Justin Brown

12/10/22

CS-320

Professor Kraya

Project Two Summary and Reflections Report

In project one, I created a mobile application for a customer. The project contained three different parts: contact, task, and appointment services. In the application, I used JUnit tests to test the code, and to provide better quality code for the customer. For each deliverable of the project, I had to align my testing techniques with the requirements given by the customer.

The first feature that I completed and wrote JUnit tests for was the contact section of the application. My approach to writing the JUnit tests was to have a test point that would test the success of my code. I would write these specifically around the constructor constraints that were given as requirements by the client. For example, in the contact class the ID string couldn’t be longer than 10 characters. So, I made a contact object that followed the requirements, then performed an assertequals test to make sure that the code behaved properly. The quality of my tests can be determined by the coverage. My coverage for the contact aspect was 93%. This shows that I adequately tested my success test point. Good coverage is considered to be 80-90%. I made sure that my code was technically sound by creating objects in my tests and making sure that the constructor behaved correctly. Next, for the service aspect, I would write tests to test the functions that I wrote aligned with the requirements. For example, contacts needed to be able to be added. So, I wrote an assertEquals test to make sure that the function call to addContacts would return true. For the confines of this project, the code was efficient enough if it behaved properly. With my tests that I did on the code, I can confidently say that my code was efficient.

The second and third features that I worked on were the task service, and the appointment service for the mobile application. My testing approach was like the first part because my test point was a success. I would code my application following the requirements given, and then wrote tests to make sure each individual part was working correctly. For example, the task object couldn’t have a name string that was longer than 20 characters. So, I would make a task object that followed this constraint and made sure the assertEquals test returned true. A better way that I could have written my test was to have a failure point to test as well. I should have made an object with a name field longer than 20 characters, and then made sure that the code produced an error. For the appointment class an interesting aspect was that you couldn’t make an appointment in the past. A better way I could’ve written my test was to make sure that the code produced an error and wouldn’t let you make an appointment in the past. I think for the purposes of this class my task and appointment features were both efficient and technically sound.

There are many different software testing techniques available to developers. The two main types of tests that I used for this project are unit testing and functional testing. Unit testing is used to test if each part of the code works as it should. The type of unit tests used for the application were JUnit tests. The other testing technique that I used was functional testing. This is used to check and make sure that the code produces the expected output. I used this to make sure that I correctly followed the requirements for the project. Some other types of software testing that I didn’t use are black box testing, white box testing, and automated testing. To briefly explain, black box testing is when the tester doesn’t know the software that they are writing test for, white box testing is when they know the software, and automated testing is using programs to automatically write tests for the code. Each type of test has a different situation where they would be the most optimal. For example, if you needed to focus on the speed of completion, an automated test would perhaps be the quickest way to achieve this.

An important aspect of being a software tester is mindset. It is crucial that you are cautious in how you go about testing. Cautious in the sense that you must try and provide a comprehensive test. For example, you would ideally want to have two different test points for your unit tests. One for success and one for failure. This would make the tests have better coverage. When writing bigger applications, there is a lot of complexity between the different parts of the code. A powerful aspect of unit testing is that it allows you to break the code up into smaller parts and test each individual part. This is important because it allows you, as the developer, to know that each part of the code is working as expected.

Limiting bias is an important concept when you are writing your own tests. As you can imagine, this is something that is of concern when there aren’t many developers working on a project. It would be easy for the developers to write tests that support code that they have written. This is a problem if the code they wrote wasn’t correct. For example, if they wrote a function to allow the user to delete contacts. If the function wasn’t correctly following the requirements given by the client, the computer wouldn’t know that. So, the developer could write tests that have high coverage but still aren’t correct.

Being quality oriented is an important trait to have as a software engineer. A lot of projects worked on in the field are used in the military, or in the field of science. It is extremely important that the software behaves as is expected, or there could be serious implications. It is important not to cut corners when writing code because that allows a higher chance for problems to occur. Moving forward in my career, I am going to try and write code to the best of my ability. This is important not only in developing a better product, but also developing a product that is trustworthy and safe to use.

References

The QA Lead, & Boog, J. (2022, December 10). *9 types of software testing in software engineering*. The QA Lead. Retrieved December 11, 2022, from https://theqalead.com/test-management/types-of-software-testing/