Writing good unit tests is critical for assuring program quality as a software engineer. A unit test is a test of a tiny piece of code or function that verifies its functionality, input/output, and behavior. In this paper, I will describe my unit testing approach for three software features, evaluate how well my approach aligned with the software requirements, defend the overall quality of my JUnit tests, describe my experience writing JUnit tests, and explain how I ensured that my code was technically sound and efficient.

The software requirements for the Contact object specified should have a required unique contact ID String that cannot be longer than 10 characters, while the contact ID should not be null or updatable. Another requirement was that it has to require a firstName, lastName, phone, and address fields that have length constraints and cannot be null. The Contact class has getter and setter methods for the ContactID fields to ensure that the code would run as required. The ContactTest class contains the test method, getContactID(), which verifies that the method returns the correct value for the contactID field. For the requirement that specified that the ContactService should be able to add, delete and update contact I included the test methods, testAdd() and testDelete(). The testAdd() method creates a new Contact object, adds it to the ContactService, and verifies that object was added successfully. The testDelete() method creates three Contact objects, adds them, then deletes the existing contact.

The second feature's requirements specified for the Task object were critical in ensuring that the TaskService operates efficiently and effectively. The unique task ID String requirement ensured that each task added to the service is unique and identifiable. The maximum length of 10 characters ensured that the task IDs are not excessively long, and the field is not updatable to avoid any unintended modifications to the task identity. The Task class, fits all of these requirements, guaranteeing that tasks may be managed efficiently within the TaskService. TaskService class allows you to add, delete, and update tasks in the service. The addTask method guarantees that the task ID is unique and that it fits the Task object's requirements. The deleteTask method deletes a task based on its ID from the service, whereas the update method updates the name and description elements of an existing task based on its ID.

For the third feature, Appointment, the requirements were that the Appointment object could not have an ID String that is longer than 10 characters and could not be null or updatable. The Appointment object must also have a Date field, that cannot be in the past or null, and the Appointment object must have a description String field that is no longer than 50 characters nor null. The unit testing approach was aligned with the software requirements as the tests explicitly verify each requirement outlined in the document. The AppointmentTest class contains nine tests, each of which checks a specific requirement for the Appointment object. The AppointmentServiceTest class has seven tests, each of which tests a requirement for the AppointmentService object. An example would be, that the AppointmentTest.testAppointmentID() method checks whether the appointment ID is created and returned correctly when an Appointment object is created, satisfying the requirement that "The appointment object shall have a required unique appointment ID String that cannot be longer than 10 characters. The appointment ID shall not be null and shall not be updatable.

I believe the Contact object was designed to be technically sound and efficient, but there were definitely areas where the code could be improved to make it more efficient and smooth. Overall, the tests provided good coverage of the code. For the Task object, I believe each test case validated the specific requirements asked and covers a variety of scenarios that could occur. The Junit tests for this feature provided full coverage of the TaskService methods, ensuring that all code paths were tested. The JUnit testing for the AppointmentTest and AppointmentServiceTest seemed effective. As they covered each requirement specified in the software requirements for the Appointment object and Appointment Service. The tests used the assertion methods of the JUnite framework to verify that the actual result matched the expected.

The JUnit tests weren’t easy, but it was easier to write as the software requirements provided a clear guideline for each requirement. The tests followed a standard pattern of creating an object or calling a method to verify. There were some frustrations when writing, as errors kept popping up, but after taking some time away and coming back with fresh set of eyes helped see what the issues were and what was causing them.