Jordan Wentworth

04/17/2022

Project 2

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

To make sure that my testing approach was aligned with the software requirements I made sure to test for each specific requirement they had. For each object, the customer wanted a character cap. For example, in the contact class the customer wanted the first name to be capped at 10 characters. In my unit testing, I created a test to make sure that the first name would not be more than 10 characters. If it was, it would throw me an error message. I did this for each object in each class.

* + 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?**

As I created each class and each object, I created a test class for each aspect of the requirements. J unit runs the code and throws me three different colors. Green means that the code was executed and are covered. Yellow means parts of the code were covered but some aspects could not be covered properly. Lastly, red means the code was not covered at all. My coverage at the end was over 60% covered. With some more time, I would make changes to receive a coverage as close to 100% as possible. However, the codes that were green were important parts of the tests, which means those requirements were tested and executed properly.

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

To make sure that my code was technically sound, I used arrays in my code. For example,

ArrayList<contact> contactList = new ArrayList<contact>();.

Using functions like contactList.add(contact);, and contactList.remove(counter);. Helped me make sure my code was sound.

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

To make sure that my code was efficient, each object was clearly labeled with notation in the code for organizational purpose. In addition, I cut out any extra could that was not necessary. I also labeled the names of the test to easily show what they were doing. To make sure that the test to make sure the phone number was only 10 characters, I labeled the test as @DisplayName(“Contact phone number is exactly 10 characters”).

The use of if else statements helped me use minimal code.

if (firstName == null) {

this.firstName = "NULL";

}

else if(firstName.length() > 10) {

this.firstName = firstName.substring(0, 10);

}

else {

this.firstName = firstName;

}.

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

A software testing technique I employed was equivalence partitioning. Testing for valid and invalid inputs into our system made sure the code was taking in and spitting out the correct data is essential. If we got an invalid input, our system would come back and say phone number was null, or last name was more than ten characters long. Another technique I use was statement testing. There were times I only focused on the executable statements.

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

The other software testing techniques I did not use were exploratory guessing and error guessing. This is my first time using jUnit testing or anything relevant to this project. With this in mind, I had no reason to use this technique with no prior knowledge.

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

Equivalence partitioning is used to test for types of inputs that will cause an error or a wrong output. Executable statements testing is practical because executables are easier to see and focusing on only those statements can create a good base for our code. Exploratory testing and error guessing tests are simple yet practical. If you have a lot of experience with specific types of systems and code that you are working on, then these types of testing is good for you. Errors in the code should be a bit easier for you than others.

* 1. **Mindset**
     1. **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.**

Since I had not experience in the project and I was solo, caution was always an issue. Bugs were going to happen, and the code was not always going to be working properly. Therefore, accurately testing the code helped. The complexity of the code was immense for each object, each method in each class. However, the interrelationships of the code were even greater. Each code depended on the last.

* + 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.**

Obviously, there will be some bias on a solo project like this. I created the code and then had to test the code. In addition, as a student who has know prior knowledge there is going to be some issues that I created. With both in mind, I am going to have times where I think I am correct and there will be sometimes where I know I am not correct because I do not have the experience. For example, I may have one way about creating a code and creating a test for it; however, a team member could have a more technically sound and more efficient way about it.

* + 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.**

In this field, being disciplined is everything. Cutting corners leads to poor quality products. Requirements will not be meet and customers will no longer want to work with you. If you find a place where you can not figure it out, you can just go around it. More time must be used to finish the product. Fatal errors and bugs can lead to the whole restart of the project. This can cost time and money, aka technical debt. Setting milestones and deadlines will ensure that the code is being worked on at a good pace. Making sure you have an effective plan in placed will ensure the quality of the product is good.