Southern New Hampshire University

CS-320 Software Test Automation & QA

February 25, 2024

Brandon Ernesto Marrero

I recently completed Project One, which involved developing a mobile application and creating unit tests for a company called Grand Strand Systems. This application included contact, task, and appointment services with specific software requirements. I will summarize my unit testing approach, reflections on my experience writing JUnit tests, and insights into testing techniques and mindset adopted during this project.

My unit testing approach, for each of the three features - contact, task, and appointment services – ensured the functionalities aligned with the software requirements. For the contact service, I focused on validating functionalities such as addition, deletion, and modification of contacts, along with enforcing constraints like maximum lengths for properties such as contact ID, first name, last name, phone number, and address. Similarly, for the task service, I employed unit tests to validate basic CRUD operations, ensuring that task ids were no longer than 10 characters, names were no longer than 20 characters, etc. In the appointment service, I concentrated on testing the creation, cancellation, and updating of appointments, specifically ensuring that appointments could not be made for past dates, in accordance with the provided requirements.

The unit tests closely aligned with the software requirements provided by Grand Strand Systems. By testing the maximum lengths of properties and ensuring constraints such as enforcing future appointment dates, I validated that the implemented functionalities met the specified requirements.

In terms of the quality of my JUnit tests, I achieved 100% test coverage for all features of the application, indicating comprehensive testing of the codebase. The effectiveness of the JUnit tests was validated by the coverage percentage using JUnit5 in the IntelliJ IDE, ensuring that every line of code was executed during testing. Additionally, the tests successfully captured edge cases and corner scenarios, enhancing the robustness of the services.

Writing JUnit tests was a smooth experience, allowing me to programmatically validate the functionality of the services. I could execute JUnit tests for any service directly from my IDE. In the event of a test failure, the IDE displayed the expected and received values, making it easier to detect and resolve errors in my test’s logic or code. To ensure technical soundness, I followed a structured approach by testing each method with various inputs and assertions. For example, in the appointment service tests, I validated the behavior of the create appointment method by providing different appointment dates and descriptions and verifying that the appointments were added successfully.

To maintain efficiency in the code, I focused on writing tests that were concise yet comprehensive. Each test case was designed to cover a specific scenario, minimizing redundancy, and maximizing test coverage. For instance, in the task service tests, I ensured that updating task details only triggered the necessary changes, optimizing the service's performance.

In terms of testing techniques, I used boundary value analysis, equivalence partitioning, and error guessing to design effective test cases. Boundary value analysis is a widely recognized black box testing technique used to identify test cases at the boundaries of input domains to ensure thorough coverage of potential errors (Myers, 2011). For instance, in the task tests, I crafted test cases to evaluate the behavior of the Task constructor when providing descriptions at the boundary values. One test case focused on validating a description exactly 50 characters long, while another tested a description exceeding this limit. These test cases were constructed based on the principles outlined in the IEEE Standard for Software Test Documentation (IEEE, 2008), which emphasizes the importance of designing test cases that cover boundary conditions to uncover potential defects (IEEE, 2008). Error guessing allowed me to anticipate potential issues and validate error-handling mechanisms. Exploratory testing and stress testing were not utilized in this project but could be beneficial for uncovering unforeseen issues and evaluating performance.

Throughout this project, I adopted a cautious mindset, recognizing the complexity and interrelationships of the code being tested. It was crucial to appreciate how changes in one part of the codebase could impact other functionalities. For example, when testing the appointment service, I ensured that canceling an appointment did not affect the integrity of other appointments in the system. I also understood that automated tests can be biased by the developer. “In most teams, the developer who builds a feature is the developer who writes the automated tests for that feature. It is all too easy for this developer, regardless of their level or experience, to write tests that are too tightly related to the specific code they wrote” (Margheim, 2020). To limit bias in my review of the code, I approached testing with an objective perspective, focusing on the expected behavior rather than personal assumptions.

Maintaining discipline in the commitment to quality is crucial in software development. Cutting corners in writing or testing code can lead to technical debt, compromising the reliability and maintainability of the software. For example, neglecting thorough testing of edge cases in the contact service could result in data inconsistencies or unexpected behavior in production. To avoid technical debt, I plan to prioritize code reviews, invest time in writing comprehensive unit tests, and regularly refactor code to improve readability and maintainability. By addressing issues promptly and iteratively, I aim to uphold the quality standards expected in the software development process.

References:

IEEE Computer Society. (2008). *IEEE Standard for Software Test Documentation (IEEE Std 829-2008). IEEE.*

Margheim, S. (2020, December 17). *How to combat bias in software testing. Test IO.* https://test.io/resources/blog/how-to-combat-bias-in-software-testing

Myers, G. J. (2011). *The Art of Software Testing (3rd ed.). John Wiley & Sons.*