Michelle Hitchcock

Alex Petitt

CS 320

02/20/2022

Summary & Reflection Report

**Describe your unit testing approach for each of the three features. To what extent was your approach aligned to the software requirements? Support your claims with specific evidence.**

My approach aligned with the software requirements by including all of the outlined specifications.

I ensured that I carefully analyzed the requirements and created methods and classes that aligned with the naming conventions and functionality outlined in the requirements.

**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?**

In order to test the task and task service classes, I composed JUnit tests that would test each component of those classes as outlined in the requirements.

For example, I created a test case that checks to ensure that the task object is being populated, that the ID is random, and checks to ensure that the illegal arguments are being thrown as they should be.

These tests ensure that not only are the Task objects being populated, but that they are populated with the correct information. These tests also ensure that if something fails, the correct action will happen.

Executing tests that not only ensure things are working, but also check to see how they could fail, helps to provide testing coverage for as much of the project as possible.

**Describe your experience writing the JUnit tests. How did you ensure that your code was technically sound? Cite specific lines of code from your tests to illustrate.**

To ensure my code was technically sound I implemented module coding by using multiple concise files. I also made sure the code was logical and used methods where appropriate. Examples of this can be found on line 41 protected void setName(String name){…} and line 50 protected void setDescription(String taskDescription){...}

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

Efficiency was one of the harder things for this project. In order to do this, I reduced redundancy in my code and kept my style clear, using logical and descriptive variables such as taskObject, and taksId.

**Reflection**

**Testing Techniques**

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

I employed software testing techniques such as equivalence partitioning, the technique of dividing the given data up into valid and invalid parts so that further testing can be done. I used this type of testing technique when partitioning the variables into valid variables and those that were invalid aka – too long, and null.

Another technique I employed was Use case testing. Use case testing is a way of testing the functionality of the program, like testing from a user’s perspective. Some examples of this kind of testing that I implemented were the tests to ensure that all variables updated properly given the input and the tests to ensure the delete feature worked.

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

I did not employ error guessing techniques in this project, this technique requires putting in tests based on what you think might go wrong. “The main drawback of error guessing is its varying effectiveness, depending as it does on the experience of the tester” (Hambling et al., 2015) this drawback also impacts if the tester is able to implement this kind of testing at all.

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

All three of the techniques that I discussed are important to different development projects. Firstly, equivalences partitioning is important, not necessarily as a test itself, but to allow for future tests to operate more smoothly. Quickly establishing what variables are valid and invalid can help identify errors, for example, if a variable is not being updated.

Use case testing is important for the final product. This type of testing ensures that the tester thinks about the project's outcome and tests based on the requirements.

Error guessing can also be a very practical type of testing when employed by the right type of developer, or development team. This type of testing relies on prior knowledge which would be built up through many projects and therefore it could help solve issues that otherwise could easily have been missed.

**Mindset**

**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 was important to be cautious during this process. I ensured that I had followed all requirements and that I had implemented tests for all the aspects outlined in the requirements e.g. testUpdateAppointmentId() and updateLastNameTest() . I also tested the features that seemed to break the system such as, setTooLongDescriptionTest() and

testUpdateDate() {

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

appointment.updateDate(this.pastDate);…

These tests ensured that the functionality was thoroughly tested. Caution was taken to ensure none were missed as this could allow for requirements to not be met, unknown issues to arise, or the program to fail in an unexpected way. For example, if I had neglected to test if llegalArgumentException was being thrown when the date was past it could result in a number of issues, ranging from a past date being allowed to be used, to it not being displayed, to a system error. Although it likely was implemented correctly the tests ensure it not only is implemented correctly but is functioning properly.

**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 is something that is hard if not impossible to completely eliminate. I attempted to limit my own biases in a few ways such as testing every function. even if I was confident in its functionality, taking breaks when assessing this code, so as to not miss anything. I feel that although every effort is made to avoid bias, there is always the chance that something obvious can be missed due to personal familiarity with one's own code, and the biases inherent to self-review. For example, a software developer in charge of testing their own code might miss testing something that they forgot about needing to implement. Because they neglected to implement it, not testing it results in no issues and so, therefore, remains missing. Although it is always good to test your own work, is it also important to acknowledge personal bias and human error.

**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.**

Discipline is ultimately one of the most important factors in producing quality work. This is true for many fields but is especially true for software engineering. Faulty code, missing documentation, and cutting corners can have far-reaching and sometimes devastating real-world effects.

Avoiding technical debt is difficult, even for more experienced software engineers with “Six in 10 engineers say tech debt slows pace of development”(Torres & Torres, 2021). One of the methods that can be employed to help is working with integrity on projects and maintaining the discipline to know what projects to accept. Another important step is team communication, proper thorough testing to ensure that each step in the SDLC is conducted smoothly.

Citation:

Hambling, Brian Morgan, Peter Samaroo, Angelina Thompson, Geoff Williams, Peter. (2015). *Software Testing - An ISTQB-BCS Certified Tester Foundation Guide (3rd Edition) - 4.7.2 Exploratory Testing.* BCS The Chartered Institute for IT. Retrieved from https://app.knovel.com/hotlink/pdf/id:kt00UC2JK4/software-testing-an-istqb/exploratory-testing

Torres, R., & Torres, R. (2021, July 13). *Software engineers suffer from technical debt, too*. CIO Dive. https://www.ciodive.com/news/technical-debt-software-engineering-developers/603234/