**Project 2**

**Summary:**

The unit testing approach that I took while writing tests for the three features for my project would be that along with writing my code for the project while thinking about the software requirements, I also made sure that for each of the requirements I had a test case that proved that the code written fully matched the requirements. An example of this would be where the requirements stated that a task object shall have a unique ID string that cannot be longer than 10 characters, I tested IDs that contained less than 10 characters and also ran tests with IDs that contained more than ten characters starting with a test where the ID had 11 characters to show that the code would throw an exception in this case. To ensure that I was testing my code was effective and that the code was being tested as much as possible I ran coverage tests on each feature and ensured that I had at least 80% coverage on each feature and worked to go above that percentage on every feature. To ensure that my code was technically sound I tested all cases of the code preforming the jobs that it was meant to using JUnit tests to make sure I could catch any errors or defects before continuing on to the next blocks of code that needed to be written. To ensure that my code was efficient I implemented things such as searching algorithms that could be used by multiple methods to be able to quickly find objects to be modified or deleted. An example of the code I used in the feature for the task service would be:

public int getIndex(String id) {

int i;

int index = -1;

for (i=0; i<taskList.size(); ++i) {

if (taskList.get(i).getID() == id) {

index = i;

}

}

return index;

**Reflection:**

The software testing techniques I employed during this project were boundary analysis, use case testing, statement coverage and branch coverage. Using boundary analysis, I tested each method I created to ensure that any input with in the boundaries would have the correct output and that any input that was outside the boundary would throw an error, which an example of how to use this technique. In use case testing I tested each method to ensure that the expected output was achieved for each use of the method to show that all were functioning as expected. Using statement coverage and branch coverage testing I could see that I had indeed wrote tests that hit as many statements as possible in my code and also took every branch that was able to be taken in my code. Using these coverage tests I was able to say with confidence that at least 80% of my code was tested and in working order. Some of the techniques that I did not use while testing my code were exploratory testing and experienced based testing. I did not use these techniques while testing my code dude to both of these techniques are less structured and require more experience in software testing which I am still fairly new at. Exploratory testing is a technique where you use your knowledge of the system to explore what has been created and find errors and defects while you do. Experience based testing is used when requirement documents are lacking or not available and a developer uses their experience to justify the testing that the code is put through which was not the case for this project. While working on this project it was hard to remain entirely unbiased due to the fact that I was responsible for writing and also testing all of my own code. It is human nature to want to be able to say that you did not make an error in your work which is what makes it hard to write the best possible tests for your own code which is why it is important to use caution and be as honest with yourself as possible when testing code that you have also been the main writer of. Knowing this after I would write a block of code for the project, I would try to take a break and step away from the code so that I could come back and try to look at the code and write tests with the mindset that it was someone else’s code that I was testing. Another way that I tried to limit bias while writing tests for my code was to think of testing my own code as a challenge to myself. In this way of thinking I was able to adopt the though process that if I write the best tests I possibly can than I will catch any defects or errors myself and can correct them so that when I am done testing and my code is seen by others it will be that much better and harder for someone else to find errors in it, thus creating a better end product in the long run and waste less time of others who would not have to correct simple errors. Being disciplined in your commitment to quality as a software engineer is extremely important if you plan on producing the best possible product for your customers. When you are disciplined in both writing the best possible code you can and also testing you ensure that the software that you produce is the best quality possible. When developing software that can affect others lives, we as developers need to remember that the more disciplined we are in writing and testing our code, the more we can affect people in a positive way and in some cases ensure that anything we create is both safe and reliable for others to use.