Jeremiah Larrabee

CS-320

Southern New Hampshire University

Project 2

**Summary**

I think I my approached followed the assignment requirements very closely. Let's look at the appointment services assignment for starters. I included checks that would make sure that the length of each variable didn’t exceed the allowed maximum. I will admit that these checks were not great because the tests that covered the areas would always have errors. These checks were in all of the assignments originally but I removed the throw command to avoid having the errors since I couldn’t figure out a fix. I also included checks to ensure that there were no duplicate ID’s and that checked worked every time. In the end I didn’t figure out how to know what my coverage percentage was exactly but from my grades on the last project I had at least 80% coverage. I ensured that my code was technically sound by running the functions in the test to ensure that they worked the correct way. Like I mentioned before my throw functions would always have errors so in that area my code was not sound. I think my code was efficient for the most part because I kept the functions short and as simple as I could to avoid over thinking things. In my original contact service submission, I had created a function that would update the contact but it was large with a lot of if statements that had multiple conditions that needed to be met. This function was not included in my final version of the code because I already had set functions that could update the contact and verify the input restrictions. Other functions simply had a single return statement that would return specific information.

**Reflection**

The main three techniques I used were unit testing, integration testing, and static testing. Unit testing is when the tester tests each individual company of the system before putting pieces together. This technique is important because it makes sure each piece works the way it is meant to and will save time testing in the later stages. This is shown best in my model tests instead of the service tests. The next step is to test how these pieces interact with each other when brought together. By ensuring they work together and give the correct outputs you are increasing the quality of the project as a whole while find potentially fatal bugs. Static testing is reviewing the code without executing it in order to find errors. There are different levels of this that range from informal review to formal inspections that are led by an individual. I’m not sure if me using the feedback you gave from the previous assignments to improve in coverage and using a map instead of an array. This form of testing is important because it increases the quality of the end product. Using the inspection will cost more in terms of time and money but should have better results because of how detailed it is.

One method that I didn’t not use in my projects is called decision table-based testing. This tests the system for different combinations of inputs and their respective outputs. This is useful for functions that have the user press a submit button after filling out data fields. For example, this could include billing information, login screens, or surveys. Another technique that I did not use is called state transition which is a black box technique that tests the output of a system based on the inputs. One application of this is a login system that only allows the user a set number of attempts. If the login is correct then the system proceeds to grant access but if it is wrong it moves to the next attempt and repeats until the limit is reached.

My mindset for this project was that things can always be improved and that bugs could be anywhere. To me this is a good mindset to have because if I thought that things couldn’t be improved then I would not grow my skills as a tester. The applications were somewhat complex because the services classes relied on the model class in order to function properly. If one had errors in it then the functions would fail.

The way I limited bias in the applications was by going over everything even if I was sure that I did it correctly. This might change in the future but as a college student I’m very aware that I will make plenty of mistakes that need to fixed even in areas I’m confident in. As a developer I think bias would be very concerning because it is sometimes hard to be objective and look at your code from a different perspective. For example, the developer could write code that would receive and validate input but decides to skim over the section because they are confident in their abilities then there could be a security risk depending on the purpose of the function.

In the real-world cutting corners will lead to issues down the road that could backfire spectacularly in your face. For example, if you cut corners in the testing phase the project could have serious bugs that could allow people to gain access to the system and cause damage to it. To avoid technical debt, I would stay focused on the task at hand during the development phase to reduce the risk of bugs and be sure to have a high-test coverage percentage during the testing phase to further reduce the risk. To further this I would have another team mate do a peer review of my approach to ensure that I couldn’t make it better.