Deje Perkins

Software Test Automation

Project Two

June 24th 2024

1. 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.

I think for my approach the requirements were aligned but just not carefully executed enough to produce an output for the test. I did meet the requirements of the ID string character lengths making sure that there is a maximum amount that can be input (10) along with the firstName String (10), lastName String (10), phone String (10), and the address String (30).

* 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?

For my overall quality of my JUnit tests I give it a 2/100. I know that I was applying much effort into my projects whether it's from my own confusion or outside interfaces where I was called into work or me just recently moving into a new apartment. I personally believe this project has been one of my least productive.

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

To ensure that my JUnit test code was technically sound was through variables that were created to hold the requirements. When calling a variable it is easier than rewriting, inserting and or deleting the variable each time one by one. For instance in the ContactTest,

Contact contact = new Contact("FirstName", "LastName", "Number", "Address");

if (contact.getContactID().length() > 10) {

fail("Contact ID has more than 10 characters.");

Im setting up a way to be able to update the Contact ID’s. Depending on the length requirements if the firstName, lastName, Number, or Address is over a certain number of characters it will reject and request a shorter amount.

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

For my codes I try to be as repetitive as possible in order to make it easier to read as well as efficient. In the TaskServiceTest the public voids for invalid ID, Name, and description have a similar test.

public void invalidID() {

Task task = new Task("00000000002", "Running", "Running around the track");

addTask(task);

System.out.println("size: " + tasks.size());

}

public void invalidName() {

Task task = new Task("0000000002", "Running runnning running running", "Running around the track");

addTask(task);

System.out.println("size: " + tasks.size());

}

public void invalidDescription() {

Task task = new Task("0000000002", "Running", "Running around the track Running around the track Running around the track");

addTask(task);

System.out.println("size: " + tasks.size());

}

1. Reflection

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

In this project I incorporated unit testing. Even though my code wasn’t sufficient, unit testing ensures that each part of the code works correctly. Then by testing individual components this made sure they were working once integrated. It also showed me where the bugs were in my code early on rather than last which helped save time.

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

I failed to do both regression and system testing. When it comes to regression testing one way it helps is when fixing a bug it tests the changes and confirms that more bugs have not been created in other areas of functionality. Then for system testing it validates the entire system making sure each component works together flawlessly. Both maintain the stability and reliability of the software and ensure that it's ready to be deployed.

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

For unit testing some practical uses would include the testing of each individual component whether it includes the classes, functions or the form validations. Another way could be the testing of core functionalities that will check the new features and make sure new or more bugs have not been created. Overall unit testing gives confidence in the softwares reliability and performance. Next is regression testing and it ensures that new changes in the code don’t impact the functionality of the software. It maintains a consistent user experience while avoiding any errors or system display problems. Like unit testing, regression maintains the stability and reliability of software throughout different development projects and will catch the bugs resulting from updates. Lastly system testing includes the overall completed and integrated software in order to evaluate the systems compliance. It verifies that the intended software system functions are correct.

1. Mindset

* 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.

I was lacking a motivated drive to prioritize my codes. It showed, and if my code's structure was improved it would have reduced errors and been maintainable. Complexity is a very large factor because it can end up in over testing and having bugs appear once another feature is introduced. Once we can understand interrelationships of the areas where the code needs to be improved errors are easier to be found once refactored, leading to a reduction of complexity.

* 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.

If I were to look over my code I think I would start reading it like how I do anything at first. If I’m reading a book and find myself not having the focus or energy, I’ll start to read the lines but not actually comprehend what I just looked at. Then I will proceed to read a single line repeatedly until I understand. With that being said I would find myself overlooking mistakes thinking that my code would work because I believe in the expected intentions that are supposed to come from it. Then another thing I feel is once I get tired and I see that my codes are passing, I won't look as hard for potential issues that could have been a result of inputting features or nay adjustments. I also think I would miss alternative ways my code could be more efficient and lead to inadequate testing.

* 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.

A major reason why it's important to not cut corners when writing and testing code is reputation. If someone pays you and expects you to deliver a system software that is flawless but instead you submit bugged and low quality code no one will want to come to you or come back. The goal is to create products that users have a great experience with and want to come back for more. Also another major contributor is the cost efficiency because going back and fixing bugs is both time consuming and can become expensive. Being patient and not cutting corners will make a high quality project and save time that could be needed elsewhere. I plan to become better at writing, clean and readable code that adheres to a quality standard. Also another way would be time management which has always been hard for me but in the long run if I practice everyday and get into a positive habit I can maintain healthier and sustainable codes.