Kyle Raja

CS 320

10/14/2022

Summary and Reflections Report

While constructing the code for the requirements of the three features, I found that they all followed similar sets of requirements as far as the user information was tested. Going through the tasks, I saw that some testing methods did vary, and I also saw the importance of each of the unit test scenarios. We can take for example the first task for Contract Service. Contract Service testing really involved testing the user input when they are entering important information such as name, street address, and phone number (just to name a few). An important test I saw was testing the phone numbers. Phone numbers in the United States per user typically are a total of 10 digits. One of the testing scenarios was to test the length of the phone number to make sure it is not too long. Sometimes a user may enter the same digit twice by accident which could trigger this test error handling. Another thing that I liked was to test if the value for phone number was null which is another word for empty. There were similar tests done for other user information that checked for the string lengths and to make sure the user did not leave important areas blank. Unit tests help ensure that the user experiences a better experience when using the application. For example, let’s say we did not have a test to check the validity of the phone number. It that application accepted that phone number, it could be problematic later down the road if that application needs the users phone number for some specific purpose. By having error handling implemented at the start of things, we can ensure that future problems can be avoided for the user.

I believe the quality of my Junit tests were quite sound. I think they handled what they were meant to do quite well. I mainly tested for input lengths and making sure no null values were entered. These are not difficult to do in code. What makes things more complex are testing for other scenarios that a user could possibly do when inputting their information. For example, on my Contact.java file lines 26-34, I show how the users phone number is tested. I used the if/else statement to check for these scenarios. I could have improved the code also to handle areas where if the user may enter a phone number that is too small. I think that the code was efficient because after I tested to make sure the phone number was not too long or null, I assigned the phone number to the private member variable of the class. I assigned this on my Contrac.java file line 68. This helped make sure that a valid value was assigned rather than a value that would cause problems later down the road.

The software testing techniques that used throughout the project incorporated a lot of OOP that helped things more clean and efficient in the code. For example, for each class whether the Contract Service, or Task, I created objects and methods. Some very efficient methods that I created were the getters and setters for these classes. These getters and setters were solely responsible for taking user information, going through a testing process like the one I described above for the user phone number, and then assign that value to an important member variable that is used throughout the rest of the application. We take the user input and set that value to the variable. Then, when we need that value, we call the get method, we access that value for what we need to do in the application. Methods of these classes are accessed through objects. I created objects from those classes so I could access the methods I created.

The uses for techniques that I described above are to ensure more clean code and have better efficiency through the application. We want our code to be as simple to read as possible. OOP things like I described above help with that.

The Mindset needed to ensure these tests involved a few things. I was careful to capture most of the scenarios that I believe a user may typically input in the application. However, more scenarios could have been tested which would increase the complexity of the application. I found that applying OOP with these test cases really helped to keep things organized. The unit tests helped to make sure also that things such as if a user was properly deleted, did happen. So basically, what the code does time and time again is to check for something before doing something. We check to make sure the person has been removed and if they have not, then we remove them. I think the caution is focused on not allowing bad data to enter the program. We do not want an invalid first name, last name, phone number, address, etc. This is because it can cause issues later down the road.

Some bias I could see from the review side is the structure of the code. I really like using if/else statements for testing especially. However, others may not find this to be a very efficient way of doing testing. There may be other ways that people want to do testing apart from if/else statements. Possibly creating try blocks or switch cases, etc. There are many ways to do testing.

Creating powerful tests is going to be more key towards being a successful developer in the future. More and more companies are going to expect developers creating unit tests so that their functionality can be constantly tested. This is because often in software development, things break. Things can break from a previous change committed, that is common all the time. By creating proper unit tests for developer changes, we can ensure that these changes can be better integrated into the application. Take for example, a developer commits a change where the phone number of the user is now required in the application. This requirement should be integrated with the unit testing also or it could cause issues in the application later when phone numbers will be used. By creating unit tests, we really are preventing a lot of headaches in the future for the developers to fix problems. Fixing problems costs the company time and money. Unit tests also help the application function without too many bugs. By creating unit tests after important functional changes to the application, developers can test and find the problems much faster and fix them before they hurt the rest of the application for clients.