Aaron Shipley

CS 320: Project 2

Southern New Hampshire University

Joseph Rangitsch

06/16/2022

In terms of writing tests for the software that was developed in this assignment, the main focus was to align all of the code and tests with the requirements of the customer. As a software developer or tester, one should keep in mind the requirements set forth by the customer during the preparatory stages of the SDLC. The stage of requirements implementation will set the customer requirements going forward due to them being written down and properly documented for the project.

With this in mind, I first reviewed the Contact portion of the project. I focused on each part in an iterative manner, tackling each portion in iterations to ensure each specific part was to standards before moving to the next portion of the project. With the Contact class, it stated the class shall have five objects within the class; ID, firstName, lastName, phoneNum, and address. Each object had specific requirements, pertaining to the length of the string data stored within them. While developing each object, a series of if statements were created to set parameters for each object. An example of this is on lines 63 – 65 of the Contact.Java file. It stated as such: “***public boolean validateContactID (String contactID) {***

***if (contactID != null && !contactID.isEmpty() && contactID.length() <= 10) {***

***return true;”.*** This if statement addressed the requirements of the contact ID. The parameters of the contact ID state in order for the statement to be true, the contact ID must not be null and the length of the string must be equal to or less than 10. The parameters were then tested in the ContactTest.Java file, where JUNIT5 tests were set up to test the parameters for each of the objects’ if statement parameters. The contact ID parameters were tested using standard JUNIT5 annotations of @Test to call upon testing of the object. The inclusion of Assertions in the file was also a part of standard testing requirements as well. On line 24 of the ContactTest.Java file, the string literals were inserted as the valid results of the Contact class. One can see the string “1234” is inserted in place for the contact ID. The test then states the expected result of the test for the string, while then providing the output of the test, to ensure the result matched the expected result. The length of the string was tested as well as if the string was empty. The testing of validation against the parameters was in another test done on lines 129 – 136.

The overall quality of the JUNIT5 tests cannot be fully understood by the output of just a successful test. The success of the tests is subjective to the overall coverage of it. A test may still provide a successful output, but the coverage may be weak, effectively letting portions of the code not be properly tested for bugs. The overall quality of the tests performed on all three iterations of project 1 yielded results that are considered to be acceptable based on industry standards (Garcia, 2017). The minimum industry standard for acceptable coverage is 80% coverage, with acceptance ranging per company, project, or other factors.

In order to ensure my tests were effective for the class and the objects within it, I ran multiple coverage tests on the project in order to see the coverage for each file. Some file coverage was at first below the 80% mark. This lack of coverage made me revisit the lower coverage files to see where the gaps were. After addressing the gaps, the files had coverage of 81% or greater, hitting the 80% benchmark. Iterating the coverage tests along with the coding, allowed me to address issues in the immediate form, making for easier revision of the code to increase coverage.

Writing technically sound code is critical to the success of a project. By writing sound code, one can ensure the project will have fewer issues in terms of syntactical errors, ease of understanding from outside sources, and universal understanding of procedures followed. I ensured my coding was technically sound by following industry best practices and standards throughout the project. Notation of the code for others to follow my process was a major portion of this. Like customer requirements, coding has its own requirements, such as notation, which developers must follow in order to be successful. Another way to ensure the code was sound, was to ensure all syntax errors were devoid of the project. The proper usage of class, object, and statement rules was followed as well. One example of this is in the TeskService.Java file. Starting on line 7, the class of TaskService is declared