Project Two: Summary and reflection Report

When creating software solutions, unit testing is crucial. Errors in software are less likely to occur, and the code quality is enhanced when problems are discovered and fixed more rapidly. For consistency with the software specifications, I used a same unit testing strategy for all three functionalities. The Contact service project  is a good illustration of a service with strict requirements, such as the requirement that each Contact object have a unique ContactId string that is no more than ten characters long and cannot be null or updatable, as well as the requirement that each Contact object have a first Name and last Name that are no more than ten characters long and shall not be null, as well as the requirement that each Contact object have a Phone field that is a string and must be exactly ten digits long and shall not be empty.

The ContactObjects also has a service attached to it that does some sloppy work with it. There was a specific need for one of the three features, and the other two features had similar demands, therefore the software had to be written to meet those standards in order to pass the unit test. My unit tests had adequate coverage since I examined each field and procedure to ensure that my test cases were accurate. The Junit exam was a great experience for me. While acknowledging that I had much to learn, I did my best to use what I did know to write technically sound code.



The preceding line of code validated the contact class parameters, and I also made a test case that determined whether or not the ContactID, getFirstName, getLastName, getPhoneNumber, getAddress, testToString was too lengthy. This method of verifying the code's efficiency was both efficient and effective.

Reflection

Testing the liveliness of the product in question was a key part of my software development process, therefore I turned to the dynamic testing methodology. This method helped me examine the software's runtime behavior and identify the weak spots caused by the many dynamic variables that are not constant. Static testing, which involves verifying the code without running it, is the second software technique that I did not use. Static testing is superior than dynamic testing because it can quickly identify faults that dynamic testing cannot, such as violations of development standards and the discovery of dependencies and inconsistencies in the program model. Because of my frame of mind throughout this project, I was able to guarantee that every test case was captured. proceeded with care while testing the numerous situations that needed to be verified. Each class object had certain conditions that needed to be met in order for me to verify the code as operating correctly, therefore it was crucial that I understand the code's intricacy and interdependencies. Areas where each class unit needs a certain something in order to operate, or the contact service that needed to do certain rudimentary actions in order to do so efficiently.

I think that confidence in one's own code prevents one from evaluating its individual components thoroughly, making bias a crucial element to be mitigated while reviewing one's code. When you create something like the variables in a class, you could justifiable that they'll work well since you made them. As testers, we should all do our best to keep it to a minimum throughout software development. A software engineer's ability to maintain discipline throughout the development process is crucial since it allows for the creation of a bug-free, high-quality product.

Avoiding millions in financial damage and even loss of life may be accomplished by testing the programs correctly and making sure everything is operating precisely before sending the code to production, making it imperative that no corners be made while creating or testing code. For instance, the Contact service was used to provide each contact a unique, immutable identifier, guaranteeing that no two contacts would ever be confused with one another.

References

Tudose, C. (2020). *JUnit in Action, Third Edition* (3rd ed.). Manning.

Appel, F. (2015). *Testing with JUnit*. Packt Publishing.