Nawras Janoudi

CS 320 – Software Test/Automation

04/20/2025

Project 2

**Summary and Reflection on Project 1 (Grand Strand Systems)**

The unit testing approach for Project One was applied to the Contact, Task, and Appointment services, focusing on validating data constraints and service functionalities in alignment with software requirements. For the Contact service, tests were designed to enforce constructor and setter method constraints, ensuring that data inputs adhere to defined rules (e.g., handling null or excessively long names, validating phone number formats). Service operations, such as adding, updating, and deleting contacts, were also validated to confirm that they function as specified. This approach ensured data integrity and proper service behavior, directly fulfilling the requirements. The Task service followed a similar strategy, validating data constraints for task names and descriptions, and verifying core service operations like adding, updating, deleting, and retrieving tasks. This ensured that data was handled correctly and service functionalities operated as expected. For the Appointment service, testing focused on validating appointment creation and management. Tests verified constructor constraints related to appointment dates and descriptions, and service operations for adding, deleting, and retrieving appointments. This confirmed that appointment management functions correctly.

The overall quality of the JUnit tests is demonstrated by their comprehensive coverage of the code and their ability to effectively identify potential issues. The tests that were created utilize assertions (assertEquals(), assertThrows(), assertNotNull(), assertNull(), assertNotEquals()) to verify expected outcomes. Tests cover both positive and negative scenarios, including valid inputs, invalid inputs, and edge cases, demonstrating thoroughness and increasing confidence in the code's reliability.

Ensuring code reliability was a primary focus during the JUnit testing process. This was achieved by writing tests that validate specific behaviors and constraints of the code. For example, tests were implemented to ensure that the Contact service throws exceptions when invalid data, such as phone numbers with more than 10 digits, is provided. Similarly, tests confirmed that the Task service correctly handles exceptions when attempting to update non-existent tasks. In the Appointment service, tests verified that appropriate exceptions are thrown when attempting to delete null appointments. These tests collectively validate the robustness of the code. Code efficiency in testing was achieved by creating focused tests that target specific units of code and avoid unnecessary complexity. Test methods were designed to test single aspects of classes, promoting simplicity and clarity. The @BeforeEach annotation was used to set up objects before each test, reducing redundancy and improving efficiency. This pattern is consistent across the Task and Appointment service tests, where tests are focused, contributing to an efficient testing process. The primary testing technique in Project One was unit testing with JUnit. Unit testing involves testing individual units of code in isolation to verify that each unit behaves as expected. JUnit's annotations and assertion methods were helpful in facilitating this. Black-box testing was also indirectly used, as tests were often written based on the requirements and specifications, without delving into the internal implementation details. For example, tests validating data constraints in the services are black-box tests because they focus on input and output. However, other testing techniques were not utilized. Integration testing, which focuses on testing the interaction between different units or modules, was not defined. While service tests implicitly test interactions within those services, explicit integration tests between services were not created. System testing, which tests the entire system as a whole, and acceptance testing, performed by end-users, were also not included in the project scope.

Each testing technique has specific practical uses and implications. Unit testing is crucial for early bug detection and code maintainability but requires developers to invest time in writing tests. Black-box testing is valuable for validating software against requirements and ensuring user satisfaction but may not uncover all internal errors. Integration testing is essential for complex systems to verify that components work together but can be time-consuming. System testing ensures that the entire system meets requirements but requires a complete system. Acceptance testing ensures user satisfaction but requires user involvement.

Adopting a cautious mindset was crucial throughout the testing process. This involved carefully considering potential failure points and edge cases. For instance, tests were designed to validate all possible invalid inputs in the Contact and Appointment services. Appreciating the complexity and interrelationships of the code was also vital. For example, the dependency of update methods on get methods highlighted the need to test both thoroughly.

Limiting bias was another critical aspect of the testing process. This was achieved by focusing on requirements and specifications rather than implementation details. However, the potential for bias is a significant concern when developers test their own code. Developers may overlook errors due to assumptions or familiarity with the code. Code reviews and testing by other developers are essential to mitigate this bias.

Finally, discipline and a commitment to quality are essential in software engineering. Cutting corners can lead to technical debt, increased debugging, and reduced reliability. For example, neglecting to test invalid inputs could lead to production bugs. To avoid technical debt, I plan to adhere to coding standards, write clean code, conduct thorough testing, perform code reviews, and refactor regularly.

In conclusion, the unit testing approach in Project One was effective, and the JUnit tests demonstrated thoroughness and quality. The reflection on testing techniques and mindset underscores the importance of a disciplined and cautious approach to software testing, emphasizing its crucial role in ensuring software reliability and maintainability.

***References***

*Introduction to JUnit 5*. (2024, January 3). GeeksforGeeks. https://www.geeksforgeeks.org/introduction-to-junit-5/

‌GarcíaB. (2017). *Mastering software testing with JUnit 5 : comprehensive guide to develop high quality Java applications*. Packt Publishing.