**Project Two Report**

The tests I wrote aligned with the software requirements of the system. For the case of the three service classes (ContactService, TaskService, and AppointmentService) I ensured that the tests recreated the functions that perform CRUD operations (Create, Read, Update, and Delete). This improved the tests effectiveness by recreating functionality that will be seen in the final version of the application. Effectiveness was measured by the result of run tests (pass/fail). Using JUnit APIs like assertEquals allowed me to compare test and application behaviour and ensure functionality requirements were met.

Tests were deemed effective by two factors. The first of those being the test coverage. Test coverage for the project was at the standard 80 percent. This fact increases the likelihood that undetected bugs will arise, as the majority of the codebase has been tested. Although it was important to have high test coverage, the effectiveness of the tests take precedence. I focused more on ensuring that the tests met both functional, behavioral and system requirements. THe quality of the tests are the most important.

I followed best practices to ensure that my tests were technically sound. This included checking for normal and edge cases. It was also important to look for failure scenarios.

When writing tests, I chose to leverage the test-driven development methodology. I chose to have the system’s requirements and behavior dictate the manner in which my tests were written. Knowing those two factors allowed me to construct tests that satisfied both business and functional requirements. The test were written before the source code was written for the application. THis choice allowed me to hone in on truly understanding the requirements of the project. It also made the project easier to write, as I knew what result I was looking to be returned in various methods. An example of this methodology in action is the testing process for updating an entity. I needed to ensure that the setter correctly updated and performed all the validation. The update functions can not function without this fact, as the setter function is called when updating an attribute of an entity.

An example of a testing methodology that it did not use was static code analysis. Through use of tools like Maven Dependency Check, a code base is analyzed for vulnerable dependencies that may compromise the security of the application.

I avoided bias in many ways. The first being confirmation bias. I had to take a step back and take myself out of the picture when potentially reviewing my code. Knowing that at the end of the day the code is meant to satisfy user needs. This means that as a developer I need to search for edge cases and not assume that code works due to familiarity. It is also important to have other developers inspect my code. Having different perspectives is important, as it can uncover edge cases and potential bugs that the original developer may miss.

Maintaining discipline is imperative when testing software. In order to meet user and stakeholder requirements, the software application itself must be functioning to expected levels. It is important to be methodical when writing tests and adhering to best practices.