Project Two

Sameeh Alqatabi

Southern New Hampshire University

**Project Two**

The software requirements for Contact Service were to add contacts with a unique ID, delete contacts per contact ID, and update contact fields per contact ID but the only fields that are updatable are first name, last name, phone number, and address. The software requirements for Task Service were to add tasks with a unique ID, delete tasks per task ID, and update task fields per task ID but the only fields that are updatable are name and description. The software requirements for Appointment Service were to add appointments with a unique ID, delete appointments per appointment ID, and add a description of the appointments. The testing approach was 100% aligned to the software requirements with 100% Coverage, 0 missed instructions. To defend the overall quality of my JUnit tests for the contact service, appointment Service, and task service I ran the tests with Coverage As to see the test coverage percentage which I had 100% coverage for both contact service and task service. This means that my tests have sufficiently tested the objects.

To ensure that my code was technically sound I wrote the code based on the requirements and performed a unit testing on the code to ensure there are error free and the appropriate results are given by the program. For example, in appointment service one of the requirements was to delete appointments per appointment ID.

void testdeleteAppointmentService() {

Appointment appointment = new Appointment("145", date, "date");

assertEquals(true, service.addAppointment(appointment));

assertEquals(true, service.deleteAppointment("145"));

The number 145 is the appointment ID. In the fourth line it is instruction to delete the appointment per ID in JUnit testing and so it did error free.

To ensure that my code was efficient I evaluated the code to see if would successfully pass. For example, in contact service one of the requirements was to update contact fields per contact ID but the only fields that are updatable are first name, last name, phone number, and address.

void testupdateContactService() {

ContactService service = new ContactService();

Contact contact1 = new Contact("123", "sameeh", "alqatabi", "8007975566",

"254 magnolia st");

Contact contact2 = new Contact("12344", "sameehgf", "alqjbn", "8004551213",

"266 magnoli444 stt");

assertEquals(true, service.addContact(contact));

assertEquals(true, service.addContact(contact2));

assertEquals(true, service.updateContactFirstName("12344", "alex"));

assertEquals(true, service.updateContactLastName("12344", "rodriguez"));

assertEquals(true, service.updateContactNumber("123", "4642348880"));

assertEquals(true, service.updateContactAddress("12344", "100 mead st"));

}

As you can see in for contact1 the ID is 123 and were trying to update the phone number in JUnit testing. In line 11 you can see we used contact ID to change the number from 8007975566 to 4642348880 and the test passed.

# Reflection

Dynamic Testing Techniques were the software testing techniques that I employed for the project. Structure-based testing techniques are focused on how the code structure works and test accordingly. Static Testing Techniques is the other software testing techniques that I did not use for the project. Static Testing Techniques or static testing is a human testing technique that does not involve executing or running the program or software product meanwhile Dynamic Testing Techniques of unit testing is to examine the methods/classes to validate functionality and make sure the behavior is as expected.

In acting as a software tester, to a high extent I employed caution because I do not believe that there are builds given or being build by me are bug-free, I questioned everything, and I only accept the build to be bug-free if I test it and found it to be defect free. It is important to appreciate the complexity and interrelationships of the code I was testing because testing helps ensure that your code works as you intend it to.

Bias would be a concern if you were responsible for testing your own code, but the goal is to focus on the quality of my tests. If you think something is right, you will try to prove why it is not wrong and you also must have valid evidence otherwise you will just have to accept that you are wrong and that is the best way to limit being bias in your review of the code.

It is important not to cut corners when it comes to writing or testing code because cutting corners at any level leads to a flawed product. There is no limit for testing until you produced a quality product. I plan to avoid technical debt as a practitioner in the field with automated testing and it is one of the most efficient ways to get rid of technical debt because automated testing makes sure that there are no bugs in the code by running multiple debugging cycles.

References

Hamilton, D. (2022). JUnit Tutorial With Examples: Setting Up, Writing, and Running Java Unit Tests.