I believe my tests were well aligned with the requirements. I wrote the tests based on the interface of the functions in question and the requirements specified in the rubric. Each specification was given one test it was expected to pass, one it was expected to fail.

Coverage percentage tarrget is typically 80% or better. While 100% coverage is ideal, achieving this in practice is often very difficult. However, in this case I was able to achieve it. Each function call involved no more than a single conditional instruction, and I tested input that would make it false and input that would make it true. There was no line of code that was missed, and no line of code that could function under conditions which were not tested. Construction of a valid object tested the true branches, and testing invalid updates and constructions tested the false branches.  
  
Non mutability was not unit tested, as declaring those fields final ensured the code would not compile if a change was attempted. Unit testing can only be done if the code will run at all. Furthermore, I was able to avoid explicitly testing for null values. This was because the Java library functions which were used to test length constraints would themselves throw an exception if used on a null object. This works fine for the moment, but there is a risk that future updates to the Java library may change this behavior.

Lines 41 and 51 in TaskServiceTest.java have a warning in Eclipse for an unused local variable. In this case, I am only interested in the results of constructing an object, not in the results of using it, so this is fine. I could supress this and other warnings that are unlikely to be a problem in my code, but leaving it in ensures that I have taken a look at that line of code and ensured that each instance of the warning is in fact OK. In other areas, for instance, if I had the same warning on line 15 in AppointmentSErvice.java, that would be a problem because the apptID variable needs to be passed to apptList.put(). Even if the put() call wasn’t throwing an error, I’d know something was wrong.

The main thing I did to ensure efficient code was to keep everything simple. While the simplest code is not always the most efficient, it often is. When it is not, simpler code tends to be easier to replace with something new without missing key functionality in the replacement. Most of my code could be good examples, for instance the function for setting a date in ApointmentSerivce.java at line 31. It simply checks the constraints and sets the variable without doing anything unecessarily fancy. Inlining the constraint check into the setter function would likely be slightly more efficient(compiler optimizaitons aside), however, if constraint checking became particularly complicated, having it in a separate function would make it easier to replace my initial pass at a constraint check with a more efficient algorithm for checking constraints. For instance, in initial development, I explicitly checked for null values when checking constraints. However, I was easily able to optimize this by realizing that the functions called to check other constraints would already throw a null pointer exception. This made my function simpler, and if there is a particularly efficient way to check for and throw an excpetion, the Java library implementors would likely do better than I would.

This was primarily white box testing. While I did not refer to internal workings except to justify the lack of an explicit test for null or immutability, it was all done with full knowledge of the internal structure thus was effectively white box. This was supplemented with static testing. Static testing by the compiler and IDE gave warnings to help me find potential flaws before unit testing, and prevented even trying to change immutable values.

Other testing was not extremely viable. For instance, while I tried to approximate a black box approach as much as I could, it was not strictly speaking possible when I wrote all the code being tested. Regression testing tests to ensure no new flaws are introduced, so it is irrelevant at such an early stage in the project where changes to a working version have not yet happened.