CS/SE 6356 Software Maintenance, Evolution & Re-engineering
Spring 2024
Assignment 4: Code Smells and Refactoring

Assignment date: **April 1**

1. Goals of the assignment

In this assignment, you will detect and analyze code smells and then, refactor some of them. The goal is to understand code bad smells, detection strategies, and detection tools, as well as refactoring.

Note: this is a team assignment. Communication with other teams is allowed, but only to address technical issues that you may encounter with the setup processes. You must explicitly document in the deliverable what help (if any) you received from other students/teams in the class.

Use the same repositories you created for Assignments #1 and #2, for the refactoring.

2. Software systems

We will use the same two systems from the previous assignments, which you should be familiar with by now: MangoDB and JEdit

3. Code smell detecting and refactoring tools

Below is a list of several tools that can be used to detect code smells and/or refactoring. Familiarize yourself with them (or at least a subset). Then pick one tool that you will use for this assignment.

NOTE: Feel free to look beyond these tools.

JDeodorant – An Eclipse plug-in that detects design problems in Java software, known as code smells, and recommends appropriate refactorings to resolve them.
https://github.com/tsantalis/JDeodorant

SonarQube - Open platform to manage code quality.
http://www.sonarqube.org/

CodePro Analytix - Software testing tool for Eclipse developers who are concerned about improving software quality
Decor (Ptidej) - Design smell detector
http://www.ptidej.net/tools/designsmells/

InsRefactor - Eclipse-plugin helping developers to identify code smells instantly and pushing developers to resolve code smells
https://liuhuigmail.github.io/tools/InsRefactor.htm

Lint - A code scanning tool that can help you to easily identify and correct problems with the structural quality of your code
http://developer.android.com/tools/debugging/improving-w-lint.html

RefactorIT - Automated refactorings, source-code metrics, audits and corrective actions
http://refactorit.sourceforge.net/

Stench Blossom - A code smell detector that provides an interactive ambient visualization
https://github.com/DeveloperLiberationFront/refactoring-tools

4. Identify and explain code smells

Use the code smells detecting tool you chose in the previous steps. Alternatively, you can detect smells manually, if you so wish. If you choose to detect smells manually, document what tools you tried, what you learned, and why you believed that manual suits better.

For each system (JEdit and Mango):

- Our goal is to detect at least 2 and at most 3 distinct **smell types** (e.g., Large Class, Refused Bequest, etc.). We want a total of 3 **instances of smelly classes/methods** detected by the tool (or manually). This means that for each smell type, we want at least 1 and at most 2 instances of smelly classes/methods.

  For example, you may choose to detect 2 instances of smelly classes/methods of the smell type: large class, and 1 instance of smelly classes/methods of the smell type: refused bequest. Or you may choose to detect 1 instance of each of the 3 different smell types. You may choose any smell types you wish, but they should be from at least 2 different categories (e.g., bloaters, couplers, etc.). At the end, the goal is to have 3 instances of smelly classes/methods for each system. Then, for each smelly code instance:

  1. Briefly describe the smell - i.e., the class, methods, attributes, etc. involved in the smell.
  2. Explain why the flagged class/method is smelly (be specific).
  3. Do you agree that the detected smell is an actual smell? Justify your answer.

5. Refactoring

In this part of the assignment, you will refactor the 3 smelly code instances from each software system (that is, 6 in total).

For each software system.

- For each smelly code instance, you identified and analyzed above:
  - Perform the necessary refactoring operations to remove the smell on the code component, either by using the refactoring options supported by your IDE, or by any of the refactoring tool you like, or manually.
  - Make sure the refactoring does not introduce any errors. Define some simple tests for that, for example.
  - Run the tool you used to detect code smells and check that the smell was removed. Also, see if your change introduced new smells related to the classes/methods refactored.
  - For each one of the smells removed:
    - List and describe in detail the refactoring (i.e., the code changes) used to remove the smell.
    - Give the rationale of the chosen refactoring operations.
    - Explain what code changes you had to do manually (if any), in addition of the changes performed with the IDE's support (or tool).

6. Deliverables

This is a team assignment, so only one submission per team is necessary.

There are two types of deliverables for this assignment:

- **Modified source code of the systems**. Push the changes to your repository for each system, in a branch called `a4_refactoring`.
- Submit a PDF document on eLearning containing:
  - List of team members and their contribution to the assignment.
  - The answers for section 4.
  - The answers for section 5.
  - Percentage of effort for each team member and description of their contributions.

7. Rubric

Item	Points
Refactoring completion	20
Answers to section 4	40
Answers to section 5	40
Total	**100**

You will receive points off if:

- You are not specific enough when describing your analysis.
- Your document contains typos and writing errors.