7-2 Project 2: Summary and Reflections Report

Armon Wilson

SNHU

CS-320 Software Test and Automation

**Summary and Reflections Report:**

This report summarizes the unit testing approach employed for the Contact, Task, and Appointment services developed for a mobile application in Project One. It details the testing strategies, code quality, and overall effectiveness of the JUnit tests, along with reflections on testing techniques and the mindset adopted during the project.

**Summary:**

**Unit Testing Approach**

The unit testing approach for all three services (Contact, Task, and Appointment) focused on verifying individual components in isolation. Each public method within the classes was tested for expected behavior, including both valid and invalid inputs. This approach ensured that each unit functioned correctly, contributing to the overall reliability of the application.

This approach aligns with the software requirements by ensuring that each service adheres to the specified constraints and functionalities. For instance, the Contact class has specific length restrictions for fields like firstName, lastName, and address. The JUnit tests validate these constraints by testing both valid and invalid inputs, as demonstrated in the ContactTest class.

A computer screen shot of a black background

Description automatically generated

**JUnit Test Quality**

The overall quality of the JUnit tests is high. The tests cover a wide range of scenarios, including boundary conditions, invalid inputs, and expected exceptions. The use of assertions (e.g., Assertions.assertTrue, Assertions.assertThrows) ensures that the code behaves as expected. The tests demonstrate a comprehensive approach to validating the functionality and error handling of each service, all with 100% coverage.

**JUnit Test Writing Experience**

To ensure the code was technically sound, each test case was designed to target specific functionalities and potential error conditions. For example, in the AppointmentTest class, the setDateTest method specifically tests the setDate method in the Appointment class by providing various date formats and checking for expected exceptions (e.g., IllegalArgumentException for past dates, NullPointerException for null dates).

A screenshot of a computer program

Description automatically generatedEfficiency in the JUnit tests was maintained by focusing on individual units and avoiding unnecessary dependencies or complex setups. Each test case aimed to be concise and focused on a specific aspect of the code. This is evident in the TaskTest class, where methods like nullTasksTest and taskLongTest efficiently test various invalid input scenarios.

**Reflection:**

**Testing Techniques**

The primary software testing technique employed in this project was unittesting. This technique involves testing individual units or components of the software in isolation to ensure they function correctly. Its characteristics include isolation of units, focus on code functionality, and use of automated testing frameworks like JUnit. Other techniques not used in this project include:

* IntegrationTesting**:** Testing the interaction between different units or modules.
* SystemTesting**:** Testing the entire system as a whole.
* UserAcceptanceTesting **(**UAT**):** Testing by end-users to ensure the software meets their requirements.

**Practical Uses and Implications**

* Unit testing is crucial for early defect detection and ensuring individual components work correctly.
* Integration testing is essential for verifying the interaction between different parts of the system.
* System testing validates the overall functionality and performance of the application.
* UAT ensures the software meets the user's needs and expectations.

**Mindset**

A screen shot of a computer program

Description automatically generatedA screen shot of a computer program

Description automatically generatedA computer screen shot of a program code

Description automatically generatedAs a software tester, I adopted a cautious mindset. I recognized the potential for errors and unexpected behavior in the code. Appreciating the complexity and interrelationships within the code was crucial for designing effective test cases. For example, in the AppointmentService class, understanding the interaction between the addAppointment and getAppointment methods was essential for testing the service's functionality.

To limit bias, I focused on testing against the defined requirements and specifications, rather than making assumptions about the code's behavior. If I were testing my own code, bias could be a significant concern, as I might subconsciously overlook potential flaws or favor certain functionalities.

**Discipline and Quality**

Cutting corners in writing or testing code can lead to technical debt, resulting in increased maintenance costs and potential system failures down the line. To avoid this, I plan to adhere to best practices, write comprehensive tests, and prioritize code reviews. For instance, consistently validating inputs and using clear, descriptive variable names can improve code maintainability and reduce the likelihood of errors.

References:

*No references utilized.*