Project 2

Joseph Daniszewski

Joseph.daniszewski@snhu.edu

CS-320

In every class designed for this project, I've incorporated the practice of unit testing, implemented through JUnit 5. These unit tests focus on examining individual components, such as methods and functions, in isolation. They can be found in dedicated testing classes, including TaskTest.java, AppointmentServiceTest.java, ContactTest.java, ContactServiceTest.java, AppointmentTest.java, etc. These tests are carfully designed to verify the accuracy of specific methods and functions within each class.

What sets these tests apart is their ability to place the code under scrutiny in isolation where they are separated from external dependencies. In this controlled environment, they aim to determine whether each method or function behaves as expected. Their primary purpose is to identify and rectify bugs in their early stages, making them invaluable in the initial phases of development. These tests aim to ensure an errorless and flawless performance of individual functions or methods and are practical for various software development projects.

Exception handling plays a crucial role, which I illustrated in TaskTest.java, AppointmentTest.java, and ContactTest.java. In these testing areas, I explore boundary testing and the handling of exceptions. Boundary testing scrutinizes how the system navigates at the outermost boundaries of input, where extremes like minimum and maximum values come into play. These tests reveal the software's capability to handle edge cases, ensuring its correctness even under extreme circumstances. Exception-handling tests serve as models of robustness and fault tolerance. Ensuring the system's appropriate response to exceptional scenarios.

While the project did not include performance testing, Performance testing is generally a vital part of the testing process. This is where a system's vigor is tested under various conditions. These include load testing, stress testing, and scalability testing. Performance testing is crucial in projects where the system's performance is a non-negotiable requirement, such as web applications, online games, or high-traffic websites. It helps pinpoint bottlenecks and ensures that the system upholds its promise of optimal performance, contributing to user satisfaction and efficiency.

In this project, I employed several software testing techniques to ensure the robustness and reliability of the codebase. One of the primary techniques used was unit testing, which involved testing individual components such as classes and methods in isolation. I created test cases to check the validity of method behaviors and validation checks within the classes. This approach allowed for efficient debugging and helped identify issues early in the development process.

Additionally, integration testing was employed to verify the interaction between different components, like how AppointmentService.java interacts with the Appointment.java class. This technique aimed to detect any potential integration issues or miscommunications between components.

However, some software testing techniques, such as end-to-end testing, user acceptance testing, and performance testing, were not employed in this project. These techniques were not relevant to the scope of this project which was primarily focused on unit and integration testing. End-to-end testing and user acceptance testing are usually used in more comprehensive projects where the entire application's functionality and user experience need evaluation. On the other hand, performance testing assesses an application's speed, responsiveness, and stability under different loads, which wasn't the primary concern in this project.

During this project, I adopted the role of a seasoned software tester, attempting to recognize the critical importance and understanding the complexity and interrelationships of the code. Understanding how changes in one class or method could impact other parts of the code was crucial. For example, when testing the AppointmentService.java class, I needed to be cautious about the data integrity and the behavior of related Appointment instances to avoid introducing subtle bugs.

In terms of limiting bias, I strived to maintain objectivity in my review of the code. I followed a structured testing plan, focusing on the specifications and requirements, rather than personal preferences or preconceived notions. Bias could be a significant concern when testing one's own code as a software developer, as one might unintentionally overlook issues or assume the code works as intended. This underscores the importance of independent testing and peer reviews to minimize bias and ensure a high-quality end product. Having said that I am sure that there were scenarios within the testing cases where I may have overlooked some cases or made assumptions due to being the writer, but I attempted to separate my testing self from my production self.

Being disciplined in the commitment to quality as a software engineering professional is very important. Cutting corners in writing or testing code can lead to technical debt, where unresolved issues accumulate and hinder future development. For example, neglecting to write unit tests for critical functions may seem to expedite the process initially but can lead to time-consuming debugging in the long run. To avoid technical debt, I plan to adhere to coding standards, conduct thorough code reviews, and prioritize testing in all my projects, regardless of their size or complexity. This approach ensures the delivery of high-quality, maintainable software, reducing future development challenges and costs.