Patrick Valencia

Software Test Automation & QA

Maylon Walker

12 December 2021

Project Two Reflection

For the overall project, I made sure that each and every aspect of the project aligned to the software requirements. I made sure to have them follow the requirements by making a list of the requirements, going through the list while creating said project, and making sure that each and every aspect was implemented or achieved on the list as I dived into the code. The main aspects that can be found in each of these requirements is that for contact service, I made sure that each contact object has a required unique contact ID String no longer than 10 characters. Additionally the contact ID cannot be null and cannot be updatable. As for the task service requirements, I made sure that each task will be added with a unique ID that is no longer than 10 characters. As well with the appointment services and class, each appointment will have a unique ID along with 10 characters.

As for the overall quality of the JUnit tests, when the project was completed, it had a coverage percentage of above 80%. This percentage is great due to it being above the generally accepted goal of 80% (Pittet). Such a high percentage leads to believe that the project does indeed cover most of the instructions laid out for it. Due to this we can tell that both the JU unit tests were effective based on the coverage percentage. As for my experience in writing the JUnit tests, it was an incredibly fun and rewarding experience. I made sure that the tests were technically sound by making sure they were structured well, and made sure that each section of the code flowed well and were efficient in the manner that they approached. For example in the code below:

public void updateAppointmentId(String id) {

if (id == null) {

throw new IllegalArgumentException("Appointment ID cannot be null.");

} else if (id.length() > APPOINTMENT\_ID\_LENGTH) {

throw new IllegalArgumentException("Appointment ID cannot exceed " +

APPOINTMENT\_ID\_LENGTH +

" characters.");

} else {

this.appointmentId = id;

}

}

This code for the appointments section of the program shows that it is both efficient as well as technically sound by the use of data structures as well as having proper syntax within the written code.

The software techniques that I primarily used for testing this project was white-box testing. This type of testing is primarily done by looking into the source code and is based on structural testing as well as logic-driven testing. While testing with the white-box testing approach, it is easier to help analyze interactions between different intercases and subsystems. It was also the best approach to testing due to it being easier to test since I was the developer for the project. The other testing methods that I did not use were the black-box and grey-box testing. Black-box testing takes the approach of the person testing having no knowledge of the program beforehand and figuring it out and seeing if there's anything wrong with the program. While this type of testing is useful in many different applications, it was not useful in my case as I was the developer for the project. As for grey-box testing, this one takes aspects of both white and black box testing into one. While it takes advantages of both while dropping the negatives of both white and black box testing, this type of testing was also impractical for my situation as I was the one testing as well as developing the program. (Smith)

For the mindset, I had to have a very open mind set in addition to a disciplined one. The open mindset involved being open to change as well as learning what needed to be done in a timely manner. As for the disciplined mindset, this one was one that involved keeping proper pace as well as making sure that each step in the development cycle was properly answered for. As for caution within this project as an acting software tester, I would say I applied a healthy dose of caution. The healthy dose of caution can be seen by the way each test was structured to account for what truly needed to be tested. In addition, each test was both efficient and had a purpose to fulfill. I tried to not be overly cautious as there is a healthy balance to where too much is not cost effective.

I tried to limit my bias in my code review by trying to be as impartial as I could. While difficult since I was both the developer and tester, I believe that I did a good job in discerning both sides. I can imagine that on the developer side there would be a great deal of bias as one would take pride in their work. This pride sometimes blind us towards ignoring small problems for the sake of pride or the feelings of accomplishment in a job well done. But once we are able to overcome these feelings we are able to truly be impartial and criticise our own work for the better of the overall project. While it is a difficult task it is imperative to always strive to be impartial as that's when the best of work is done. When it comes to being disciplined as a software engineering professional, one must be able to stay focused in addition to always being on task.

Having such a refined discipline will lead to an overall better foundation when developing or even just maintaining already existing code. As well, not cutting corners with writing or testing code helps tremendously in the long run. Starting off and continuing on the right foot is what leads to a premium experience as well as development process with applications and software creation. Additionally, if one does not properly develop and cuts corners in the beginning it could lead to a costly error down the road. So it's much better to have a great beginning foundation than it is to save time by cutting down on that foundation. I plan on avoiding technical debt as a practitioner in this field by being disciplined in the manner in which I do my work. I take pride in what I create and cutting corners to try and skip certain steps instead of doing it right in the beginning would negatively impact me more than help me.

Works Cited

Pittet, S. (n.d.). *Introduction to code coverage*. Atlassian. Retrieved December 12, 2021, from <https://www.atlassian.com/continuous-delivery/software-testing/code-coverage>.

Smith, A. (2020, May 15). *Difference between black-box, white-box, and grey-box testing - dzone performance*. dzone.com. Retrieved December 12, 2021, from <https://dzone.com/articles/difference-between-black-box-white-box-and-grey-bo>.