Tiffany Montero

CS 320

Project Two

During the testing process for Project One, I followed the software requirements line by line. In Contact, Appointment, and Task, there are variables which must meet a length requirement and cannot be null. Each variable had 3 tests, one for a valid entry, one for an entry which was too long, and one for a null entry. Contact also had a variable, phoneNumber, which had to be 10 characters in length. This one was tested differently to ensure the variable was only assigned if it was the appropriate length. In the invalid entries, I tested to ensure the exception was thrown. This confirmed that the variable was not assigned unless it met the requirements.

As my skills with JUnit testing improved, I was able to get a higher percentage coverage. After reviewing each program one final time, I was able to get over 90% coverage, with Appointment having 100% coverage. I regularly checked the percentage of coverage to ensure each test was efficient and help reduce redundancy.

To ensure the code was technically sound, I used an array as the data structure for the service portion of each program. I also used proper naming conventions such as TestIdTooLong, getName, setId. I also used comments regularly to help readability. This means if someone else were to work on those programs at a later date, they would be able to understand it easier. In regard to efficiency, I tried to prevent too much code repetition, if possible. In Contact, Task, and Appointment, I put the variable requirements in the setters, and used the setters for the constructor. Otherwise, the code would have been repetitive. This also helps increase readability and helps prevent the code from getting too cluttered.

The software testing techniques I applied in Task, Appointment, and Contact were white box testing and black box testing since it was structure and specification based. I regularly tested the code’s behavior via different inputs both by running the code and by JUnit testing. Each program also included boundary value analysis since there were requirements that had to be met for each variable in all 3 programs. To test for these requirements, I tested inputs outside of the required bounds to ensure the program didn’t accept such data. I did not use any security testing. If these programs were to be implemented and accessible to a customer/employee base, proper security testing would be a crucial step.

My mindset while creating Project One was willing to experiment, technical, and open. I ran and tested my code regularly and ensured there were no bugs in my code. If I found myself getting frustrated, I would take a break and decompress since it would affect the quality. While testing the code, I aimed to have the coverage percentage as close to 100% as possible. I did aim about the minimum required and wanted to make sure all requirements were met.

In order to be efficient when testing a program, there cannot be a bias. When you develop your own code, there is a certain level of bias that naturally comes with it. This was hardest for me when building and testing Appointment. I was unfamiliar with Date(), and struggled greatly with implementing it. My JUnit tests were failing, and it turned out that I wasn’t initializing it properly. I tried to remain unbiased by accepting test failures but would regularly check for syntax errors as well to ensure I had applied it properly.

It is crucial to be committed to quality in anything you do, especially when developing software. Cutting corners could cause unpredicted behaviors in your program. Not following proper commenting and naming conventions could make the code near impossible to read in the future, even for the original developer. This would compromise the quality and performance of the product in the long run. It could also cause for delays if the software needed to be updated in the future. To prevent technical debt, I plan to implement agile methodology into my development. This means running my code often, starting with most important requirements, and keeping an open line of communication between the development team and the client.