Jon Marvin

12/10/2024

CS 320

SNHU

**Summary**

The unit testing approach that worked the best for me was the assertion package. Assertions allow you to test your program against what the value is expected versus what the actual value is, whether the included conditions are true or false, and whether a value is null or empty. I tried to align my approach to the software requirements by testing each method against a condition. For example, in the contact and contact service code, I included tests to check if any value was null or did not adhere to the length requirements. If the length of a name, phone number, or address were too long, the assertion would be thrown, and the user would receive a message telling them to fix it.

I knew that my tests were effective when the code coverage came back over 90%. Since it is nearly impossible to get code coverage to 100%, I set a goal of 85% and followed through with the tests until that goal was reached. 90% code coverage means that almost all of my code was successfully tested with the test classes that I added. With no errors found, I was confident that the classes and all methods contained within were functioning correctly. Unfortunately, 90% code coverage does not guarantee that there are no bugs in the code, but rather that 90% of the code that was tested passed.

I know my code is technically sound because I mapped the requirements to variables and used the variables in the test code. It helps that I used appropriate naming for functions, methods, and variables being that finding and calling aptly named variables is easier. Not only does this save time when writing the tests, but it also makes the code look cleaner. Having random strings in each of the tests would make it look messy and harder to read. For example, I had variables for names, phone numbers, addresses, descriptions, etc. along with counterparts that did not meet requirements to ensure the test would catch any issues in the user input. I also used spacing to separate methods, tests, and variables so everything would be easy to find if any maintenance was needed. The following code is the variable mapping for the TaskServiceTest code.

**void** setUp() {

id = "1234567890";

name = "This is Twenty Chars";

description = "The task object shall have a required description.";

tooLongName = "This is way too long to be a task name";

tooLongDescription =

"The task object shall have a required description String field that cannot be longer than 50 characters. The description field shall not be null.";

}

To make my code efficient, I repeated tests to hold the variables and assert everything being run. This made it easy to achieve the full scope of testing while not rewriting every test. Below is an example of the assertAll function used in the same TaskServiceTest class.

@Test

**void** deleteTaskNotFoundTest() **throws** Exception {

TaskService service = **new** TaskService();

service.newTask();

*assertAll*("Only delete task if found",

()-> *assertEquals*(1, service.getTaskList().size()),

()-> *assertThrows*(Exception.**class**, () -> service.deleteTask(id)),

()-> *assertEquals*(1, service.getTaskList().size())

);

}

**Reflection**

The testing technique that I used the most was unit testing with JUnit tests. Unit testing can help you discover issues in methods that may not be running how they should. There were a few issues that popped up for me, specifically in one module, and after some feedback, I was able to correct the issues and complete all tests with no errors. Assertions were also a helpful way to test my code using Boolean expressions to consider values along with comparing test values to the requirements. The assertAll function allows you to group assertions and run through each one rather than completing one and skipping to the next.

I prefer unit testing to system testing because it is easier to find errors in the code. That is not to say that system testing is wrong, but more that newer Java developers may find unit testing to be easier to interpret than system testing. While unit testing checks just the specific methods for errors, system testing tests the entire code and lets you know if errors exist. More experienced developers may be able to read the error messages and understand what mistake was made, but I am not that advanced with Java. Regression testing is another testing method that can be helpful if your code is constantly changing. If values or functions need to be updated regularly, regression testing is the best bet to catch errors.

When working with languages that I am not proficient at, I tend to employ extreme caution. I will constantly try to find resources on functions that I am using or trying to create, and I try not to include anything that I do not understand. Though the requirements were not difficult to code, I have anxiety about Java from past experiences. Due to my inexperience with Java, I probably did include a good deal of bias in my code. It was not my intention, but there may have been a few methods that I simplified only to get them to pass tests rather than optimizing them. Experienced developers may get complacent and assume their code is good enough to the point that they choose not to test certain parts of it due to confidence or ego. The code may be fine and operating perfectly, but there is no way to know without testing.

Being disciplined is important because everyone can make mistakes. An easier way to write code may not be a better way. Being thorough in your execution could save time and money that would otherwise be used rewriting code that had not passed tests because the developer wanted to finish faster. Cutting corners is a good way to start over. When one part of the code must be updated, if corners were cut, it may not synchronize with the rest of the code. Being methodical in each block of code will ensure that everything works together how it should. I plan to avoid technical debt by following good coding and documentation practices such as clean coding and leaving comments for anyone else who may be using or working with the code. I also plan to stay up to date with packages that are used regularly. Many packages are updated on a schedule and certain aspects may be phased out. Testing is the most important part of coding. Writing a code that is supposed to do something can be a wonderful thing if it works. If code is not tested properly, there is no guarantee that it will provide the expected outcome.

# References

Codecademy. (n.d.). *Code Coverage*. Retrieved from Codecademy: https://www.codecademy.com/article/code-coverage

Dovgol, A. (2024, November 13). *Code Quality: Definition & How to Improve Code Quality*. Retrieved from DevCom: https://devcom.com/tech-blog/code-quality-definition-how-to-improve-code-quality/#:~:text=As%20coding%20tends%20to%20be,the%20quality%20of%20the%20code.

*Software Testing Techniques*. (2024, September 26). Retrieved from GeeksForGeeks: https://www.geeksforgeeks.org/software-testing-techniques/