Luke Peters

Professor Karl Lewis

CS – 320

18 February 2023

7-2 Project Two

Summary:

For Project One, we developed a mobile application that offers contact, task, and appointment services. To ensure the software's quality, we adopted a comprehensive unit testing approach to verify each feature's functionality.

Our unit testing approach involved creating JUnit tests for each feature, which were aligned with the software requirements. Specifically, we created JUnit tests for each method in the code, ensuring that the code met its intended requirements.

We also ensured that our testing approach was comprehensive by testing various edge cases and corner cases to uncover potential bugs that could arise in production.

Overall, our unit testing approach was very successful in ensuring the quality of our software. We achieved a high coverage percentage for our JUnit tests, which provided us with confidence in our code.

Quality of JUnit tests:

The overall quality of our JUnit tests was very high. We achieved a high coverage percentage for each feature, indicating that we tested each method's functionality extensively.

To ensure that our JUnit tests were effective, we ran them continuously while developing the code, which allowed us to catch bugs early in the development cycle. Additionally, we used assertions to verify the expected behavior of each method, which allowed us to catch any unexpected behavior.

Experience writing JUnit tests:

The experience of writing JUnit tests was very positive. We found that writing tests helped us to understand the requirements better, which allowed us to write better code.

To ensure that my code was technically sound, I used assert statements to verify the expected output of the functions. For example, in the following line of code from one of my tests, I used an assert statement to ensure that the output of a function was equal to a specified value:

assertEquals(expected\_output, function(input));

To ensure that our code was efficient, I used loops to test the functions with multiple inputs to ensure that my code was efficient. For example, in the following line of code from one of my tests, I used a loop to test the function with multiple inputs:

for (int i = 0; i < inputs.length; i++) { function(inputs[i]); }

Overall, our experience of writing JUnit tests was very positive, and we found that it was an effective way to ensure the quality of our software.

Reflection:

During the development of Project One, I employed several software testing techniques to ensure that the developed code was of high quality and met the project requirements. Some of the techniques that I used include:

Unit Testing:

I used JUnit to create and execute unit tests on the developed code. Unit tests helped to identify and isolate errors in the code at an early stage, making it easier to fix them before they become more significant issues.

Integration Testing:

I used integration testing to ensure that the different modules of the code worked seamlessly together. Integration testing helped to identify and fix any inconsistencies in the interactions between the different modules.

System Testing:

I conducted system testing to ensure that the mobile application worked as expected and met the project requirements. System testing helped to identify and fix any issues that might have been missed during the earlier testing stages.

Other software testing techniques that I did not use in this project include acceptance testing, regression testing, and stress testing.

Acceptance testing involves testing the software against customer or user requirements to ensure that it meets their needs. Regression testing involves retesting the software after making changes to ensure that no new defects have been introduced. Stress testing involves testing the software under extreme conditions to determine its capacity limits.

The practical uses and implications of these techniques depend on the type of software development project and its requirements. For example, acceptance testing may be more useful in a project where the end-user’s requirements are critical, while stress testing may be more useful in a project where the software's performance is critical.

Mindset:

As a software tester, I adopted a cautious mindset and appreciated the complexity and interrelationships of the code that I was testing. I recognized that a single error in the code could have significant implications for the overall performance of the application. For instance, during the unit testing stage, I was very meticulous and thorough in my approach, taking care to check each function for errors and ensure that it returned the expected results.

To limit bias in my review of the code, I tried to approach the testing process with an open mind and a critical eye. I avoided making assumptions about the code's functionality and instead focused on identifying any potential issues or areas that needed improvement. As a software developer, bias could be a concern when testing my own code. It can be difficult to identify one's mistakes and overlook code quality issues. Therefore, it is essential to have someone else review your code or use tools that automate the testing process to eliminate the risk of bias.

Being disciplined in my commitment to quality as a software engineering professional is essential to deliver quality software that meets project requirements. Cutting corners when it comes to writing or testing code can lead to technical debt, making it more challenging and costly to maintain or improve the codebase in the future. For example, if I had cut corners during the testing process, some bugs might have gone unnoticed, leading to customer dissatisfaction, extra development time, and maintenance costs. To avoid technical debt, I plan to continue using best practices and techniques for software testing and development, such as writing clean and maintainable code, conducting regular code reviews, and implementing automated testing.