Hutch Ellis

Professor Omar Toledo Lopez

CS 320 Project Two

13 April 2023

Summary and Reflections Report

When it comes to unit testing there are several different approaches available to take. There is black box testing, gray box testing, white box testing, bottom-up testing, top-down testing, and incremental approach just to name a few. I found that using the white box testing approach was most useful for the contact, task, and appointment services. Using the white box testing approach seemed the most logical to me because I had to write the code and wanted to ensure my code was free of errors while also ensuring that all the JUnit tests were passed. Before creating test cases I first read over the guidelines of whichever assignment I was working on and ensured that my test cases lined up with what was stated within the directions. In the task service I was given certain requirements for the task and task service classes. For the task class, I had to ensure each task had a unique task ID that could not be longer than 10 characters and could not be null or updatable, ensure each task required a name that could not be longer than 20 characerd and could not be null, each task also required a task description that could not be longer than 50 characters and could not be null. For the task service requirements class, I had to ensure that tasks could be added using a unique ID, ensure tasks could be deleted, and that the name and description of tasks be updatable using the task ID. I used these requirements to help develop my code and they gave me a promising idea of where to start. Following the guidelines and ensuring that the JUnit tests alligned with my written code was a fantastic way to not only ensure that my code was free of errors, but it also helped ensure that my code was functioning as intended. When starting the appointment service assignment, I was given similar requirements as with the task service assignment. For the appointment class, I was required to create a unique appointment ID string that could not be over 10 characters, not be updatable, and should not be null. I also had to create an appointment date field. This was one of the key differences with this assignment, I had to use java.util.Date to ensure that appointment date was not in the past. I also had to create an appointment description that could not be null or longer than 50 characters. With this assignment I kept getting errors on two of my JUnit tests but after a quick review of my code I found that I had accidentally put an extra character in the appointment ID, so it was 11 characters. The testing was being used to ensure the appointment ID could not be greater than 10 characters, so it was constantly throwing errors with each test. The appointment service class required that appointments be able to be added or deleted using the appointment ID. This assignment really showed me how useful testing your code really is because if not for the test I would have turned in code that contained errors and did not function properly. The third and final assignment, contact service, had similar requirements as the other two. For the contact class, I had to ensure a unique contact ID that could not be null, could not be updateable, and could not be greater than 10 characters. I also had to provide a first name and last name string each with a character limit of 10, a phone number string that had to be exactly 10 digits and provide an address field that could be no longer than 30 characters. The contact service class required that the user be able to add contacts with a unique ID, be able to delete contacts using the contact ID, and that the user be able to update the first name, last name, number, and address of contacts. Being that all these assignments share similar guidelines I found it easiest just to stick with the same unit testing approach throughout all three assignments. As mentioned earlier I used the white box testing approach because I wanted to ensure that the code that I had written was functional and free of error. When it comes to the testing techniques, I found that for the contact service the best technique to use was statement coverage. Using statement coverage allowed me to ensure that each statement within the code was executed at least one time. This technique was the right decision for the contact service since it consisted of managing and creating a list of contacts. For the task service assignment, I found the branch coverage technique to be the most logical option since it consisted of creating, updating, and deleting tasks. I wanted to ensure the program had full functionality and could carry out each of the assigned requirements. For the assignment service assignment, I found the unit testing technique to be the most logical because this assignment relied on the certain condition that the appointment is not in the past and I wanted to ensure that the code was free of basic and simple errors. There are several other testing techniques I could have used for this project. Some other testing techniques I could have used are mutation testing, security testing, dynamic analysis, and static analysis just to name a few. Mutation testing makes slight changes to the code to see if the tests can detect them. Security testing focuses on vulnerabilities within software’s code and architecture. Dynamic analysis testing evaluates the behavior of the software system during its runtime to check performance and functionality. Static analysis does not actually execute the code, it instead analyzes source code and design documents. Out of all the white box testing techniques I feel as though I chose the ones that are the most fitting for the assignments. Had I been working on a larger project with more in-depth code some other techniques would have been more practical.   
When acting as a software tester, it is always important to employ caution to help ensure the success of the project. Issues not found during testing could significantly impact the application's performance. To avoid simple mistakes, I made sure that my test cases and my code aligned with the project requirements. It is important to appreciate the complexity and interrelationships of the code because it makes it easier to find potential issues that could arise. To avoid bias during the review of my code I was sure to have an open mind the entire time and consider every scenario. I believe that testing your own code could lead to some type of bias because you may overlook simple mistakes that you have made without realizing and they could eventually come back to haunt you. Being disciplined in my commitment to quality as a software engineering professional it important because it will help ensure that my code meets my client’s requirements. Cutting corners in writing or testing code can lead to issues and could end up costing a lot of money. To avoid technical debt as a practitioner in the field, I plan to use only the best practices. One of the most important things to do is to address potential issues early in the development process and document any new issues that arise later in the development process. Doing this ban helps eliminate common errors from happening later down the road within the project and helps to ensure that the code is excellent quality and has appropriate functionality.

**References**

Khandelwal, A. (2019, November 16). *7 Different Types of White Box testing techniques: White box Testing Tools*. TestingGenez. Retrieved April 14, 2023, from <https://testinggenez.com/types-of-white-box-testing-techniques/>