Summary and Reflections Report

Malcolm Williams

Southern New Hampshire University

CS-320: Software Test Automation & QA

Professor Norman

February 25, 2024

Summary and Reflections Report

## Summary

## A screen shot of a computer program Description automatically generatedAs a software engineer for Grand Strand Systems, I was responsible for developing and testing back-end services for a mobile application. I delivered three services: contact, task, and appointment, each with specific requirements. I implemented each class according to the requirements and designed the appropriate test cases to verify that my code met every specification. I designed the test cases to challenge my code, ensuring that almost every line was executed. This resulted in satisfactory test outcomes, which demonstrated the accuracy of the code and the effectiveness of the tests. I am confident in the code quality because my tests covered 91.2% of the service code. The remaining 8.8% consists of small and uncommon logic branches where a review is sufficient to ensure adequate code quality. The service code is validated to be technically sound by comparing the actual output to the expected results of the functions. When a unit of code produces a value that matches the expected value, the unit passes the test. The efficiency of the tests checking each requirement is illuminated in examples such as this case that verifies a task may not be created with an invalid name. It utilizes Junit’s parameterized tests to check the name length and a null name simultaneously.

## Reflection

To test my software, I used a technique called unit testing, a testing methodology that focuses on a single unit of software. The units are usually the size of a single function, which helps to ensure that the smaller building blocks of an application work as expected, which unearths and eliminates issues early in the development process. The developer who writes the code is usually the same person who will create unit tests to verify the code, by comparing the function’s output to the expected output defined by the developer.

Although I extensively used unit testing to verify these deliverables, many other testing methods can be applied when developing software to help create clean and correct code. One method is system testing, which is related to unit testing. Instead of validating a small unit of code, a larger system within the software is tested. The system may be the whole application or a sub-system, but it has a larger scope than unit testing. Another method is integration testing, which is performed to verify that two or more systems communicate and function together properly. Individually, those systems may seem ready, but integration testing ensures that they will work smoothly in combination. A third, but not final form of quality assurance is the performance testing of your software. Performance testing is the process of measuring how your software performs on different platforms and with various resources available to the software. This includes normal operations and extreme cases. This form of testing can help to find bottlenecks and even potential denial of service attack points.

All the mentioned methods of testing have practical uses during the development of complex software. Unit testing is especially useful in finding bugs early in the development process and works to foster a workflow with continuous quality assurance. The value of integration testing may become apparent after developers have completed multiple modules of an application and it is time to integrate the modules to work together to form a system. System testing will be very helpful in demonstrating to stakeholders that the software meets the requirements before its release. Performance testing is very important in most cases, but it is more critical when developing software for mobile devices where hardware resources are limited.

While writing test cases, I was careful to not write too many or too few test cases. In my initial delivery of the contact service, I was not aware of Junit’s ability to use parameterized tests and wrote many tests that could have been simplified. After learning this, I reduced the number of tests while keeping the same level of coverage in my final deliverable of the backend services. Since unit testing is done by comparing output to expected results, how your code works and what is expected can be influenced by developer bias. I tried to reduce as much bias from my tests by ensuring that I achieved a high coverage with JUnit’s coverage tool. If I had relied only on my assumptions of how the units should function, I might have missed critical logic paths that could cause software defects.

The software that developers write has real-world consequences. There are many amazing technologies to be admired, but when done incorrectly, one can hope that it only causes inconvenience or disruptions, but developers can unintentionally create safety hazards and security vulnerabilities if the quality of their code is not sound. A bug in a game may make the game less enjoyable and cause a company to lose revenue and reputation. A bug in a rocket launch sequence can cause the loss of life of the astronauts on board. Both are undesirable, but the range of possible disasters from poor code quality varies by environment and should not be overlooked so remaining disciplined in the commitment to code quality is essential.

References

GeeksforGeeks. (2017, August). *Types of Software Testing - GeeksforGeeks*. GeeksforGeeks. https://www.geeksforgeeks.org/types-software-testing/

Carpenter, A. (2021, October 28). *What is Integration Testing?* Codecademy News. https://www.codecademy.com/resources/blog/what-is-integration-testing/

Black, R. (2018). *system testing*. SearchSoftwareQuality; TechTarget. https://www.techtarget.com/searchsoftwarequality/definition/system-testing

*Performance Testing | Software Testing - GeeksforGeeks*. (2019, April 18). GeeksforGeeks. https://www.geeksforgeeks.org/performance-testing-software-testing/

‌