Johnny Lewis

CS-320-R4839

04-28-2024

Professor Kalysa Wilson

CS-320 Project Two

My approach to unit testing was aligned with the software requirements. Each requirement dictated the creation of a specific test case to ensure that compliance was met. For instance, the requirement that the appointment date should not be null was tested by attempting to create an Appointment instance with a null date, which was designed to trigger an IllegalArgumentException. Similarly, the requirement to prevent setting the appointment date in the past was validated by setting the date to a moment slightly earlier than the current time and expecting the same exception. These tests not only validated adherence to the specifications but also ensured that any improper inputs were handled gracefully, proving the class's robustness in maintaining data integrity. I similarly validated the other features, such as the contact and task classes.

My goal with the JUnit tests was to ensure that my code met the requirements established. In this case, is the character limit on a name or description, in addition to it not being null. The priorities of the tests I created were to assert or look to see that the conditional statements I established for each object were ensured to meet the software requirements. To ensure my code was efficient, I made the tests in my Appointment class straightforward and focused. For example, using a new Appointment(pastDate) directly tests whether past dates are correctly rejected without any extra steps. I also used simple checks, like assertEquals("Expected the future date to be correctly retrieved", futureDate, appointment.getAppointmentDate());, which directly verify if the appointment date is set and retrieved correctly. This method keeps the tests running quickly and makes them easy to understand and maintain. These and similar tests made it very simple and easy to verify the validity of the code to ensure it is working correctly as intended.

I employed unit testing as the primary software testing technique in this project. Unit testing involves isolating the smallest pieces of testable software, such as functions or methods, from the rest of the codebase to ensure that they are working correctly. For example, I tested individual methods like the constructor and setters in the Appointment class to verify their functionality independently. This technique ensures that each component operates correctly in isolation, catching errors early in the development process. It's beneficial because it simplifies debugging by isolating the source of errors from specific parts of the code. Unit tests are written and run by developers, often using frameworks such as JUnit, making them a cornerstone of agile development practices where frequent changes to the codebase are common.

           Apart from the unit and integration testing methodologies that I employed in my project, various alternative testing procedures might considerably augment the project's general quality and user contentment. System testing is essential because it assesses the overall operation of the system in a setting that closely resembles actual usage, ensuring that all integrated components perform as a unit. Acceptance testing, which prospective end users frequently carry out, assists in verifying that the system satisfies the stated business objectives and is easy to use. Applications intended to manage large volumes of user traffic must undergo performance testing to see how well they can continue functioning steadily and quickly under pressure. Last, security testing is essential for programs that handle sensitive data since it finds flaws that can result in security breaches. These methods each provide unique insights into system reliability and readiness, enhancing confidence in the software before it reaches the end-user.

I adopted a cautious and detail-oriented mindset as a software tester when working on this project. It was crucial to be thorough because missing even minor errors could lead to more significant problems. For example, ensuring that the Appointment class correctly interacted with other courses like AppointmentService and the Date class was vital. This not only required testing each class individually but also making sure they worked well together. Understanding the complexities and how different parts of the code related to each other helped me ensure that the entire application functioned smoothly without any hidden issues.

Reducing bias in code review is a critical aspect of software testing. I tried to question my assumptions and rely on objective testing outcomes rather than subjective judgment to achieve this. This practice is particularly crucial when testing one's code, as it's easy to overlook flaws in something you've developed. For example, a developer might assume their data processing function is flawless, but unexpected errors could be noticed with rigorous testing. To maintain high quality and avoid shortcuts, I plan to implement continuous refactoring and leverage automated testing tools. These practices keep the codebase clean and efficient and minimize the risk of accumulating technical debt that can make future changes arduous and costly. Reflecting on this project, I've realized the importance of a disciplined approach in software testing. Great attention to detail and an openness to feedback have reinforced the need for comprehensive testing and ongoing improvement. Moving forward, I'll continue to prioritize rigorous testing and best practices to prevent technical problems and ensure high-quality software development. This experience has underscored the importance of thoroughness and quality in delivering reliable and sustainable software solutions.

References

Factor, P. (2020, May 18). The database development stage. Redgate. https://www.red-gate.com/hub/product-learning/sql-compare/the-database-development-stage?topic=database-builds