**SNHU**

**CS – 320**

**7 – 2 Project Two**

**Jorge Sierra Alzaga**

1. ***Summary***
2. *Describe your unit testing approach for each of the three features.*
3. *To what extent was your approach* ***aligned to the software requirements****? Support your claims with specific evidence.*

My testing approach was directly aligned with the software requirements as it tested the functionalities and constraints required. For example, there were constraints on certain variables, were there was a character limit, or it could not be null, which was tested, checking that an illegal argument error was thrown in each case.

1. *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?*

After submitting my project, I detected some errors, which I have now corrected, greatly improving the quality of my Junit tests. Each of the test classes achieved a 100% coverage percentage in their respective class, thus the Junit tests were very effective.

1. *Describe your experience writing the JUnit tests.*
2. *How did you ensure that your code was* ***technically sound****? Cite specific lines of code from your tests to illustrate.*

To ensure my code was technically sound, I took several aspects into consideration, such as legibility, processing times, and scalability. The last two reasons were why I chose to use a HashMap as my main data structure. I used basic functions to ensure the main conditions were met, such as “assertThrows” and “assertTrue”:

@Test

**void** testCreateContactSuccess() {

Contact contact = **new** Contact("123456", "Bob", "Mike", "1111 E Road Street", "4802929112");

*assertTrue*(contact != **null**);

*assertTrue*(contact.getContactId().equals("123456"));

*assertTrue*(contact.getFirstName().equals("Bob"));

*assertTrue*(contact.getLastName().equals("Mike"));

*assertTrue*(contact.getAddress().equals("1111 E Road Street"));

*assertTrue*(contact.getPhoneNumber().equals("4802929112"));

}

@Test

**void** testCreateContactContactIdFails() {

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

**new** Contact("12345678901", "Bob", "Mike", "1111 E Road Street", "4802929112");

});

}

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

To ensure my code was efficient, I used functions whenever there were repeating lines of code, for example, I created functions to check the validity of an input, based on the number of characters that was allowed in each instance:

**private** **boolean** validateInput(String item, **int** length) {

**return** (item != **null** && item.length() <= length);

}

I also used if/else statements to check conditionals and throw any needed errors. I checked my code against different scenarios to ensure all the requirements were met, which can be seen when checking if the provided appointment date is in the past, thus being invalid:

**private** **boolean** validateDate(Date date) {

**return** (!date.before(**new** Date()));

}

1. ***Reflection***
2. *Testing Techniques*
3. *What were the software testing techniques that you employed in this project? Describe their characteristics using specific details.*

The testing techniques used on each of the milestones were similar, mainly falling into the category of white-box and black-box testing. White-box testing tests the internal structures of the application, while black-box testing tests the functionality. An example of the white-box testing technique is providing an invalid input, which is expected to follow the validation path in the code and return an error. On the other hand, with the black box testing technique, an object would be added to the data structure and then removed it, testing that the object was effectively deleted.

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

Some of the testing techniques that were note used include performance testing, stress testing and experience-based testing. Performance testing determines how the application performs under a particular workload, while stress testing overloads the application until it reaches a breaking point, which is used to determine usage limits. Experience-based testing are test cases made by the tester, in which the application is analyzed against different scenarios of usability. None of these techniques were applied to the application as it was only needed to test against the requirements.

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

The previously mentioned techniques are to be used depending on the needs of the system, with white-box testing being used when the tester knows the internal functionality of the application, therefore implementing test cases for each expected outcome. Black-box testing can be used when the tester knows what the outputs would be, without necessarily knowing the internal functionality of the application. Performance testing and stress testing can be used when there are performance and usage requirements. Experience-based techniques can be used to determine the behavior of the application under circumstance that fall within and outside the usage scope of the system.

1. Mindset
2. *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.*

When working on this project I adopted a mindset that relied in organization, as everything was laid out in the requirements, I tried to check my code as I wrote it and doing an overall test whenever I finished a class. I was as cautious as I could, increasing it overtime, when I notice I was letting errors slip through, for example, at first, I was not checking for letters in a phone number input. In the real world, each of these errors can cost us our job as programmers, or the way our ethics are perceived.

1. *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.*

As a software developer, when writing and testing your own code, there is always going to be bias, as it is something intrinsic to us as human beings. I think it is important to recognize that bias will always be there to an extent and act to minimize it. In my case in this project, I always tried to find the most effective way to perform the testing and cover all the requirements, whether it was looking at documentation or searching for similar cases in Stack Overflow, I tried to have different perspectives.

1. *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 our commitment to quality as a software engineering professional is paramount, as the quality of our work will be what defines us as professionals. The way we project ourselves, and what our work says about us is what will keep us with a job, but outside of the economical reasons, it is ethically right. In order to avoid technical debt as a practitioner in the field, I think it is important to recognize that we don’t know how to do everything, being open to feedback and always learning, as this will allow us to keep improving.