Aaron Bevans

CS 320 Project 2: summary and reflections report

For this project, I developed features that included corresponding classes and unit tests executed with JUnit. My testing methodology consisted of validating and testing functional and architectural specifications needed for the customers. Internal logic correctness was verified by utilizing white-box testing, as well as black-box positioning to confirm input and output limitations. I then developed each class to include necessary properties and methods to maintain and modify data as required.

ContactService was created with unit tests that confirmed each contact contained an assigned ID, while the system bypassed null and invalid datasets. Specific requirements were validated utilizing test methods, like testAddContactValidData() and testDeleteContactById(). TaskService class tests validated length of names and descriptions met specified requirements and were not null. Methods such as testUpdateTaskDescription() were designed to confirm updates were applied as required. The task ID was also designed to be immutable. Additional service classes contained tests that verified their purpose.

The AppointmentService class tests confirmed that its directive was fulfilled by using logic that created appointments with the required dates and times set to only present or future dates. Logic such as “assertTrue(futureDate.after(new Date())); assertThrows(IllegalArgumentException.class, () -> new Appointment("123", pastDate, "Routine checkup"));” aligned with requirements that scheduled events needed to occur on confirmed future dates. The software properties and logic were developed and tested to align with customer requirements.

Properties such as IDs, names and addresses met required character amount limits and were opposite of null. JUnit tests restricted test cases to be constrained to boundary and negative testing. Upper-bound violations were validated by attempting to apply exceeding lengths to contacts with stringent length requirements and reviewed correctly thrown errors. 20 character limits were applied to the TaskService and AppointmentService tests that successfully complied to their required constraints, every JUnit test was designed to conform to at least a single requirement to completely meet expected functional coverage.

Overall quality and test coverage were paramount throughout development, resulting in metrics above the threshold of 80%, and were seen at 85% test coverage. Both the nominal and edge cases were figured well above the highest percentage range. JaCoCo was used to analyze code coverage metrics, and to validate branches such as constructors, setters that were executed during tests. The quality and reliability of the tests were validated by the consistent assertion and coverage results. Utilizing the very best software testing tools resulted in well written code that functioned on industry standard levels.

Constructing quality systemic test designs was the result of learning the JUnit testing framework. Failed tests were the most optimal approach to eventually obtaining bug free code. Failed tests were initially used to validate and fulfill expected requirements before applying fixes. Implementing the correct logic before calling the function was key to successful execution. Parameterized tests were utilized to test various data complexes effectively. Reduced code duplication and enhanced efficiency were produced by the modularity of each test case. Each test was independent and repeatable due to the isolation of each method.

Unit testing and boundary values were analyzed, and where unit tests isolated each class and method to be tested independently using different system components. External APIs and databases were not relied upon, which made for controlled environments and utilizing data structures within memory. Field constraints were verified by implementing boundary value analysis by comparing maximum character limits for names and addresses. Limit enforcement was implemented by testing a contact with exactly ten characters for the first name and then with twelve characters. Integration testing was considered, but not included. Integration testing would combine services such as linking contacts to appointments to verify interoperability. UI and persistence layers would benefit greatly while utilizing integration testing.

Early-stage defect detection is most effective for unit testing, whereas integration and system tests are used for ensuring a holistic performance in larger software applications. This project reinforced disciplined requirement-filled software testing. Careful specification, structured JUnit design and validation iteration, then delivered successful requirement criteria with metrics that reflected effective system test design.

Maintaining a focused and analytical mindset allowed me to magnify the accuracy of how and what properties needed to be modified and data needed to be returned. I reminded myself of the fact that even small issues in software validation can lead to issues for users in the long run. Updates to ContactService were crucial in creating code interrelationships and affected method dependency for contact integrity, where each change needed revalidation. Regression testing was also important towards each code modification. This assignment helped strengthen my focus on software development correctness, and enlightened me on what it takes to create efficient and robust applications as a software engineer.