Project Two: Summary and Reflection

Benjamin Cleary

SNHU

CS-320 Software Testing, Automation and QA

Prof. Omar Toledo Lopez

Summary

My approach to ensuring that my code fulfilled the requirements and then tested to verify that alignment was fairly straightforward. When designing and writing the software, I had the requirements up on another screen so that, at any time, I could reference the actual requirements for the specific portion I was coding and once I had completed a section, I would look back and consider whether or not the code I built would fulfil the requirement. I did run into an issue earlier in the project where I was attempting to build code that could run some remediation if it didn’t receive proper data and still run instead of the code recognizing the bad data that was outside of the requirements and throw an exception. Once I cleared up my own confusion on what was expected from the code, I was able to remediate that section and more effectively code the subsequent parts of the project.

Once I had switched modes and began building the tests for the software, I still had the requirements up as I was writing the tests and breaking down the requirements into simple yes or no questions in terms of what would be acceptable and what would not. This translated well into the overall test coverage that the test cases achieved. With over 90% coverage achieved by the test case, which is well above the industry standard of approximately 80% coverage, I knew that my test cases were exercising and testing almost all possible outcomes and branches of logic embedded in the code and would properly operate in the case of correct data or return exceptions when in receipt of incorrect data.

The way the test cases’ code was written also played into the overall effectiveness. Including standard conventions for the naming of variables such as camel case and variable/method names that are descriptive enough to understand. Additionally, initializing test objects needed to test the variable service classes before the test method vice initializing a new test object for each test allowed for more code reuse and limited the lines of code to what was necessary without adding additional lines of code. To efficiently execute the tests, I tested that the code could execute and create an object with known good inputs. After that, I was able to test for each possible bad input and know that the bad input was, in fact, the variable that was being tested and could troubleshoot and correct the code quickly as there was no confusion on where the potential errors were.

Reflection

There were a number of testing techniques utilized when developing and executing the test cases. Decision testing was exercised to ensure that any type of if, or, and not statements executed correctly and that the expected output based on the decision point was correct both in the affirmative and negative case. This ensures that the code is properly executing the logic built into the program and exists in almost all programs and is a vital part of any project. Statement testing is vital to ensure that the various variable assignments and other operations worked correctly. Once again, this is in all programs and is vital to proper operation. State transition was used to ensure that the state of an object changed as expected. In the case of the services, there were a number of array lists that were expected to either add or remove entries and state transition testing ensured that the state of the list changed as expected. This plays a part in many projects in checking on changes to program objects, ensuring that an objects state changes as expected. Input partitions ensured that the correct types of inputs were accepted by a program and did not, for example, accept a string for an integer value as that would cause issues when the program tried to operate on that variable. This could cause a number of issues that might be difficult to track down or rectify in more complex projects. Boundary value analysis was used to ensure that variable values on the boundary, either just under or just above, are handled appropriately based on the over or under of the value. This can help prevent variable overflows or other unintended issues such as memory or storage overages which can affect the program’s efficiency and resource management.

There were also some testing techniques that were not used in the development of these test cases. Decision table testing takes into account a number of conditions and verifies that either an action or condition results from the sum of those initial conditions. This can affect more complex projects as many programs require more complex inputs and logic to properly function and cause confusing outputs if not properly addressed. Use case testing takes a specific use case and tests the success of completing or failing of that use. Many requirements can stem from one use case and might individually pass testing but once those interactions are compiled in one order or operations or another, they use case might not operate as expected. This issue starts to address the overall functioning of a project and can reveal systemic problems with it if the individual actions of the code are successful but due to either a design flaw or oversight can wreck the progress of a project.

During the development of this project, I had to be mindful of the way that the software code interacted so that I could properly test the software. In this case, the software was required to have a service change values on the individual contact or task object which meant that I would be writing code to use the service to change the value but then validating that the individual object held the correct value instead of validating something with the service changed. These kind of complexities have to be accounted for so that a test can actually verify that the software is functioning properly.

Another aspect that I had to consider was my own bias as the developer. With the requirement of not having a null variable, it was a conscious effort on my part to ensure that I actually tested that the value would not take a null assignment. I had to fight my assumption that a null value could not make it through the system or that there was a check for null value upstream from the code that was under test. That natural assumption of obviously invalid data not reaching my code is a bias that could have caused serious issues if it was not properly tested.

During the entire project, I had to remain disciplined in my approach to designing and implementing proper testing. Firstly, ensuring that I was thorough in my testing so that coverage was sufficient to have faith in the final code and that the code with operate as intended. Additionally, maintaining a solid testing regime will ensure that the basis of what the software will do with given inputs is understood and with the inevitable revision and update comes to the software, the software and test design will be able to adjust properly and reduce the time required to refactor and update the code and tests.