**Abigail Sibazeu**

**CS-320**

**June 21, 2024**

**Project Two: Reflection**

In this report, I will summarize my unit testing approach for the Contact Service, Appointment Service, and Task Service features developed for a mobile application. Additionally, I will reflect on my experience writing JUnit tests and the quality of those tests.

The unit testing approach for each feature was comprehensive and methodical. For the Contact Service, the tests were designed to cover the primary functionalities of adding, deleting, and updating contacts. Each test method instantiated the ContactService and created a Contact object with valid data. The tests verified the functionality by using assertions such as assertNotNull for additions and assertNull for deletions. This ensured that the contact was correctly added to or removed from the service.

Similarly, the Appointment Service's unit tests focused on the core operations of adding, deleting, and retrieving appointments. The tests created AppointmentService instances and Appointment objects with future dates to validate the functionality. Assertions like assertNotNull and assertNull were used to confirm that the appointments were correctly added and deleted, respectively.

For the Task Service, the tests included methods for adding, deleting, and updating tasks. The TaskService was initialized along with Task objects. The tests verified the presence or absence of tasks using assertNotNull and assertNull assertions, ensuring the tasks were correctly managed by the service.

The unit testing approach was well-aligned with the software requirements. The tests ensured that all core functionalities—adding, deleting, and updating—were thoroughly tested. Each test case corresponded directly to a specific requirement, verifying that the service methods performed as expected. This alignment ensured that the application met the specified requirements and provided the intended functionalities to the user.

The quality of the JUnit tests can be defended by their high coverage percentage. The tests covered all critical methods and scenarios, providing comprehensive validation of the service functionalities. For instance, the ContactServiceTest included tests for each major operation: adding, deleting, and updating contacts. This thorough coverage ensured that any potential issues were identified and addressed during the testing phase, contributing to the overall reliability of the application. Writing the JUnit tests involved creating clear and concise test cases to ensure technically sound and efficient code. The process required a deep understanding of the service functionalities and the ability to anticipate potential issues. For example, in ContactServiceTest.java, the following test case ensured that the addContact method functioned correctly:

@Test

public void testAddContact() {

ContactService service = new ContactService();

Contact contact = new Contact("1234567890", "Julien", "Stewart", "0123456789", "123 Main St");

service.addContact(contact);

assertNotNull(service.getContact("1234567890"));

}

This test verified that the contact was not null after being added, ensuring that the addContact method performed as expected.

The primary testing techniques employed in this project included unit testing with assertions such as assertNotNull and assertNull to verify the presence and correctness of service operations. These techniques ensured that individual units of code worked as intended and provided a solid foundation for identifying and fixing issues early in the development process. Other testing techniques that were not employed in this project include integration testing and system testing. Integration testing involves combining multiple units to test their interactions, which can help identify interface issues between different parts of the application. System testing evaluates the entire system against the specified requirements, providing a final validation before deployment.

Each of these techniques has practical uses and implications for different software development projects. Unit testing is ideal for early-stage testing and debugging, while integration testing is essential for validating interactions between components. System testing ensures that the complete system meets the specified requirements and is critical for final validation before deployment.

Adopting a cautious mindset throughout the project was crucial for ensuring thorough validation of each service. I appreciated the complexity and interrelationships of the code. For example, ensuring that adding a contact did not inadvertently affect other contacts required my careful consideration and testing. This cautious approach helped me identify potential issues early and ensured the reliability of the application.

Limiting bias in my review of the code was another important aspect. Writing tests that covered both positive and negative scenarios helped me ensure comprehensive coverage. For instance, testing both valid and invalid inputs provided a more complete validation of the service functionalities. As a software developer, I recognize that self-testing can introduce bias, but peer reviews and automated tests help mitigate this concern. Maintaining discipline in my commitment to quality is vital to avoid technical debt. Cutting corners in writing or testing code can lead to long-term maintenance issues. As a practitioner, maintaining comprehensive test suites and adhering to best practices helps me avoid technical debt. Regularly updating tests to cover new features and refactoring code to improve quality ensures ongoing code quality and reduces the risk of future issues.

This summary and reflections report encapsulates my unit testing approach, alignment to requirements, quality of tests, and the mindset I adopted during the project. The testing strategies I employed ensured robust and reliable service functionalities, reflecting a disciplined approach to software engineering. By maintaining a high standard of quality in testing and development, I can deliver reliable software solutions that meet the needs of our customers.

**References**

Beck, K. (2003). *Test Driven Development: By Example*. Addison-Wesley Professional.

Gamma, E., & Beck, K. (2019). *JUnit 5 User Guide*. JUnit.

Meszaros, G. (2007). *xUnit Test Patterns: Refactoring Test Code*. Addison-Wesley Professional.

Myers, G. J. (2004). *The Art of Software Testing*. John Wiley & Sons.