Mao Christie

August 21, 2024

CS-320

Project 2

My testing approach for task services, appointment services, and contact services involved writing tests to check that software requirements were met and constantly referring to the software requirements. Each set of code (task, appointment, and contact) had various software requirements like checking for uniqueness, length limits, and not being null. While writing code, I followed along with the requirements, adding in each one as I went through the requirements. Then I wrote tests for each class that tested these requirements to make sure they worked as expected. To check if my tests were effective, I used coverage to make sure I was covering all my code. At the end of writing my tests, I had 100% coverage for appointment and contact services and 99% to 100% coverage for task services.

To make sure my code was technically sound, I wrote tests that would create class objects with the correct parameters for each class. An example of this would be using “assertTrue” and “assertEquals” to check if an object was being created with the correct parameters. I also checked through tests to see if objects with invalid values for parameters would not be made by using code like “Assertions.assertThrows” to check if a new object with the wrong parameters would throw an error or not.

The software testing techniques I used for this project were dynamic techniques like structure-based testing and functional testing. The project had 6 different classes with Task, Task Services, Appointment, Appointment Services, Contact, and Contact Services. I employed unit testing by creating a Junit test for each of these classes and then creating tests for each function within the class. Unit testing is software testing that tests “individual components or functions of a software application” (geeksforgeeks, 2024). By testing each component or unit, I made sure each unit was functioning as expected. Functional testing is testing that tests the system “against the functional requirements and specifications” (geeksforgeeks, 2024). I used functional testing by testing to make sure my code for each class aligned with the requirements set by the client. Those requirements I checked for were creating an object with parameters that had limits on length and type, creating a related service class that could create a unique object that was then added to a list of objects, updating specific variables, and deleting an object from the list. Structure-based testing is testing that looks at the code to check for missing gaps in tests. A structure-based test I used was code coverage. While writing my tests and at the end, I checked my coverage to make sure I was not missing any code that needed to be tested and thinking of all the different scenarios that needed to be tested. Throughout testing, I built tests that covered all statements and decisions for each class.

These software testing techniques I used are important because they make sure my code works as expected, without problems. Checking code coverage is important to ensure each line of code is functional and works as expected. Without it, there could be untested code that performs unexpectedly when the client uses it. Functional testing is important because customer satisfaction is dependent on whether the project meets the requirements set by the client. A project that doesn’t meet specifications and requirements could leave a customer dissatisfied. For a successful project, each component or unit of the system should work as expected which is why unit testing is important. Without it, the system could behave unexpectedly when used by the client.

Some software testing techniques I didn’t use are security testing, performance testing, and usability testing. Security testing tests the system for any vulnerabilities and checks how the system holds up against potential threats. If there are security-related requirements set, security testing would test if the system meets those requirements. Performance testing tests aspects of the system like speed, scalability, and stability. For this project, a performance test could have been to check the speed and efficiency of searching through the list of objects in the Service classes. Usability tests test if the system is user-friendly. For my project, this could have involved the formatting of dates for the Appointment class or changing the way unique objects are created so the user doesn’t have to input a unique ID themselves.

While working on this project, I was cautious at each step of creating a class and testing. It was important to make sure the code for a class like Task met requirements and then make sure the Task Service class aligned with those same requirements since it would be creating and updating a Task object. It was important to appreciate the complexity and interrelationships of the code I tested because many of the functions that I tested were used for both classes. For example, the Task class had a function called “getName” that I also used while testing the Task Service class as well. I tried to limit bias while reviewing my code by using tests to actually check if what I had written was performing as expected. I can imagine that in a project for a client, testing my own code could be a concern because of bias. Assumptions on whether my code is clear, efficient, or functional could create biases on what tests I write and use.

It is important to be disciplined in my commitment to quality as a software engineering professional because quality coding makes it less likely for defects and escapements to be missed. Quality coding makes it less likely a client would be dissatisfied with the product given and makes the coding process more efficient. By cutting corners when writing code or tests, it can make the process of creating a system less efficient because bugs and defects may not be discovered until later in the process, which could make fixing issues more time-consuming. Also, it makes it more likely a final product leaves the client dissatisfied. To avoid this, I plan to ensure I work on code thoroughly without cutting corners.

Reference:

Geeksforgeeks. (2024, June 12). *Unit Testing – Software Testing.*

<https://www.Geeksforgeek>s.org/unit-testing-software-testing/

Geeksforgeeks. (2024, May 29). *Functional Testing – Software Testing.*

<https://www.geeksforgeeks.org/software-testing-functional-testing/>

Dev. (2024, May 28). *Structural-Based Testing: A Simple Guide.*

https://dev.to/tiuwill/structural-based-testing-a-simple-guide-5035