**Summary and Reflections Report**

Matthew Tyson

Southern New Hampshire University

CS 320: Software Test Automation & QA

Jonathan Norman

June 17, 2023

**Summary and Reflections Report**

In this report, I am a software engineer for Grand Strand Systems, a software engineering company that focuses on developing and testing back-end services. I recently completed an assignment in which I developed a mobile application for a customer and delivered the contact, task, and appointment services.

I will provide a summary and reflections on my unit testing approach, experience writing JUnit tests, and the overall quality of the tests conducted during the development of the mobile application for our customer. This report will also discuss the testing techniques employed and the mindset I adopted during the project.

**Summary**

The project involved developing a mobile application for a customer, focusing on implementing contact, task, and appointment services. As a software engineer at Grand Strand Systems, I was responsible for the development and testing of the back-end services. The project aimed to meet the specific requirements outlined by the customer and deliver a high-quality mobile application.

***The Approach***

For each of the three features (contact, task, and appointment services), I followed a systematic unit testing approach. This involved identifying the individual components and functionalities of each feature and designing test cases to verify their behavior. I used JUnit, a popular Java testing framework, to implement and execute the tests.

**Alignment to Requirements.** My unit testing approach was highly aligned with the software requirements. I ensured that each test case addressed a specific requirement or functionality outlined in the project specifications. By mapping the requirements to the test cases, I could verify that the implemented features met the intended objectives.

**Effective Tests.** The overall quality of my JUnit tests was high. I measured the effectiveness of the tests by calculating the coverage percentage, which indicates the proportion of code covered by the tests. I aimed for comprehensive coverage to identify and rectify any potential defects or issues. By reviewing the coverage reports generated by the testing framework, I confirmed that the majority of the code was adequately tested.

***Experience***

Writing JUnit tests was a valuable learning experience for me. I gained proficiency in structuring test cases, setting up test fixtures, and asserting expected results. I followed the Arrange-Act-Assert (AAA) pattern to ensure the tests were readable, maintainable, and easy to understand (Kralj, 2023). Regularly writing tests improved my code comprehension and helped me identify potential flaws early in the development process.

**Technically Sound Code.** To ensure the technical soundness of the code, I adopted several best practices while writing the JUnit tests. For example, in the contact service tests, I verified that the contact creation, retrieval, and modification operations were functioning correctly. By inspecting specific lines of code in the tests, such as the assertions that validate the expected results, I ensured the code adhered to the required specifications and performed as intended.

**Efficient Code.** Efficiency was a crucial aspect of the code I developed, including the JUnit tests. I strived to minimize redundant operations and unnecessary computations. For instance, in the appointment service tests, I carefully selected test data that covered different scenarios to maximize the effectiveness of the tests while keeping the execution time within acceptable limits.

**Reflection**

During the project, I had the opportunity to reflect on my approach to software testing and the lessons learned from the experience. It was a valuable exercise that allowed me to assess my strengths and areas for improvement in delivering high-quality software solutions. Through this reflection, I gained insights into the importance of comprehensive testing, effective utilization of testing techniques, maintaining a balanced mindset, and upholding discipline in software engineering practices.

***Testing Techniques***

In this project, a variety of testing techniques were employed to ensure the accuracy and reliability of the developed mobile application. Techniques such as functional testing, boundary value analysis, and equivalence partitioning were utilized extensively. Functional testing involved validating the expected behavior of individual features, ensuring they functioned correctly according to the defined requirements (McGovern et al., 2003a). Boundary value analysis helped identify and test critical boundary conditions, enabling better coverage and identification of potential defects (Limited, 2023). Equivalence partitioning aided in grouping input values into equivalent classes and selecting representative test cases to minimize redundancy (Rajkumar, 2023).

**Techniques Employed.** The testing techniques employed in this project encompassed both black-box and white-box approaches (McGovern et al., 2003a). Black-box testing techniques were utilized to validate the functionalities from an end-user perspective, focusing on the input-output behavior and the expected results. This included functional testing, where different scenarios were tested to ensure the desired outcomes were achieved. White-box testing techniques were also employed, examining the internal structure and logic of the code. This included code coverage analysis using tools like JUnit, which helped identify areas of the code that were not adequately tested and facilitated improvements in the test suite.

