CPSC 4260 Refactoring and Software Design

Spring Quarter 2025

Individual Project – Make your code smell detector and refactoring tool!

Code Submission Due: 6/3/2025, 11:59 PM.

In this project, you will create a program – Code Smell Detector and Refactoring YOURSELF! Isn't it exciting?

First, your program should be able to detect the following code smells:

1. Long Method/Function
   In the context of this project, we define ***15 Lines of Code*** as the threshold for a long method/function. In other words, if a method/function contains 16 or more lines of code, we will say the code smell Long Method/Function exists. Please note that a blank line is NOT considered as one line of code.

2. Long Parameter List
   In the context of this homework, we define ***3 Parameters*** as the threshold for a long parameter list. In other words, if a method/function contains 4 or more parameters, we will say this method/function has a long parameter list.

3. Duplicated Code
   As we discussed in this class, there are different types of duplicated code. In the context of this homework, you will need to adopt a metrics-based approach. To be more specific, we will use **Jaccard Similarity.** DO NOT need to consider semantic duplicated code. We define **0.75** as the threshold.

Second, IF duplicated code is detected (in the context of our project, it means two similar code fragments executing the same functionality), your tool needs to be able to refactor the duplicated code.

Third, you need to provide a **graphical user interface** to perform all the operations mentioned above.

Operation Flow:
1. Start your program and a GUI is generated.

2. Users upload one single file through GUI.

3. Your tool analyzes the code in the file. Give users a prompt if any of the three code smells exist.

4. If duplicated code exists, give users a prompt asking them if they want to refactor the code. If users do that, your program needs to be able to "produce" a file containing the code after refactoring.

5. Users close the program through GUI.

Please notice:
1. You can use **ANY** language you want. You can implement **ANY** way you want.

2. You do **not** need to consider anonymous functions (such as Lambda function).

3. The duplicated code detection will **only** focus on function levels. For example, there are two same ***for*** loops inside the main function, but we will not identify these. The detection focuses on the comparison between different functions, but not the code inside one specific/main function.

```
int main() {
    int n, t = 10;
    for (int i = 1; i <= n; ++i) {
        if(i == 1) {
            cout << n << ", ";
     }
     for (int i = 1; i <= t; ++i) {
        if(t == 1) {
            cout << t << ", ";
     }
    return 0;
}
```

4. Lines of code calculation. Here are some examples of how you can calculate lines of code. In this project, as long as the line is not blank, we count it as one line. For example,

```
int main() { for (int i = 0; i < 10; i++) { cout << “Hi” << endl; }}
```

The number of LOC is: 1

```
int main() {

}
```

The number of LOC is: 2

```
int main()
{

}
```

The number of LOC is: 3

```
int main() {
    for (int i = 0; i < 10; i++) { cout << “Hi” << endl; }
}
```

The number of LOC is: 3

5. We will suppose all the code are self-documenting. In other words, you do **not** need to consider any comment code.

6. For the Long Parameter rule: if one of the parameters is an array or tuple, it should be considered as a single parameter only.

```
def function(x = [7,8,9,10,11]):
```

This function has only a single parameter, x. It is not a long parameter list (i.e., it has just one parameter, not five)

7. All the testing code will be valid. In other words, there will be **no** bugs in the testing code.

8. For any of the code smells, the use of API calls (such as OpenAI API, Gemini API, or any other AI tool) is not allowed. No credit will be given for this project is you use API calls.

Project Grading Breakdown

Your individual project weighs 18% of your final grade. In other words, 18 points out of 100 points. There are two grading parts for this project: mid-point check and final demo.

Individual Project Mid-Point Check (by 11:59 PM, May 20)

For your individual project mid-point check, each student needs to submit an essay to give a self-reflection on your individual project. The essay must follow the following format:

- **Fonts**: Your essay should be word processed in 12-point Times New Roman fonts.
- **Double space**: Your entire essay should be double spaced, with no single spacing anywhere and no extra spacing anywhere. There should not be extra spaces between paragraphs.
- **Heading**: In the upper left corner of the first page of your essay, you should type your name, your class, essay purpose, and the date, as follows:

  Your Name

  CPSC 4260

  Individual Project Self-reflection

  May 20, 2025
- **Margins**: Your essay should have a one-inch margin on the top, bottom, left, and right.
- **Align Left**: The text of your essay should be lined up evenly at the left margin but not at the right margin. In your word processor, choose "Align Left." Do not choose "Justify."

Your essay cannot be shorter than 2 page.

Breakdown	Points	Note
1. Your current progress	25	Briefly discuss your current progress toward the completion of the project.
2. Challenges	40	Briefly discuss what challenges you have met during your development practice
3. Solutions	40	Briefly discuss how you overcome the challenges you met.
4. Programming Language	5	At this point, you need to finalize the programming language in which you will build your refactoring tool. Please explicitly specify the programming language you are using for the application you are building.

Please note that this mid-point check weighs 3% of your final grade. In other words, if you get 100 based on the breakdown above, it will become 3 points in your final grade. If you get 50, it will become 1.5 points in your final grade.

Final Individual Project Submission (by 11:59 PM, June 3)

Please submit all your original code in a compressed format (.zip) on **Canvas**. Even if there is only one file, you must still submit it in a zip file on Canvas.

Additionally, you need to name your zip folder in the following manner:
`FirstName_LastName_IP.zip`

For example, if your name is Nayan Makwana, then your zip folder must be named:
`Nayan_Makwana_IP.zip`

All your code submissions will be screened based on AI-aided plagiarism tools. Confirmed plagiarism leads to an F in this class and an academic integrity report to our university.

Individual Project Demo

On June 4th and June 5th, each student will need to come to our grader to demo your project face-to-face. The grader will ask you questions based on your implementation.

A detailed demo schedule is uploaded on Canvas. Below is a grading breakdown for the demo. The location of the demo will be announced as soon as our library is open for reservation.

Breakdown	Points	Note
1. UI for the program	20	1. GUI is provided and it allows users to test your program.
2. Code Smell Detection	50	2.1 Long Function detection is accurate based on the threshold provided. 15 points 2.2 Long Parameter List detection is accurate based on the threshold provided. 15 points 2.3 Duplicated Code detection is accurate based on the metric you chose. 20 points
3. Code Smell Refactoring	30	3. Refactor duplicated code (structural duplicated code).

Please note that this demo weighs 15% of your final grade. In other words, if you get 100 based on the breakdown above, it will become 15 points in your final grade. If you get 50, it will become 7.5 points in your final grade.

Coding Suggestions

The code you write should be:

1. Clean Code:
    a. Readability: Write code that is easy to read and understand. Use meaningful variable and function names and follow a consistent coding style.
    b. Modularity: Break your code into small, modular functions or classes, each with a specific responsibility. This enhances maintainability and makes it easier to reason about each part of the code.
    c. Comments: Add comments where necessary but in a minimal manner, explaining complex logic or algorithms. However, keep in mind that the code should be self-explanatory, and comments should not state the obvious.

2. Code Smell:
    a. Avoiding Long Methods: Since your project involves detecting and refactoring long methods, ensure that your code adheres to this principle. Break down complex functionalities into smaller functions or classes.
    b. Parameter List Length: Similarly, be mindful of the length of your function parameter lists. If a function has too many parameters, consider refactoring or grouping related parameters into a data structure.
    c. Duplication: Ensure that your code is free from unnecessary duplication. If there are repeated blocks of code, consider creating functions or using other mechanisms to eliminate redundancy.

AVOID ANY CODE SMELSS AS MUCH AS POSSIBLE!
