SWE6733 – Emerging Software Engineering Processes

Assignment 2

Dr. Reza Parizi

Group 2:

Matt Decker

Devika Kandel Neupane

Ifeanyi Okoye

Cameron Page

**Assignment 2 Final Methods and Final Thoughts**

The following is a breakdown of how group 2 worked through assignment 2. We will outline our methods and how we approached the problem as well as our final thoughts on mob programming.

**Project Github:** <https://github.com/Decker-Matthew-R/Group2-GildedRose-Refactoring-Kata>

**F-2-F Tools:** For our mob programming sessions, we decided to use Teams video chat with our group. We met and pushed changes back and forth using git. We felt in-person, face to face sessions would be more effective, but due to group members locations out of state and constraints with our schedules, Teams was the only viable option to complete the assignment.

**Complete the Gilded Rose Refactoring Kata:**

1. Our final product is on our group github: <https://github.com/Decker-Matthew-R/Group2-GildedRose-Refactoring-Kata>
2. We have added test cases to our program. These are also located in the github repo and these tests (screenshots and test coverage statistics) will be addressed in the **Test Case** portion of this write up.
3. Our code coverage is addressed in the **Test Case** portion of this write up.
4. See a screenshot of our unit tests in the **Test Case** portion of this write up. A screenshot is also located in our github repo.
5. We used code smells and test cases to refactor our code during our mob session. Our mob session can be viewed here: <https://youtu.be/SQqcKc8iVXY> A link is also posted on our github. We also have the refactored code on our github.
6. We also did a separate mob session to add updated features for the ConjuredItems, per the customer request. We have also included a test case for the conjured items. Here is a link to our second mob session: <https://youtu.be/SQqcKc8iVXY> A link is also on our github. Please see the **Test Driven Development** section of this write up to see how we used TDD to drive the implementation of our new feature.

**Test Cases/Testing**

We tested our code in our IDE (Visual Studio Code) using a series of test cases. You can see from our mob session, that we tested the code in small chunks as we were refactoring. Due to technical difficulties, code had to be passed around and tested by one person as we had issues setting up environment variables and Junit. This made testing the code a clumsy process. Our session would have been more efficient if we were all able to test, however the team overcame the technical difficulties and were still able to refactor code that passed all unit tests.

Below is a screenshot of our IDE showing all code passes:

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

As you can see, one test (the test meant to test the whole program) is still failing. I’m sure this is due to an assert.equal substring comparison not being exactly right of the gold.txt file provided by the original repo. This test should pass if all other tests pass. Also, when you compare the manual expected output of the gold.txt file and our actual output, you can see they are identical, thus the test is passing after a manual review of the output code and is just a red herring.

Below is a screen shot showing we achieved 100% code coverage using our test suite:

Graphical user interface

Description automatically generated with low confidence

Looking at our main source java file, we achieved 100% coverage on GildedRose.java. This showed we used a comprehensive test suite and were able to completely verify our refactor.

**Test Driven Development**

We refactored this code with Test Driven Development. Simply put, we developed tests that showed the original code passed these tests. We made sure that we had excellent code coverage (100% in this case). This ensured that as we refactored, we were still keeping the functionality and spirit of the original code. Using TDD ensured that as we performed our refactoring exercises and our code changed, the outputs did not change. This cemented the idea that our refactor was correct and going as intended. Once the code was refactored, we developed a test showing that conjured items degrade twice as fast. As we built our new method, we ran it against the test to ensure it was doing what we intended it to do.

**Final Thoughts**

This was a very good exercising in refactoring and mob programming. For our specific group, we found mob programming to be sort of clunky in terms of accomplishing work and this was due largely to our group having varying experience with refactoring, Java, and unit testing. These varying skillsets made our mob programming sessions long. As a group, our communication was excellent, and everyone contributed in different aspects to the assignment. We also found mob programming to be difficult given the length of time required to complete sessions and aligning everyone’s schedules. If we were doing this for our jobs, it would have likely been easier to find common blocks of time that could be shared for mob sessions, however given our job requirements, life requirements, and additional assignments/classes that we were juggling, it was difficult to find those common blocks of time to complete the assignment in a mob programming setting. Overall, this was a great opportunity to learn a new method of programming and sharpening some skills such as testing and