Ivan Arroyo

12/9/24

Professor Turft

CS-320

Summary Report

Throughout the project, I utilized unit testing for all features in the Appointment, Contact, and Task services. Each test case was created in order to check certain functionalities and ensure they met the given constraints. For instance, tests like testAddAppointment and testDeleteAppointment in the AppointmentServiceTest.java file made sure that the system was capable of handling appointments with proper descriptions and timestamps. The JUnit test efficiency was checked through coverage analysis and the test ultimately achieved 100% coverage. This meant the code path was properly covered, even in extreme cases such as the invalid ones. Lastly, assertions such as assertTrue were implemented to check conditions such as whether an appointment was deleted or a task was properly updated.

My approach in writing JUnit tests involved best practices that ultimately aimed at producing technically correct code. For instance, in the ContactServiceTest class, each method was written in a way that would simply address one singular aspect of the contact class in order to make it clear and concise. This is due to the use of @BeforeEach which ensured there was consistency in the tests that were conducted as it provided a clean, predictable environment. It also helped avoid duplication of code, as initializations were done once and test cases were arranged in a modular manner as seen in ContactServiceTest class. Apart from decreasing the time that it took to complete the task, it also made it easier to debug and maintain the code in the future.

Several testing types were implemented throughout the project, these include black box testing, white box testing, and mocking. Black-box testing was very valuable in assessing input-output relationships at the beginning of the process while white-box testing allowed us to check for specific internal code paths. Mocking helped in creating controlled environments with real data without the need for other systems. Other approaches that could’ve been useful include boundary testing and stress testing. Although they were not utilized, boundary testing would have checked constraints such as the maximum characters allowed for a task description while stress testing would have checked how the system reacts under heavy workload. Each of these techniques has different effects in different contexts, which is what makes them extremely useful in the real world.

When testing, I applied that of a real-world testers mentality to enhance the overall quality of the system. I found this to be extremely crucial in order to avoid bias and I made sure to not make any assumptions about the code's behavior at any point. I maintained discipline throughout it all as I followed the set protocols and ensured that I documented the test cases properly. Some of the measures I took included monitoring the test coverage metrics on a frequent basis as well as adhering to the best coding standards. One of the key lessons learned was to be very cautious especially when dealing with edge cases as when I was deleting a contact, I had to ensure the ID was valid and that it did not refer to an imaginary contact. This approach helped me identify many vulnerabilities while also making the testing process more reliable.

One of the major lessons learned in this project was how to balance between the need to be comprehensive and the need to be brief. I understand it is important to seek completeness, however, I also understand it is not practical to test every condition that may occur. I used the Godoid approach whereby I only targeted high-risk areas of the code such as null inputs or improper data formats since they are likely to expose severe weaknesses. This approach made the testing process efficient which was extremely useful in moments when I found time to be a concern. Ultimately, the approach helped me to focus on the most important sections of the code and thus contributed to the overall efficiency of the testing process without sacrificing the quality.

Another important lesson I learned is that proper documentation and communication are extremely important. Simply writing and executing tests is not enough, every outcome needs to be documented and its implications need to be explained thoroughly in order to allow potential stakeholders and team members to understand the system's status. Documentation promotes cooperation and ultimately makes it easier to find who did what and where in the testing process. In the future, I plan to use these proper communication measures to check whether the functionality is correct and to support my potential team and the overall project.

This project has truly taught me that proper testing along with the proper approach are extremely crucial in software development. Thus, by applying proper testing methods and keeping a proper attitude, I managed to create a stable and maintainable system. In the future, I will try to implement stress and boundary testing to enhance the reliability of my applications. These strategies will allow me to deliver quality software products that meet all required standards.