Project 2 : Reflection

For each of the 3 components of our previous project there were certain parameters each variable had to abide by. Within the code it can be demonstrated with any of the “ == null” when throwing an illegal argument exception and also setting a maximum character content for each string associated with the variables in EACH the components of Task, Contacts, Appointments. JUnit test quality within my code was at least on par with 73% overall coverage I do believe with my worst being the TaskServicetest class which was <10%. Examples were testing each component of the initial (getter/setters) for appropriate parameters limits. By purposefully passing/failing JUnits based on character lengths or no entry.

A direct example of proper JUnit test code, Assertions.assertThrows(IllegalArgumentException.class, () -> {

new Task ("Djohn", "Honoraryian12", "This Name is Random");

This was use to fail a test for the last name variable. This succeeded. Demonstrating how technically sound the JUnit test was able to pass this specific test with a green check mark upon its run config. Also demonstrating efficient code this is a very simplistic line to read that performed its allotted task well.

Within this project J-Unit test cases were utilized by asserting claims that would/would not pass through the code. For example throwing a variable description in that is longer than it’s allotted length. This is done by first setting the length of the variable before creating an assertion of this variable in a length exceeding its limitation. A form of testing we did not perform was a performance test which benchmarks software components. This can demonstrate whether or not code will be fast when under a high load. Another form of testing that we did not utilize within this class is boundary value analysis. This tests boundaries between partitions including max, min, inside/outside boundaries, typical values, and errors. Performance testing allows for someone to know if code is efficient and practical for real world utilization while BVA tests to make sure the range of assumptions will test well within a software.

Caution within coding during my stand in as a software tester was done particularly in the code itself. Trying to be as concise and to the point as possible. As there are multiple answers to any coding assignment based on one's methods, not all methods are created equal. When I say this I mean, depending on the way you code… testing can be more or less of a headache. I tried specifically to keep my code as simplistic and to the point as possible, because the more variables, methods, and roundabout solutions you use; the harder it is to write successful tests. Specifically for the parameters we were given for each step in the project there were multiple ways to initialize the variables with their set parameters. The means in which was demonstrated within the tutorial video we had initially was the method I chose to stick with after having found multiple others, as it took less space and headache.

Personally I do not know how someone could be bias toward their own code UNLESS they’re conceited to begin with. I would find myself more along the lines of always doubting my code and tests… I am still quite the ‘noob’ to coding so that may change in the far future. Discipline is probably the most important value a software developer can have as coding/programming as a profession in general takes a lot of discipline to be successful at, from my own experience discipline is probably one of the largest factors to propagate a software engineer’s future. Cutting corners when testing code can lead a team or yourself to REDOING work hence wasting time, effort, and money. Testing after diligently working code (as you work) can lead to far less steps backward as you have then verified a component as functional. Technical debt can then be avoided by due diligence, patience, foundational development, and practice. By performing Junit tests throughout your work you can avoid having to later go back over functions that you had coded much further back in time.

REFERENCE:

*Software Testing Techniques with Test Case Design Examples*. Guru99. (n.d.). https://www.guru99.com/software-testing-techniques.html.