For this assignment I will reflect on my testing approach for the project submitted last week. The basis of testing stems from the software requirements determined in initial stages of planning. For the different milestones we were given a clear set of instructions of what to implement and test for. There were opportunities I found with the requirements to improve on what was asked for but remembered that this was practice for real world applications where we do not always have the budget or time to expand the scope of the project. I dialed back on what I was to program to made sure that only what was asked for was delivered. This also decreased the code based which allowed for greater testing coverage with less test cases. An example of an expansion idea that I retracted was implementing a Singleton pattern for each service class. This would have limited only one instance of the service to be created and not have multiple arrays generated. From research this would have also been much more difficult to test for. Forum posts suggested that Junit tests do not work the same for Singleton classes and would be much more difficult to create. With the foundation for the code base simplified and aligned with the requirements I was able to create Junit tests that were simple, straightforward, and had a high code coverage. The code coverage average of the project was around 80% which is what forum posts suggested is the industry target. The code coverage identifies the amount of code that is being tested against in Junit tests. It is important to strive for a higher number when able to ensure most of the program has test cases.

Writing Junit tests was initially difficult but once I was able to learn the syntax and methods used it became much more clear how I can leverage it to be useful. When creating the application I made sure that I was programming primarily to meet the requirements. This made each method fairly straightforward. Many methods were getters and setters or managing an array list. I was able to ensure that my tests were technically sound by identifying what exceptions would be thrown if something goes wrong and the return type. By focusing on these two concepts I wrote test cases to ensure each method fails and returns how it was designed to. Examples of this can be seen in the “AppointmentTest” class. I identified how each parameter could be entered incorrectly and created a test for it. If the ID supplied is too long or a null value then an exception will be thrown as expected. Going into the efficiency of my tests I made sure to create them in a way that isn’t computationally intensive and is portable. Each test for the constructors of the POJO classes creates an object with an incorrect parameter type and validates that it throws the correct exception. To decrease typing I created variables with a date and description that would be used often in each test. Being able to utilize a variable instead of typing out the string cut down on a lot of time. Lastly, most of my tests were created in a way where I could copy and paste them for similar test cases. I only had to substitute out the variables that I was testing against each time. Designing each method to be portable in this way cut down on having to type out each test case.

Next I will reflect on the testing techniques used in the project. Static techniques employ the use of specialized tools or review of the project without compiling and running the program. Formal review is used in large enterprise projects and validates the design documents before programming begins. In my simple design I used a form of this by looking at the specification document and creating a list of requirements. I then double checked that I captured everything in my notes before starting work on the project. Another static testing tool I used was code coverage. This tool is built into the eclipse IDE and identifies what percentage of the code base has been tested against. I made sure to cover at least 80% of the code base so that there are less lines of code that are more likely to produce unexpected results. Using the requirements gathered earlier in the project I then employed specification based techniques to create test cases. I knew what each method should return or an exception that needed to be thrown before the code was even written. Some of the test cases I was able to write before the code was even finished. The most common type of analysis I used was boundary analysis with character limit in the constructors of each POJO class. I had a test case for inside the boundary and outside to make sure the constructor behaved as expected.

There are many other types of techniques that I could have used but were not needed for the project. Equivalence partitioning is the idea of grouping similar types of input together to cut down on the number of test cases. If the inputs are expected to behave the same then there is no need to create a test of each one and is much more efficient to group them together. Decision table testing is another technique where all input conditions that can occur and all the actions that can arise from them are structured into a table. This creates an easy way to identify test cases for each business rule. Along the same vein of diagrams is utilizing flow charts to create a visual structure of the code base. Pseudocode can be used to supplement this to create a non-technical diagram of how the program should operate that is approachable by people that do not read programming languages. By mapping out the program, test cases and reviews can be made simpler by testing each of the branching paths or decisions in the logic.

The final piece I will reflect on for Project One is how my mindset has changed regarding testing over the duration of the course. As I have been developing my skills as a software engineer I have mostly focused on creating programs and making sure it works in ideal conditions. Now that I know how to use Junit and have learned some techniques I feel much more equipped to plan out projects and clean up so that I can catch situations that may arise in fringe cases. Testing needs to be incorporated from the start in order to catch costly mistakes. Simply creating tests at the end of a project is not sufficient and can become a large problem. Using caution when implementing things is also important to the success of the project. Creating diagrams and various levels of review can catch some of the problems mentioned earlier so that the application is aligned with the customers specifications and does not have fundamental flaws in the design.

Creators of any kind have sentimental attachments to what they create. This can lead to bias towards a program by the software engineer that created it and makes it harder to spot flaws. Separating the tester from the creator helps eliminate this and allows a different perspective to evaluate the work that has been done. They should not have any attachment to the code and be able to offer constructive criticism to the engineer that will give them guidance on how to fix or improve it. From personal experience I have had to outreach to professors and peers for help on a couple projects I have work on in school. As the designer I was not able to see what was wrong with what I wrote and needed someone else to point me in the right direction. Because of situations like this I value code review and peer programming as methods of quality control to help remove bias.

All members of the production pipeline are essential to creating an application that is quality and meets the needs of the client. It is easy to skip steps that are repetitive and boring. However, these tasks were generally created in order to ensure secure and proper programming. It is important maintain discipline in solid programming techniques and testing to create a quality product. I plan to keep learning the skills that I need to make my work the best that it can be. Increasing familiarity with all of the steps of the production pipeline and how my work effects others is a start that I have learned in other fields that I have been employed in. By learning what others do, you can change your workflow to not only benefit the task that you are involved with but further down the line as well. I also plan on learning more design patterns and notation that are common and maintaining discipline to follow them even if it isn’t convenient. I have learned a lot over the course of this class and will take the information and mindset into future courses and my career.

Source:

1. Hambling, Brian Morgan, Peter Samaroo, Angelina Thompson, Geoff Williams, Peter. (2015). *Software Testing - An ISTQB-BCS Certified Tester Foundation Guide (3rd Edition).* BCS The Chartered Institute for IT. Retrieved from  
https://app.knovel.com/hotlink/toc/id:kpSTAIST01/software-testing-an-istqb/software-testing-an-istqb