Project Two: Summary and Reflections Report

William Marshburn

Southern New Hampshire University

CS-320 Software Test, Automation and QA

Professor Pettit

April 3, 2022

**Summary**

All tests were based off the requirements given to complete the assignment. The requirements stated to develop contact and task service objects to be manipulated or updated within certain parameters. For examples, we were instructed to create a first name, last name, or just a name field that could only be a certain length. We were also tasked with allowing the objects to have a unique ID to which could not be updated. In the screenshot below, we utilized JUnit to test various inputs and we validated the requirements were met by having passed all the tests. Below are screenshots of just the taskService and task classes, results were the same for the contactService and contact classes.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Overall, I feel the quality of the testing was adequate. I tested multiple scenarios and incorporated additional test cases beyond just verifying a String did not go beyond a certain length (tests were included with one letter in a string to ensure this would pass and function since it fit in the requirements). In the screenshots above, all tests ran without failure indicating that there were no issues with the applications thus far.

I generally like to save and run the code frequently to ensure there is little chance for error to arise and if there is, I can go back and fix it. This becomes easier because I won’t have to sift through the whole application to fix a simple error that could have been caught earlier. One area that was generally difficult and took additional time but ended running error free was the validation and construction of the objects. I ensured technicality by verifying and creating a constructor that only creates an object that fit the requirements.

Graphical user interface, text, application

Description automatically generated

Building off my answer from the above paragraph, given my experience with code thus far, usually I would validate within the class that would contain the main method. After reviewing other techniques and sifting through the requirements, I changed things up and included validation in the class that is responsible for creating and constructing the appointment or task objects. This makes the code run efficiently because now, I do not have to include validation or error checking when working in the task or appointment service classes. This helps me organize my thoughts better and I can focus just on the methods to make the service classes operate correctly.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Reflection – Testing Techniques**

Throughout the completion of the milestone assignments, several testing techniques were used. Static, dynamic, unit, structural and white-box testing were the main techniques employed. Static testing is considered manual review of the actual code while developing it, say review of the lines of code following the completion of a method. Dynamic testing was performed via giving the program certain input or data to see if we could get certain output, one desires following completion of the program. At a high level, white-box testing is considered the testing of a system when the internals are known, so since the requirements were given, the developer technically knows how the program is supposed to work (Hambling, Morgan, Samaroo, Thompson, Williams, Peter, 2015). Finally, unit testing is the process of testing small units of the program, specifically testing each method to ensure a proper outcome, this was conducted with the Junit testing suite and getting a passing test for each unit helps one ensure development is going well (Hambling, et al., 2015).

There were several techniques that have not yet been implemented when completing the milestone assignments. Non-functional, black-box, integration, system, and acceptance testing were not used during development of the module milestones. Non-functional testing is the process of seeing how the program utilizes hardware resources, while the program is generally developed with this in mind, this was not actually tested. Furthermore, black-box testing is the opposite of white-box testing, when performing black-box testing the internals of the program are not known, which was not the case (Hambling, et al., 2015). Integration, system and acceptance testing are all beyond the range of unit testing and incorporate the larger system and other components. These techniques were not employed.

As projects expand and requirements change, different testing techniques may serve a certain purpose or using certain testing techniques over others may fit the bill. While one may consistently use certain techniques, think dynamic and static, there are other techniques that should be employed to ensure that the whole system is working correctly and is as bug free as possible. Errors might arise even if one may think that everything is properly accounted for and checked. For example, unit testing just encompasses small units and methods, integration testing begins to test modules containing several units and combining these modules for testing to make sure there are no errors when they are used together. Other techniques such as structural, functional and non-functional testing become very important as the program grows because we want to construct an efficient program, if the program does not run correctly and uses too much of the system resources, then it would be hard for the user to be satisfied. Lastly, it is important to note that testing should be frequent and consistent and in most situations be performed as early as possible. This helps development of the application is not slow or waste additional budgetary expenses later in the software development life cycle.

It is also important to note that it is very important to construct a program that both meets the requirements that the user provides and that the program is designed in a way that the user wishes the program to be utilized, think verification and validation (Hambling, et al., 2015). While we may code the program a certain way and we may develop certain aspects to our liking, it is always important to base the developed program on what the user is requesting. As one is doing this, testing is used to validate the creation, once a fully functioning application is passing all the tests following employed testing techniques, one moves a step closer to providing a quality application for the consumer.

**Reflection – Mindset**

A cautious mindset was adopted and used throughout the development of this application. Each step in the creation of this application was taken with the specified requirements and end user in mind. Developing the tests for this application was straight forward but became more difficult as the complexity increased. This is said because when one begins to work with multiple files and classes and each of the classes borrow from other class methods, one error or neglect of a certain relationship can cause problems when developing the application. It is always important to appreciate the complexity and relationships of the code because if that becomes neglected, problems can arise, and defects can occur. This can be illustrated via the creation of the methods that error check the inputs. For example, if there were no proper validation for an appointment being scheduled in a past date or if a description or name length was not validated this would cause a defect and the application then would not be meeting the requirements.

Bias is a concern, and a reduction or exclusion of bias was attempted throughout the development and testing of this application. One big cause of bias may be completely unintentional, what one person may deem as “common-sense”, non-complex or simple another may look over and not exactly understand what is happening or prefer to do things a different way. It is hard to have the same perspective someone else may have because one may simply not be able to look through that lens. Every scenario for testing was considered when creating the unit tests, this was relied heavily on to eliminate the chance of bias. It is important to have additional testing from other sources completed to fully eliminate bias but, in this case, this was not possible. When this scenario occurs, one way to shrink bias as much as possible is to test, test often, test multiple scenarios, and continue this pattern until the completion of the application. Coverage of Junit testing also helps with reducing bias, since developing unit tests help monitor and test outcomes, ensuring a certain percentage of coverage will further help eliminate bugs, reduce bias, ensure a certain depth of testing is achieved, and develop an adequately operable program.

Being disciplined is something that one can benefit from not only as a software engineering professional but in any profession. Without proper discipline, quality declines significantly. It is very important to not cut corners because this will increase the chance of errors, dissatisfaction, and ultimately cost time and money down the road. My favorite example of this, in a non-software related example, is painting a car. If one does not properly prep the car and account for all the factors going in to properly paint the car, then quality issues occur. Initially, the paint will be fine, but a few months down the road, the paint cracks and looks horrible. At this point the car winds up back in the shop and must be re-painted at the business’ expense, which in the software world increases technical debt. So, it is best to take the time, implement proper design, and testing standards while creating software. As I move forward, I will practice, test and code often while following current software development/engineering guidelines and best practices to ensure I continue to create practical, safe and quality software.

**References**

Hambling, Brian Morgan, Peter Samaroo, Angelina Thompson, Geoff Williams, Peter. (2015). *Software Testing - An ISTQB-BCS Certified Tester Foundation Guide (3rd Edition).* BCS The Chartered Institute for IT. Retrieved from  
https://app.knovel.com/hotlink/toc/id:kpSTAIST01/software-testing-an-istqb/software-testing-an-istqb