Project 2:

Reflection, summary and report of testing

Joshua Flores

SNHU

CS-320

The software requirements outlined in Brian Hambling et al.'s Software Testing Foundation were tested by following the rule of ensuring the large percentage of decision execution branches. Specifically, testing the branches between true and false conditions outlined in the JUnit coverage tab in the IDE that I used, to ensure that each unit met the requirements specified in the assignment. The test methods checked that requests adhered to certain standards, such as not allowing null values or character values of a certain length, and used assertThrows that made a differentiation between IllegalArgumentException and Error exceptions, to be able to know whether one branch versus the other was being used. Running these tests through the use of the debugger revealed that over 95% of the coverage quality was achieved, with the remaining 5% being covered in other @Test methods, such as the creation of Contact or Service Objects. The @Test method was designed to verify the creation of Contact.java and Service.class objects, followed by testing of getters, setters, and other methods to ensure quality code that met the project's requirements. To confirm that each branch was met per method in the classes, I followed all decision paths and made sure that the code was efficient by using boilerplate code for verifying the data being passed to the differing data fields being updated as an argument, thereby reducing redundancy of this boilerplate code and having to implement it multiple times for different data. For this reason, this helped reduce redundancy in allowing certain methods that worked on a variety of data to be tested once in contrast with it to be tested multiple times. For example, when updating the task field in TestService.java at lines 42 and 50, I passed the argument 'String field' to indicate the field being updated, which helped streamline the code. Overall, I am confident that my code is technically sound because I verified each branch for methods in the classes and ensured that they met the required standards by using asserts for the requested code quality, such as ensuring no null values. For instance, in lines 27 to 29 of my testVerifyTaskID code, I made sure that the IllegalArgumentException.class was thrown, as well as the Error.class, for the first two possibilities. Additionally, the last possibility was a boolean value that indicated whether the task ID met the required standards, and did not throw either error. By following all decision paths, I ensured that my code was robust and met the specified standards.   
 For this first summary, the testing approach was aligned with the software requirements outlined in Brian Hambling et al.'s Software Testing Foundation and was effective in ensuring that each unit met the requirements specified in the assignment.

Reflecting upon these tests, I focused on implementing a similar working code that was previously tested on my previous milestones, however, I did it by unit and did what I would consider Test-driven development since I made sure to write the tests before developing the code. I used a V-shaped model, which is the opposite of the typical waterfall model, where the code is developed first and then tested. In the TDD that I implemented I followed the V-shaped model that was in ways a reversed V-shaped model in developing the first part of the tests since I wrote the tests first and did the code after writing the tests, to ensure it passed the requirements of the tests. I also employed agile development by writing one piece of code, testing it, and then moving on to the next piece. This allowed me to successfully follow and develop as would be performed in an agile framework environment, to develop code quickly. In addition, I utilized Java collections libraries to enhance my skills.

Although I did not perform black box testing, it would have been useful to test the software from the user's perspective without knowing the inner workings of the program. The practical uses of the techniques I employed include adapting to different data types and writing modular programs with each unit working as intended. Test-driven development helped me learn more about the inner workings of Java libraries and the standard library and the necessities for developing efficient code. In reviewing the techniques, the development of software requires proper testing techniques to ensure that the code meets the standards set for it. Test-driven development is one such technique that ensures code is developed to meet requirements and that tests are run before the code is developed. By implementing test-driven development, I was able to develop efficient code that met the requirements and passed the tests. Additionally, I was able to learn more about the Java library and standard libraries that are necessary to develop good code. Overall, the techniques employed in this project can be applied to future projects to ensure the development of quality software that meets the required standards.

The mindset that I tried to adopt when developing the code for testing these software modules was particularly trying to look at a new way of testing something. Although, I initially looked to copy some of the code previously used I also made sure to find other possible ways to make sure something follows the standards of coding. Such as whenever I employed creating the software for each module in a different way such as making extensive use of collections in the second portion of the project. I tried to do this because although the standards of the code were particularly similar, I made it so that I could test the software in a way that might be tested on java's collections libraries, which could not be tested in the exact similar manner of implementing the other code. I tried to be cautious in thinking that the tests from previous tests would work the same on other the new portions of the project. This is important because simply copying and pasting might introduce a bug somewhere if you are not careful and may lead to an oversight of your code. It is important to realize this because if I chose to employ the testing methods for the TaskService.Java class and the other Service classes it would perhaps not work as implemented in the others because of the use of collections in the TaskService.java class. I had to go about checking the collection of objects in the TaskService differently than checking the other services.

Finally, it is important to be disciplined in writing your tests for the software that you are assigned, because of the simple fact that perhaps although you might use one function for example two percent of the time in the entirety of the software, it might still end up being the case that the function ends of being used one hundred percent of the time for some users of the program. For example, if you wrote a program to run various equations, in which the users would be mathematicians, they could all employ different equations, however, they would probably not all be the same. Ensuring that coverage is met is a good way to therefore keep your program sound. It is important to secure and fully commit to testing every unit in the program, whether it might be something as simple as ensuring that the program sets the name correctly. Because even though that function in the program might seem that it is too simple to mess up on, it might be the exact location that may cause an error that perhaps is not easily spotted; The program might still compile and have no compilation errors, however, there are use case errors. I will make sure to avoid this by testing the functions that I write in future programs to develop a habit of implementing a strong sense of testing within my programs.

References

Hambling, Brian, et al. *Software Testing: An ISTQB-BCS Certified Tester Foundation Guide*. Fourth, BCS, The Chartered Institute for IT, 2019.