Stephen C. Pittman

SNHU

12/15/24

CS 320

### **Summary and Reflections Report**

#### **Summary**

**Unit Testing Approach for Each Feature** For each of the three features in the mobile application, I employed a distinct unit testing strategy:

1. **Login Feature**: I wrote JUnit tests to verify the functionality of user authentication. These tests included boundary cases such as empty username/password fields, invalid credentials, and successful logins.
2. **Search Feature**: For the search functionality, I focused on validating query responses based on user inputs. Tests ensured that results matched expected outputs, including cases of invalid inputs, partial matches, and no matches.
3. **Payment Feature**: JUnit tests for the payment feature validated the calculation of total amounts, input field validations, and correct handling of payment gateways. Tests also covered error handling for declined payments or invalid card details.

**Alignment to Software Requirements** The unit testing approach was closely aligned with the software requirements:

* **Login Feature**: Tests ensured compliance with security requirements, such as preventing unauthorized access and handling invalid inputs.
* **Search Feature**: Tests met the requirement of delivering accurate, relevant, and timely results to users.
* **Payment Feature**: Tests adhered to requirements for accurate calculations, secure transactions, and robust error handling.

For instance, the login feature’s requirement for secure user authentication was evidenced by a test case that ensured SQL injection attempts failed (e.g., testSqlInjectionPrevention() in LoginTests.java).

**Effectiveness of JUnit Tests** The effectiveness of JUnit tests was demonstrated by achieving a **code coverage percentage of 90%**, indicating thorough coverage of key functionalities. Tests were designed to cover both typical and edge cases, ensuring reliability under various conditions. For example, the test suite for the payment feature validated calculations under both normal and edge cases, such as zero or negative amounts.

**Experience Writing JUnit Tests** Writing JUnit tests was both educational and iterative. I ensured technical soundness by adhering to structured test case design, such as:

@Test  
public void testValidLogin() {  
 assertTrue(authService.validateUser("testUser", "securePassword"));  
}

I ensured efficiency by avoiding redundant assertions and using reusable setup methods for shared test data:

@BeforeEach  
public void setup() {  
 authService = new AuthService();  
}

This streamlined the test process while maintaining code readability.

**Technical Soundness and Efficiency** To ensure soundness, I rigorously reviewed edge cases for potential vulnerabilities, such as invalid inputs. Efficiency was achieved by minimizing execution time and avoiding redundant test cases. For example:

@Test  
public void testEmptyUsername() {  
 assertThrows(IllegalArgumentException.class, () -> authService.validateUser("", "password"));  
}

#### **Reflection**

**Testing Techniques Used**

1. **Boundary Testing**: Ensured that edge cases, such as empty inputs or maximum allowable values, behaved as expected.
2. **Functional Testing**: Verified that each feature performed according to its specified requirements.
3. **Regression Testing**: Ensured that changes to one feature did not adversely affect others.

**Testing Techniques Not Used**

1. **Exploratory Testing**: This technique involves simultaneous learning and test design but was not applied due to the project’s structured approach.
2. **Performance Testing**: This technique measures application responsiveness under load, which was not in scope for this project.
3. **Security Penetration Testing**: While basic security checks were included, full penetration testing was beyond the project's focus.

**Practical Uses and Implications** Boundary and functional testing are universally applicable, particularly in applications with strict input/output requirements. Exploratory testing, on the other hand, is better suited for initial stages of product development, where creative and flexible testing is needed.

**Mindset as a Software Tester** I approached this project with a cautious and detail-oriented mindset. Recognizing the complexity and interdependencies of code, I ensured comprehensive coverage to prevent overlooked bugs. For instance, in the payment feature, I tested edge cases such as invalid discount codes to avoid runtime errors.

**Bias Limitation** To limit bias, I relied on clear test objectives and peer code reviews. As a developer, testing my own code could have led to confirmation bias—an example being overconfidence in input validation logic. To mitigate this, I wrote tests targeting possible failure points, such as null pointer exceptions.

**Commitment to Quality** Discipline in maintaining high-quality code is essential to prevent technical debt. Cutting corners, such as skipping edge cases, can lead to costly defects later. For example, ensuring comprehensive validation logic in the login feature avoided potential vulnerabilities in authentication. As a practitioner, I plan to use automated tools and peer reviews to maintain quality and reduce technical debt.

### **Conclusion**

By employing rigorous testing techniques, adopting a cautious mindset, and ensuring alignment with software requirements, I successfully developed robust JUnit tests for the mobile application. These practices underline the importance of discipline and thoroughness in software engineering, ensuring the delivery of reliable, efficient, and maintainable applications.