Brad Farfaglia

CS 320 Project Two

The software requirements for this project dictated my testing approach. Each requirement set forth has a specific test to verify that the function works properly. The requirements for each class gave maximum lengths for each attribute and also stated that they could not be null. My tests checked to make sure that these requirements were met, with tests designed to add an attribute that was null above the maximum length, throwing an error when working correctly. The service classes also have requirements for adding, deleting, and updating information. To ensure that these functions are working, I designed tests that simulate each action and only allow changes if the aforementioned class requirements are met.

To ensure overall quality of JUnit tests, I made sure that coverage was at least 80%. The purpose of the tests was to verify that the classes handle input properly. I tested each field with an input that did not meet the proper criteria. For example, I tested a first name that was longer than ten characters. If an error was not thrown from the invalid input, then the test would fail.

I made sure that my code was technically sound by following proper syntax and indentation rules for Java. For example, in lines 17-38 of the Appointment class, all if statements are properly indented and bracketed so that the system knows exactly how the code is supposed to flow. Since I am still a beginning programmer, I wrote one line at a time and verified that there were no errors before moving on to the next line. This way I didn’t have to go searching the entire code for any errors that popped up.

One way in which I made my code efficient was to reuse code that had already been proven to work in previous classes. The functions and JUnit tests for the Appointment class are copied from the Task class, which in turn was copied from the Contact class. This cuts down on the amount of time that I have to spend coding, giving me more time for testing and debugging. It also makes the code cleaner and more uniform, being easier for an observer to read and understand what it is trying to accomplish.

The primary software testing technique I used was unit testing, specifically JUnit testing. The purpose of this testing is to isolate each part of the program and test them in terms of requirements and functionality. This is why there are many individual tests that focus on a single metric such as length of input or whether or not the input is null. I also used static testing to manually review the code to look for defects before running the program.

Other testing techniques that I did not use include integration testing, system testing, and acceptance testing. Integration testing involves testing two or more combined parts of an application to see if they function together correctly. It can be done in two ways- bottom up, where you take unit test cases and combine other units to construct higher levels of functionality, or top down, where collaboration between the highest level modules is tested first, with lower level modules following after. System testing is performed once all components are integrated and tests the system as a whole to ensure that the application meets all requirements. The testing environment is either the actual production environment or as close to the production environment as possible. Acceptance testing involves testing the application with a set of pre-written scenarios and test cases. This type of testing is the gauge that determines whether or not the application meets specifications and satisfies the client’s requirements. Acceptance testing is intended to find any bugs that may result in major errors or system crashes.

It is important to me to exercise caution and to be thorough when writing software. If my tests aren’t as extensive as possible and I don’t cover all of my vulnerabilities, then the holes in my program can be exploited and innocent people may be harmed because of it. For this reason, I take testing very seriously. It’s important to understand how my code works together and what it is capable of. The better I understand it, the better I can protect everyone else from misuse. For example, it was important to understand exactly what I was testing with the JUnit tests. If I thought that I was testing whether or not a contact was successfully created rather than intentionally introducing an invalid input to try and get it to fail, that could lead to a fundamental misunderstanding of the program that could have major consequences.

I do my best to limit bias by having different programmers do the coding and the testing. Unfortunately, that was not an option for this project. I did both the programming and the testing. One technique I utilized to combat this was to do the coding and testing in separate sessions. Once I was done coding, I worked on another project and then came back and looked at what I had written as if it was done by someone else. This ended up being a good way to separate myself from the testing process.

Being disciplined is important to maintain the quality of the software that I create. Discipline may be tough in the short-term, but it will make my life easier in the long-term. It is critical for getting my programming skills to a high level and maintaining that high level. The hard work that I put in to being disciplined will pay off exponentially in the future. It is also vital in avoiding technical debt. Technical debt is incurred in several ways such as prioritizing speed over quality, taking shortcuts to meet deadlines, not fully understanding a solution, releasing code without proper testing, and many more. These decisions may seem like good ideas at the time, but they are more than likely to cause problems in the future and accumulate to the point where they cause more work than if you had just addressed the issue properly the first time. I plan to avoid technical debt by always prioritizing quality over speed, keeping my skills sharp so that I have a high level of understanding, and complete the full testing process with no shortcuts.