Jorge Rodriguez

8/23/2025

CS-320

Project 2

This report is to give a summary and reflection on the testing strategies I had applied when developing the code for the customer. Throughout the project, my main focus was to analyze the software requirements and implement JUnit tests to confirm that the features from each area have met the expectations given. In my report, I will provide an overview of the unit testing that is performed for Appointment, Contact & Task features.

Within the Appointment Feature, I used JUnits to verify that the objects within the "Appointment" enforces the constraints placed inside, such examples include. A valid appointment will be accepted with a correct ID as well as description and future date (“testValidAppoinmnet”). For any invalid inputs, such as (‘null’) ID, they are rejected (‘testInvaildId’). Within the “AppointmentService” i made sure to test service-level operations, example include: to add and retrieve an appointment (“testAddAppoinment”) anf in order to avoid duplicating an appointment (“testAddDuplicate”). With the approach I utilized, I was able to ensure that the requirements regarding the "Appointment" validity and behavior were met.

Next is the Contact Feature, within which I verified objects creation as well as field level constraints. Examples include: For a valid “Contact” it sets all fields (“testValidContact”). For Null or invalid IDs it will trigger expectations (“testInvalidContactID”). To update fields, it was tested to confirm mutability rules (“testUpdateFields”). Within “ContactService”, to validate I ;

Adding new contacts (“testAddContact”), to reject duplicates (“testAddDuplicateContact”).

Lastly, the “Task” class was put through several tests as well, such as the IDs, names and descriptions following the requirements given. I confirmed the Valid creations by using (“testValidTaskCreation”). And with Invalid IDs or null inputs, it will trigger expectations with (“testDeleteContact”). In “TaskService”, in order to verify the tasks can be added uniquely and be able to be retrieved I included this :

(“ String taskId = tempTask.addUniqueTask(fullName, description);

assertNotNull(taskId, "Task ID should not be null");” )

Each test is directly aligned with the requirements that were defined for the features: the IDs must be unique and non-null (e.g., testInvalidID in TaskTest) The .Contacts must include valid phone numbers and addresses. And the Appointments must occur in the future. These were consistently reflected in JUnit assertions like: (“assertThrows(IllegalArgumentException.class, () -> new Appointment(null, futureDate(), "Checkup"));”)

The quality of my JUNit tests were effective because I covered many of my bases such as including, Happy paths (valid creation and retrieval).Negative cases (nulls, invalid lengths, duplicates). Boundary conditions (future dates in Appointment.) With my coverage reports they reported high code coverage, confirming that the critical logic paths were properly validated.

For my experience writing JUnit Test, I was put in a position where I had to think like an end user and anticipate potential incorrect usages. FOr example, it may be an easy mistake to add a duplicate within the Contacts in ContactServcies. By putting myself in this position, I came to the conclusion that I should incorporate strict duplicate checks, so the service can overwrite any existing data that is present. To ensure technical soundness, I relied on assertions and exceptions checks in order to validate constraints, one example : (“assertThrows(IllegalArgumentException.class, () -> new Task(null, "Name", "Description"));”)

This example demonstrates the system's defense against invalid data. For ensuring efficiency, it was maintained by avoiding any and all redundant testing logic. One example comes from my “TaskServiceTest” the tasks within were cleared after each test after using the following code

(“ @AfterEach

void tearDown() throws Exception {

tempTask.clearAllTasks();

}”). With this it kept the test independent and helped avoid conflicts.

For the testing techniques I utilized I used Unit Testing, this focused on the individual classes such as Task, Contact and Appointment. Boundary Testing, I checked the limits such as null IDs, empty strings and or future dates. I also utilized exception testing, this confirmed that any invalid inputs raised an exception. Out of the techniques I could have used but decided not to utilize they include, Integration Testing, this would test how multiple components interact (not necessary in this isolated project). System Testing, this tests verifues the complete end-to-end system.

Regression Testing, by re-testing after changes, this is mainly useful in ongoing projects with evolving requirements. These techniques would be critical in larger systems where modules are more interconnected. BUt with the project at hand, I did not see much promise in utilizing these particular tests.

In order to complete this Project, I had to put myself in a mindset of a detail oriented developer. I put myself in this mindset with the assumption that every possible invalid input can and will be entered. An example of such, when testing appointment with a past date this helped confirm that the class in question correctly ensured “future only” scheduling. In order to limit bias, I had tested not only valid but also invalid cases, instead of assuming that my code worked regardless. Being a “developer” its easier to envision my work as successful, but as a “ tester” I actively tried to find errors and faults to “break” the system. The bias would come into concern If I only focused on the happy path. For example, if I only tested a contact without thinking of checking for duplicates, I would have missed a critical flaw within the service design.

To be disciplined in testing, it ensures long-term software quality. By skipping negative tests or cutting corners would create technical debt, where hidden bugs could cause failures later.

One such example is failing to validate IDs could allow duplicate tasks, leading to data inconsistency. In order to avoid this, I plan to write tests in parallel with implementation,use automated testing tools to keep quality consistent. Having this mindset helps me ensure reliability and reduces the cost of bug fixing in future stages. .