My testing approach aligned with the software requirements 100%. For the Contact Class I ensured that the unique contact ID string cannot be longer than 10 characters, the contact ID shall not be null and shall not be updateable. The first name string with the same requirement, and the last name string. The phone string must be exactly 10 digits and shall not be null and the address field must not be longer than 30 characters and the address field shall not be null.

For the Task Class requirements, I ensured that the task ID String shall not be longer than 10 characters and shall not be null and not be updateable. The name string cannot be longer than 20 characters and shall not be null. The description string field cannot be longer than 50 characters and the description field shall not be null. An example of this from the ID string is as follows: “0, Math.min(toString().length(), 10));”.

For the Appointment Class requirements I ensured that the object would have a unique ID and string that is not longer than 10 characters. That the date field could not be in the past and that the field is null. Also a description field that is not longer than 50 characters.

The JUnit test for contact service and task coverage was 100%. This is how I knew that the tests were successful. I ensured the code was technically sound by performing tests within Eclipse and Junit. An example of code from the Task Class is the following: “private Task searchForTask(String id) throws Exception {

int index = 0;

while (index < taskList.size()) {

if (id.equals(taskList.get(index).getTaskId())) {

return taskList.get(index);

}

index++;

}

throw new Exception("The Task does not exist!");

}”

I ensured the code was efficient by only writing what was necessary. An example of this is: “public void newTask() {

Task task = new Task(newUniqueId());

taskList.add(task);

}”

I ensured that the code was technically sound by performing the JUnit test. An example of this is: “@Test

void testNewAppointment() {

AppointmentService service = new AppointmentService();”

Software testing techniques I employed include static testing by reviewing the code before executing it, and dynamic testing by executing the code through Eclipse and JUnit. Static testing involves reviewing the code and finding errors before executing, and dynamic testing involves executing the code.

Other test techniques include Black-box and White-box testing. Black-box testing is a technique where the internal workings of the software are not known to the tester. The tester only focuses on the input and output of the software. White-box testing analyzes the internal structure and internal design of the code.

Black-box testing is a good idea when the user does not know the internal workings of the software, so this would be beneficial when the developer doesn’t want the testers to know the inner working of the project. White-box testing would be good in the situation that the developer does want the inner workings to be known. Static testing is always good for saving money and time because errors are found before the code is executed, and dynamic testing is almost necessary because the code needs to be executed to make sure there are no errors.

As a software tester I used caution when making sure to static test before performing dynamic testing. For example, I would review the code, test through Eclipse, then test through JUnit. I limited bias by being open to criticism and review of the code by someone else. There are many reasons to be disciplined in commitment to quality as a software engineering professional. From loss of life, money, property, time, etc. there are so many catastrophes that have occurred due to an oversight within a program. There are missile defense systems that have failed causing loss of life for example. I plan to always use caution and thoroughly test any programs I create.