Nyzheir Warner

CS-320

06/21/2025

7-2 Project Two

***Part 1: Summary***

My approach to testing was very closely aligned with the requirements defined in the software requirements document. I made sure that each functional requirement included in the documentation had corresponding Junit test cases to check the validity of the implementation. For example, look at a line from the 4-1 milestone requirements document.

“The task object shall have a required name String field that cannot be longer than 20 characters. The name field shall not be null.”

This line of the documentation explicitly states the Name field must not be longer than 20 characters and should not be null. We can see that in the testTaskName method:

A computer code with colorful text

AI-generated content may be incorrect.

Here we are explicitly testing to make sure this functionality is in place by testing that when the Field is longer than 20 characters or is null, we expect it to throw an Illegal Argument Exception which does confirm the requirements have been properly implemented and tested.

I believe that the overall quality of my tests is high as I made sure to include every requirement stated in the documentation as well as testing these requirements against bad and good data to make sure that it is working as expected. Although I am not using tools to explicitly test the percentage coverage of my tests, I think it would be a high amount around 90-100% as we are testing all code inside the base and every method is being used in these tests to verify the functionality of our code meets the requirements set for it.

To ensure that my code is technically sound I made sure to follow proper Java coding practices and the principles of OOP. For example, we can look at the Task class specifically lines 4-6:  
A close up of text

AI-generated content may be incorrect.

Here we declare the fields needed for the Task object. We can see that all the fields are declared private making them only accessible through controlled getters and setter methods following proper OOP principles. Additionally, we can see that the ID field is explicitly declared final, following the requirement stated in the documentation that the ID should not be updated.

To ensure that my code is efficient I made sure to extract code into its method to be reused where I would see the same code repeated throughout other methods. For example, in TaskService.Java we can see a method called getTask on lines 32-37:

A screen shot of a computer code

AI-generated content may be incorrect.

This method checks if a given taskID exists within the data structure and returns the Task object or throws an exception if not it’s not present. This piece of code removed redundant code that would be needed in multiple methods making my code more efficient. Additionally, I made use of the @BeforeEach annotation in JUnit to efficiently initialize shared testing objects in the TaskService class.

A computer code with colorful text

AI-generated content may be incorrect.

***Part 2: Reflection***

Employed Software Testing Techniques

1. Unit Testing
   1. Test small, isolated pieces of code individually.
   2. Validates logic is correct and input validation and error handling are done properly.
   3. I deploy this testing technique in the AppointmentService deliverables which has AppointmentTest and AppointmentServiceTests which test individual classes Appointment and Appointment Service this is called unit testing in which I am looking for expected exceptions on invalid parameters.

Not Used Software Testing Techniques

1. Integration Testing
   1. Tests the interactions between two or more components of a system.
   2. Ensure that different parts of the system work together as expected.
   3. This type of testing focuses on the interactions between services, classes, or modules.
   4. This test is used to catch errors that only appear when components are combined IE when data from one component is required for another test so make sure that necessary information is transferred correctly
2. Performance Testing
   1. Includes load testing, stress testing, and memory testing.
   2. Identifies potential performance bottlenecks or issues.
   3. Measures system metrics such as stability of the system, and performance under load, and can include things like system scalability or responsiveness tests.

Practical Uses and Implications

1. Unit testing is commonly used in a lot of software projects because it allows you to verify the correctness of individual components of the system and is essential to catch bugs early. Integration testing becomes more important as the system grows in complexity where multiple modules and services must interact. Performance testing is important for applications to ensure that they perform as expected under certain conditions like high traffic or other resource-intensive tasks, this testing ensures that the system remains responsive and stable under these varying conditions. Using these testing techniques appropriately helps reduce future issues and improve the product's overall quality, leading to a more reliable product and better user experience.

When working on these projects I tried to make sure I was following the requirement documentation as closely as possible. The relationship of the code is also important as they are dependent on each other so stringent attention to detail and testing is necessary. For example, in our Task Service it is important we properly implement and test the method in the Task object as they directly influence how the Task Service operates and will function during testing. This is also how I tried to limit my bias by implementing the technical requirements verbatim. All in all, I think is imperative that we as testers practice strict discipline in order to deliver the upmost quality in our product. Working in the field I think following the proper coding guidelines for your platform along with assuring your code follows requirements will ensure that you avoid technical debt and deliver products that deliver complete experience.

References

Brian Hambling, Peter Morgan, Angelina Samaroo, Geoff Thompson, & Peter Williams. (2019). *Software testing: An ISTQB-BCS certified tester foundation guide - 4th edition*. British Computer Society.