**Software Test Automation & QA**

**Project Two**

**by Ron Causey**

Summary

In order to ensure that the software would meet the client’s requirements I read through the requirements and specifications several times. I noted down all crucial information so that I could create the most efficient and accurate tests as well as code the software itself to fit their needs. For example, you can see on lines 18-23 of the TaskTest file that I carefully test that the unique IDs cannot be updated per the client’s request. I also run several other tests throughout the TaskTest file verifying that the ID, name, and description cannot be null and that they cannot surpass the desired character length.

In the Task class file, you can see on lines 9-18 that the code not only meets the requirements but helps their program be a little more secure. The input validation that I coded will help protect against potential attacks the client could face. If we were to simply allow a user to fill the name field with as many characters as they’d like it could lead to problems. That being said, I would like to code in some more secure input validation to protect against other attacks such as cross-site scripting, SQL injection, and other common vulnerability attacks the program could come up against.

The quality of my JUnit tests is definitely improving. The resources that were provided to us in each of the modules helped a lot as well as YouTube videos. The coverage percentage was over 80% in each of my assignments which was very encouraging, and I was happy to see how high my total percentage was for the final project. I will be using testing more often in my programs and will be sure to use it early after seeing how beneficial this is. Previously I had only done dynamic testing by simply running my program after a little coding and seeing if it worked.

**Reflection**

The main testing that I did was static testing where I would review the requirements, review the code base, refactor code, expand upon the code, and then write tests that would ensure those requirements were met. While we did explore some dynamic testing it is not something that I fully implemented at this stage in the application development. There are several other testing techniques which we did learn about but didn’t implement. Those testing techniques are gorilla testing, end to end testing, acceptance testing, security testing, and performance testing.

* **Gorilla Testing**: Gorilla testing is used to check to see how robust the application would be in a real-world environment.
* **End-to-end Testing**: End-to-end testing is another where we test the application for real-world scenarios. This can be things like interacting with databases, types of hardware, and also networks.
* Smoke Testing:Smoke testing is good for new builds and tests the basic functionality of an application.
* **Acceptance Testing**: Acceptance testing is when the customer gets to inspect the application to confirm all functionalities of the application work and that it meets their requirements.
* **Security Testing**: Security testing helps tests the application for vulnerabilities that hackers can manipulate.

**Performance Testing**: Performance testing involes different types tasks like stress testing, scaling the application, and checking response time.

It goes without saying that every step of the software development life cycle is important to making a high-quality final product. That being said, I feel that many people underestimate just how much of an impact the testing phase of the software development life cycle is. It is much more than just making sure a program can be used without crashing or that navigating isn’t buggy. Software testing can help ensure that you have secure code, that the code runs well, and it also saves time and money.