Kaitlyn Lush

CS 320

Project Two

The approach I took while writing my tests varied depending on what feature I was working on. Since requirements were given for each feature, I wanted to make sure those requirements were being met. While compiling my unit tests I made sure that they directly aligned with the requirements given for each feature. For example, in AppointmentTest.java the description for the appointment could not exceed 50 characters. Therefore, I made sure that the test implemented that by adding this to my test code:

**assertThrows(IllegalArgumentException.class, () -> appointment.setDesc("This test will make sure that the appointment description does not exceed 50 characters"));**

Another example is in my contactTest.java where the first name could not exceed 10 characters. For this I made sure to include a portion that would test for that:

**@Test**

**@DisplayName("Create a contact with a first name longer than 10 characters")**

**void testContactFirstNameTooLong() {**

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

**new contact("A42197", "Kaitalionate", "Lush", "5028645215", "688 Puppy Street, Dog City");**

**});}**

I think that the overall quality of my JUnit testing could definitely use some work since I did have some errors in the process. I think that with more practice and time using them the quality would improve.

In order to ensure that my code was technically sound by using data lists such as arrays as well as many common algorithms such as add, equals, and length factors. I also made sure to add tests in my JUnit tests to figure out how each of these features worked. For example, in my taskTest.java I made sure to test that the variable was being set properly:

**@Test**

**void testSetName() {**

**Task task = new Task("987654321", "CheckThis", "Check this grocery list for items");**

**task.setName("CheckThis");**

**Assertions.assertEquals("CheckThis", task.getName());**

Also, in my AppointmentServiceTest.java I created a test case to ensure that the appointment was being added:

**@Test**

**void addAppTest() {**

**Appointment appointment = new Appointment("987654321", date, "Appointment description goes here");**

**assertEquals(true, AppointmentService.addApp("987654321", date, "Appointment description goes here"));**

By using assertions such as “assertEquals”, “assertTrue”, and “assertThrows” I was able to catch any errors that were being thrown.

I am still learning how to code properly and definitely have room to grow but I did as much as I could in order to ensure that my code was efficient. While coding I made sure that any errors that popped up were fixed. I also used test cases to verify that certain things didn’t already exist before adding them. For example, in my TaskServiceTest.java, I tested to see if the task was found before deleting:

**@Test**

**void testDeleteTaskNotFound()throws Exception {**

**taskService taskService = new taskService();**

**taskService.addTask("CheckThis", "Check this grocery list for items");**

**assertEquals(1, taskService.getTaskList().size());**

**assertThrows(Exception.class,()-> taskService.deleteTask("987654321"));**

**assertEquals(1,taskService.getTaskList().size());**

During this project there were a couple different software testing techniques that I used while developing. A could of the techniques that I used in the project are black box and white box testing. JUnit testing would be considered white box testing. This form of testing allowed me to see how my program was performing. It helped to provide insight into what was working in my program and what was not. Black box testing was employed also because most of the testing that I used was testing the functionality of the program as well as its behavior.

There are of course a lot of software testing techniques that I did not use for this project. One of these would be security testing since this project did not call for the use of security testing. Security testing is used to find parts of the code that could lead to security issues. It allows the user to find any flaws that could potentially cause leaks of internal and private information about the users. I also did not use exploratory testing in this project. Exploratory testing involves the tester actively searching for any defects in the program without any specific test plan in mind.

While working on this project the mindset that I adopted was that of growth. My coding skills definitely have room for improvement so while working with JUnit testing for the first time I wanted my mindset to be learning from my errors. I employed caution while writing my code by trying my best to ensure the code was error free before testing the program. I also researched as much as I could to gain further knowledge of how to properly write this project. It’s important to appreciate the complexity and interrelationships of the code I was testing because they all work together to provide a working program. If one portion of the code is wrong, then the whole thing will not function properly. While working through the project I tried to fix as many errors as I could from the initial milestones. This way I could be confident that the program would function.

I also tried to limit bias in my testing by testing for several different factors of one function. For example, in my task test I created several different test cases for name, task ID, and description. I did this by writing test cases for instances where they would be too long and whether or not they were null. By limiting your bias, you can ensure that the program is functioning properly regardless of if you know it will run correctly or not. Bias could be dangerous when testing your own code because if you were only to test the factors you knew would work you would be missing errors for things that didn’t run properly. By testing the inputs for multiple different instances, you can be sure that you are finding any errors that may occur within your program. It will also help the program to run more efficiently.

It is extremely important to be disciplined in your commitment to quality since you are creating a program for other people to use. By being disciplined you can be confident that your program runs correctly and efficiently. It is vital that we don’t cut corners as software developers because we want to be sure that our work is functioning properly and so that it doesn’t come back later needing any errors you didn’t find the first time to be fixed. In order to avoid technical debt as a practitioner I will be sure to get my coding skills up to a higher level. I will also be sure that I am using all of the techniques I have learned to test my code properly. By testing correctly and frequently it will allow me to have confidence in the work I put out.