Jennifer Swinton

Week 7 Summary Report

Desmond Springer

Aug 2024

1. **Summary**
   1. Describe your unit testing approach for each of the three features.
      1. To what extent was your approach **aligned to the software requirements**? Support your claims with specific evidence.

For Contact Class requirements, a unique Contact ID of less than or equal to 10 characters, not null and not updatable was tested using:

public void setContactID(String contactID) {

if (contactID.length() > 10) {

throw new IllegalArgumentException("Contact ID cannot have more than 10 characters");

It also required first name, last name, and address strings with respective length restrictions and null checks:

@DisplayName("Contact ID cannot have more than 10 characters")

void testContactIDMoreThanTenCharacters()

This was applied to each step of the programing, ensuring that each of the software requirements was addressed individually.

* + 1. Defend the overall quality of your JUnit tests. In other words, how do you know your JUnit tests were **effective** based on the coverage percentage?

I’ve implemented methods that test both valid and invalid operation cases - addition and deletion of contacts, as well as cases when the user tries to delete a non-existing contact.

A test checks if a Contact can be added successfully, which is a good scenario to test to ensure that the ContactService works correctly.

Another good practice used in the tests is the use of @Test annotation and descriptive method names making it easy to understand what each test is doing.

The TaskServiceTest class includes tests for possible scenarios such as adding, deleting, and updating tasks.

Set up Before Each Test uses @Before to instantiate TaskService before each test to ensure that every test is isolated and doesn't affect other tests.

Positive and Negative Scenarios were included as tests for both positive scenarios (like adding a task) and negative scenarios (like adding a task with an existing ID or deleting a non-existent task). This helps ensure handling both kinds of input.

Validation of Functionality shows that after performing an action, you're validating whether the action has had the expected effect. For example, in the testAddAppointmentDuplicate(), it is verifying that the Appointment is not already existing.

Exception Testing is also checking for exceptions where needed to ensure that the service is throwing exceptions when it should.

* 1. Describe your experience writing the JUnit tests.
     1. How did you ensure that your code was **technically sound**? Cite specific lines of code from your tests to illustrate.

The JUnit tests are also technically sound and test the methods effectively. For example, testing for inputs exceeding character limit:

void testContactIDMoreThanTenCharacters() {

Exception exception = assertThrows(IllegalArgumentException.class, () -> {

Contact contact = new Contact("12345678901", "John", "Doe", "0123456789", "123 St."); });

String expectedMessage = "Contact ID cannot have more than 10 characters";

This test checks if an exception is thrown when an ID greater than 10 characters is provided, ensuring that the validation in the setter method is working correctly.

* + 1. How did you ensure that your code was **efficient**? Cite specific lines of code from your tests to illustrate.

The code is efficient as it uses setter methods for setting contact details with validation right at the point of insertion. This immediately invalidates wrong inputs and thereby reduces the chances of problems at a later stage.

task.setName(name);

task.setDescription(description);

The use of JUnit tests ensures that errors in individual pieces of code are spotted much quicker which improves the efficiency of the overall development process.

1. **Reflection**
   1. Testing Techniques
      1. What were the **software testing techniques** that you employed in this project? Describe their characteristics using specific details.

* I used primarily Junit testing to ensure each part of the classes worked appropriately. These tests are fast, and repeatable. They can help to identify bugs or other problems early.
* I also wrote code in short blocks and debugged and ran the code to test for any issues in the writing, such as typos, missed commas, etc. This makes it easier to identify the location of a problem if there is only small blocks of new code.
  + 1. What are the **other software testing techniques** that you did not use for this project? Describe their characteristics using specific details.
* **UAT testing** is not used. The code is not compiled into something that is user friendly yet and therefore not practical for this type of testing.
* **Regression testing**, like UAT is not yet at a point where it is feasible to apply. These two types of tests are used to test end results being delivered to the consumer.
* **Integration testing** is not used since it is testing segments of the application for the expected behavior.
  + 1. For each of the techniques you discussed, explain the **practical uses and implications** for different software development projects and situations.
* **UAT testing** is end user style testing to ensure that the deliverables match what the end users needs/want.
* **Regression testing** ensures that new changes do not impact the code outside of the small area that it is designed to modify.
* **Integration testing** is more difficult to write than Junit tests and has strict order to run and can not be done simultaneously. Each unit is validated and then combined with the next.
* **Unit testing** is writing and running repeatable automated tests for early detection of code errors. It is used in development by developers mostly although it can be done by testers. It helps reduce code of bugs by identifying them early in the process.
  1. Mindset
     1. Assess the mindset that you adopted working on this project. In acting as a software tester, to what extent did you employ **caution**? Why was it important to appreciate the complexity and interrelationships of the code you were testing? Provide specific examples to illustrate your claims.

Software testing isn't just about finding defects but also understanding the complexities within the software. This guides the overall quality assurance process, maximizing the effectiveness and efficiency of the testing process, leading to a better final product. I applied caution by carefully checking each step of the code, and ensuring full coverage of unit testing as I went along. I also made sure to apply changes suggested after a code review, such as that while I added a uniqueness check for the ID in the ContactService code, I failed to add it in the Contact code initially. By applying the feedback I was able to better improve my project.

* + 1. Assess the ways you tried to limit **bias** in your review of the code. On the software developer side, can you imagine that bias would be a concern if you were responsible for testing your own code? Provide specific examples to illustrate your claims.

Limiting bias in reviewing code is able to be done in several ways. Since I did not have another person to test my code, I couldn’t use independent testing. I used a checklist of items to verify and Junit testing to automate some of the testing. I also used feedback received to improve the code for the final version.

* + 1. Finally, evaluate the importance of being **disciplined** in your commitment to quality as a software engineering professional. Why is it important not to cut corners when it comes to writing or testing code? How do you plan to avoid technical debt as a practitioner in the field? Provide specific examples to illustrate your claims.

Being disciplined in commitment to quality is important because it is your reputation as a dependable developer and company on the line. Well written code is more reliable, efficient and functional and leaves customers more satisfied. It also helps to keep costs down by fixing bugs as they occur rather than after the software is released.

I intend to prioritize quality, ensuring that even under pressure, not to cut corners and produce low quality code, including proper in-line documentation. This can be ensured by regularly reviewing my code and taking frequent, short breaks to clear the mind. Continuous testing helps early detection of bugs.