Summary and Reflection Report

Charles A. Beavers

SNHU

CS-320: Software Test Automation & QA

10/15/2023

# Summary

My approach to developing the features in this course was fully aligned with the software requirements outlined for these projects. All attributes and services were implemented with the desired effect established by the requirements and all tests to ensure the functionality were passed successfully. Specifically, the unique ID requirement was met by creating an incremental numbering object from the   
“AtomicLong” class. This method ensures that the ID will not reach a length of 10 digits and incrementing ensures that the ID will always be unique. Testing for this requirement finds that each object created correctly assigns an ID to itself upon creation using the format specified. Each requirement was tested in a similar way and achieved successful test results that indicated proper functionality.

The percentages achieved for the coverage tests were greater than ninety and ensured that most, if not all, portions of the code were passed through a conditional statement resulting in a Boolean value that determines validity. The coverage tests were required to be above eighty percent as a minimum and this threshold was met and passed significantly resulting in more confidence of my JUnit tests.

To ensure that my code was technically sound, I used tests that allowed me to violate the verification barriers and determine whether the code catches the issue. To do this I used the line   
“**Assertions.*assertThrows*(IllegalArgumentException.class, ()-> new Task(null, "Writes story");”**. This line checks whether an exception is thrown when an object is created with a null value, which is invalid per the requirements. Using variations of code with this method, breaching the bounds of verification functions and checking for exceptions, is how I determine confidence in my code.

To reduce the amount of code needed, I used a switch case to handle the update method for the service types and a selection option to consolidate all updating code to one method. In addition to this, including validation methods within the setters of object variables helps to reduce code clutter and excessive runtimes or, in other words, be more efficient.

## Reflection

The testing techniques I used on the features in this course were unit testing and experience-based testing. Unit testing refers to JUnit tests that help determine outcomes of algorithms. These JUnit tests were implemented in each feature or project and were the primary source of determining valid outcomes of the programs. The experience-based testing refers to moments in which I had identified similar possibilities due to previous project outcomes and now know to test for said possibilities.

While I used a “white-box” testing method (unit tests) and an “experience-based” testing method, one testing method remains at large. “Black-box” testing is the other main form of testing that deals with the system. “Black-box” testing compares the outcome of an entire application or system with a presumed value or functionality. This technique helps to determine if all parts of an application or system will function properly when used together to achieve a specific outcome. Methods of this technique include equivalence partitioning, boundary value analysis, use case testing, and more.

Each technique has its own place among most software projects but various situations and/or stages of the SDLC may require specific techniques. Incremental development would require using unit testing to ensure that each component or increment of code is functioning properly before moving on to a broader stage. This would be an example of *white box* testing which ensures inner functionality. “Black-box” testing, however, would be geared more towards stages at the end of the SDLC where the entire application can be tested to compare its output with expectations or to determine if the application is even usable by the target audience. Acceptance testing is a form of “Black-box” testing that is used near the deployment stage and is geared towards future users of the application or system to find flaws in usability. “Experience-based” testing can be used during any situation as it is quite vague. Experience from past applications and systems can help programmers or testers to spot vulnerabilities or defects at any point in the SDLC.

Reviewing my mindset during these projects as a software tester would determine an excessive amount of caution. I felt wary of every test and verified each test multiple times and reviewed output as needed. Being the tester means that it is my job to ensure everything functions properly or is fixed if not and this can lead to cautious behavior. This is especially true when considering the complex relationships that the code contains. Without understanding the behavior and different relationships crafted by the code, testing becomes nearly impossible because you can’t make a proper presumption about the outcome.

To limit bias in the review of my code, I don’t think of the code as my own. Focusing less on the idea of who wrote the code and more on the idea of functionality, it makes it easy to see the code as a generic piece of work that needs to function. Testing your own code on a regular basis, however, could most certainly be an issue, especially when the code covers a project you are proud to have developed. You may determine that there is nothing wrong with your work when in fact there are plenty of issues and potential problems for the future that others may have noticed.

Maintaining your integrity and dignity as a software developer or tester is important to ensure that all software you create is functional and safe to use. Quality should be at the forefront of software development and cutting corners is extremely impractical as one cut-corner can lead to a cascade of issues that span multiple areas. To avoid technical debt in my career as a software engineering professional, I plan to implement time management techniques that allow for adequate amounts of time to complete projects correctly and with quality.

References

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2019). *Software testing: An ISTQB-BCS certified tester foundation guide*. British Computer Society.