Individual Assignment 02
V.1.3

Part 1: Using JDeodorant to refactor design flaws

Description:

In the previous assignments, you were able to locate several design flaws using InFusion, in this assignment, were interested in fixing some of them. You will choose at least 2 design flaws instances, from 3 different design flaws types (6 in total). You will use JDeodorant to come up with potential refactoring operations to fix them. Since JDeodorant sometimes gives you many recommendations on how to fix the same flaw, based on your understanding of the symptoms of design problems, choose the necessary refactorings that might solve these problems. Finally, you will check whether the problems have been actually solved or not using InFusion.

Task:

For this exercise, you will need to use JDeodorant. You will have JDeodorant analyze one version of a JAVA software of your choice (you can use your previous assignment project if it is feasible). You will then be asked to make some refactoring decisions and report them and their impact. Follow these steps:

1. Install the IntelliJ plugin for JDeodorant or Eclipse plug-in for JDeodorant
2. Run JDeodorant on a project of your choice and select 2 instances of each of the following flaws types:
    1. God Class
    2. Feature Envy
    3. Long Method
3. Now, you can look at the refactoring recommendations by JDeodorant, choose which ones to be executed, keep refactoring until you process all your chosen instances.
4. Now use InFusion. Input your refactored project and double check if the instances that you have just refactored are seen as infected with the flaws.
5. Report your findings: Chosen flaw instances, chosen refactoring operations, refactoring results. (Provide screenshots as well).
6. Add to the report a concise comment about your experience with JDeodorant (positives, negatives and other comments).

Part 2: Using Refactoring Miner to detect refactorings

Description:

The purpose of this assignment is to understand how to detect refactorings and design improvements, performed by developers as part of their daily development activities. For this, we will use Refactoring Miner. Refactoring Miner is a library/API written in Java that can detect refactorings applied in the history of a Java project.

Task:

For this part, you will need to use Refactoring Miner. You can either run the mining tool to detect all refactoring in all the history of a given project. Or you can use the Chrome extension, which will help you detect refactroings for a given commit. You will have it analyze one version of a JAVA software of your choice (you can use your previous assignment project if it is feasible). You will then be asked to report the refactoring decisions, performed by developers, and then measure their impact using the Understand tool. Follow these steps:

1. Install the Refactoring Miner or Chrome extension
2. Run the tool on a project of your choice and select 1 commit preferably containing a large number of refactorings (more than 5).
3. Report the number and types of commits detected, explain the intent behind them if possible.
4. Now use Understand to analyze the impact of these refactorings. Input your project checked out before the commit and after the commit. Run Understand on the two checked out instances.
5. Check the CK metric values for the files associated with refactorings (and not the all the files in the project).
6. Analyze the impact of these refactorings on the metrics, and report your findings.

Part 3: Writing Test Cases Using JUnit

Description:

Learning objectives:

- Create unit tests and fix defects. Use JUnit for unit-level testing to confirm defects are fixed.
- Use EclEmma to measure the coverage of Unit Testing.

Tasks:

- Eclipse requires Java to be installed on your machine.
    - Recommended Java Version 8 (1.7.0 and above required).
    - Software installed on local computer:
- Import the CoffeeMaker Junit project in Eclipse.

- Write test cases for the classes Inventory, Recipe, RecipeBook. Make sure to achieve 100% coverage.
- There are 5 defects in the project source files, you're required to find, fix and submit any five (5) defects in project CoffeeMaker:
    - For each defect: explain in 1-2 sentences of why it was a defect and what you did to fix it.
        - Format:
            - Screenshot of failed unit test (caused by defect)
            - Screenshot of code which caused the defect (with defect shown)
            - Screenshot of code after being fixed
            - Screenshot of same unit test passing
    - Hint: code coverage does not mean that you tested all of the possible cases, it just means that you tested at least one case per method
- Example of what is considered a defect:
    - Unreachable code
    - Wrong/unexpected outputs or results that do not match the comments of the method
    - Code that does not handle edge cases (such as index out of bounds)
    - Anything else that you think is a defect (along with a valid argument is provided explaining why it is a defect)

Note: getters and toString() methods are only intended to be tested to get the desired coverage. One test method per each getter/toString() is sufficient.
Note: Methods are not the same as asserts. Each method should, ideally, have one assert.

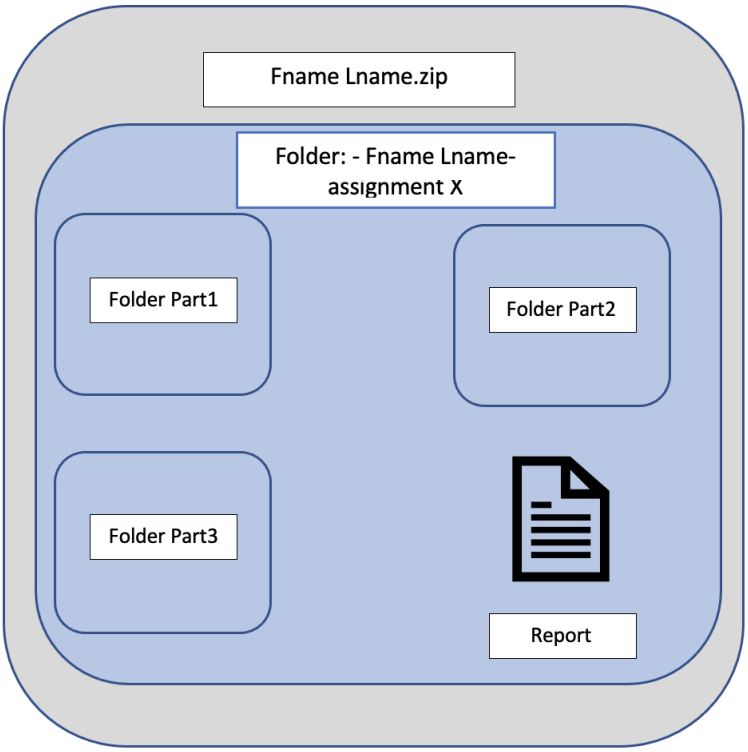
Grading:
Part1.
Fixing 6 design flaw instances from 3 different types - 30%
Part2.
Choosing a commit - 5%
Reporting refactorings and their impact on CK metrics- 15%
Part3.
Coverage per class:

- 100% 10%
- 80%-99% 5%
- <80% 0%

Errors (20%):

Submission:

Submit 1 pdf file containing your report for all 3 parts. For part 1, submit your corrected defects and your comments about the tool. You also need to provide a link to the source code of the project you studied. For Part 2, submit the chosen commit (screenshot), detected refactorings (screesnhots) and their impact on metrics (metric values before and after the refactorings). For part 3, it should contain your screenshots of defects and your explanation of their correction. Submit a screenshot of your coverages in the pdf as well. Attach your project source code after adding the test cases. The following figure explains how your submission should be structured:



Change Log

V.1.3 2022_02_03 Added the link to IntelliJ JDeodorant.

V.1.2 2021_02_22. Combining refactoring through tools with Unit testing.

V.1.0 2021_02_10. Initial verification by Instructor.