ADDIS ABABA SCIENCE AND TECHNOLOGY UNIVERSITY

COLLEGE OF ENGINEERING

DEPARTMENT OF SOFTWARE ENGINEERING

Software Evolution and Maintenance

Project 1: Common Refactoring Methods

Group Member	ID
1. Selihom Demeke	Ets1159/13
2. Yihun Shikuri	Ets1317/13
3. Yodahe Ketema	Ets1328/13
4. Yordanos Seyoum	Ets1371/13
5. Yordanos Yirgu	Ets1374/13

1.Introduction to Refactoring

Refactoring is the process of systematically improving the internal structure of existing code without changing its external behavior. It enhances code readability, maintainability, and adaptability, making it easier to modify and extend while reducing the risk of introducing defects. Over the past 30 years, refactoring has evolved beyond simple code transformations to encompass architectural, model, and requirement-level improvements. This expansion reflects the growing complexity of software systems and the need for structured approaches to ensure long-term software quality.

According to Abid et al., refactoring research has focused on key areas such as determining optimal refactoring timing, recommending appropriate refactoring techniques, detecting opportunities for code improvement, and validating refactored code. Despite significant progress, the field remains fragmented across different domains, making it difficult to establish a unified framework. Their systematic literature review, which analyzes over 3,000 studies, highlights current trends and gaps in refactoring research, emphasizing the need for further exploration to enhance automation, scalability, and best practices in software maintenance.

2. Types of Refactoring and Their Characteristics

Refactoring techniques can be broadly classified into the following categories:

A) Extract Method Refactoring

Definition: Involves breaking down a long method into smaller, more manageable methods to enhance readability and maintainability.
Characteristics:

- Improves code modularity
- Enhances readability and reusability
- Reduces code duplication

Example: Splitting a method that performs multiple tasks into separate methods that handle each task individually [2].

B) Inline Method Refactoring

Definition: Replaces a method call with the method's content when the method is too simple and does not add clarity.
Characteristics:

- Simplifies code when unnecessary method abstraction exists
- Reduces method call overhead
- Enhances code comprehensibility

Example: Removing an unnecessary function and directly implementing its logic where called [3].

C) Move Method Refactoring

Definition: Moves a method to a more appropriate class to improve cohesion.
Characteristics:

- Enhances class organization
- Reduces dependencies and coupling

- Improves encapsulation

Example: Shifting a method from a utility class to a domain-specific class where it is more relevant [4].

D) Rename Method Refactoring

Definition: Changes the name of a method to make it more descriptive.
Characteristics:
- Enhances code readability
- Improves developer understanding
- No impact on code execution

Example: Changing "process()" to "processUserPayment()" for better clarity [5].
E) Pull-up and Push-down Methods

Definition:
Pull-up: Moves a method to a superclass to eliminate redundancy.
Push-down: Moves a method to a subclass when it is only relevant there.

Characteristics:
- Increases code reuse
- Strengthens class hierarchies
- Reduces duplication

Example: Extracting a method common to multiple sub-classes and placing it in a super-class [6].

3. Comparison and Contrast of Refactoring Techniques

Refactoring Type	Purpose	Benefits	Drawbacks
Extract Method	Breaks down large methods	Improves readability, reduces duplication	Can create excessive method calls
Inline Method	Replaces method calls with code	Reduces unnecessary abstraction	Can make code harder to modify
Move Method	Moves a method to a better-suited class	Increases cohesion, reduces coupling	Requires careful dependency handling
Rename Method	Changes method name for clarity	Enhances understandability	No direct impact on performance
Pull-up/Push-down	Adjusts method hierarchy	Encourages code reuse	Can disrupt existing class structures

4. Limitations of Refactoring

While refactoring enhances code quality, it has limitations, such as:

Increased Development Time: Frequent refactoring can delay new feature implementation.

Risk of Introducing Bugs: Without proper testing, refactoring may lead to unintended functionality changes.

Higher Maintenance Costs: Over-refactoring can lead to unnecessary complexity and maintenance overhead [7].

5. Best Refactoring Approach

The best refactoring technique depends on the software context. Extract Method is widely used due to its ability to improve modularity and maintainability, whereas Move Method is essential for reducing class dependencies. Automated refactoring tools and deep learning-based techniques are emerging to improve refactoring efficiency and accuracy [8].

6. Conclusion

Refactoring plays a crucial role in software engineering, improving software quality by enhancing maintainability, readability, and modularity. Choosing the right refactoring technique depends on project requirements, complexity, and code structure. While refactoring offers many benefits, it should be performed strategically to avoid unnecessary overhead and risk.

References

[1] C. Abid, M. Wiem Mkaouer, and M. L. W. Minku, "30 Years of Software Refactoring Research: A Systematic Literature Review," *arxiv.org*, 2022. [Online]. Available: https://arxiv.org/abs/2007.02194. [Accessed: Mar. 26, 2025].

[2] Y. Zhao, W. Wu, and Y. Fei, "An Architecture Refactoring Approach to Reducing Software Hierarchy Complexity," *J. Softw. Evol. Process*, vol. 35, no. 5, pp. 1-15, 2023.

[3] J. Oliveira, R. Gheyi, and L. Teixeira, "Towards a Better Understanding of the Mechanics of Refactoring Detection Tools," *Inf. Softw. Technol.*, vol. 151, no. 10, pp. 1-22, 2023.

[4] A. Nandini, R. Singh, and A. Rathee, "Code Smells and Refactoring: A Tertiary Systematic Literature Review," *Int. J. Syst. Eng.*, vol. 14, no. 1, pp. 45-62, 2024.

[5] P. Naik, S. Nelaballi, and V. Pusuluri, "Deep Learning-Based Code Refactoring: A Review of Current Knowledge," *J. Comput. Inf. Syst.*, vol. 63, no. 3, pp. 234-250, 2023.

[6] "Supporting Single Responsibility through Automated Extract Method Refactoring," *Empir. Softw. Eng.*, vol. 28, no. 12, pp. 1-18, 2023.

[7] "Impact of Refactoring on Software Maintainability: A Case Study Using Static Analysis Tools," *J. Softw. Evol. Process*, vol. 34, no. 9, pp. 1-20, 2022.

[8] "A Systematic Review of Machine Learning Techniques for Code Refactoring Recommendation," *J. Syst. Softw.*, vol. 192, no. 4, pp. 1-25, 2022.