Brylene Patrick

CS 320 Software Test Automation& QA

Project Two: Summary and Reflections Report

**Summary**

Each of my testing methods was implemented following the customer’s requirements. For the task service application, there were three constructors such as the task ID, name, and description. Then, I applied conditional statements to put a limit on how many characters the users can input and ensure that the prompts are not null. Along with the conditional statements was also the application of the function, IllegalArgumentException, where if the conditional statements are not met, then an error message is outputted. I applied the same approach to the contact service application to ensure that it addresses each requirement.

The overall quality of my JUnit tests for the contact service and task service was successful because both produced more than 90% coverage for the test files. I ensured that each test method followed the customer’s requirements such as the number of characters allowed per task object and avoiding null input. In addition, the methods for adding, deleting, and updating the contacts and tasks were tested by inputting test cases to ensure it performs the functions properly. Throughout the tests, different test inputs were applied in the application where one meets the conditional statements and one that doesn’t make sure that it detects both.

**Reflection**

1. Testing Techniques

During the first milestone, I implemented a test method to check for null and character length input. With the character length, I set up restrictions to allow for a certain number of characters allowed with the implementation of throwing IllegalArgumentException if the input exceeds. Using the IllegalArhumentException function helps with checking that the user has valid input to avoid irrelevant and cluttered data in the system. In the second milestone, I implemented an assert function where it would detect if the input didn’t meet the conditional statements leading to a successful test. The assert method ensures that each test unit successfully detects invalid input. During the third milestone, I imported the java.util.Date to use the function before(new Date) to ensure that the appointment date field input is not in the past.

Software testing techniques that I didn't use are fixtures and test suites. Fixtures are implemented with a fixed set of objects. These objects are used as a baseline for running JUnit tests with the main goal of ensuring a fixed environment where tests generate repeatable results. Fixtures include the setUp( ) and tearDown( ) methods. The setUp( ) method is run before every test execution while the tearDown( ) method is run after every test method. Fixtures prevent repetitive writing of test input since values are already set up. Test suites are a bundle of unit test cases that are run together using the @RunWith and @Suite annotation. These annotations are attached with reference to JUnit classes. The test suite allows for the aggregation of test cases from classes in one place and compiling them together.

1. Mindset

The mindset I adopted while writing the code was being detail-oriented and cautious to ensure that each of my tests and test cases align with the requirements. For example, in the contact.java file, I implemented a test method to output an invalid first name input if the user’s input is null and has more than 10 characters. To further test this method, in the contactTest.java file, I implemented a test method to check if it detects both correct and incorrect input. The correct input was tested using the assertTrue function while the incorrect input was tested using the assertThrows with IllegalArgumentException function. It is important to understand the complexity and interrelationships of the code because each test methods in the files work together to generate a successful JUnit test application.

To limit bias in my review of the code, I utilized unit tests in thorough conjunction with the requirements to ensure that each test is successful. For example, in the appointmentTest.java file, I implemented a test method to check for past date input. I strictly followed the methods to construct this function through the provided resource to ensure that I write a technical function.

It is important to adopt a disciplined trait as a software developer to provide a great quality application. Cutting corners leads to a negative impact on the application because it births many risks of missed details and requirements. To avoid technical debt, I plan to be always detail-oriented and continue my education to maintain competency in the field.