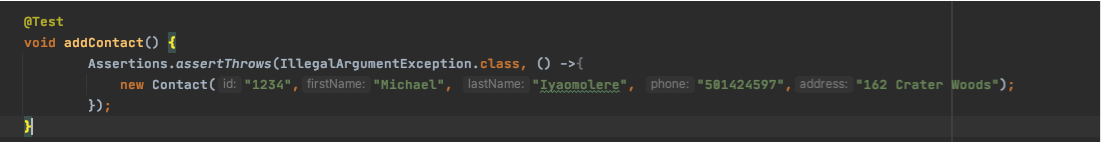
Unit testing is significantly important in regard to creating software solutions. It assists in minimizing the possibility of application bugs as well as enhancing code quality by rapidly searching for flaws and fixing them. I believe that my approach to unit testing for the three features was aligned to the software requirements. A good example was the contact service, which had certain needs such as the contact object must have a unique contact ID string , must not be larger than ten numbers, and must not be null. The same thing goes for the first and last name string and a phone string is also needed to be precisely ten digits and not null. Finally, an address field is needed no longer than 30 numbers and not null.

The contact object includes a service that implements crud operations with contact object. All of these were certain needs for one out of three features, and the other features had demands that were alike. Therefore, the software needs to be executed with these needs in order to pass the unit test. The overall quality of my unit test was good, and I tested every field and operation in order to examine the validity of test cases. My experience with the Junit test was really great. I know I still need to learn more, but I used the knowledge I have acquired to make sure that my code was technically sound.



The code above was examining the validity of the parameters of the contact class and I developed a test case that was examining if the testID was longer. This was a great approach to verifying if my code was efficient.



This code is also evaluating an operation on the contact service that will add a new contact. One of the software techniques that I utilized was the dynamic testing method that tested dynamic behavior of the code. I utilized this technique in order to verify the behaviors of the different variables that weren’t constant and to discover the weak areas throughout software runtime. The other software technique that was not utilized was static testing that tests code but with no execution. The main difference between dynamic and static is that identification of defects can’t be simply discovered by dynamic testing like development standard breaches and detecting dependencies and inconsistencies in software models.

My mindset adopted to this project because it allowed me to make sure that I included all of the test cases. I used caution in regard to testing several test cases that needed to be examined. It was critical to appreciate the complexity and interrelationships of the code that was tested since every classes objects had a few demands that were required to be met in order to ensure the code was functioning properly such as the areas where every class unit had certain needs to be capable of working or the contact service that included certain crude operations to be executed correctly.

I feel that code review has bias that can be a significant factor that must be minimized because the individual thinks that he is confident about the code written and just won’t conduct any tests of the code units. For instance, when developing class variables, you might be sure that since you were the creator the code must be correct. Discipline in regard to ensuring quality as a software engineer is really important because this makes sure that the process of developing a program is flawless and doesn’t include any coding flaws or errors. It is also important to not cut corners in regard to developing and testing code since a lot of money can be lost and also lead to deaths of innocent lives. Therefore, ensuring that code is created and tested correctly prior to the code being released can definitely help prevent all of this from happening. For instance, in the contact service, we needed to make sure that every contact had a unique ID that isn’t updatable in order to make sure that every contact cannot be interchanged with another contact.

**References**

Pittet, S. (n.d.). *Introduction to Code Coverage*. Atlassian. Retrieved February 26, 2022, from https://www.atlassian.com/continuous-delivery/software-testing/code-coverage

DeLoach, E. (2017, December 4). *How to Evaluate, Manage, and Avoid Technical Debt*. Metova. Retrieved February 26, 2022, from https://metova.com/how-to-evaluate-manage-and-avoid-technical-debt/