Module 7: Project Two

Scot C. Batton

CS-320 Software Test Automation & QA

Robert Tuft

December 10, 2023

In Project One, I developed a mobile application that delivered contact, task, and appointment services to a customer. During the development of these three services, I employed different unit tests which allowed me to test customer requirements, code integrity, and total coverage of the code prior to delivery and implementation of the mobile application.

The approach to my unit testing for the three features was aligned to the software requirements of the customer. For the contact services within the mobile application, I created JUnit tests to create, delete, and update the contacts for the customer. There were also tests that check for the validity of the data that was being input and would not allow for the improper information to be input. Within the task service of the mobile application, the JUnit tests were very similar to the contact services. The tests were written to create, delete, and update tasks just like for the contacts. The task unit tests also checked for empty task names and for any invalid due dates. The approach for the appointment services were also checked using JUnit tests that met the customer requirements. The requirements were to create, modify, and delete appointments within the application. The tests also checked for any appointments that overlapped and if the times were incorrect.

After the completion of my code base and tests, I was able to complete JUnit testing and testing of coverage percentage of the code requirements. I played around with the code and tests to try and get the highest percentage of coverage for each service in the application. The coverage for the three services went from 62% to 79%. This coverage was lower than anticipated or required for Project One but showed that the coverage was greater than half and that the tests covered most areas of interest to the customer.

While writing the JUnit tests for Project One, there were steps that were taken to ensure that the code was technically sound and efficient. Within the contact service there were a few things that I did to ensure technically sound code. I initialized a new instance of ContactService before each of the test methods.

@Test

public void testAddContact() {

ContactService contactService = new ContactService();

Contact contact = new Contact("12345", "John", "Doe", "1234567890", "123 Main St");

I used this to be able to keep the environment clean and allow for better testing. I was also able to isolate each of the tests from the other so that the tests and test results are independent from one another which makes the tests very reliable.

With the AppointmentServiceTest code I was able to use the @BeforeEach annotation to ensure that the environment I was testing in was clean prior to testing each test method.

@BeforeEach

void setUp() {

appointmentService = new AppointmentService();

validDate = new Date(System.currentTimeMillis() + 3600000); // Set to valid date (1 hour from now)

validAppointment = new Appointment("1234567890", validDate, "Valid description");

}

I also used assertions from Jupiter to validate my results against the actual results of the test which can help to ensure that the test is true, and the code base is correct based on the customer’s requirements.

For the TaskServiceTest, I used the same assertions from JUnit that I did with the AppointmentServiceTest to ensure the outcomes of the tests are the same and the actual results. I also used the try-catch blocks to be able to handle the expectations that were expected from the tests.

try {

taskService.updateTaskField("456", "name", "NewName");

fail("Expected IllegalArgumentException for non-existent task ID");

} catch (IllegalArgumentException e) {

}

I used this exception handling when I added duplicate ID’s or non-existent ID’s.

The code that I wrote also had to be efficient along with being technically sound. I used HashMap to store my contacts. Using HashMap to store contacts allows for easy and efficient lookup and retrieval of the information.

import java.util.HashMap;

import java.util.Map;

public class ContactService {

private final Map<String, Contact> contacts;

public ContactService() {

this.contacts = new HashMap<>();

}

This implementation allows for efficient code and also is effective when it comes to extracting and implementing the functions that the code asks for.

The software testing techniques that were used in this project allowed me to ensure quality and reusable code for the customer. I used JUnit testing to make sure that I was able to test for the customers requirements such as adding, updating, deleting, and modifying the contacts, tasks, and appointments within the application. I also was able to use error handling testing to ensure that the code would function if an incorrect input was added. This test was able to make sure that the system was able to handle these changes and present the proper answer.

While I used many forms of testing within the project, I did not use other forms because the scope of the task did not require it. I did not perform any type of performance or integration testing on the codebase. The performance of the system was not a requirement from the customer and was merely focused on the functionality of the system. I also did not have a requirement of integrating the different modules of the code within each other, so I did not use integration testing.

The mindset that I adopted for this project was one of optimistic caution. I was optimistic for the opportunity to complete the requirements that the customer had asked for and to be able to make quality tests and code that can be implemented, reusable, and reliable. I was also aware to act with caution to provide customers with their requirements but also provide code that covered all things in a safe and testable manner. When I updated the contact field, I had to make sure that the code could handle it and that it did not affect any other parts of the code.

I was able to limit by bias when reviewing my code by relying only on the information provided by the customer. My bias is able to be removed when I realize that the customer is the only opinion that matters and that as long as I reach the objective that they set than I am doing my job. This will help remove the bias of testing my own code and seeing the larger picture of what the actual purpose of my code is.

Discipline within the code and commitment to quality as a software engineering professional is as important as the code itself. The ability to not cut corners is a skill that must be adhered to and practiced. To be proficient and in demand, the discipline to stay within the requirements of the customer, allow ego to remain at bay, and to avoid being too technical when it calls for a small, sound, precise code will allow for greater efficacy of code and a better job as a practitioner.