Jacob Casas

CS-320

12/11/22

Project Two

While unit testing the three features, I tried to keep the software requirements in mind. For example, the contact class required a unique ID, a first name, last name, phone number, and address. The ID, first name, last name, and phone number were all required to be ten or less characters and could not be null. The address field was required to be thirty or less characters and could not be null. While conducting my J-Unit testing, I created tests for all these conditions. For instance, to test if the ID is null, I used the following:

// Testing if ID is null

@Test

        void testIdNull() {

            Assertions.assertThrows(IllegalArgumentException.class,() -> {

                new Contact(null, "Peter", "Parker", "9497246281", "11 Candy Lane");

        });

        }

I tried to test every requirement made by the customer. Overall, my J-Unit testing was effective as we can see while conducting the tests. While running the tests, all are completed apart from a few in ContactServiceTest.java. We also know that the testing is effective by looking at the coverage which was 68.8%. While this number could improve, it still shows us the effectiveness of the J-Unit testing.

By using J-Unit testing, I was able to ensure the code was “technically sound.” J-Unit testing allows you to break down the code into small fragments and test each piece. In return, this allows you to make sure that all your code is working properly. For example, I showed you how I tested if the ID was null above. However, if you remember, the ID was also required to be ten or less characters. This means that this is another fragment of my code for me to test. I did so by writing the following code:

// Testing if ID is too long

        @Test

        void testIdTooLong() {

            Assertions.assertThrows(IllegalArgumentException.class,() -> {

                new Contact("12345678901", "Peter", "Parker", "9497246281", "11 Candy Lane");

            });

        }

Another thing to keep in mind while writing code is efficiency. I was able to ensure this by using steps to produce quality code. One step to ensuring quality code is to make use of comments and trying to create readable code. I did my best to make sure everything that needed to be commented was as well as writing the code in a simple readable way. For instance, here you can see my use of comments as well as well formatted readable code:

@Test

    void testAddTask() {

        TaskService taskService = new TaskService();

        // Testing if Name Null

        Assertions.assertThrows(IllegalArgumentException.class,()->{

            taskService.AddTask(null, "Example Description");

        });

        // Testing if Description null

        Assertions.assertThrows(IllegalArgumentException.class,()->{

            taskService.AddTask("Peter Parker", null);

        });

        // Testing if Name too long

        Assertions.assertThrows(IllegalArgumentException.class,()->{

            taskService.AddTask("Peter Parker Peter", "Example Description");

        });

There are many different software testing techniques that can be utilized however, for this project I used structural testing, otherwise known as white box testing. White box testing helps ensure that your application is functioning properly. GeeksforGeeks.org states, “White box testing techniques analyze the internal structures the used data structures, internal design, code structure, and the working of the software rather than just the functionality as in black box testing.” While using the structural testing method, I used statement coverage to test all the statements of the code.

On the contrary, a couple of testing techniques that I did not utilize are exploratory testing and error guessing. The reason exploratory testing was not utilized is because it relies on little to no planning with the best testing results possible. For this project, planning was more crucial than speed. Error guessing was not utilized either because it relies on the experience of the developer. This method is where an experienced developer who is familiar with the application might know off the top of their head where the issue might lie in the program due to their familiarity with the application.

Every testing method that has been mentioned has applications and may be applied in a variety of software projects and circumstances. White box testing, for instance, may be utilized by any company trying to boost test coverage. The number of undiscovered problems in the code will decline because of this testing. For an application where the developer is familiar with the code, error guessing can also be employed. This could turn out to be a quicker approach than others. If necessary, exploratory testing can be utilized to quickly understand the product.

While working on this project I took on the mindset of using caution. The reason for this is because while working with so many connected files, I did not want to overcomplicate things for myself by including non-required features. It is always important to be able to appreciate the complexity of the code being tested because you do not want to make mistakes that could jeopardize the program. For instance, if you had a volleyball game, you would not want to underestimate the other team as this often leads to a loss. Just like the example, you would not want to underestimate the complexity of the code as it can lead to mistakes.

As I was the sole developer for this project, trying to limit any bias in my review is crucial. I did this by using J-Unit testing instead of relying on things such as error guessing. I do see bias as being a concern while testing your own code and a fresh set of eyes is almost always beneficial. Staying committed to creating quality code is vital as a developer. Quality code makes it easier for other developers to read and edit your code as well as ensuring functionality and execution.

Resources

*Software engineering: White box testing*. GeeksforGeeks. (2022, March 9). Retrieved December 11, 2022, from <https://www.geeksforgeeks.org/software-engineering-white-box-testing/>

*Software testing techniques*. GeeksforGeeks. (2021, March 1). Retrieved December 11, 2022, from https://www.geeksforgeeks.org/software-testing-techniques/