Summary and Reflection

My testing approach is not 100 percent aligned with the Contact and Task Service software requirements. But it’s very close to software requirements and I made sure to write JUnit test cases to test out every single component. When I wrote the code for both Contact Service and Task Service assignments, I made sure that attributes are not null and meet the length requirements specified in the assignments. I achieved that by writing if statements to verify those requirements and if they failed to meet the requirements, I wrote error handling exception to make sure the program does not break. In my code, I wrote the IllegalArgumentException if the requirements are not met. Overall quality of JUnit tests is good, and I feel like I could have written more unit test to increase coverage near to 100 percent. Each unit tests covers at least testing one method with different output to make sure it’s working as expected. Well, looking into JUnit test coverage, I was confident enough that my program will not break down easily and it ensures correctness and efficiency of the code. Even though, we can’t cover every single corner especially with large program, but enough to ensure that program will run safely. To make sure my code was technically sounds, I wrote the code with the help of IDE to implement the best technical practices and wrote comments to explain what each method functions are. Examples from my program would be private attributes I created to ensure that they cannot be access outside of the class and getters and setter to fetch and make changes to attributes values. Latest IDEs are very helpful in helping you write efficient code and same goes for Eclipse IDE, which I used for writing my Java program. It ensures that you have sufficient code and point out unnecessary lines of code to make the program efficient and less ambiguous.

Throughout Modules Three, Four, and Five, I have written JUnit Test cases based on the requirements that was provided to me by the clients. Before working on these test cases, I had very little knowledge about the JUnit Test cases and as I progressed through each module, I have learned several software testing techniques by researching on how to write effective Unit Test cases for software products and through Instructor feedbacks, which helped me tremendously in writing efficient and effective test cases for my project. Before proceeding to write my Unit Test cases, I thoroughly analyze and read the client’s requirements, and I jot down the important requirements that stands out. For instance, Client does not want ID variable to be null and want to be under certain length. So, when I am writing code, I keep in mind to include the mechanism to prevent user or system from leaving the field empty or exceeding the length of the field. When it comes to writing JUnit test cases for my program, I wrote one JUnit test case for each class to narrows down the tests and make sure each component is working as expected. Then in each JUnit test case, I wrote one test method for testing out each method in the classes and for some I wrote several test methods as code gets more complicated because each test method does not suffice enough to be sure that my code won’t break down. Lastly, when I run the JUnit Test cases against my program, I checked all the Test cases passed and make sure coverage is near 100 percent. Test Coverage, which I did not know in Module Three and It is very helpful because it gives insight which lines are testing and not testing. This is the best way to improve your test coverage to make you check every aspect of the code before you deploy your code into production.

I should have written more Test cases to increase coverage near to 100 percent by planning for test cases ahead of time before implementation. Test takes more time than developing the code and I feel like I didn’t spend enough time testing my code before submitting it. I should have start testing with small part of the code then move onto the larger components. Try to run each test method several times using RepeatedTest () tag to make sure is not going to break.

So far what I have discussed, it is applicable for every project and use cases. Read the requirements carefully and thoroughly. Then analyze it and write down the important requirements, which can be helpful in writing test cases. Start writing test cases as you are developing the software.

For this project, I tested the program as an end user and tried different inputs to match the output with results I was expecting. Keeping that in mind helped me in writing better test cases and cover more ground that left untouched. As a Software Developer, we must ensure end-user satisfaction and making sure that product is error free. As I was writing user test cases, I was looking at test cases from user perspective instead of mine. There are concepts, we as programmers can only understand and users no clue what that means. As software get more complicated, it gets harder to write test cases to cover single area of the code and we can’t promise 100 percent test coverage to clients instead we have to focus on the quality of the software test cases.

Definitely, there will be bias when you are testing your own code. You will think your code is error free and there should not be errors or bugs in the code. But, when someone else test your code, they happen to find many errors in your program. Same goes for me, when I finish writing my code and started writing test cases, I told to myself that there won’t be any errors because I wrote it. But, when I perform thorough testing, I actually found few bugs in my code.

I believe software quality is a vital aspect of Software Engineering Professional and we can’t neglect writing high quality and clean code. I try to keep my code as simple as possible. So, others can understand my code better and make future updates. I follow these practices whenever I write code. As we have seen throughout the history, there have been many incidents related to software bugs, which resulted in personal injuries and financial losses that’s why we can’t cut corners when it comes to writing or testing code.