**Summary and Reflections Report**

For Project One, I implemented a unit testing approach to ensure the reliability and functionality of the mobile application's contact, task, and appointment services. Each feature was tested comprehensively, covering various scenarios and edge cases. The unit testing approach closely aligned with the software requirements, validating the correct execution of operations such as contact creation, task assignment, and appointment scheduling. The overall quality of the JUnit tests was high, with extensive coverage and clear assertions. Writing the tests using a test-driven development (TDD) approach enhanced the development process, aiding bug identification and code improvement. Measures were taken to ensure technical soundness, such as input validation, error handling, and adherence to coding best practices. The code was also designed for efficiency, minimizing unnecessary computations, and leveraging appropriate data structures.

To give specific evidence using the Contact project requirement, I developed unit tests to cover the creation, retrieval, updating, and deletion of contacts. The tests focused on verifying the correct execution of these operations and handling various scenarios, such as empty fields, duplicate entries, and boundary cases.

The unit testing approach aligned closely with the software requirements for contact services because the tests validated that contacts could be managed effectively, and the desired functionalities, such as adding and updating contact information, were implemented correctly. For example, the requirement stated that contacts must have a unique Contact ID, and the corresponding test case checked if the system correctly rejected duplicate entries.

The experience of writing JUnit tests for Project One was valuable and educational. It allowed me to gain a deeper understanding of the application's behavior and early detection of potential issues. By following a TDD workflow, writing tests before implementing functionalities, I was able to clarify requirements and improve code design.

I ensured technical soundness by performing thorough validation of inputs, error handling, and adhering to coding best practices. For example, in the Appointment Services Test, the following line of code checks that an Illegal Argument Exception is thrown when attempting to delete a non-existent appointment.



I aimed for efficient code by minimizing unnecessary computations and leveraging appropriate data structures. In the Contact Services Test, the following line of code verifies that a new contact is added correctly when adding users.

A picture containing screenshot, text, font

Description automatically generated

Throughout the project, I utilized various software testing techniques to ensure the quality and functionality of the mobile application. One of the key techniques employed was unit testing, which involved testing individual code units in isolation. Unit testing allowed me to verify the correctness and behavior of small units such as functions or methods. By isolating dependencies through mocking and stubbing, I could focus solely on the unit under test, enabling faster and more frequent test execution. The use of assertions and expected outcomes ensured that the units performed as expected, enhancing the reliability of the application.

While the project extensively utilized unit testing and mocking techniques, other testing techniques such as integration testing and acceptance testing were not specifically implemented.

Integration testing focuses on verifying the interactions and interfaces between different components or modules of the system. It ensures that the components work together effectively as a whole system and identifies issues related to component integration, data flow, and interface functionality. Acceptance testing involves testing the software in realistic user scenarios, mimicking real-world user interactions and usage scenarios.

Throughout the project, I approached the task with a meticulous mindset as a software tester. I understood the importance of being cautious and paying attention to detail to ensure the reliability and quality of the code. I exercised caution in several ways. Firstly, I thoroughly reviewed the software requirements to gain a deep understanding of the expected functionality and behavior. This helped me design effective test cases that covered a wide range of scenarios. Secondly, I focused on testing edge cases and boundary conditions. By considering extreme input values and boundary limits, I aimed to uncover any unexpected behaviors or vulnerabilities in the code.

For instance, when testing the Task Services, I paid attention when creating the Add Task function. By considering the interrelationship between task creation and updating tasks functionality, I crafted tests that covered cases where the tasks are correctly added and that the proper fields can be updated based on the specified requirements.

In conducting the code review, I made conscious efforts to limit bias and maintain objectivity. To mitigate bias, I focused on the requirements and specifications while reviewing the code. This approach allowed me to evaluate the code against the intended functionality rather than my personal preferences. In a professional setting, I can imagine if I were responsible for testing my own code as a software developer, bias could indeed be a concern. Being personally invested in the code might inadvertently lead me to overlook potential issues or gaps. Also, due to familiarity with the codebase, I might overlook certain aspects or assume that certain parts of the code are error-free. This bias can prevent me from thoroughly testing those areas and identifying potential defects.

Discipline in upholding a commitment to quality is very important in the field of software engineering. Cutting corners when writing or testing code can result in a host of negative consequences. It can lead to increased technical debt. By rushing or compromising on code quality, I would accumulate technical debt, making future development and maintenance more challenging and costly. Also, neglecting proper testing and quality assurance would reduce the reliability and stability of the software. Undetected bugs and issues could lead to software failures, poor user experiences, and damage to the reputation of the product or company.

To avoid technical debt and maintain a high level of quality, I plan to adhere to a set of practices. Firstly, I will continue to follow established coding best practices, such as writing clean and readable code, adhering to naming conventions, and implementing modular and reusable components. Secondly, I will prioritize comprehensive testing, including unit tests, integration tests, and acceptance tests, to ensure thorough coverage and validation of the software's functionality.

Resources:

García, Boni. (2017). Mastering software testing with JUnit 5. Packt Publishing.