Sam Blanton

CS-320

Johnathan Norman

10 December 2023

Project 2

**Summary**

In Project One, my unit testing approach for the Contact, Task, and Appointment services was designed to closely align with the software requirements. I focused on testing all possible scenarios, including both normal and edge cases, to ensure comprehensive coverage and functionality. This project was a great exercise for to learn how to create functional and maintainable code, along with learning how to use junit to write tests for my code to ensures all cases are covered.

For the Contact service, I scrutinized field validations and unique ID enforcement, ensuring each contact met the predetermined criteria. Similarly, in the Task service, I emphasized the immutability of task IDs and validated the correctness of task attributes. The Appointment service required careful consideration of date validations and appointment uniqueness, which I made sure to test for.

During Project one I took several steps to make sure that my code met all the requirements of the assignment. For instance, in ContactTest, I not only tested valid contacts but also validated each field against null and size constraints. This was mirrored in TaskTest and AppointmentTest, where edge cases and specific scenarios like past dates for appointments were also assessed.

My coverage was 87%, indicating that most lines of code were executed during testing, which means the is low risk of untested or faulty logic.

Writing the JUnit tests was a very new and different experience for me. I did my best to ensure technical soundness by consistently using assertThrows for validation tests, as seen in TaskTest, and assertEquals for verifying successful updates in TaskServiceTest. I tried to keep things efficienct by avoiding redundant code, as exemplified in ContactServiceTest where tests were concise yet comprehensive.

**Reflection**

**Testing Techniques**

1. **Employed Techniques**: I primarily utilized positive and negative testing. Positive testing (valid inputs) ensured that each class and method behaved as expected under normal conditions. Negative testing (invalid inputs) helped identify potential breaks and weaknesses in the code. For instance, in AppointmentTest, negative tests for past dates and null values assured that the class handled such scenarios correctly (GeeksforGeeks, 2020).
2. **Non-Employed Techniques**: Stress and performance testing were not used. These techniques, while crucial for assessing how systems perform under heavy load, were outside the scope of this project focused on unit testing.
3. **Practical Uses and Implications**: Positive and negative testing are fundamental in any development project for verifying functional correctness. If your code does not account for invalid inputs it can cause your application to break, for example if your database only accepts date objects and you let them submit a string, it could cause the application to break. Stress and performance testing are more specific, often employed in scenarios where the application is expected to handle high volumes of data or traffic. This would have been something we would do had we gotten closer to finishing the application, and is something you do before deployment.

**Mindset**

1. **Caution and Complexity Appreciation**: In testing, I adopted a cautious mindset, aware that overlooking minor issues could lead to larger problems. This project also made me more aware of how complex even simple code can be, this is something you can't appreciate until you start writing tests. This was really shown when testing features like the unique ID checks in ContactService, where one faulty logic could impact the entire system.
2. **Limiting Bias**: To limit bias, I approached the code as if I were not its original author, questioning every function's reliability. It is really difficult writing tests for your own code because you are so familiar with it and I was quick to assume it was already right even before the codes. For instance, while writing tests for TaskService, I had to consciously challenge my assumptions and make myself validate each method rigorously.
3. **Discipline and Commitment to Quality**: The commitment to quality and discipline in avoiding shortcuts was important for me. Cutting corners in software development can lead to technical debt, making future modifications more challenging and error-prone. To avoid this, I ensured each test case was thoughtfully written and covered a wide range of scenarios, as seen in AppointmentServiceTest.

In conclusion, this project reinforced the importance of thorough testing in software development. It was a valuable lesson for me in the need for meticulousness, objectivity, and a commitment to quality, all of which are essential traits for a professionals in this field. This class has given me a new appreciation for what it takes to make sure that code is thoroughly tested to make sure you do not start development with faulty code, and so that your new code doesn’t break what is already working.

Sources:

GeeksforGeeks. (2020, October 17). *Difference between Positive Testing and Negative Testing*. https://www.geeksforgeeks.org/difference-between-positive-testing-and-negative-testing/