Brandon Parkerson

CS-320

SNHU

23 Feb 2025

Project Two

1. Unit Testing Approach for Each Feature
2. Contact Service: I developed unit tests that focused on validating contact creation, update, and deletion operations. Using JUnit, I wrote tests to cover both expected behaviors (e.g., valid contact information) and edge cases (e.g., handling of null or malformed input). The tests directly mirrored the requirement to ensure data integrity and proper error handling. For example, a test verifying that a contact’s email address meets formatting rules was implemented like this: (ContactServiceTest.java line 32) assertTrue(“Email format is invalid”, contact.isEmailValid());.
3. Task Service: The task service unit tests were designed to validate task assignment, priority sorting, and deadline management. Parameterized tests were used to verify a range of task inputs without duplicating code. These tests ensured that all business rules were thoroughly validated.
4. For the appointment feature, I focused on time-bound validations, ensuring that appointment scheduling did not allow for overlapping or invalid dates. Exception testing was a critical part of this module. By testing for correct date handling and conflict resolution, the tests validated key requirements related to scheduling logic. Example: assertThrows(InvalidAppointmentException.class, () -> appointmentService.schedule(null));.

B. Defending the Quality of JUnit Tests

1. The overall test suite achieved a high code coverage percentage is about 90% as measured by our JaCoCo reports. This high coverage is indicative of comprehensive test cases that include both typical use cases and edge cases.
2. The tests were designed to catch regressions early. For instance, every change in the business logic triggered immediate feedback via our continuous integration pipeline, confirming that no functional requirements were compromised.

C. Experience Writing The JUnit Tests

1. Writing the JUnit tests was a rigorous yet enlightening process. Initially, mapping out the various edge cases required careful analysis of the requirements documentation. It was more rewarding than I thought. I ensured technical soundness by following best practices: clear naming conventions for test methods, using setup methods to initialize test data, and leveraging assertions to confirm expected behavior.
2. Efficiency was achieved by refactoring repetitive test code into parameterized tests and utility methods. In the task service tests, this reduced code duplication and improved readability.

Reflection

1. Testing Techniques
2. Unit Testing: Characteristics focuses on testing individual components in isolation. I applied both black-box testing and white-box testing. Unit tests are essential for quickly detecting errors in small, isolated parts of the code and are ideal for projects with clearly defined requirements.
3. Integration Testing: Although my primary focus was unit testing, I ensured that the individual modules, like contact, task, and appointment, interacted as expected. Integration tests are useful when verifying that interfaces between components work correctly.
4. Techniques Not Used: Performance testing was not used because the focus was on functional correctness rather than system scalability.

B. Mindset and Professional Discipline

1. Catuious Approach: In acting as both developer and tester, I maintained a mindset that anticipated potential failure points. I rigorously validated input parameters and error states to ensure that even unexpected scenarios were covered.
2. Cutting corners in testing can introduce long-term maintenance issues—commonly known as technical debt. I ensured that every feature had a corresponding set of tests before considering it complete.

The unit testing approach implemented for Project One not only aligned with the detailed software requirements but also set a high standard for code quality and robustness. By applying techniques such as parameterized tests, exception handling, and comprehensive coverage analysis, the resulting test suite provided immediate feedback and increased confidence in the application’s functionality. This experience reinforces my commitment to rigorous testing and continuous quality improvement in all future projects.