Jasper Conneway

CS 320

Professor Gaston

August 13, 2024

Summary and Reflections Report

I believe my testing approach aligned with the software requirements for all three features. I started my testing by just running the tests under a regular configuration. When I knew I was near the end of my testing I ran coverage to see what percentage of my class was not being tested. Here I was able to recognize if I missed any testing.

The overall quality of my JUnit tests for the Contact, Task, and Appointment features are of good quality. They are simple and easy to understand. They test to find defects – something wrong in the code. I know they were effective because the coverage tests returned 100% for all my JUnit tests.

I ensured my code was technically sound by testing for what could go wrong in my code rather than what I knew wouldn’t return an issue. I tested everything that a user should not provide as input for an item. For example, in my ContactTest class, I tested the reaction my code had to one type of invalid input:

@Test

void testContactIdTooLong() {

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

new Contact("10111095689", "Jasper", "Conneway", "1232356914", "123 Gully Rd 4801");

});

}

The code is technically sound or rational to ensure the contact ID is not too long and is handled correctly.

My code was also efficiently effective. ContactServiceTest ensures that contacts truly add contacts, update them, and delete them. One of the tests covers the situation in which a user tries to update a contact that does not yet exist. The following code is how I tested for an instance such as this:

@Test

void testAddContactByUpdate() {

ContactService contacts = ContactService.getInstance();

contacts.updateContact("103423", "LBDOG", "CBHOUSE", "1800OURDGS", "LIVnAT Our House");

assertTrue(contacts.findContact("103423").getFirstName().equals("LBDOG"));

}

During the project, my approach aligned with the software requirements by using different types of testing techniques. I employed white-box testing and black-box testing. Unit testing allowed me to verify the code’s quality. It helped to determine if the code I wrote functioned properly without causing errors and informed me if the code handled invalid input properly. I used black box testing for the Appointment feature because the date class was not familiar to me. I looked at Oracle to determine what I needed to make the class run properly. I educated myself on how the different methods behaved. Since the initialization was deprecated, I did use the suppress warnings.

A type of testing I did not use was non-functional testing. Non-functional testing tests the performance, usability, security, and reliability typically with load testing, stress testing, and usability testing. I think the testing I utilized for the project was the best for this situation.

The mindset I adopted when working on this project was one of excitement. This was the first class I’ve had the opportunity to create my own JUnit tests for the classes I create. I did employ caution because we want to ensure everything, or as much as possible, is tested before promoting the product we create. It is important to appreciate the complexity and interrelationships of the code you are testing. This is because when we do not prepare for things we often miss them. If we only test what code we expect our users to input, we may not have prepared for the invalid input they may give us. Another way I handled the complexity is by creating a find function for each feature rather than having repetitive code in every function to find an object throughout the features. I tried to limit bias by thinking outside of what I would want out of a feature like this. This is why I included the add contact when trying to update one that does not exist. I also limited bias by testing my code extensively. We cannot expect ourselves to be incapable of making mistakes. Unlike computers, it happens – we just need to prepare for it. I noticed my delete object functions were only deleting the local object and accounted for this. I found the error and fixed it.

Ultimately, it's important to have quality code and to test for quality. Cutting corners only hurts ourselves and others and we end up ruining reputations. Our customers and clients expect the best from us and deserve the highest quality products. I plan to avoid technical debt by following these values and asking questions or for help when I need it.