**Other Techniques.** Although the project incorporated several effective testing techniques, certain techniques such as performance testing, security testing, and usability testing were not employed. Performance testing evaluates the responsiveness and scalability of the system under different loads and stress conditions. Security testing focuses on identifying vulnerabilities and weaknesses in the software's security measures. Usability testing assesses the user-friendliness and intuitiveness of the application from the end-user's perspective. While these techniques were not utilized in this specific project, their importance should not be overlooked, as they play significant roles in ensuring a comprehensive evaluation of software systems in different contexts.

**Uses and Implications of Techniques.** The techniques employed in this project have wide-ranging uses and implications in various software development projects and situations. Functional testing is a fundamental technique used to verify that software features operate as intended. It helps detect defects, ensures compliance with requirements, and enhances the overall quality of the software. Boundary value analysis and equivalence partitioning assist in focusing testing efforts on critical areas and efficiently selecting representative test cases. These techniques are especially valuable when requirements are complex or numerous, allowing for thorough testing while optimizing resource utilization. Additionally, code coverage analysis and white-box testing techniques provide insights into the internal workings of the software, aiding in identifying potential vulnerabilities, improving code quality, and reducing the risk of defects.

***Mindset***

Adopting a specific mindset while working on this project was crucial for successful software testing. Among the key aspects of the mindset were caution, bias awareness, and discipline. By exercising caution, I approached testing with a meticulous and thorough mindset, ensuring that no potential issues were overlooked. Appreciating the complexity and interrelationships of the code being tested was important in understanding the system as a whole and avoiding assumptions or oversights that could lead to defects. For example, when testing the appointment service, I carefully considered different time zone scenarios, daylight saving adjustments, and edge cases to ensure accurate scheduling and consistency in handling time-related operations.

**Caution.** Caution played a significant role in mitigating risks during the testing process. It involved meticulous test case design, extensive boundary testing, and rigorous data validation. By paying close attention to details and being mindful of potential corner cases, I aimed to uncover hidden defects and ensure the stability and reliability of the software. Caution also extended to thorough error handling and exception testing, allowing for proper handling of unexpected situations and maintaining the integrity of the application.

**Bias.** Bias can be a concern when reviewing code, and it was essential to approach the testing process objectively. As a software tester, it was crucial to identify potential issues and provide constructive feedback without favoring personal preferences or preconceived notions. By focusing on the quality of the code and adherence to requirements, I minimized bias in my review and evaluation. For example, when assessing the implementation of contact retrieval functionality, I emphasized validating the correct retrieval of contact information without being swayed by personal opinions about the user interface design.

**Discipline.** Maintaining discipline in software engineering practices is vital for delivering high-quality solutions. Cutting corners during code writing or testing can lead to technical debt (*Technical Debt*, 2021), making future maintenance and enhancements challenging. Therefore, I was committed to following best practices, adhering to coding standards, and conducting comprehensive testing to ensure code correctness and robustness. By documenting and organizing test cases, maintaining clean and readable code, and regularly reviewing and refactoring the test suite, I aimed to minimize technical debt and facilitate future development and maintenance efforts.

**References**

Kralj, K. (2023, February 10). Arrange Act Assert: The Best Structure for Highly Successful Unit Tests in C#. *MethodPoet*. <https://methodpoet.com/aaa-in-unit-testing/#:~:text=In%20unit%20testing%2C%20the%20Arrange-act-assert%20%28AAA%29%20pattern%20is,step%20verifies%20that%20the%20expected%20results%20were%20achieved.>

McGovern, J., Tyagi, S., Stevens, M., & Mathew, S. (2003a). *Java Web Services Architecture*. Morgan Kaufmann.

Limited, Q. T. P. (2023). Boundary Value Analysis: The Ultimate Tool for Robust Input Validation in Software Testing. *QAble*.  <https://www.qable.io/boundary-value-analysis/#:~:text=Boundary%20Value%20Analysis%20is%20a%20popular%20technique%20for,due%20to%20incorrect%20assumptions%20about%20the%20system%20behavior.>

Rajkumar. (2023, January 1). *Equivalence Partitioning Test Case Design Technique*. Software Testing Material. <https://www.softwaretestingmaterial.com/equivalence-partitioning-testing-technique/#:~:text=Equivalence%20Partitioning%20is%20also%20known%20as%20Equivalence%20Class,from%20each%20group%20to%20design%20the%20test%20cases.>

*Technical Debt*. (2021, September 2). <https://www.productplan.com/glossary/technical-debt/>