**Name: Fabian Weiand**

**Date: 2/21/21**

**IT 320 Module 7 – Project Two Submission**

**SUMMARY**

During the completion of the Contact Service, Task Service, and Appointment Service applications, specific requirements had to be met. These software requirements provided the basis the for the Junit test that were written. Each test was designed to ensure that a specific requirement was met. For example, the ID fields of the Contact, Task, and Appointment objects needed to consist of a String that holds no more than 10 characters and is also unique. From this simple requirement statement, we can extrapolate three specific tests. First, the field needs to be able to hold a string and cannot be empty. Second, the field should not allow string longer than 10 characters, Third, the values for the field cannot be repeated in subsequent objects. Three individual tests were built, one to ensure that the object can be stored and retrieved, another to ensure that id longer than 10 characters is rejected, and lastly that duplicate id throw exceptions.

Coverage percentage for testing is the amount of code in the system under test that is being used for any of the tests. Test coverage is especially useful to finding untested parts of our system under test. As a rule, a test coverage rate of 80% or above is considered reasonable (Garcia 2017). In our contacts and tasks application 96% of lines of code are covered which mean the Junit tests that were designed exercise almost the entirety of the system.

To ensure that the application was technically sound, a different test method for each requirement was created and the tests covered every aspect of the contacts and task class objects. For example in the testContactIDTooLong() test method creates an object and tries to pass it an invalid ID. It uses a Assertions.*assertThrows*() method from the JUnit library to assert that Illegal argument exceptions are throw with invalid input. If this test passes it can be assumed that the program can correctly ID’s over 10 characters long.

The efficiency of the application can be seen in the constructor for the contacts and task class. When the object is created the arguments passed to the constructor are validated using a series of if statements. For example this line of code that checks the taskID for the required input; if(taskID == null || taskID.length()>10). If specific arguments do not meet the requirements, then illegal argument exceptions are thrown and this means that no objects with invalid arguments are created in the first place making for overall more efficient code.

**REFLECTION**

In working on this project, we explored various testing techniques and their uses. In Module Three, Four, and Five we focused on JUnit testing and built unit tests for an application that manages contacts, tasks, appointments in java. Unit testing is a testing process where individual units of a software are tested. The purpose is to validate that each unit of the software performs as designed. For example, the contacts class of our application needed to be tested for several requirements such as id length, uniqueness, etc., and a specific test was written for each of these requirements.

Another testing technique that was used during these Modules is called Integration testing in which individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. In the case of our application, the individual tests were grouped into classes. These classes were then assimilated into a test package, which allowed for System Testing.

Some other software testing techniques that we did not use are Reviews and Acceptance Testing. A review is a systematic examination of a document by one or more people with the main aim of finding and removing errors (Hambling et al 2015). Acceptance testing is where a system is tested for acceptability to the end user. Each of the testing techniques described above may be more of less important given the circumstances. Test objectives, exit criteria, contractual obligations, available documentation, and limited time and/or budget are limited are all factors that affect the testing techniques that are emphasized. On top of these consideration a good testing technique helps minimize the risk of a defect while minimizing resource usage and cost.

The mindset I adopted during the completion of this project was to be thorough and specific. As a software tester it is important to write code meets the requirements set by the requirements document. Each test was designed to check specifically that the applications meet each requirement and functions as expected. It was important to be cautious especially when writing test for similar yet slightly different requirements. For example, the contact and task service requirements dealt exclusively with string objects so when writing the code and tests for the Appointment Service application a new variable had to be considered. This new Date variable need a different test than the others and in writing the test it was important not to miss this change.

Another important aspect of the mindset in approaching the development of this project was to stay unbiased. It is easy to write a test that passes, however, the principle behind testing is not to write a test that passes but to actively try and find defects. Since I acted as both the developer and the tester more intimate knowledge with the code might also lead to overseeing errors that might be obvious to non-developers.

Technical Debt is one of the major reasons behind adopting this mindset of being thorough, cautious and unbiased. This concept revolves around the idea that choosing an easier solution, in this case, not double checking that all the requirements were met, missing specific test parameters, or writing biased tests, would results in more time spend reworking the code later. Often taking the extra time to make sure your tests are accurate will mean finding errors sooner and avoiding the cost of a defect in your code later in the development cycle.

**REFERENCES**

García, B. (2017). Mastering Software Testing with JUnit 5. Packt Publishing

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