Patrick Knight

Southern New Hampshire University

CS – 320 Software Test Automation and QA

8/14/2021

Project Two

I began designing my unit tests by reviewing the requirements that were requested. To ensure that the requirements were met, I used them as a framework for my tests. The Task tests that I created would be an example of how I used the requirements in my testing. One of the requirements was for there to be a task description that had to be a String data type that could not be longer than 50 characters and could not be null. In my tests, I checked this requirement by creating instances of the task object with Strings that were less than 50 characters, more than 50 characters, and null. Text

Description automatically generated

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

To ensure that the tests were effective, I went and made tests that covered each requirement. If the requirement requested a string that could not be longer than 50 characters and not null, then I tested the software with strings that were null or more than 50 characters. I made sure that for each of the requirements there was a test that checked for each outlier. Also, I tested each function within the software to meet 100% code coverage.

For technically sound code, I made sure to follow the requirements that were laid out before me. I implemented tests that would check for the requirements by making calls to the different functions that were within the software. Within the code were mutator methods that would update the different fields. I made sure to test these methods to ensure that they adhered to the requirements.

Graphical user interface, text, application

Description automatically generated

I wanted to maintain efficiency when designing the code. The service classes required a way to store multiple versions of their objects which meant that I needed a storage container for these objects. However, I did not know the number of objects that would need to be stored at any given time. This resulted in my use of an array list for my storage. An array list can be resized at any time. Whenever the array list became full, I could simply resize it to store more objects. Graphical user interface, text

Description automatically generated

The software testing techniques that I used were static testing, statement testing, and function testing. Static testing is the process of reviewing the code that has been written without executing it. This typically involves a review process where members and non-members of the development team review the piece of code. (Hambling, Morgan, Samaroo, Thompson, & Williams, Static Testing, 2015) Statement testing is the process of testing the different lines of code for their ability to meet requirements. If a requirement asks for a string no longer than 50 characters, a test can check how the code reacts to a string that is longer than 50 characters. (Hambling, Morgan, Samaroo, Thompson, & Williams, Test Design Techniques, 2015) Function testing involves testing the various methods and functions of the software. A mutator method that updates a field would need to be tested to ensure that it still meets the requirements laid out. (Hambling, Morgan, Samaroo, Thompson, & Williams, Life Cycles, 2015)

The first software testing technique that I did not use was the method of decision testing. Decision testing involves testing the different decisions that are within the code. This would require testing for how the code reacts to a true and false decision. The next form of software testing was state transition testing. This form of the test involves how objects change within a system. (Hambling, Morgan, Samaroo, Thompson, & Williams, Test Design Techniques, 2015) The final form of testing not incorporated was the non-functional tests. Non-functional tests involve testing the performance and security of the software. (Hambling, Morgan, Samaroo, Thompson, & Williams, Life Cycles, 2015)

The different test methods listed above have practical uses. Static testing can be used to catch errors before the software is even running. This means that errors are caught and handle early in the project. Functional testing and statement testing checks the methods and individual lines of code for correctness. However, these two forms of testing can become time-consuming. State transition tests for how objects change within a system. These different modes would need to be tested to ensure they meet requirements. Each decision within a project needs to be tested for both true and false decisions. This is important because decisions within a project can affect other decision trees. Finally, non-functional testing is important because the software needs to be tested for security and performance.

Caution was taken into consideration during the process of testing. It was important since the services had relationships with the objects. The task service was able to create and modify task objects. When testing, I needed to make sure that these objects maintained their requirements. Also, caution was taken when implementing the Date object from the Utility library. I was inexperienced when using this object and had to use caution and research when testing the requirements. This also led to me creating my own validation method to check for dates that had passed.

Text

Description automatically generated Bias can affect the ability to see mistakes in our code. This can lead to errors making their way into production if we are not careful. My method for limiting bias is to view software that I have written as if it was written by someone else. When viewing someone else’s work, it can be easier to be objective. While I was writing and testing the contact and contact service, I made plenty of mistakes. I was not reviewing my code objectively and ended up leaving errors within. Since I did not test the deletion of a contact from a list of multiple contacts, I ended up missing an error that was in my original code.

Discipline is important when working as a software engineering professional. Letting errors and defects out into production can be detrimental to a company. These errors and defects can cost companies money, time, and customers. When cutting corners in testing, we can miss these errors and defects in the software. Reviews and testing throughout are a method to avoid technical debt. Errors and defects found later in development can lead to technical debt. Technical debt is the cost of refactoring code after it has already been written. (Technical Debt, n.d.)

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2015). Life Cycles. In *Software Testing - An ISTQB-BCS Certified Tester Foundation Guide* (pp. 37 - 59). BCS The Chartered Institute for IT.

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2015). Static Testing. In *Software Testing - An ISTQB-BCS Certified Tester Foundation Guide* (pp. 60 - 76). BCS The Chartered Institute for IT.

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2015). Test Design Techniques. In *Software Testing - An ISTQB-BCS Certified Tester Foundation Guide* (pp. 77 - 138). BCS The Chartered Institute for IT.

*Technical Debt*. (n.d.). Retrieved from ProductPlan: https://www.productplan.com/glossary/technical-debt/