Hadi Abdelaal

CS-320

Dr. Kraya

**Reflection & Summary**

Unit testing could help minimize the risk of mistakes in the software and enhance the general quality of the source code by detecting faults and repairing them right from the ground up. Whenever it comes to developing solutions for software, testing for units is critical. It reduces the likelihood of software mistakes and improves code quality by detecting and correcting issues promptly. The unit testing technique I used for the three features was similar since I adapted it to meet the software's needs. A significant instance was the Contact service, which possessed a few particular demands such as the Contact object requiring distinctive contact ID strings that cannot be more time than ten characters and are unable to be null, they additionally required the contact object to demand a firstName and lastName that are unable to be more time than ten characters and shouldn't be null, they also desired a phone number field the fact that is a string and has to have precisely ten numbers and shouldn't contain null.

I wrote unit tests for the contact, task, and appointment services within the consumer's smartphone app. I used a mix of black-box and white-box testing to guarantee the tests were efficient. Submitting data and confirming that the right output has been generated was part of the black box assessment. White box testing entailed inspecting the program's code and checking that the reasoning was correct. I additionally utilized JUnit test cases to look for unusual situations and faults in the source code. As a result, I managed to cover 99 percent of the code, confirming its effectiveness and efficiency. Developing the JUnit tests was an enjoyable endeavor for me. I was enabled to guarantee the technical soundness of my code by confirming that the reasoning was accurate and that the intended outputs were delivered. I utilized a mix of strategies, such as refactoring and deleting unnecessary code, to guarantee the code was efficient. JUnit assertions were also useful in ensuring that the desired outcomes were obtained.

In addition to sharing some of my Junit testing methodologies, I'd like to say that my encounter with the Junit testing was amazing. I recognize that I nevertheless have plenty to learn, so I attempted to apply what I had previously acquired to guarantee the code was theoretically good. We developed taskTest, which checked the correctness of the parameters required for the contact class by utilizing an example case and making sure the testID hadn't become too lengthy, as seen in my code beneath.

A computer screen shot of a program

Description automatically generated

ContactService was employed to test an action that creates a fresh contact. The code following demonstrates how I evaluated the program for efficacy in bringing all three aspects altogether. Also, as I mentioned above there is a lot to be learned but I guess I have done the best I could to evaluate my code by this kind of JUnit testing.

A screenshot of a computer program

Description automatically generated

**The Reflection:**

The Dynamic testing approach, which involves looking at the dynamic characteristics of program code, was one of the software development strategies I employed. I utilized this approach to examine the actions of many changing factors that are not consistent and identify weak points.

Throughout the course of program execution. Another software technique that I did not employ was static testing is a method of evaluating code without running it. The primary distinction between static and the disadvantage of dynamic testing is that it is difficult to detect flaws.as violations of standards for development and the discovery of interdependence and anomalies in versions regarding the software are put forward. I employed numerous software approaches, but among them was the white box testing technique, often known as dynamic testing, which includes examining the code's actions. When the application was running, I also examined the variables that were not constant to locate the weak spots. Statically testing, a method I failed to employ, is another approach for software evaluation in which you test before ever activating the code. The explanation for this is that dynamic testing enables the detection of any mistakes, faults, or irregularities identified within the computer code when it is executing.

With the mentality(mindset) I used in this endeavor, I was possible to assure that all of the test cases were captured. Whenever it came to evaluating the numerous cases to be verified, I was cautious. It was critical to understand the intricacy and interdependence of the code I was evaluating since each of the class instances had particular demands that had to be met in order to certify the code as running properly. All of that necessitates the individual understanding of what each component of the code accomplishes. For example, locations in which each unit of the class requires a special condition for operation or the contact service that has certain specific core activities to carry out properly.

Whenever it comes to evaluating your code, I think bias is a big component that must be eliminated, because the individual feels confidence in the program they developed and so will not perform sufficient testing of the code's components. For example, when defining variables in a class, you may be certain that the source code is successful since you wrote it. Whenever it comes to testing software, we ought all endeavor to keep it to a minimum.

As a software development specialist, discipline is essential for assuring excellence because it is what ensures a smooth process of producing a program with fewer defects and code mistakes. It is critical not to cut corners while creating or evaluating code since millions of dollars and even lives may be saved by fully evaluating the scripts and verifying everything works correctly before delivering the code to operation. For instance, when utilizing the Contact service, guarantee each contact has a unique ID that is not updatable, and this is carried out to guarantee each contact is distinctive and cannot be switched with another contact to avoid duplication.

Also, to add to the above, When it came to the attitude I used while completing this project, I was very careful and cognizant of the code's intricacy. I attempted to prevent prejudice in my code review by extensively examining the code's performance utilizing inspections of code and unit tests. As a programmer, it is critical to be cognizant of the possibility of prejudice and to verify your own code thoroughly.

Ultimately, when creating and testing code, it is critical to be consistent in one's dedication to reliability. To minimize technical liabilities, it is critical that you avoid taking corners when creating or testing code.

**Resources:**

1. *JUnit Tutorial With Examples: Setting Up, Writing, and Running Java Unit Tests*. (2018, August 2). Parasoft. https://www.parasoft.com/blog/junit-tutorial-setting-up-writing-and-running-java-unit-tests/