Dillon Belanger

CS 320 SNHU

Project Two

10/15/2023

My testing approach was able to align with the software requirements, this was because of the methods which were used and implemented into the code. For example, within the contact class there was a requirement for first name and last name not being longer than 10 characters. For this there was a JUnit implementation to test for input of characters that were too long. Thus, aligning with the software requirements and maintaining the inputs to make sure they were correct. The task class specified that the id for the task could not be longer than 10 characters. So, the TaskTest JUnit had a method that was used to throw up arguments if the characters were greater than 10.

Quality was seen to be improving within the JUnit tests. This was helped by the continued development and included resources every module to improve the tests each week. This resulted in higher coverage percentages in the contact tests than previous task tests. There was also a significant improvement in the coverage percentage of the service tests. Because of these positive percentages, this shows more of the functions were being covered when testing. These can be used again in the future for the statements, maintaining the implementation within the test cases.

I was able to use arrays to ensure the code was correct and structurally sound. This allowed arrayed to be used for a list for the strings within the contact class, in tandem with algorithms for equals, add and length. Using length confirms and equal algorithms for the names and lengths for the services, this made sure they were all output correctly. There were also assertions used such as assertequals, asserture, asserthrow, and so on.

In order to ensure the code was efficient and developed correctly I referenced tutorials within the provided resources and error checked the code while developing it. Going back and forth with each error within the code to make sure it stayed correct through the development process. Techniques helped along the way, such as declaring variables and adding checks for ids to see if they were used prior to being added. Within contact service, it was tested for contact input and incorrect inputs. This was done using assertions and equal arguments to see which tested valid and which tested invalid.

For the milestones in each module, I used black box and white box testing. This is because all the techniques were specified and/or structure based. Black box techniques “derive test cases directly from the specification or some other kind of model of what the system should do. This was said by Knovel, explaining that black box techniques have equivalence petitioning for valid and invalid inputs. This also includes decision tables for conditions, and state transition for test events. Structure-based tests were used primarily for coverage testing to analyze the components. This puts the tests into sections which then get tested for better accuracy. These tests involve statement coverages, path coverage, and branch coverage. These are used to “explore system or component structures”.

Another technique that can be used is experience-based techniques. This “uses the user and tester 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 sides of errors – this is where the experiences come in” and determines the output. This type of testing was not used as much as the other techniques. This technique involves much error guessing and expanded tests. Having prior experience to determine which tests are best to analyze the code. This also includes tests which do not have specifications.

When completing this course, I tried to be experimental and improve on the code with each module. I attempted to use many references and tutorials to make sure I was doing the code correctly and minimizing my errors. In the beginning my coverage percentages were low but as the code was developed more the coverage percentage was increased more to above 80% which was the requirement. This was ensured by additional testing for high product quality.

I tried to limit bias with this review of all the code. This was done by having multiple tests and doing them multiple times even when the tests were correct and functioning. Rather than using assumption I tried to use intelligent guessing. This shows how bias can be a concern for code developed by yourself. If you test one part and not the other because of bias, you could miss faults in the code which you did not believe were there. Such as testing for character length but not id length which would result in the requirements not being met. Testing all the inputs validates the inputs and invalid inputs to confirm all the testing.

The quality of coding and the final product is important in softare development and engineering. It upholds the integrity and reputation of one's coding. Making sure the code is always done correctly and not half hazard is what preserves the quality of the product in the end. Pushing for more tests and quality, keeping communication strong between developers and clients, and agile methods all incorporate to maintain the technical sound.

**References**

Hambling, Brian Morgan, Peter Samaroo, Angelina Thompson, Geoff Williams, Peter. (2015). *Software Testing - An ISTQB-BCS Certified Tester Foundation Guide (3rd Edition) - 4.1.1.1 The Test Development Process (K3)*. BCS The Chartered Institute for IT.

Retrieved from <https://app.knovel.com/hotlink/pdf/id:kt00UC2IS4/software-testing-an-istqb/test-> development-process

*Software Engineering Code*. ACM Ethics. (2018, December 19). <https://ethics.acm.org/code-of-> ethics/software-engineering-code/