that the software aligns with the user's goals. By using an iterative development process, I continually test and refine the software, ensuring that it remains aligned with user expectations throughout the project lifecycle.

#### **How do I approach designing software?**

When designing software, I start with a clear understanding of the requirements and ensure that the architecture is modular, scalable, and maintainable. I break down the software into components or services, design with reusability in mind, and choose appropriate data structures and algorithms. I also apply best practices such as adhering to design patterns and principles like SOLID to create a well-structured and adaptable solution.