Matthew Wilt

CS-320

Project Two – Summary and Reflection

[Matthew.wilt@snhu.edu](mailto:Matthew.wilt@snhu.edu)

*Summary*

The testing approach aligned to the software requirements because it tested the required

methods that were implemented in the code. In the task class, the requirements stated that the

first name and last name cannot be longer than 10 characters. Therefore, the Task Test JUnit

implemented the assertion:

public void testTaskID() {

String validTaskID = "987654321";

task.setTaskID(validTaskID);

assertEquals(validTaskID, task.getTaskID());

}

The requirements for the appointment class specified that the appointment id cannot be longer

than 10 characters. Therefore, in the Appointment Test JUnit, the following was used to

check if the appointment id is greater than 10 characters.

@Test

public void testApptId() {

String validApptId = "123456789";

appointment.setApptId(validApptId);

assertEquals(validApptId, appointment.getApptId());

}

The quality of the JUnit tests is slowly improving. The resources that were given to us in

the modules helped me a lot. The coverage percentage was higher in the appointment test than in

the task tests. However, in both functions, the service tests had a much higher coverage

percentage. After running the code with the Junit, it highlighted the lines of code that were

included in the coverage percentage. All of mine were above the 80% required. I intend to use

these more often to ensure that each statement is being implemented in at least one test case. I

also encountered some issues in my contact tests during the build. I figured it out after turning in

project one. My hypothesis is that there wasn’t a Junit4 or Junit5 folder installed into that

package. I am not aware of how to add it after the fact. It was intuitive in the other two packages

This has been my first introduction into testing software. I had to read and re-read, watch and re-

watch the resources. The more I practiced it the more it started making sense to me. I judged the

code to be technically sound by the coverage percentage. Once it met the criteria, I knew it was

as efficient as I could make it at this point in my learning.

*Reflection*

The software testing techniques that I employed for the milestones would be under the

black box and white box categories because they were all specification-based or structure-based.

Black box techniques include equivalence partitioning to test for both valid and invalid inputs,

decision tables to test conditions and actions, and state transition testing used to test events that

change the state or generate outputs, use cases that are made from test cases, and boundary

values to test boundaries. Structure-based testing was used a lot in the coverage tests to analyze

components and in the if then statements. It is used to break down the tests into sections to be

tested. Structure-based techniques include statement coverage, path coverage, and branch

coverage. Structure-based testing techniques “are used to explore system or component

structures at several levels” (Hambling et al., 2015). The testing techniques that I did not use for

the milestones are the experience-based techniques. According to Knovel, these techniques “use

the users’ and the testers’ experience to determine the most important areas of a system and to

exercise these areas in ways that are both consistent with expected use (and abuse) and likely to

be the sites of errors – this is where the experience comes in” (Hambling et al., 2015). The

specific techniques include error guessing and exploratory testing. Error guessing is where prior

experience is used to determine which tests would be best to analyze the code. Exploratory

testing is used to test areas that are lacking specifications. I omitted these techniques due to my

limited experience with testing. The techniques described above include black-box techniques,

white-box techniques, and experience-based techniques. Black-box techniques are used when

code segments have clear-cut functionality and is generally used for outsourced testing. White-

box techniques are used when the desired outcome of the product being tested is thoroughly

understood. Experience based techniques are to “identify special tests that may not be easy to

capture by the more formal techniques” are implemented “where specifications are either

missing or inadequate and where there is severe time pressure” (Hambling et al., 2015). Each of

these techniques are implemented based on their practical uses and implications for different

software development projects and situations.

I ensured that the code was written appropriately and tested the code accordingly.

Following the guidelines within the module, I knew I needed to be thorough with the classes and

the testing in Junit. I wanted to make sure I followed the rubrics guidelines. The mindset that I

adopted working on this project was analytical, experimental, and learning oriented. I

employed caution through extensive research and testing by watching tutorials and implementing

many trial and error runs. I attempted to limit bias in my review of the code by testing everything

multiple times. I could see how bias would be an issue if I were responsible for testing my own

code. For example, if I only tested the function which checked that the length of the first name

was no longer than 10 characters and chose not to test if the ID, I may not have caught the fact

that the ID test was not working. Testing multiple inputs such as both valid input and invalid

input versus just one or the other also helped to limit bias. It is important to be disciplined in the

quality of code as a software engineering professional because it helps you focus on creating

cleaner code. It is important not to cut corners to avoid compromising quality and performance

of the finished product. To avoid technical debt as a practitioner in the field, I plan to implement

agile development techniques which test code often, consistently push for high-quality,

demonstratable software production, and keep communication between the client and the

developers open. The agile methodology will help mitigate technical debt.

References

Hambling, Brian Morgan, Peter Samaroo, Angelina Thompson, Geoff Williams, Peter. (2015). *Software Testing - An ISTQB-BCS Certified Tester Foundation Guide (3rd Edition).* BCS The Chartered Institute for IT. Retrieved from  
 [https://app.knovel.com/hotlink/toc/id:kpSTAIST01/software-testing-an-istqb/software- testing-an-istqb](https://app.knovel.com/hotlink/toc/id:kpSTAIST01/software-testing-an-istqb/software-%09testing-an-istqb)