Noah Khomer  
CS 320 Software Test and Automation  
Instructor: Desmond Springer

October 20, 2024

**Summary and Reflections**

**Report for Project One**

**Introduction:**

Recently I completed the development of a mobile application which had three main components to it, contact, task and appointment management for Grand Strand Systems. This report will provide a summary of my unit testing approach for each of those features. I will also provide my writing experience with JUnit Testing while reflecting on the different testing techniques employed. Lastly, this report will focus on evaluating the mindset. Adopted during this project and the strategies that were used to make sure bias is limited and code quality is optimal.

**Unit testing approach:**

The mobile application I was tasked to develop three core services, Contact, Task and Appointment service. Each of these services was designed to provide unique functionalities to fulfill requirements which were critical to the design and testing process for the application.

**Contact service testing approach:**

The Contact service was a type of class which I designed to manage contact data to allow users to create, update and delete their contacts by unique IDs.

The main approach of designing this type of system was the implementation of CRUD operations like create, read, update, delete. For each of the operations, I wrote a special unit test to make sure that data integrity and consistency is maintained. For contact service testing, I made sure to test edge cases like updating the nonexistent contacts and even trying to add duplicate contacts.

**Alignment to requirements:**

The testing strategy made it closely aligned with the requirement because each test case directly reflected the expected behavior that was from the software. For example, in my testing I tested that the contacts would invalid length name and phone numbers should result in exceptions or errors like required in the requirements.

**Task Service testing approach:**

For the task service testing approach one of the most important requirements was to implement a class that had functionalities to manage tasks with unique, and non-modifiable fields/Ids. The reason for this was simple because fields such as name and description shouldn’t be able to be changed but at the same time it was important to make sure length requirement was taken into account so that it could correctly be added, updated and even deleted.

**Alignment to requirements:**

All the single test cases designed in the Task Service class were made sure to be taken from project specifications. For example, when it came to testing the length of the task name, I made sure that tasks with name longer than the allowed characters were automatically rejected by the system. This development is what led to the supporting of the software requirements

**Appointment Service Testing Approach:**

Appointment Service class was also made to make sure that appointments were managed by utilizing unique, non-modifiable fields and their descriptions. However, one of the key elements that was required to be in place was that appointment dates could not be set into the past. The main reason for that was to make sure there are fewer errors when it comes to booking the appointments. Additionally, I also made sure to test the operations for appointments to be CRUD specific with valid IDs.

**Alignment to requirement:**

The test for this appointment service was fully aligned with the requirement to make sure that the integrity of the appointment dates and descriptions was maintained. For example, appointments in the past were not allowed to be created, which would in turn reduce the margin of error. Think about it, you want to book an appointment but accidentally select the past year and can no longer sort it properly at an office. Errors such as these were greatly reduced while also ensuring critical requirements of the service were met.

**Quality of my Tests:**

The overall quality of my J unit test was measured by the coverage percentage gained. My project achieved a 54% coverage trade. I do acknowledge that the contact package was not fully tested, which resulted in 0% coverage for that part of the code base. The gap in my coverage testing significantly impacted the final results that were to be achieved. For to improve, I will focus on developing edge cases for invalid input and boundary testing for the contact fields.

**My experience writing J unit testing:**

My experience in writing the J unit test was very rewarding because it was one of the first times I had experienced how J unit testing worked. There were a lot of times where I did infact write programs in Eclipse, but I had never installed a package for J unit or tried to work with it. I learned a lot of new things, for example that **arranging acts assert pattern**. Below is a sample of the code I'm going to talk more about:

**@Test**

**public void testValidContactCreation() {**

**ContactService contactService = new ContactService();**

**assertTrue(contactService.addContact("Emily", "Clark", "9876543210", "456 Oakwood Drive"));**

**}**

The main beauty of this code is the pattern. The pattern helped me in maintaining clarity and organizing my test in a way to make sure that my code was technically sound by following best practices of limiting repetition and clear assertions. Moreover, I focused on gaining efficiency by avoiding the unnecessary setup code in my test, which only allowed me to focus on the required operations for each test case.

**Testing Techniques Used:**

One of the main testing techniques used in my software was boundary testing. In this technique, the main thing that is being tested is the limit of the valid input values. For example, as mentioned earlier, one of the things I had to make sure of was the maximum allowed length for the contact names and even their task descriptions.

**Testing techniques not used:**

There are many testing techniques that I did not use in my software. One of these examples can be the integration testing. In this type of testing, the main thing that is testing is how Contact, Task and Appointment services interact with one another. It is important to remember that the Contact, Task and Appointment service were made for one application, not 3 separate applications. Therefore, integration testing would be helpful to detect issues which are coming by combining these on a large-scale application basis.

**Mindset:**

As a software tester, my mindset was significantly impacted because I had to focus on caution complexity while also limiting my bias and focusing our commitment to quality. For example, I ensured that I remain cautious. While testing the mobile application, especially when dealing with interrelated services. For example, one of the things I noticed was that the task and appointment services had overlapping functionalities which required careful handling of the system. By understanding the complexity of the relationships between task and appointment services, I took extra steps to ensure that I made a test which covered various interaction issues.

When I was writing tests for my own code, I was aware of the potential biases that can come while developing your own code, such as overconfidence. It is important to remember that psychologically Everyone thinks the work they put in is the best. The same can be very true about my own code. Therefore, to counter this bias, I intentionally wrote a special test design to challenge my own code. For example, I deliberately tried to test invalid inputs and edge cases that were not obvious.

Lastly, I made a commitment to the quality. One of the things to remember is that in software engineering you should not try to cut corners, especially in the testing phase. The reason for this is because cutting corners in the testing phase can actually lead to significant issues during production and cost you even more. For example, if I had not tested the edge cases for the task group sessions, users may have experienced crashes on their invalid data input attempts. Therefore, one of the mindsets I have developed is to. Avoid technical debt by making sure rigorous testing standards are part of employing any type of automated software for excellent quality.

**Conclusion:**

Therefore, in conclusion, Project 1's experience has provided me with a lot of valuable insights for the importance of utilizing unit testing for different applications. Although in my project there is room for a lot of improvements, especially in my package percentage, one of the things I have learned is that my overall approach to the testing was aligned with the project requirements. That can be seen with the quality of the J unit test being validated through the coverage of over 56%. Moving forward, I have decided to apply these principles to my future projects to make sure both technical and unbiased testing is committed.

**Work Cited**

Knight, A., Knight, A., says:, tyrionlannister26, & says:, D. T. (2020, July 7). *Arrange-act-assert: A pattern for writing good tests*. Automation Panda. https://automationpanda.com/2020/07/07/arrange-act-assert-a-pattern-for-writing-good-tests/

*OpenDSA data structures and Algorithms Modules Collection*. 3.11. Code Coverage In JUnit - OpenDSA Data Structures and Algorithms Modules Collection. (n.d.). https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/junitcoverage.html

Schmitt, J. (2021, December 3). *Unit testing VS integration testing*. CircleCI. https://circleci.com/blog/unit-testing-vs-integration-testing/

What is boundary testing? (plus how it works and examples) | indeed.com. (n.d.). https://www.indeed.com/career-advice/career-development/what-is-boundary-testing