Final Project

SNHU Final Project: Summary and Reflections Report

Tiana Dinh

Southern New Hampshire University

CS-250: Software Test, Automation

Professor Robert Tuft

20 October 2024

**Summary and Reflections Report**

**Summary**

I used a thorough unit testing strategy for every feature while I was developing the application for the client in Project One. A crucial factor to consider was how well my testing strategy matched the software specifications. In-depth information about my unit testing methodology, JUnit test efficacy and quality, my writing experiences, and the steps I took to guarantee code efficiency and technical soundness are all covered in this report.

**Unit Testing Approach**

I created unit tests for every feature that covered a range of scenarios and edge cases. These tests evaluated the software requirements' functionality, error handling, and boundary conditions in detail. I paid close attention to input validation, made sure that data was processed and output accurately, and considered how users would interact with the system.

**JUnit Testing**

A thorough test coverage analysis was used to confirm the efficacy and quality of my JUnit tests. I made sure my tests covered a broad range of possible code paths. A high percentage of coverage was found in this coverage analysis, demonstrating how thoroughly I tested various code branches and functionalities.

**JUnit Tests Writing**

I definitely had to combine meticulous planning, execution, and validation when writing JUnit tests. Using the software requirements as a guide, I began by identifying critical test scenarios. Next, for every scenario I had identified, I methodically created tests, frequently using assertions to confirm predicted results. As an example, I carried out tests to verify data transformations and contrasted the actual and expected outputs.

**Software Requirements Alignment**

My approach was as sound as I could make it in terms of aligning with the software requirements. I made an effort to develop tests that would verify that incorrect inputs would result in the proper error messages, thus directly addressing the need for error handling. Along with tests to confirm accurate data processing of various data sets, I also wanted to make sure that data processing was done correctly. In order to verify compliance with the required behavior, I also want to have assessed user inputs and system outputs.

**Technical Efficiency and Sounds**

I went over the format of my tests very carefully to make sure it was technically sound. I made sure that method calls and assertions were used to faithfully simulate event-driven interactions. In order to ensure that every test was independent of outside influences and self-contained, I also wanted to isolate dependencies. The code was kept efficient by using well-thought-out design and implementation. I used short and effective code for input validation and verification. Additionally, I wanted to concentrate on streamlining and streamlining the execution of data processing algorithms.

**Reflections**

**Techniques for Testing**

I used a range of software testing methods for the project, such as functional testing with JUnit and whitebox and blackbox testing. I was able to thoroughly verify the code's internal logic, interactions with other components, and adherence to functional requirements thanks to these techniques.

**Other Techniques for Testing**

Due to the project's limited scope and the IDE environment, I used a variety of testing techniques, but I refrained from system and non-functional testing. Non-functional testing would have looked at things like performance, scalability, and security, while system testing would have validated the behavior of the entire application as a whole.

**Implications**

The project-specific context informed the selection of testing methodologies. Internal code logic was examined in white box testing, whereas user interactions were simulated in blackbox testing. Unit-by-unit validation was made easier by functional testing using JUnit. System testing is essential for integrated applications, as it guarantees smooth component interaction, when viewed in this larger context.   
In order to guarantee the best possible performance, user experience, and security for production-grade applications, non-functional testing is essential.

**Mindset**

I approached testing with a thorough and cautious mindset. I was forced to simulate a variety of scenarios during testing because I could see how complicated the code was by itself.   
For example, in order to guarantee consistent system responses, I ran tests that simulated user inputs across various states.

**Discipline for Commitment**

In software engineering, maintaining discipline and a dedication to quality is crucial. Cutting corners runs the risk of affecting the functionality and stability of the finished product. I carefully tested each code component in my project with the goal of avoiding technical debt. I intend to maintain this discipline as a future practitioner by implementing rigorous testing and ongoing code review procedures.

**Self-Review and Bias**

I worked hard to reduce bias in the code review process. by closely adhering to software requirements and testing plans. When I test my own code, familiarity could lead to biases.  
As a result, impartial testing requires both external input and objective validation.

To sum up, I tried my hardest to be as detailed as possible when it came to software testing the application, making sure it complied with the requirements and used a range of testing methods. By methodical planning and execution, I made sure everything was technically sound and efficient. I continued to focus on thorough testing, exercise caution, reduce bias, and strive for objectivity. Maintaining a commitment to quality, I aimed to stay away from technical debt by following strict testing guidelines. These reflections have given me important new understandings of the complex relationship between software development and testing.

References

Prica, P. (2023, January 29). *Discipline within Software Engineering - Pavle Prica* - Medium. *Medium*. https://pavleprica.medium.com/discipline-within-software-engineering-36f13f870e24

Barney, N. (2024, July 29). *6 ways to reduce different types of bias in machine learning*. Enterprise AI. https://www.techtarget.com/searchenterpriseai/feature/6-ways-to-reduce-different-types-of-bias-in-machine-learning