**Summary and Reflections Report**

**Summary**

**Unit Testing Approach for Each Feature**

In developing the contact service, I focused on ensuring that contact information adhered to constraints, such as validating the contact ID, first name, last name, phone number, and address. Unit tests verified the creation, update, and deletion functionalities, directly aligning with software requirements. For example, tests ensured that the contact ID was neither null nor exceeded ten characters, and that phone numbers contained exactly ten digits.

Similarly, in the task service, I validated the task name and description to ensure compliance with requirements. Tests checked for null values and enforced character limits of 20 characters for the name and 50 characters for the description. By systematically handling various invalid inputs, I ensured that the service could accurately manage task creation, updates, and deletions.

**Alignment to Software Requirements**

My unit testing approach was meticulously aligned with the software requirements. Each test was designed to validate specific constraints, capturing all edge cases and potential input errors. For instance, the contact service tests focused on ensuring that all fields met their respective length and nullability requirements, directly aligning with project specifications.

**Overall Quality of JUnit Tests**

The quality of my JUnit tests is demonstrated by the high coverage percentage across all features. The tests covered all possible branches and handled edge cases, such as null inputs and invalid lengths. The contact service tests ensured that all setters and constructors were adequately tested, with assertions checking for exceptions where necessary. The high coverage percentage indicates that the tests were effective in identifying potential issues before they could manifest in the production environment.

**Experience Writing JUnit Tests**

Writing JUnit tests for this project was challenging but rewarding. Ensuring that all edge cases were covered, particularly for input validation, required a methodical approach. I used the assertThrows method to ensure that invalid inputs, such as null values or overly long strings, resulted in appropriate exceptions. This approach ensured that my code was technically sound and handled unexpected inputs gracefully. Additionally, I utilized parameterized tests to reduce redundancy and increase efficiency, enabling me to test multiple scenarios with a single method.

**Reflection**

**Testing Techniques**

I primarily employed unit testing techniques, focusing on validating individual methods and their interactions with the rest of the code. Characteristics of these techniques included testing isolated components to ensure they behaved correctly under various conditions. For example, I tested individual setter methods in the task service to confirm they handled valid and invalid inputs appropriately. While unit testing was the focus, I did not employ other techniques such as integration testing and system testing, which would involve testing multiple components together and the entire system, respectively.

**Mindset**

Throughout this project, I adopted a cautious mindset, recognizing the complexity of the code and its interdependencies. For example, when testing the task service, I was mindful of how changes to the task name or description could impact other parts of the system. This caution was essential to avoid introducing bugs or overlooking potential issues.

To limit bias, I approached testing with an objective mindset, ensuring that I did not assume my code was flawless. Writing tests that challenged my assumptions, such as testing for null inputs, helped me identify potential flaws that I might have otherwise overlooked.

As a software engineering professional, it is crucial to maintain a strong commitment to quality. Cutting corners in testing can lead to significant issues, resulting in technical debt that is costly to resolve. In this project, I avoided technical debt by thoroughly testing each feature, ensuring that all requirements were met and that the code was both robust and maintainable. Going forward, I plan to continue this disciplined approach, ensuring that I always prioritize quality in both development and testing.

**References**

**Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019). Software testing: An ISTQB-BCS certified tester foundation guide (4th ed.). BCS Learning & Development Limited.**

**Boni Garcia. (2017). *Mastering Software Testing with JUnit 5 : A Comprehensive, Hands-on Guide on Unit Testing Framework for Java Programming Language*. Packt Publishing.**

***Junit - Test Framework*. Tutorialspoint. (2024). https://www.tutorialspoint.com/junit/junit\_test\_framework.htm**