Lillian Martinez

February 23, 2025

CS 320

7-2 Project Two

1. Summary:

For each of the three features my unit testing approach followed a structured method that helps to ensure the accuracy and alignment with the software requirements that were needed for the project. I began by looking at the functional specifications and identifying any of the key test cases that would confirm the expected behaviors needed.

For feature one which is the contact service, I made sure that the unit tests I created were made to check that contacts were updated, added and deleted correctly. For feature two, which is Task Service I tested the methods to make sure that tasks could be created, edited and removed as expected and lastly, for feature three, Appointment Service, I wrote tests that would check appointments were scheduled, modified and canceled as expected.

My approach was to align closely with the software requirements that each feature needed, because each test case mapped out a specific function, that would ensure that all expected behaviors would be validated. By designing these tests on requirement-drive criteria, it helped to minimize any risks and any functionality gaps.

To defend the overall quality of my Junit Tests, I relied on the code coverage metrics and the assertion effectiveness. I used the coverage percentage to measure how much of my code was exercised during testing. A high coverage percentage, such as 80% or above would indicate a strong test effectiveness, and I struggled to confirm that and also to ensure that they covered edge-case scenarios. While in contact service I helped to ensure that the character limit was not longer than 10 characters, I followed the requirements needed for each feature.

While writing Junit tests I needed a more structured process that would require more consideration of logic, while this is the first time writing JUnit tests I learned a lot. When it comes to ensuring that my code was technically sound, I tried to adhere to the best practices such as writing code modularly, in sections and testing as I went. While Also using reusable test methods, and assertions correctly and validating any of the expected outputs. What I did to ensure code efficiency, is that I avoided any redundant test cases and worked on test execution time.

1. Reflection:

In this project, I employed several different testing techniques to help to ensure the reliability and correctness of the code and tests. First technique is Unit Testing: Using Junit, I tested the individual components in isolation to verify that each of the functions would perform as expected. This technique, though proved a bit of a struggle at first, helped me to validate small units of code efficiently. White-Box Testing is also another technique that I used that means that I have access to the internal structure which is the cod/logic for each feature and also designing tests that would specifically target different parts of the code, and since I used Junit testing this also falls underneath white-box testing because I am writing tests based on the internal working of functions rather than just testing inputs and outputs blindly. This type of testing helps to ensure that the system met the requirements for each feature.

Some software testing techniques that I did not use was Exploratory testing, instead of manually testing the software through an unscripted test case, I relied on the structured unit tests. This kind of testing is based useful for uncovering unexpected issues. Another technique I did not use for this project and for the features required, I did not do any regression testing which is testing that means running or adding tests to ensure an old functionally isn’t broken by new changes, since the features I was working on it was more unit testing and debugging rather than regression testing.

Each technique helps to ensure different purposes in software development, Unit Testing is most essential for identifying any of the issues early in development which I found is more widely used in agile and test-drive development (TDD). Exploratory testing is useful for usability and security testing, where predefined test cases may not uncover all of the potential errors or vulnerabilities. Regression testing is particularly valuable for projects that have frequent updates, ensuring that any new features added to a project wouldn’t break existing functionality.

While working on this project, I did have to adopt a critical and detail-orientated mindset to help identify any of the potential errors that may occur with each feature. One thing I developed was having to be cautious and overlooking an issue in unit testing can cause problems later down in a sort of deployment for the project.

Some ways that I tried to eliminate bias in my code is by using automated tools to analyze test coverage rather than relying on myself, also looking at feedback that my instructor provided, which I did struggle to implement. Bias can be a major concern when developers test their own code, they may avoid testing certain edge cases or assume that their logic is correct, when it may not be. A fresh perspective should always be welcomed such as independent code reviews, this helps to identify any overlooked issues that may occur.

Finally, the importance of maintaining discipline in testing is a fundamental part of software engineering. Any high-quality software enhances the reliability, security, user trust, and reducing the risk of costly bugs in production. As a practitioner in the field a commitment to testing ensures the maintainability, well-tested code will be easier to update and extend, scalability, robust testing allows the software to grow and also professional integrity, poor testing can have some serious consequences like in healthcare, finance, or security, making quality assurance of the utmost importance.