Treyton Davis

CS-320

October 14, 2023

Summary and Reflections Report

**Summary**

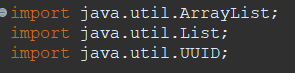
For each feature that was built for the system, I tried to ensure each requirement was included in the main code. For each requirement, I tried to ensure that there was a test built for each to ensure coverage across the entire system. For each of the classes, I ensured that there was a constructor for a few variations of what data could be entered into the system to ensure that each field has some form of data being entered into the system.

According to the coverage report from the system tests, there was only 6.3% left that wasn’t fully tested. I ensured that the tests were covering as much as possible to confirm no issues were able to be found within the system. Seeing as there was a 93.7% tested confirmation, I believe the system was tested as much as possible without compromising any parts of the system. I did realize after everything was finished that I only tested to really see if it worked the way it was supposed to. I never tried to create tests that could fail instead of passing everything. Even though there was only 6.3% missed, that accounted for 227 missed instruction that shouldn’t have been missed.

A screenshot of a computer

Description automatically generated

I ensured that my code was technically sound by ensuring that any APIs used wouldn’t have any vulnerabilities within the code. All APIs that I used in the process of developing the system were Java libraries that are already included in the software without requiring me to install any new versions of Java, nor having to build the program through Maven. I also ensured that Contact, Task, and Appointment classes had a way to verify that the ID, name, and descriptions could verify that each of the variables were able to be checked before assigning the variables to the list to store in the system.



The code for both the system and tests are as efficient as possible by ensuring there are no duplicated code, while also ensuring that everything is able to communicate between the files and connecting to the database. I also used an ArrayList to allow for storage of the data instead of connecting a whole database to ensure that the data can be used quickly.

A close up of a text

Description automatically generated





**Reflection**

The main technique that I employed in this project for testing was unit testing. The way that I used this was by building one piece of the program to allow for that code to be tested while the next piece of software could be built to allow the program to be built as soon as possible. One technique I thought about using but didn’t use was integration testing. The reason I didn’t use this was because only two pieces were connected at all. I also thought about acceptance testing but didn’t use this even though I should have because I would’ve known that I needed to test against failures instead of just passes. Unit testing helped me by allowing each piece of classes to be tested without another piece needing to be tested at the same time.

As a software tester, it was my responsibility to ensure that the software I tested was safe and effective. To achieve this, I was extremely cautious in my approach. I followed every step of the testing process carefully, paying close attention to the results and any potential issues. I also closely monitored the software's performance, looking for any unexpected or unintended consequences resulting from the changes I had made. I was particularly cautious when deploying the software in production or real-world scenarios, making sure that any bugs or issues had been identified and resolved before the software was released to users or made available to the public. In addition, I was constantly evaluating the software's quality assurance standards, making sure they were being met and that the software was meeting the requirements of the customer. In this way, I was able to ensure that the software I tested was reliable and safe for use.

Limiting bias in review of one’s own code can be very hard. Everyone thinks that their own code should be perfect from the beginning when it could be the worst code ever written. Bias could be a concern because someone might think the code works correctly even though it could be a hidden defect waiting to be released to the public.

Quality of code is a major component to software engineering. It could be the difference of having perfect tested code, or each line of code being buggy that could cause an issue across the whole system. It’s important to not cut corners when writing or testing because if a corner gets cut, it could leave a bug or defect in the system that could’ve been resolved before releasing the product to live production. I plan to avoid technical debt by allowing myself time after writing the main part of the code to be able to write self-built tests to ensure that the code gets tested before it gets sent to the main testers to allow them to reduce their work instead of forcing them to test everything themselves.