Jhariel Almonte Ventura

April 11, 2023

Software Test Automation

Project Two – Summary and Reflection Report

As stated in module five’s journal, although I did not implement the practices intentionally, I implemented the following testing techniques: integration testing, unit testing, and functional testing while developing my tests for each individual requirement. Each class’s member function was individually tested with valid and invalid inputs. For instance, the Task class has a member variable taskID that could not be longer than 10 integer values, the input was validated by using values longer than 10 integers and null values. With respect to the coverage percentage, I noticed an increment in the coverage percentages as we progressed through the modules and features, in addition to an improvement on the quality of the code itself, as during the first feature I was not testing the user input in the setters, which essentially defeated the whole purpose of validating the user’s input when the objects were created. However, despite in the last feature implementation I made sure to validate the user’s input in the setters, as well as test these member functions individually in my unit tests.

I believe the code of the first two features could’ve been more efficient, as I was not validating the user input in the setters, which ultimately ends up in a logical error by allowing the user to change the object’s member variables with invalid values such as firstName longer than 10 characters in the Contact class.

if(firstName == null || firstName.length() > 10) {

            throw new IllegalArgumentException("Invalid input, must be less than 10 characters");

        }

public void setFirstName(String firstName) {

        this.firstName = firstName;

    }

However, I changed the logic of the setters in the final Mobile Application project by validating the user’s input in the setters, which improved the efficiency of the code and its logic, which were tested as well.

public void setFirstName(String firstName) {

        if(firstName == null || firstName.length() > 10) {

            throw new IllegalArgumentException("Invalid input, must be less than 10 characters");

        }

        this.firstName = firstName;

    }

Among the techniques I implemented while coding the tests, I used integration testing, functional testing, and unit testing, which consists of testing each individual component to verify the logic and functionality of these components works as intended. I implemented this technique by testing the logic and data structure involved in the base classes of each milestone. Functional testing consists of verifying the logic of each function, making sure that it compiles and yields the expected result. I implemented this technique by testing each function in both the base classes (Task, Appointment, Contact), and the service classes. Integration testing is the process of testing components as they are integrated. I employed this technique by testing each function as I implemented them, to make sure the logic of the base classes such as Task, Appointment, Contact classes were working correctly before transitioning into the Service classes.

There are multiple testing techniques I did not implement, such as static testing as no documentation was required for this particular project, black-box testing as I was essentially playing each role from back-end to the user-end side of the project, hence, I was aware of the functionality of the program, defeating the whole purpose of black-box testing.

As I stated in module five’s journal, I believe as developers we should formulate our approach to projects depending on the complexity and the scope of the project itself and personal preference, or in the case of a more professional environment, by direction of the team lead/product owner. In our case, I did not believe it was necessary to formally document the testing process, bugs or create “tickets” per say; however, I believe in complex projects performance testing would be necessary to ensure the quality and scalability of the code.

From the beginning of the implementations, or development “cycle”, I was aware that I would be making some mistakes while creating the tests as I have not done it before, but I was always aiming to test each basic functionality and the fundamental logic of each class and their member functions as these worked in conjunction with “Service” classes to create the basic functionality of each feature. However, while testing the code, and looking back at the code, at some point I told myself “What was I thinking while writing this line?” and immediately went back to the drawing board and made changes such as the setters input validation…somehow that slipped my mind while creating the code, which takes me to my next point: quality of the code improved progressively through each implementation. However, this small logical error in a relatively small project, could result in billions of dollars in loss, looking at you Y2K bug, as this resulted in a global technical hassle caused by a bug in the integer values of the calendars in programs, which affected date-sensitive applications and organizations that relied on these programs such as banks and nuclear plants…all because of an invalid value…