Ashli Campbell

Southern New Hampshire University

CS 320

Project Two: Summary and Reflections Report

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.

There were multiple software requirements that I needed to align my code with; for example, for the contact service, the Contact class requirements included strings that could not be longer than 10 characters, could not be null and not updatable. The Contact service required the ability to add, update or delete contacts. The appointment service was similar in its requirements, where strings couldn’t be over a certain length and couldn’t be null, and the Task service required the option to add, delete or update tasks.

* + 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 made sure to test for each requirement to ensure that the JUnit tests were effective - striving for the goal of 100% coverage. There were tests for strings that were invalid, i.e. too long, that my tests caught.

* 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.

Technically sound code ensures that the syntax is correct and that there are tests for invalid input.

@Test

void testTask() {

Task task = new Task("123456789", "Homework", "4-1 Milestone");

assertTrue(task.getTaskID().equals("123456789"));

assertTrue(task.getName().equals("Homework"));

assertTrue(task.getDescription().equals("4-1 Milestone"));

}

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

One example of keeping my code efficient was my test for an ID of a length that was over the maximum length allowed:

@Test

void testTaskIDTooLong () {

Assertions.assertThrows(IllegalArgumentException.class,() -> {

new Task("123456789", "Homework", "4-1 Milestone");

});

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

The testing technique that I employed was black-box JUnit testing. Unit testing allows you to test small blocks of code, which helps you to keep your code free of bugs. When adding additional features it helps to check and make sure it’s still functioning as intended, which reduces the amount of time the programmer spends debugging. In milestone three, I employed feature testing to test object-to-object interactions.

The specification-based black-box technique is used to test specific cases when you know what your system should do, and makes use of things like equivalence partitioning, which groups similar inputs together to reject unwanted input.

* + 1. What are the **other software testing techniques** that you did not use for this project? Describe their characteristics using specific details.

I did not use state transition testing, which is a black box technique used to create test cases for possible events and changes in state of each input. The test is used to trigger changes in state to prevent invalid transitions.

* + 1. For each of the techniques you discussed, explain the **practical uses and implications** for different software development projects and situations.

State transition testing is best used when system features are expected to transform from one state to another. It also tests the dependency on values in the past. An example of a transition state is resetting the time at a certain time, or the date on a certain date. This type of testing is not always reliable, but it generally covers all possible conditions.

* 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.

It is important to appreciate the complexity and interrelationships of the code being tested because inadequate testing often leads to inadequate software. Without it, defects and errors may fly under the radar, and software is deployed that doesn’t fully meet customer expectations. This could even open customers up to security concerns, if for example the software manages credit card information or other sensitive data, which can ruin the company’s reputation and trust. There is also an example in our textbook of the electrical provider’s billing system change that caused the blackout of a major city (Hambling, et al, 2015). These are just two of many examples of harm that can be caused by software defects.

* + 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.

Bias in review of code can be risky. If you are over-confident in your code then you won’t be able to see and understand issues in code for what they are.

* + 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.

Cutting corners may lead to faster development and a quicker release of the product, but inadequate testing can lead to huge errors down the line. This can pose problems not only for the consumer’s use of the product, but if there is sensitive data or control then it can be a security concern. It is better to construct a realistic timeline for product release and conduct all necessary testing rather than skip testing to move the project along.