Victor Gomez

June 17, 2023

Instructor Ramsey Kraya

CS-320-T5518

Project Two Reflection

My approach for unit testing the three features for this assignment included thoroughly reading Grand Strand Systems requirements and ensuring that all their requirements were fulfilled. In order to achieve this, I had to design different aspects of the program to meet the necessary specifications required. These specifications included things such as: there could be no nulls, a character limit for different inputs needed to be followed, and I made sure that appointments could not be made for dates that had already passed. The overall quality of my JUnit tests was exemplary. This can be seen by the coverage percentages of all my JUnit tests being more than 80%.

My experience writing JUnit tests was exceptional, and I thoroughly enjoyed it. To ensure my code was technically sound and efficient I made sure to include throws. This helps the JUnit tests to recognize thrown errors and thus makes the job of the JUnit tests much easier. An example of this from my code can be seen here:

public Appoinment(String appID, Date appDate, String descForDate) {

if(appID != null && appID.length() <= 10) {

this.appID = appID;

}

else {

throw new IllegalArgumentException("Invalid ID");

}

*@*Test

public void testAppID() {

Assertions.assertThrows(IllegalArgumentException.class, () ->

{new Appoinment("A12123321654", new Date(), "Gomez");});

}

*@*Test

public void testAppIDNull() {

Assertions.assertThrows(IllegalArgumentException.class, () ->

{new Appoinment(null, new Date(), "Gomez");});

}

In this example, you can see that in the main code if an appointment ID (appID) is null or greater than 10, then an IllegalArgumentException will be thrown. The JUnit test is able to recognize this throw and determines if the test passes or fails given the criteria.

As far as software testing techniques are concerned, I employed assumptions and asserting exceptions. These I felt were the easiest to use because I could catch any piece of code that was thrown and I could find if certain conditions were expected. In my appointment tests I used assumptions when I wanted to test if my addAppoinment object worked. I would use assertTrue to test if it was working in my JUnit. I also tested with asserting exceptions by placing IllegalArgumentExceptions in the code for the JUnit test to catch the throw and check if it was correct or not.

Some examples of software testing techniques that I did not use were conditional tests, and tagging and filtering tests. Tagging and filtering tests focus on tags that can be filtered to discover and execute code later in the testing method. Some of the criteria for this test is a tag must not be null or blank, a trimmed tag must not contain a white space, and it must also not contain ISO control characters. Conditional testing involves testing certain types of branches with certain types of tests. An example of this could be a test can deactivate all tests or individual tests depending on a certain key factor such as operating system or key word.

When acting as a software tester, it is imperative to employ caution. It is important to execute caution because a bug or faulty program that slips by could have grave consequences such as the loss of a life, monetary resources, or reputation. It is also important to appreciate the complexity and interrelationships of the code I was testing because they all work together to complete their mission to successfully make an appointment for a unique individual. I tried to limit bias in my review of my code by being willing to recognize that my programs could have errors in them. I made sure of this by testing all aspects of the code. In terms of the role bias plays for a software developer who is responsible for testing their own code, I believe there is great concern. Software developers should never test their own code. As previously stated, it is important to be disciplined when it comes to quality because of the severe consequences that could occur from a lack of discipline. Moving forward, to avoid technical debt as a practitioner in the field I plan on testing early and often. This will ensure that I exercise caution and discipline when it comes to the quality and integrity of my code.