Philip Trinh

SNHU/ CS-320

07-29-2023

5-2 Journal: Software Testing Techniques

Throughout milestones Three, Four, and Five, I was employed to assist the client in creating a mobile app that fulfills all their requirements. I utilized various coding techniques using Java applications and conducted software testing to ensure the developed app's quality and functionality meet all the client’s requirements. Below is a detailed description of the testing techniques used for all milestones, along with examples, and an explanation of why I employed these techniques:

In the mobile application development project, three modules were implemented: Contact and ContactService in Milestone One, Task and TaskService in Milestone Two, and Appointment and AppointmentService in Milestone Three.

The first technique I used for testing my code specifically is to test in small increment portion, this is known as unit testing. As mentioned by author Oliver on this topic, “A unit test is a type of software test that focuses on components of a software product. The purpose is to ensure that each unit of software code works as expected. A unit can be a function, method, module, object, or other entity in an application’s source code.” (Moradov, 2023). In summary, unit testing was employed in all the modules to test individual units of code in isolation. It focused on verifying the correctness of each class's methods and their behavior with different inputs.

Example from ContactTest class: Here the contact ID and contact first name are separated into 2 different tests and tested individually for functionality instead of combine into one test.

A screenshot of a computer code

Description automatically generatedSecond, I incorporated fail assertion techniques to test my code. This as described by author Ramesh “The fail assertion fails a test throwing an AssertionError. It can be used to verify that an actual exception is thrown or when we want to make a test failing during its development.” (Fadatare, n.d.). In the test, it is used when limited condition was already set and I only need to know if the test will pass or fail.

A screenshot of a computer code

Description automatically generatedExample from ContactServiceTest class: this technique is used to set desired fail conditions to when the appointment Id length is longer than 10 characters or the appointment description is longer than 50 characters. The test will simply fail if the conditions are not met.

Beside fail assertion test, the test I used mostly must be assertEqual. Like fail assertion test, as implied by its name, assert equal set conditions for if the value of expected value does not meet the set value. However, it is a more detailed test in that if another value does pass but does not meet the value set, it will still fail. If the test does not fail but gives another value within range, fail assertion will pass the test.

A screenshot of a computer code

Description automatically generatedExample from TaskService Class: assertEquals was used to ensure that the new Description is match indeed written as “Updated Description” exactly as input. If a letter is off, the test will fail.

Following with another Junit test is the assertNotNull which was used to verify that a tested object is not null, ensuring its existence and proper initialization. When this assertion is applied, a test passes successfully if the object is not null, indicating that it is correctly instantiated and contains a valid value. However, if the object is found to be null, the test fails, indicating a problem with object instantiation or initialization.

Examples from Appointment class: The date and description object are tested using getter to ensure it is initialized properly (not null).

A screenshot of a computer program

Description automatically generated

Next, I used boundary test to focus on testing the boundaries or edge cases of input values. The goal is to verify how the system behaves when inputs are at the extremes of their valid ranges or when they approach limits.

A computer screen shot of a program

Description automatically generatedExample from AppointmentTest class: the appointment date is being tested to see whether it is handling is correct when a date in the past is provided by compared to the current date which was set ahead of time.

Lastly, I used a manual test on the console after running a Junit test. This is the most basic testing technique by verifying the expected results showing on the console and going over my codes to fix any unexpected results or in lines error that is warning by the system.

A screenshot of a computer

Description automatically generatedExample from appointment Service: It shows the result which was set to print out on a console for manual verification.

I believe that the various techniques that I used are best to test my code and ensure quality as well as functionality of the client’s requirements. However, there are other testing techniques I could have incorporate but was not used mentioned, those techniques are listed below:

Assertions.assertThrows: This technique is used to verify that a specific piece of code throws an exception under certain conditions. It can be useful when testing methods that are expected to raise exceptions in response to specific inputs. Practical use includes verifying the code can handle exception validate error and handle it properly. This helps to ensure the project is functioning as expected and programmed to handle problem certain lines of expected problems when arise.

Assertions.assertTrue/False:While the assertTrue/False method is used for simple true/false assertions, it was not explicitly used in the provided examples. Instead, assertions like assertEquals were employed for more specific verifications. Practical use to validate condition and ensure certain state of the application is set correctly. This help to enhance logic of the code and give straight forward check when problem occur.

AssertSame/NotSame: “Performing Selenium testing often requires you to compare two different objects passed as parameters in a method. This is to determine whether they refer to the same object. In this case, you can use JUnit assertSame(). An assertion error is displayed if the two objects do not refer to the same object.” (lambdatest.com, n.d.). AssertNotSame is the opposite. Mostly in the test I used to test one condition as a time and haven’t refer to other object in comparison for same or not same. In practical, this testing technique is more commonly used when two objects being compared are the same instance or different instances, respectively. This test ensures the object is unique and detects any duplication.

Integration Testing: Integration testing validates the interaction of multiple components, ensuring seamless functioning of the entire application. Practical use to ensure multiple classes can work together smoothly. This ensures cohesive in the code and each class can get information from one another if program to work together. Since I didn’t have to run an entire application yet and just utilized Junit test to test individual units, integration testing wasn’t used.

Security Testing: assesses the system for potential vulnerabilities and ensures that it meets security requirements. It is used to make sure the system is free of potential bugs or flaws that can be potentially exploited. The test ensures the code is clear and protects application data from attacks. the security testing wasn’t required yet, so this has not been tested.

Performance Testing: evaluates how the system performs under specific loads or stress conditions. This helps identify weakness, data load, stability, and scalability. Normally would be run when the project is complete or reaches a certain point. The system has not yet been incorporated to work together so this test has not been run.

In conclusion, the combination of these testing techniques will lead to more reliable and high-quality software. I believe that each testing technique is important and serves a specific goal to contribute to the success of the mobile application project. However, it is always good to consider additional testing techniques like assertions for exceptions (assertThrows) and specific true/false assertions (assertTrue) to further strengthen the testing strategy. By doing so, I believe the software can be developed correctly to meet user expectations and withstand real-world application.

REFERENCES

Moradov, O. (2023, July 26). Unit Testing: Definition, Examples, and Critical Best Practices. BrightSec. Retrieved July 28, 2023, from <https://brightsec.com/blog/unit-testing/>

Fadatare, R. (n.d.). JUnit Assert.fail() Method Example. JavaGuides. Retrieved July 28, 2023, from <https://www.javaguides.net/2018/08/junit-assertfail-method-example.html>

"JUnit Testing Tutorial: A Comprehensive Guide With Examples and Best Practices." LambdaTest Learning Hub. Retrieved July 28, 2023, from <https://www.lambdatest.com/learning-hub/junit-tutorial>