Project 2

Randy Marcelino

Southern New Hampshire University

[CS-320-T3691 Software Test Automation & QA 22EW3](https://learn.snhu.edu/d2l/home/964285)

When creating tests the best way is to build them off of the requirements needed by the end-user. In light of that, I used the requirements given to me by the Grand Strand System to create the tests in the milestones. As an example In module 3, they wanted to test how to add and delete contacts so I made a test that tries to use the methods for adding and deleting from the ContactService code. And in module 4 the requested requirement was to update the name and description of a task which I created the test to try values that would break the method and see if an exception was thrown. I am fairly confident in the JUnit coverage of the contact and task services not only because it covers the bases that were set out by the company, but I made the testing strings make sure that they would break the program in one way or another to make sure that it is catching all possibilities set by the company.

When it comes to the code being technically sound you would want to look for code covering most if not all the bases for a test that could happen to the method. I would reference the method setPhone I made to set and update the phone number of a contact it goes as follows: String regex = "[0-9]+";

if (phone == null || phone.length() != PHONE\_LENGTH) {

throw new IllegalArgumentException("Phone number length invalid. Phone Number must be " + PHONE\_LENGTH + " digits and cannot be null.");

} else if (!phone.matches(regex)) {

throw new IllegalArgumentException("Phone number can only be numbers");

} else {

this.phone = phone;

}

This code does well to cover the needs of not having the phone number be too long or too short and not have letters in the number. On the other hand, when it comes to the efficiency of the code you could look for coding where things are done more efficiently such as code from any of my set classes for example:

if (address == null || address.length() > ADDRESS\_LENGTH) {

throw new IllegalArgumentException("Address cannot be longer than " + ADDRESS\_LENGTH + " characters or null");

} else {

this.address = address;

}

Where practically all the check statements for the string address are done in one line rather than multiple if-else statements. When creating tests for code you want to make sure that they cover the requirements and that they are coded efficiently.

Software testing techniques are a set of testing methods that can be used to ensure that the product is well tested and completes the functions that it was created to do. While I was creating the milestones I employed unit testing and acceptance testing. When implementing unit testing you check the code itself manually to make sure there is no error and that it is returning the results that you expected. As an example of unit testing, I created the addTask function and I manually ran the code to make sure that it created a task and added it to the list. The other type of testing that I conducted was acceptance testing which is when you test that the code actually completes what it was set out to do. As an example of this, I made sure to test the setting variable methods in milestones for the requirements such as cant be null and cannot exceed a certain length.

There are some testing methods that I did not use that would have been a good practice to have used such as security and performance testing. For security testing you would typically look that the program is safe and secure against attacks as an example you would include a hash checksum verification to make application secure for the customer data and information. The other testing method would be performance testing which is making sure that the speed of the program is not bogged by certain loops or algorithms.

I did not use these two testing methods simply because I did not need to for the milestones the security is for a more complete web application that would require the need to secure data, and for performance I had no big loops that would bog down the speed of the program that did not use them. They were more non functional testing the acceptance and unit are functional meaning they test more of how the code should work. Typically the functional requirements are what is tested first and prioritized simply because they are what is set out in the requirements and what is more important the non functional like speed and security can come after the program is functioning and doing what the user would want it to do.

While working on this project I set out to accomplish what was asked of me as a software tester. Now what I mean by this is when the client asked for the class to take in certain information and have them meet a set criteria such as not be null I accomplished just that. This may be considered as doing the bare minimum since there of course could be more scenarios to be cautious of but, I was trained in my previous employment to not create more than was requested because the client may not have wanted that or it may require more resources than are provided. I will say that through creating these tests I developed a deeper understanding of how the program should work and in a sense reassessed what the program was made to do. In the process also found a neat way to test the program for all its parts all at once, normally I would test every piece but with JUnit test I can test out all parts with the push of one button making it much more convenient.

When I was writing my test to make sure that I would not include bias in my work such as imparting data that would be preferential to my program passing I would create the tests specifically to look for what the client wanted. The data that I would pass to these tests, especially the ones that would cause failure, I made sure that they had data that would trigger one or more of the exceptions. But bias is almost impossible to avoid as an example of a type of bias “Confirmation bias is one of the most accepted ideas in psychology today. It is a type of bias that drives you to favor information that confirms your previously existing beliefs“. Which can appear in day to day life and in development, an example is to test for what you were told to and once it passes believe that it is complete which is the typical way to program. But making sure that other bases are covered when testing is very important for the code to be more secure especially when it deals with customer data which is sensitive.

When dealing with a system with customer data it is vital to be careful with what you make and touch because one false move can reveal all that data to the wrong person and cost you and your company to lose money and trust from clients. For this reason when you work in a corporate company that deals with private info they make you go through the sensitive data courses to teach you to be careful or they set up standards to be followed and require code reviews for anything to be pushed. When discipline and security of a program are tested is when the technical debt begins to mount which is “when development teams take actions to expedite the delivery of a piece of functionality or a project which later needs to be refactored. In other words, it’s the result of prioritizing speedy delivery over perfect code”. Amounting technical debt can result in disaster because a team will forgo some if not all the testing to meet a deadline which produces weak products in return with bugs and issues. I plan to give myself ample time to complete a project and make sure to keep an open communication with the client about the process.

**References**

Batchelor, K. (2020, June 15). *Council post: Addressing confirmation bias in software development*. Forbes. Retrieved February 16, 2022, from https://www.forbes.com/sites/forbestechcouncil/2020/06/16/addressing-confirmation-bias-in-software-development/?sh=41aa015d4d43

*Technical debt*. ProductPlan. (2021, September 2). Retrieved February 16, 2022, from https://www.productplan.com/glossary/technical-debt/#:~:text=Technical%20debt%20(also%20known%20as,speedy%20delivery%20over%20perfect%20code.