Katelyn Rials

CS-320

**Project Two**

**Summary**

For every feature in the application, I structured my testing based on the requirements provided by the customer. It was crucial to ensure that each requirement had corresponding tests to validate its functionality. I aimed for comprehensive coverage of the code, ensuring that all units within it were thoroughly tested. This approach was vital to guarantee that every aspect of the code operated as intended.

I focused on testing each function to confirm its proper functionality. For instance, in lines 131-220 of the ContactServiceTest, I tested the update feature, verifying that each function correctly updated the relevant variable with the appropriate information. This attention to detail ensured that the update functionality performed as expected.

Efficiency in code was another priority, and I achieved it by ensuring that the code executed only necessary actions without any extraneous additions or errors. This efficiency is displayed in lines 76-92 of the TaskServiceTest, where I tested the Delete Task function. Here, I confirmed that the function accurately deleted the specified information, thereby maintaining the code's efficiency and effectiveness.

**Reflection**

*Testing Techniques*

In this project, I used various software testing techniques which include unit testing, integration testing, and regression testing. Unit testing involved testing individual components of the application in isolation, ensuring their functionality according to specifications. Integration testing focused on verifying interactions between different modules to validate the application's overall behavior. Regression testing aimed to detect any unintended side effects of code changes by retesting previously implemented features.

Other software testing techniques not used in this project include acceptance, stress, and usability testing. Acceptance testing involves validating whether the application meets user expectations and business requirements. Stress testing assesses the application's performance under extreme conditions to identify potential bottlenecks or failures. Usability testing evaluates the application's ease of use and user experience.

Practical uses and implications of these techniques vary based on the project's context and requirements. For instance, unit testing is beneficial for detecting defects early in the development process and facilitating code maintenance. Integration testing ensures that individual components interact correctly within the application, preventing integration issues. Regression testing helps maintain application stability and reliability across multiple releases. Acceptance testing validates that the application meets stakeholders' expectations, while stress testing ensures its resilience under high loads. Usability testing identifies usability issues that may affect user satisfaction and adoption.

*Mindset*

Working on this project, I adopted a cautious mindset as a software tester, recognizing the importance of understanding the complexity and interrelationships of the code being tested. Appreciating these complexities helped me design comprehensive test cases and anticipate potential edge cases or failure scenarios.

As a software developer responsible for testing my own code, bias could be a concern if I fail to recognize potential blind spots or overlook alternative perspectives. For instance, I might unintentionally prioritize testing positive scenarios over negative ones. To mitigate bias, I would leverage code reviews by peers or utilize automated testing tools to provide objective feedback on my code quality and test coverage.

*Importance of Discipline in Quality Commitment*

Being disciplined in my commitment to quality as a software engineering professional is crucial to ensure the reliability and maintainability of the software I develop. Cutting corners when writing or testing code can lead to technical debt, increasing the likelihood of future bugs, maintenance challenges, and reduced overall productivity.

For example, neglecting thorough unit testing may result in undetected defects that surface later in the development lifecycle, requiring more time and resources to fix. Similarly, rushing through code reviews without addressing potential issues can lead to the accumulation of technical debt, hindering code maintainability and scalability over time.

To avoid technical debt, I plan to prioritize quality in every stage of the software development process, from requirements analysis to deployment. This includes allocating sufficient time for thorough testing, conducting code reviews regularly, and refactoring code to maintain its cleanliness and readability. By investing in quality upfront, I aim to minimize the need for costly rework and ensure the long-term success of the software products I contribute to.