BONUS

3E:

The method of employing specialised software tools to automatically rewrite the source code of a software system without altering its exterior behaviour is known as automatic refactoring. The objective of automatic refactoring is to enhance the code's design and maintainability while lowering the possibility of making mistakes or degrading the system's operation.

Automatic refactoring is typically performed by an automated tool that analyses the source code of a program and identifies code smells or anti-patterns that indicate areas of the code that could benefit from refactoring. In order to enhance the code's design and maintainability while maintaining the program's functionality and behaviour, the tool then automatically makes the appropriate changes to the code.

Examples of automatic refactoring techniques include renaming variables and methods to improve their clarity, extracting common code into separate methods or classes, simplifying complex conditionals or loops, and optimising code for performance. Automated refactoring tools can improve the quality of the software system while saving developers time and lowering the possibility of human error by automating these operations.

We are going to state an approach for automating the process of refactoring code according to the requirements of this project (i.e addressing design smells with consideration for code metrics). As refactoring requires altering the structure and behaviour of the code, automating the process can be difficult. Yet the following actions could be taken to automate the refactoring procedure:

- Define Design Smells and Code metrics: Identifying the design smells and code metrics you wish to solve is the first step. Code that is duplicated, has lengthy methods, has big classes, and other design smells are some examples. The same is true for the numerous code metrics that can be used to assess the quality of code, including cyclomatic complexity, code coverage, and maintainability index.
- Define Refactoring Patterns: One of the simplest approaches to automate refactoring is to create common patterns that can be recognised and refactored automatically. A recurrent structure or behaviour that may be automatically identified and changed is referred to as a refactoring pattern. A class that may be divided into two or more classes or a group of methods that can be combined into a single method is one such example of patterns. You can create algorithms to find and use these patterns once you've found them.
- Use a Refactoring Technique: The next step is to decide which refactoring approaches can be utilised to fix the discovered design smells. Among the refactoring approaches are move method, extract method, and extract class. Each method aims to enhance code quality and solve particular design smells.
- Continuous Evaluation/Testing: Doing automated tests before modifying the code is crucial to ensuring that it continues to function after refactoring. Automated tests can assist in finding any problems that may develop as a result of the code changes.

- Implement Automated Refactoring Tool: The next stage is to create an automated refactoring tool after establishing design smells, metrics, refactoring strategies, and developing tests. When design smells are found in the code, this tool can scan for them and then use the proper refactoring technique to make the code better.
- Continuous Integration and Delivery: It's crucial to incorporate refactoring into the continuous integration and delivery process to guarantee that the codebase remains well-refactored. This entails executing tests automatically, evaluating the quality of the code, and using appropriate refactoring techniques.

In short, automating the refactoring procedure can speed up the process and raise the standard of the codebase. You can develop a system that ensures code quality and preserves the structure of the codebase over time by defining patterns, using refactoring tools, utilising machine learning, and including refactoring as part of the continuous integration and delivery process.

Machine Learning to Refactor the code:

Now let us try to understand how to refactor the code using machine learning in detail:

First we need to analyse the source code which is like *Static analysis* that entails looking at a software application's source code without running it. Static analysis is used to find potential flaws in the code, including errors, inefficiencies, and security holes. We can utilise static analysis tools to analyse the code and find sections that need restructuring in order to automate the process. So, code smells, which are frequently seen signs of shoddy programming, can be found using static analysis tools. Duplicate code, lengthy methods, complex conditionals, and many other things are examples of code smells. Static analysis tools can make suggestions for altering the code to increase its quality by identifying certain code smells.

Now it is using machine learning; a neural network is one type of machine learning method that can be applied to code restructuring. A dataset of code samples and their associated refactored versions can be used to train a neural network. In order to generate suggestions for reworking new code, the neural network can learn the patterns in the refactored code. To make things more appropriate, we can use static analysis and machine learning approaches to automate the refactoring of code. Code smells and areas that need to be refactored can be found via static analysis. Based on patterns discovered from previously refactored code, machine learning techniques can be utilised to give recommendations for refactoring. The codebase can then automatically implement the changes. So, integrating static analysis and machine learning techniques can provide a powerful approach to automate the process of refactoring code. The method automates the laborious and error-prone operation of refactoring code, which can save time and enhance the quality of software development.

Static analysis and machine learning methods can be used to automate the reworking of code, although this method has certain drawbacks as well. Training machine learning algorithms on extensive codebases can be difficult, which is one constraint. Another drawback is that machine learning algorithms may not be able to fully understand the complexities of the code, and they may occasionally produce recommendations that are inappropriate. Therefore, it is essential to validate the recommendations generated by machine learning algorithms before applying them to the codebase.

In conclusion, Refactoring code can be automated to save time and boost software development quality. The method outlined in this method uses machine learning and static analysis to find places that need refactoring and produce refactoring recommendations. The combination of these methods can offer a potent method for automating the refactoring of code. Before implementing the recommendations produced by machine learning algorithms into the codebase, it is crucial to validate them.