Abby Ramatowski

Software Test Automation & QA

10/13/2021

Project Two

**Summary: Testing Approach**

The submitted application included Contact, Task, and Appointment features, each of which were tested individually using JUnit tests. The purpose of the JUnit tests were to ensure that each class would work the way it was intended against the given requirements. For example, within the Contact class, the requirements were that Contacts should be added that have a Unique ID, first name, last name, phone number, and address. Each field should be less than or equal to 10 characters long and should not be null. JUnit tests were used in the ContactTest class to test when all fields are entered correctly and a contact is successfully created. Tests were also created when creating a contact fails due to a Contact ID, or another entered field, being too long. Similar tests were created when testing the success or fail of a contact being updated or removed. These types of tests were carried over to each test class throughout the application.

**Summary: Quality**

I followed best practices to ensure that my JUnit tests were effective. I tested only one unit at a time and made sure that I tested each method in the classes. Each test was independent from the previous one and test names were clear and consistent throughout each test class. I also made sure to test the exceptions so that everything was thoroughly tested.

**Summary: Technically Sound Code**

I tested often and thoroughly to ensure that my code was technically sound. For example, lines 26-33 in the Task class include a mutator that allows the Task Name to be set. In order for the name to be set, it must be less than or equal to 20 characters long, and must not be null. If these requirements are not met, than an IllegalArgumentException is thrown and the task cannot be set or updated with the entered name. In the TaskTest class, both the success and failure to meet the requirements were tested when creating a task. In lines 12-20 in the TaskTest class, the length requirements for the task name and all other fields were met and the task was successfully created. Lines 29-34, the task create fails because the name was too long.

**Summary: Efficient Code**

In order to ensure that my code was efficient, I made sure to meet all of the functional requirements without using unnecessary operations or variables. Maps were used within the ContactService class, TaskService class, and AppointmentService class to minimize operations while ensuring that contacts, tasks, and appointments can be added, updated, and removed while meeting the functional requirements.

**Reflection: Testing Techniques**

JUnit testing was implemented within test classes for the Contact, Task, and Appointment model and service classes. JUnit tests were used to test the individual components of the application and to ensure that each component performs the way that it is intended to before the units are integrated together for further testing. For each test class that I created, I tested the mutators established in the model classes and made sure that contacts, tasks, and appointments are either successfully created when all requirements are met, or that the create fails when one, or more, of the requirements are not met.

**Reflection: Other Testing Techniques**

Integration, system, and acceptance testing were not implemented within the application. Integration testing is the next level of testing after unit testing is completed. This is when the individual units are integrated and the purpose of testing is to ensure that the data flows together. System testing involves testing the complete system with all areas fully integrated. Its purpose is to see if the system meets the requirements when fully integrated. Acceptance testing allows the customer to test the system and provides confidence in the system.

**Reflection: Practical Uses and Implications**

Each of the testing techniques should be completed in order to ensure that the system works, is secure, and meets requirements. Unit testing is the first testing to be completed in the testing process to ensure that each individual unit works before being integrated together. The main practical use for unit testing is to detect errors quickly and before units are integrated together. The practical use for integration testing is to expose errors in the interaction between integrated units. The practical use for system testing is to verify that the application performs tasks as they are designed to. Acceptance testing is to ensure that the client or customer approves the application.

**Reflection: Mindset**

While working on this project, I adopted the mindset of a program tester. Rather than focusing solely on the development of the code, I was focused on creating code that could be tested, and thoroughly and efficiently testing that code. I employed caution when creating my JUnit tests. While creating the tests, it seems that JUnit tests can do a lot in terms of testing, but should only test each given requirement for each feature within the program. It is important to limit bias when reviewing code. When developing code, I tend to assume that it is correct and easy to understand, however, this is not necessarily always the case when another person is reading my code. When testing my own code, it is important to try to look at it as someone who did not create the code so that I can eliminate bias.