**7-2 Project Two: Summary and Reflections Report**

Anthony Vigil

CS-320: Software Test Automation & QA

Southern New Hampshire University

12/5/2022

The approach utilized was unit testing, where the features were evaluated according to the client’s requirements for the software. When working with the Contact Service testing, one of the requirements was for both the last name and first name to be no longer than 10 characters, otherwise it should throw an exception. On my first attempt, I revised the requirement but did not include it in the code. After revision, I implemented the requirement to throw an exception or invalid argument. The other requirement included the Appointment Service to have a date that is not set in the past. Unit testing ensured that the data would not be before the present day, utilizing the hint that was given in the guidelines such as Date java utility. For Task Service, the requirement asked for a unique ID to be created with additional values. Unit testing ensured the requirement was met by utilizing the unique ID utility. After all testing was done and functional, the coverage percentage was 87.3 percent. According to Kralj (2022), the accepted average percentage should be above 80 percent (MethodPoet). This confirms that the testing was able to establish the effectiveness of the software according to its requirements.

I have not experienced writing JUnit tests prior to this course or the projects so this was all fairly new to me but very interesting to learn about. The JUnit tests allowed me to learn how the code can be analyzed for its functionality and quality according to the client’s requirements. JUnit testing is a useful tool for many so I can see why companies use it to establish software. The code I created is organized and uses the correct structures to implement all the requirements. An example is shown below of the Appointment Service class:

**public** **void** delete(String appointmentID) {

Appointment temp = getAppointmentList().remove(appointmentID);

// If null, delete

**if** (temp != **null**) {

System.***out***.println("Deleted Appointment ID: " + appointmentID + ", Description: " + temp.getDescription());

}

// Throw exception

**else** {

**throw** **new** IllegalAccessError("delete is not possible");

}

}

The method used in the delete appointment ID uses an if-statement that will loop by first getting the list and removing the ID, then the statement checks if it is not null then delete the appointment. The else statement will throw an error and it is organized with proper spacing. Another example would be when checking if the description on Appointment Service is more than 50 characters. The code is shown below:

**void** CheckDescription\_Length\_More50() {

**try** {

*App* = **new** Appointment("App05", **new** ~~Date~~(2021, 01, 02),

"For Surgeon Doctor,For Surgeon Doctor,For Surgeon Doctor,For Surgeon Doctor,For Surgeon Doctor,For Surgeon Doctor");

}

**catch** (IllegalArgumentException ex) {

Assert.*assertEquals*(

"The description cannot be longer than 50 characters..", ex.getMessage());

}

}

The method does a try and catch where the new appointment is checked for the length of 50 characters. When it is more than 50 characters then it will throw an illegal argument exception with a note stating that the description is longer than 50 characters.

Testing the code and keeping the lines at a minimum allowed for maximum efficiency when implementing new appointments. Creating new appointments would add the appointmentID, appointmentDate, and the description. The code for it is like the one used when implementing the delete appointment statement but instead it is added. The code is shown below:

**public** **void** addAppointment(String appointmentID, Date appointmentDate, String description) {

// Adds

**if** (!getAppointmentList().containsKey(appointmentID)) {

getAppointmentList().put(appointmentID, **new** Appointment(appointmentID, appointmentDate, description));

}

// If already added, throw exception

**else** {

**throw** **new** IllegalAccessError(

"Appointment ID:" + appointmentID + " can't be added because it's already present");

}

}

The technique used for this project was the white-box testing technique. In *Black Box vs. White Box Testing*, it says “White-box testing is based on knowledge of the internal logic of an application’s code” (Ashtari 2022). It is used when trying to determine if the structure and logic are faulty, but since we know what the code is meant to do then we are certain about its functionality.

The method I did not utilize is the black-box testing method. In *Black Box vs. White Box Testing,* it says “Black-box testing uses only the specification to identify use cases, while white-box testing uses the program source code as the basis of test case identification” (Ashtari 2022). It uses code coverage, fault injection, and mutation testing. When using black-box testing, there is no knowledge of the structure so the output might not be analyzed. In this case, for the project, I knew the output, so this method was not utilized to help ensure the functionality.

When using the black-box testing method the testing team understands the requirements. It is then followed by analyzing and setting valid inputs for the predicted outputs. That information is then used to make test cases which check the functionality of the code. When using the white-box testing method, a specific section is tested and is executed to test the system. According to Boni Garcia (2017), he states “It determines if the program-code structure and logic is faulty” (p.23). All the testing will be recorded, and issues will come up to be fixed. White-box testing ensures that the code will be high in quality and integrity.

I set myself into a learner’s mindset for the project. I had no previous experience testing the quality and functionality of the code. That includes not having experience with creating java projects with JUnit testing, different packages, and utilizations. I practiced caution when modifying objects, testing and checking for errors. Any modification can result in the code not functioning as well as bad structure. Finishing the project helped me understand the importance of the testing and the complexity of the process. Each step is made to ensure functionality and patience is required to complete the tasks.

On the software developer side, I kept the code readable to make easier edits. I kept any bias towards the project limited to ensure the completion. From the previous milestones, I kept revising and analyzing the code that was completed to ensure efficiency. Working on this project and its entirety helped me focus on every aspect of the process while also being very involved in learning new material. Bias can play a role in determining when the code is effective enough. For example, a software developer can think after spending weeks on a code that is completely solid but can have flaws that were left unchecked due to the inconvenience the developer will feel.

Being disciplined and committed to providing the best quality for the software is very important due to it being used by many others for its functionality. It is important not to cut corners as it could cause a huge inconvenience or error to the users in the future. Catching errors early allows for more time to be invested in the perfection of the software. I plan to avoid technical debt by utilizing the best practices in coding such as data structures, comments, or organizational plans. This does not only benefit the company I am working for, but also creates a sense of reliability for myself by providing good work skills.

**References**

Ashtari, H. (2022, September 29). *Black box vs. white box testing: Understanding 3 key differences*. Black Box Testing vs. White Box Testing. Retrieved December 5, 2022, from https://www.spiceworks.com/tech/devops/articles/black-box-vs-white-box-testing/

Boni Garcia. (2017). *Mastering Software Testing with JUnit 5 : A Comprehensive, Hands-on Guide on Unit Testing Framework for Java Programming Language*. Packt Publishing.

Kralj, K. (2022, June 29). *Acceptable code coverage: Is 80% the ideal goal?* MethodPoet. Retrieved December 5, 2022, from https://methodpoet.com/acceptable-code-coverage/