There was a dependency between Appointment Service and Project One that I could not remove but other than that, Project One was more successful than the other projects. The coverage was 88.6% so my JUnit tests were pretty effective. I could have fixed the Task Service coverage had I designated more time to it and less to Contact Service. Task Service.java had coverage of 0% which brought the whole Project One down. Below are lines of my code that did not meet the requirements:

for (Task obj : tasks) {

System.out.println(obj);

}

service.addTask(new Task("0000000001", "Singing", "Wedding Engagement"));

System.out.println("Delete Task ID #0000000002");

service.deleteTask("0000000002");

System.out.println("Update Task ID #0000000003");

service.update(new Task("0000000003", "Running", "Jogging with friends"));

Code that did work from AppointmentService:

private Appointment searchForAppointment(String id) throws Exception {

int index = 0;

while (index < appointmentList.size()) {

if (id.equals(appointmentList.get(index).getAppointmentId())) {

return appointmentList.get(index);

}

I ensured my code was technically sound and efficient by testing, checking the errors and coverage, testing again, and continuously repeating the process until I obtained the results that were required (80% or higher). Using JUnit testing I was able to ensure the expected outputs were correct by using a combination of methods and JUnit assertions. Again, more time on the Task Service could have brought the coverage up more but 88.6% still meets requirements. The Appointment Service package gave me the fewest problems within Project One but the dependency mentioned above did affect my coverage.

Software testing techniques I employed during this project include JUnit testing. JUnit testing is used to test code and ensure it runs properly by breaking the code down into smaller sections and finding the error within those smaller sections. Software techniques not specifically focused on for this project include system testing, functional/non-functional testing, and acceptance testing to name a few. System testing ensures the software is ready for use by double checking for any errors in the code after integration testing and acceptance testing are complete. Functional testing tests the functional requirements of the code, or the requirements that influence the way the software behaves. Non-functional testing is the testing of non-functional requirements: security, usability, and reliability. Acceptance testing is the last phase of software testing and ensures the system is ready for actual use.

Each of these techniques has its own practical uses and implications. JUnit testing ensures bug and error free code and is best for repeatable automated tests. System testing involves putting the whole system together and testing it as one piece to ensure the software meets every standard. Functional/non-functional testing are used to ensure the software meets all requirements and is user friendly, secure for confidential information, and reliable. Examples of acceptance testing include alpha and beta testing. Alpha tests are internal and point out the obvious defects and beta testing is external testing before the product is rolled out for use.

The mindset I had starting on this project was that it would be fairly easy to complete since all the pieces had already been built. The individual pieces were more challenging since each one was made up of its own interrelationships. I am not confident in my ability to write code so my bias leans more towards something being wrong with my code. I am my own worst critic when it comes to programming. My lack of confidence helps because I feel the need to double and triple check my code. However, sometimes I get overwhelmed with it, so peer review is beneficial.

It is imperative that one stays disciplined when writing code. Cutting corners when writing code can lead to a program that is difficult to test resulting in missed errors and threats to the system. It can also result in code that is difficult to maintain through updates and changes made to the system. In the “real world,” error-free code ensures confidential information is not shared with the wrong parties, financial transactions can take place over a secure connection, and systems that provide essential services (power, water, phone, etc. companies) continue to work. I plan on avoiding technical debt by following best practices used throughout the industry and being willing to learn the best practices of the company I work for even when they are different from those of the industry as a whole.