**4-2 Journal: Unit Testing Approach and Writing JUnit Tests**

Winnie Kwong

Southern New Hampshire University

CS-320 Software Test Automation & QA

Professor Toledo

September 23rd, 2023

**Meeting Software Requirement Expectations**

The testing approach I used to align with the software requirements is understanding the business context, defining the testing objectives and metrics, and executing the testing plan. Before implementing any code, the first step is to determine the scope of the assignment to ensure I understand the goals, priorities, and constraints. For example, the first assignment for module three was to create a system for creating, adding, deleting, and modifying an account. In module four’s assignment, the object was to create functions for tasks within the application to add, update, and delete task objects. Both assignments consisted of restrictions that must be defined and implemented to achieve the software requirements. When executing the testing plan, I could log a history of what was tested and their results, which further helped me determine which branches of the system were covered and if all the branches in the code had been met.

**Meeting Software Quality Expectations**

The overall quality of my JUnit tests for contact service and task service meets the standards portraying examples of best practices of external and internal attributes. All test files prove functional correctness and user error protection because each test’s coverage was 100%. When test coverage is 100%, all functions can accomplish the specified objectives while covering all branches that may lead to vulnerabilities or other unexpected scenarios that can hinder the system. Besides satisfying end-user experiences, the test files are also beneficial for the developer because of their aspect of maintainability. All files were proofread to ensure all classes, methods, and variables were readable, simple, and consistent to reflect their purpose.

**Ensuring Software Code Was Sound**

I ensured my code was technically sound by implementing software testing principles and applying JUnit guidelines. Testing positive and negative scenarios is essential because it makes the code more stable and reliable. In the file TaskTest.java, block codes 28 to 34 and on lines 93 to 116, test the description objective’s correct and incorrect outcome to ensure the code is protected from the end user’s mistake, increasing the security of the code. Throughout the test files, assertions detect small and subtle errors. Covering the code with assertions minimizes logical errors while increasing data reliability for recipients to receive the matching data that the sender communicated. Another example to be considered sound is by applying best practices that include reducing code smells. The code smell used in the file TaskServiceTest.java on lines 57-65 is an example of appropriate naming conventions. The function tests for taskID duplication whose name is consistent with its function and excludes long or meaningless identifiers. Ensuring my code was sound allows the code to have better quality and performance.

**Meeting Efficiency Within the Code**

I ensured my code was efficient by constructing clear and consistent coding while using proper comments. In the file ContactTest.java from lines 22 to 36, the first objective is to test and ensure the code is initialized. Providing comments helps identify where those bugs are and makes it much easier to read. Also, throughout all the test files, such as in ContacTest.java from lines 104 to 134, the function tests all limitations of when a phone number is inputted. The section is an example of efficiency because it tests small pieces of code in isolation so developers can quickly find and isolate broken code. Having an efficient code reduces resource consumption while making the code more reliable and faster.