Jessica Ramirez

October 13th, 2024

CS320

Professor Phillips

7-2 Project Two

In Project One, I undertook the development and testing of three key features: the Contact Service, Task Service, and Appointment Service. This report outlines my unit testing approach for each feature, evaluates the quality of my JUnit tests, and reflects on the techniques and mindset I adopted throughout the project.

For the Contact Service, my unit testing approach focused on validating key functionalities such as the creation, modification, and deletion of contacts. I implemented various tests to ensure the service met the specified software requirements. For instance, the testAddContactSuccess method verified that valid contacts could be added without exceptions, while testDeleteContact confirmed that deleted contacts could not be updated, reflecting the requirement that once a contact is removed, it should no longer be accessible. The alignment of these tests with the software requirements is evident; they directly assess the core functionalities that the application needed to support.

The Task Service was approached with similar rigor. I crafted tests to validate successful task addition, update, and deletion. The method testAddTaskSuccess checks the successful creation of a task, while testUpdateTaskNameFailure ensures that attempts to update a non-existent task throw the appropriate exceptions. This thorough testing ensures that the software operates within defined parameters, directly addressing user needs for reliable task management.

In the Appointment Service, I focused on validating appointment details such as date and description. For example, testInvalidAppointmentDate ensures that appointments cannot be set in the past, adhering to the requirement that only future dates are valid. This method, along with others, ensures the robustness of the service in handling user input, particularly in edge cases.

Overall, my JUnit tests achieved over 90% code coverage, which indicates a high level of testing thoroughness. This coverage was particularly strong in areas concerning error handling and boundary conditions, demonstrating the effectiveness of the tests in identifying potential issues before deployment.

Writing the JUnit tests was an enriching experience that enhanced my understanding of both the code and the testing process. To ensure the technical soundness of my code, I utilized assertions like assertEquals to compare expected and actual values, which confirmed that the services were functioning as intended. For instance, in the Task Service tests, I validated that the name of a task was updated correctly using assertions that explicitly compared the expected output to the actual task state. This methodical approach helped me identify discrepancies early in the development process.

In terms of efficiency, I structured my tests to avoid redundancy. For example, common setup code was encapsulated in the @BeforeEach method, which ensured that each test started with a clean state, reducing the likelihood of errors caused by state carryover. This practice not only streamlined the testing process but also enhanced the clarity of the tests, making it easier to understand each test's intent.

Throughout this project, I employed several software testing techniques, primarily focusing on unit testing and boundary testing. Unit testing involved validating individual components of the services in isolation, which is critical for identifying bugs early in the development cycle. For instance, I confirmed that tasks could only be created with unique IDs through targeted tests. Boundary testing was also employed to ensure that inputs stayed within acceptable limits, such as checking that task names did not exceed the specified character count.

Techniques that I did not utilize include integration testing and user acceptance testing. Integration testing focuses on the interaction between different components of a system, while user acceptance testing assesses the software from an end-user perspective. Although these techniques are vital in comprehensive testing strategies, the scope of Project One necessitated a focus on back-end functionality, making unit testing the priority.

Throughout the project, I adopted a cautious mindset, recognizing the importance of appreciating the complexity and interrelationships of the code. This perspective allowed me to anticipate potential issues arising from interconnected components, ultimately leading to more robust tests. For example, while testing the Appointment Service, I considered how changes in appointment management might affect related services, such as task scheduling. This holistic view was essential in developing reliable software.

To limit bias in my review of the code, I approached testing as an external reviewer rather than as the developer. This separation allowed me to identify issues that I might have overlooked had I been too closely tied to the code's original intent. I made a conscious effort to critique my work objectively, understanding that bias could cloud judgment and lead to missed defects.

Finally, the importance of discipline in maintaining quality as a software engineering professional cannot be overstated. Cutting corners in writing or testing code can lead to significant technical debt, which can jeopardize the long-term success of a project. To mitigate this risk, I plan to prioritize thorough documentation and peer reviews in future projects. These practices not only ensure that code is understandable and maintainable but also foster a culture of accountability and continuous improvement within the team.

In conclusion, the unit testing approach I employed for Project One was methodical and thorough, addressing each service's requirements effectively. By leveraging appropriate testing techniques and maintaining a disciplined mindset, I was able to produce a high-quality software product. As I continue my journey in software development, I remain committed to upholding these standards and continually improving my practices to minimize technical debt and enhance software quality.