Truman Wilson

7-2 Project Two

The unit testing approach I applied to all three features was an example of white-box testing. Since I had to rewrite most of my assignments to ensure they worked correctly, I conducted some research for additional help in writing the service classes and tests. I also made necessary changes to the three regular classes. Referring to the software requirements, I ensured I replicated the requested features. For instance, when writing the Appointment class, I ensured that the appointment ID was no longer than 10 characters, not null, and updatable. I created private variables, setters and getters, and a constructor to tie everything together. In the setters, I included an if statement that would display an error message if the input was null or exceeded the character limit. If the input was valid, it would meet the requirement. This approach was applied to all the other classes I wrote as well.

The overall quality of my JUnit tests would likely be around 50%. I didn't initially realize that I was supposed to test for failures, so I primarily focused on testing the project when it worked as expected. This round of writing JUnit tests was more technically sound, as I learned how to write better tests that aligned with the requirements. The main area for improvement is testing for failure scenarios. I ensured the code was efficient by rewriting the services in a different way. Initially, I had implemented the service using a hashmap, which I believed could work, but I struggled to get it to function properly. I then reviewed examples of how others wrote their code, and once I developed my own version, I found it easier to write JUnit tests. For the tests, I wrote:

void testDeleteContact()

{

ContactService service = new ContactService();

service.newContact("5678", "John", "Doe", "987-654-3210", "4567 Oakwood Dr");

service.newContact("6789", "Jane", "Smith", "321-654-9870", "1234 Pine St");

service.newContact("7890", "Mike", "Johnson", "111-222-3333", "6789 Maple Ave");

assertEquals(3, service.getContactList().size());

service.deleteContact("5678");

assertEquals(2, service.getContactList().size());

service.deleteContact("6789");

assertEquals(1, service.getContactList().size());

service.deleteContact("7890");

assertEquals(0, service.getContactList().size());

}

This test to makes sure that the contact is deleted; which works in this scenario.

void testCreateNewContact() {

ContactService service = new ContactService();

service.newContact("5678", "John", "Doe", "987-654-3210", "4567 Oakwood Dr");

Contact contact = service.getContactList().get(0);

assertEquals("5678", contact.getContactID());

assertEquals("John", contact.getFirstname());

assertEquals("Doe", contact.getLastName());

assertEquals("987-654-3210", contact.getPhoneNum());

assertEquals("4567 Oakwood Dr", contact.getAddress());

}

This tests to make sure that the contact is created; which works in this scenario.

As previously mentioned, I approached writing the tests with the intention of implementing white-box testing for the service. I was aware of the project requirements and kept them in mind while developing the software. Although I considered potential scenarios during testing, I recognize that this is an area where I could have planned more thoroughly.

While working on this project, I had to adopt a new mindset to run tests and apply them to the code I wrote. This approach was initially unfamiliar, but over time, I became more comfortable with it. I also realized that I lacked caution during the process. For example, I considered making global variables private instead of public but didn’t fully address that. Additionally, I didn’t account for all possible values when testing. While I ensured the objects I created passed the tests, I didn’t test for failure scenarios. I also recognized that I created unnecessary objects that weren’t needed in some of the services I wrote.

I believe I need to focus on whether I’ve written the code clearly, without ambiguity, and avoided including unnecessary elements. As I’ve advanced in programming, I’ve noticed that this has been a challenge for me. At times, I’ve written code in a longer, more complex way when a simpler, more efficient approach was available.

It's crucial not to cut corners when writing or testing code, as you are learning valuable skills for future employers. A lack of attention to detail in programming reflects the developer’s work ethic and character, which can negatively impact their reputation, whether intentional or not. I believe the best way to improve as a practitioner in the field is through consistent practice.