7-2 Project Two

Lauren-Ann Javier

CS 320

8/14/2024

Southern New Hampshire University

My unit testing approach for the Contact, Task, and Appointment was thorough and careful. For the Contact class, I focused on testing the code, then updating it as I go, and retrieving the contact details, making sure that each field (e.g., firstName, lastName, phone, and address) was correct. For instance, the tests were to verify that invalid phone numbers or names that are too long would trigger specific exceptions, supporting the software’s input requirements. In the Task class, my testing centered around updating task classes and verifying due date limits. I made sure that tasks could not be created with past due dates and that status transitions, such as from "Incomplete" to "Complete", worked as needed. The Appointment class required careful testing of date handling and description validation to ensure that appointments were scheduled correctly, and descriptions were within the required length. These tests aligned closely with the software requirements, as proven by the detailed mapping of tests to each functional specification provided in the project guidelines.

The effectiveness of my JUnit tests is supported by the coverage percentage, which was pretty high, showing that most of the code paths were tested. I used statements such as assertThrows(IllegalArgumentException.class, () -> new Contact("tooLongName", ...)) to make sure the application handled the wrong inputs as expected. By covering both typical and edge cases, I made sure that the tests were not only thorough but also effective in catching potential issues. Writing these tests was a detailed process, where I carefully planned each test case to cover various parts that were needed, ensuring that the code ran as intended under all circumstances. For example, in the AppointmentTest, I included tests to handle invalid date formats and boundary cases like leap years.

The primary software testing technique I used in this project was unit testing. Unit testing focuses on testing individual parts or methods in isolation, which is usually effective for validating specific needs and functions within each class. The characteristics of unit testing include its granularity, allowing for a detailed examination of each method, and its ability to quickly identify problems within specific areas of the code. For example, unit tests enabled me to verify the task code on its own, separate from other system components, when testing the Task class. This approach made sure that the logic was solid before moving on to integration.

Other software testing techniques that I did not use include integration testing and system testing. Integration testing is used to verify that different modules or components work together as intended. It is essential to identify issues that come up when individual units interact, such as making sure that data flows correctly between the Contact and Appointment classes. System testing, on the other hand, evaluates the entire system’s fulfillment of specified requirements. It assesses the system’s behavior, making sure that all components integrate seamlessly and perform correctly in a production-like environment. While these techniques were not applied in this project due to their range, they are crucial in larger projects where the interaction between many components and the overall system behavior are very important. These techniques help find issues that may not be obvious in isolated unit tests but become clear when the components are integrated.

During this project, I adopted a mindset of caution and thoroughness. I have made many mistakes before rushing through the process of making my code which led to careless mistakes that I could have avoided if I had just been careful and taken my time. This cautious approach was necessary to avoid overlooking potential issues, particularly in areas where different components interacted, such as the scheduling logic in the Appointment class. For example, I was mindful of how changes in one part of the code, such as updating a contact’s details, could impact related classes like appointment scheduling. To limit bias in my review of the code, I involved my friend in code reviews and relied on automated testing tools to confirm my statements. This approach helped lessen the risk of overlooking errors due to familiarity with the code. Bias is a significant concern when a developer tests their own code, as they may unconsciously ignore potential mistakes. For instance, I initially believed that the date validation logic in the Appointment class did not have any errors, but my friend’s review and automated tests revealed edge cases I had missed, highlighting the importance of an unbiased review process.

Maintaining discipline and assurance of quality is important in software engineering, as cutting corners can lead to technical mistakes, which can compromise the software’s long-term stability. To avoid technical mistakes, I plan to stick to best practices such as regular code refactoring, continuous integration, and comprehensive documentation. For example, in this project, I ensured that all classes and methods were well-documented, making it easier to maintain and update the code in the future. Additionally, by consistently writing and running unit tests, I aim to catch issues early in the development process, preventing them from escalating into more significant problems later on. This disciplined approach not only ensures the quality of the current code but also lays a strong foundation for any of my future developments, reducing the risk of gathering technical mistakes.

References

Assignment 3-2

Assignment 4-1

Assignment 5-1