Jacob Virgilio

10/19/2024

CS-320

Project Two

**Describe your unit testing approach for each of the three features.**

**To what extent was your approach aligned to the software requirements? Support your claims with specific evidence.**

**Defend the overall quality of your JUnit tests. In other words, how do you know your JUnit tests were effective based on the coverage percentage?**

I used these tests to keep as close to the requirements as possible. An example of these requirements is the naming parameters, which state that the full name must be less than 20 characters, required, and not null. In my code, I tested to make sure to test a name that is valid, a name with a null string, a string longer than 20 characters, and an empty string. Alongside this example, other areas were coded in this same manner depending on the requirements. Testing was balanced for readability and overall coverage. 100% coverage was not planned for in these tests, since most constructors had more than 3 parameters set. Setters and getters were targeted, mainly testing for overall valid or invalid strings or constructors.

**Describe your experience writing the JUnit tests.**

**How did you ensure that your code was technically sound? Cite specific lines of code from your tests to illustrate.**

I tried to make sure the code was overall readable and concise. A good example of this is ContactTest’s valid constructor test:

@DisplayName("Test valid constructor")

@Test

public void testGoodConstructor() {

String contactId = "1";

String firstName = "George";

String lastName = "Foreman";

String phoneNumber = "1234567890";

String address = "919 Grill Street";

Contact testContact = new Contact(contactId, firstName, lastName, phoneNumber, address);

assertEquals(1, testContact.getContactID());

assertEquals(firstName, testContact.getFirstName());

assertEquals(lastName, testContact.getLastName());

assertEquals(phoneNumber, testContact.getPhoneNumber());

assertEquals(address, testContact.getAddress());

}

I felt like this was one of the safest and easiest ways to complete this test. Later in the program, other tests are run to find specific errors not covered by this test.

**How did you ensure that your code was efficient? Cite specific lines of code from your tests to illustrate.**

In ContactServiceTest, all related tests are done with similar sets of information, differentiating based upon errors that the specific test is trying to check for an example of this is testing addContact:

void testAddContact() {

String firstName = "George";

String lastName = "Foreman";

String phoneNumber = "1234567890";

String address = "919 Grill Street";

ContactService test = new ContactService();

assertTrue(ContactService.contactList.isEmpty());

test.addContact(firstName, lastName, phoneNumber, address);

assertFalse(ContactService.contactList.isEmpty());

assertEquals(0, ContactService.contactList.get(0).getContactID());

assertEquals(firstName, ContactService.contactList.get(0).getFirstName());

assertEquals(lastName, ContactService.contactList.get(0).getLastName());

assertEquals(phoneNumber, ContactService.contactList.get(0).getPhoneNumber());

assertEquals(address, ContactService.contactList.get(0).getAddress());

}

**What were the software testing techniques that you employed in this project? Describe their characteristics using specific details.**

The types of testing techniques used were mainly static testing, which involves interpreting code and writing specific error handling conditions for multiple different outcomes. I was attempting to follow the specific design specifications in the attached documentation as I built the test around the programs. I also made sure to have the code not only throw an exception or an error, but to correct the linked problem. In this most recent assignment, my AppointmentServiceTest testing program confirms the exception type the area was testing before throwing the relevant error.

**What are the other software testing techniques that you did not use for this project? Describe their characteristics using specific details.**

Integration testing was never used in these programs. Since all of these applications are treated as standalone segments, total integration testing was not included. Automated testing was also not used, as all errors are manually input.

**For each of the techniques you discussed, explain the practical uses and implications for different software development projects and situations.**

Integration testing is important in large programs that have multiple different sections created on an individual basis. These different sections must merge and cooperate well, or the entire program will not work as specified. Automated testing comes into play with a more complex array of input. The automation of these tests will save on time and manpower, sometimes reducing possible human error in testing sections. Static testing is a common practice, and is used in almost all projects, since it is a simple sort of “check your work” type of test. All of these test have a reliable use in most different projects and situations, with some of them being more effective in larger projects than others.

**Assess the mindset that you adopted working on this project. In acting as a software tester, to what extent did you employ caution? Why was it important to appreciate the complexity and interrelationships of the code you were testing? Provide specific examples to illustrate your claims.**

Caution and time are extremely important aspects of this project and as an active software tester. I tried to make sure that the connections between the testers and base code had a relatively broad range of coverage to help prevent any unwanted code execution. Time is extremely important, because the more time you spend trying to predict the actions of the user base, the higher chance you have of completely mitigating possible unwanted inputs. Some specific examples of this are not only testing for an empty description, but an incorrect description as well, including too long or too short.

**Assess the ways you tried to limit bias in your review of the code. On the software developer side, can you imagine that bias would be a concern if you were responsible for testing your own code? Provide specific examples to illustrate your claims.**

I tried to limit bias by taking examples of other websites. For instance, a password requirement might need a capital letter, a number, and a special symbol. This can be translated to this project by stating that a description or name needs to be not null, between 5 to 20 characters, and avoid using special characters. Bias would be a little bit of a concern if you were testing your own code in the sense that you cannot always remember everything that you did previously, this is why in line comments are important.

**Finally, evaluate the importance of being disciplined in your commitment to quality as a software engineering professional. Why is it important not to cut corners when it comes to writing or testing code? How do you plan to avoid technical debt as a practitioner in the field? Provide specific examples to illustrate your claims.**

Being disciplined in your commitment to quality is of utmost importance as a software engineering professional. This is not only important to you, in the fact that updates to almost any software will be required. Updating a poorly written piece of software is indefinitely harder then updating a well made piece. This also goes for any other professional you are working with, since they will rely on your code to be readable and work well. Avoiding technical debt is challenging in some regards, especially if you are using an older software or engine, but preventive maintenance is required regardless of software. Using the correct format and most recently updated software is a good way to avoid technical debt. An example of this is updating to the correct version of Java to help prevent errors.