**Project Two: Summary and Reflections Report**

Ryan McFarland

Department of Computer Science, Southern New Hampshire University

CS-320 Software Test Automation and QA

Dr. Cross

October 17, 2021

The following concerns my recent assignment developing several services for a client mobile application. First, I will describe the approach I used to create an end product aligned with the requirements. For each service developed I was provided requirements that defined the operations and operational limits of each service. The provided requirements allowed my work to proceed quickly and development was focused on the sole goal of meeting the application requirements. For instance, the first service worked on was the Contact service. I was provided a list of five contact class requirements and three contact service requirements. The contact class required a contact object with a required contact ID string that could not be longer than ten characters and could not be updated. The contact object required name strings limited to ten characters which could not be null. The contact object also required a phone number string that is exactly ten digits and not null. The last requirement was an address field no longer than thirty characters and not null. After reading through the requirements, I was able to begin work on the contact class object by writing pseudocode. For instance, the first requirement was for an object with a unique ID that is not null or updatable. The pseudocode to guide the actual code began with “create contact object” and the next line could have been something as easy as “create contact ID with unique value that is not null or updatable.” After writing pseudocode for each requirement and fleshing out the class and class service the test development began. The test development was also guided by the project requirements. To test that I had truly created a contact object resistant to null fields, I would begin with pseudocode again by simply writing a line like “test that assigning null value to field produces error.” The pseudocode, nearly the requirements verbatim, and many times kept as a code comment, was fleshed out with actual code to create the objects under test and an attempt is made to assign illegal values or perform illegal operations like the following:

//test first name parameters, cant be null or more than 10 characters

@Test

void setFirstNameTest() {

Contact contact = new Contact();

contact.setFirstName(firstNameT);

assertAll("firstNameTest",

()-> assertEquals(firstNameT, contact.getFirstName()),

()-> assertThrows(IllegalArgumentException.class,() -> contact.setFirstName(null)),

()-> assertThrows(IllegalArgumentException.class,() -> contact.setFirstName(longFirstName))); }

To ensure that the Junit tests were effective based on the coverage percentage I attempted to create a class which would run all of the tests simultaneously and run this class as coverage. This did not work well for me because of my inexperience with this method, however, I was able to simply right click the package containing my code and run the package as coverage and obtain a result for code coverage with this method. To ensure that my tests did not skew the coverage results, because the coverage percentage also include the code in the tests, I simply subtracted the test code from the coverage result to determine that greater than eighty percent of my class code had in fact been exercised dynamically. To achieve technical soundness and efficiency I sometimes used print statements to understand which objects were being instantiated and in what order and I also learned to order tests using the order annotation like in the following example:

@Test

@DisplayName("Testing for appointment ID too long.")

@Order(1)

void testIDTooLong() {

Appointment appointment = new Appointment("This is an appt description.");

if(appointment.getApptID().length()>10) {

fail("Appointment ID is longer than 10 characters.");

}

}

Software testing techniques used in this project include a variety of both static and dynamic methods. Static testing included observing that denoting an atomic long used for ID generation as final precluded any subsequent attempt at altering the ID and satisfied this aspect of the requirements. Primarily, functional testing was used to align the project with the specifications and was the best choice because functional tests are, “also called specification-based testing: testing against a specification,” (2021). The tests were focused on determining if inputs were within the correct boundaries and that the structure of each element, whether a service or item, complied with the specification document. Another way to classify the techniques used in the project would be as a black-box technique versus a white-box technique. Black-box techniques are properly termed specification-based and white-box techniques are properly structure-based (2021). Structural testing, addressing the direct question of how much testing has been carried out, was also included in the project and a coverage report was developed (2021). Software testing techniques that were not used include non-functional testing such as comparing the system under test with a defined industry standard (2021).

During the course of this project my mindset was best described as cautious. I say this because at certain points, like attempting to implement a class that would run all tests, I did not have a solid understanding of scope and other concepts as it related to testing. In sum, I am a beginner with Junit testing so at times I had to resort to using rudimentary techniques like printing the contents of an object to make sure that the code I had written had accomplished its purpose. I tried to limit bias by doing things like including a test for a redundant ID although it was impossible to create one based on the way that I had designed the code. Although I was confident of the quality of my code, I still implemented tests that attempted to induce unexpected behavior to qualify my work. If no outside review of my code occurred, I would have been tempted to save time by not testing aspects of my code and it would increase the likelihood of having released defects in the project. It is important to be disciplined when attempting to implement an engineering process that is efficient and thoroughly vets the product from start to finish. As I mentioned, despite my confidence in having understood the implications of a final designation for a variable, it was important to test the code nonetheless. These sort of decisions to take the long way to complete a project properly, versus a shoot from the hip mentality, will guide my decisions well into the future. A final product aligned with specifications is only possible by strictly adhering to and applying specifications to future projects. A favorite technique I have learned and will include in any future development work is to use the requirements as a guide for creating pseudocode.

# References

Hambling, B. M., Samaroo, P., Thompson, A., & Williams, G. (2021). *Software Testing - An ISTQB-BCS Certified Tester Foundation Guide (3rd Edition)*. Retrieved from app.knovel.com: https://app.knovel.com/hotlink/pdf/id:kt00UC2GO3/software-testing-an-istqb/deciding-when-enough