ADDIS ABABA SCIENCE AND TECHNOLOGY UNIVERSITY

COLLEGE OF ENGINEERING

DEPARTMENT OF SOFTWARE ENGINEERING

COURSE: Software Evolution and Maintenance

Title: Research Paper Review on Different Code Refactoring Methods

Group Members Name	ID
1. Melat Zenebe	ETS0826/13
2. Mariamawit Chala	ETS0800/13
3. Lelena Damtew	ETS0750/13
4. Mahlet Seyoum	ETS0795/13
5. Miheret Berhanu	ETS0872/13
6. Mahlet Gizew	ETS0793/13

Submitted to: Mr. Dereje

Submission Date: April 2, 2025

Introduction

Refactoring is the process of reorganization of code that is present already without altering its external behavior. Refactoring is significant software engineering practice that enhances code performance, readability and maintainability. Effective refactoring practices improve software security, scalability and responsiveness to change, hence refactoring as critical aspect of software evolution and maintenance. This paper discusses well known refactoring practices based on existing research to uncover their impact on software quality and security.

General Refactoring Techniques

A few general refactoring techniques have been seen and evaluated by recent research studies. Those refactoring techniques usually focus on lessening the complexity of th e code, avoiding smells in code, and promoting modularity.

1.Extract Class and Extract Superclass

When a class performs multiple responsibilities, it can be refactored using the Extract Class method. This technique moves related attributes and methods into a new class, improving separation of concerns. Similarly, Extract Superclass creates a new superclass to accommodate shared behavior among related classes, promoting code reuse and maintainability [1].

2. Consolidate Duplicate Conditional Fragments

The Consolidate Duplicate Conditional Fragments refactoring technique focuses on identifying and merging repeated code segments that appear within multiple conditional branches of a program. By consolidating these duplicate fragments into a single statement, developers can significantly reduce code duplication, which not only streamlines the codebase but also minimizes the risk of inconsistencies and errors that may arise from having multiple copies of the same logic[1]

3.Move Method

The Move Method refactoring technique is applied when a method is more relevant to another class than the one it currently resides in. By moving the method to the appropriate class, it enhances encapsulation and cohesion, leading to a more intuitive object-oriented design [2].

4.Inline Method

Inline Method is the inverse of Extract Method, used when a method's body is as clear as its name, making the method redundant. This technique reduces unnecessary method calls and simplifies the codebase, particularly in cases where excessive abstraction impairs readability[1].

5.Method Encapsulation

Method Encapsulation is a refactoring technique that involves restricting direct access to certain methods within a class, ensuring that they can only be accessed in a controlled manner. This improves **data security, maintainability, and modularity** by enforcing clear boundaries between different components of a program. By using access modifiers like **private, protected, or public**, developers can control how methods interact with other parts of the system, reducing unintended side effects and improving overall code reliability [3].

6.Input Validation

Input Validation is a security-focused refactoring technique that ensures user inputs are properly checked before being processed. By validating input data, developers can **prevent security vulnerabilities** such as **SQL injection, cross-site scripting (XSS), and buffer overflow attacks**. This technique involves enforcing constraints on user inputs, such as **type checking, length restrictions, and pattern matching**, to maintain data integrity and application security. Proper input validation not only enhances security but also prevents unexpected system failures caused by invalid data [3]

7.Code Simplification

Code Simplification is a refactoring method that focuses on reducing unnecessary complexity in the codebase. It involves **removing redundant code, eliminating**

duplicate logic, and restructuring convoluted code patterns to enhance readability and maintainability. This technique helps developers quickly understand and modify code, reducing technical debt and improving performance. Common approaches to code simplification include **removing dead code, replacing complex conditionals with simpler logic, and using meaningful variable names** [3].

8.Pull Up Method and Push Down Method

The Pull Up Method refactoring technique consolidates similar methods from multiple subclasses into a single method in the superclass, reducing redundancy and promoting code reuse. In contrast, the Push Down Method moves unique methods from a superclass to its subclasses, minimizing unnecessary coupling and ensuring that each subclass contains only relevant methods. Together, these techniques enhance code organization, clarity, and maintainability within the class hierarchy.[7]

9. Rename Method

The Rename Method refactoring technique involves changing the name of a method to more accurately convey its functionality and intent, thereby enhancing the overall clarity of the code. By selecting a name that better describes what the method does, developers can make the codebase more intuitive and easier to navigate for themselves and others who may work on the code in the future.[8]

10.Composite Refactorings

Brito et al. (2023) introduce the concept of composite refactorings, which involve applying multiple refactoring techniques in sequence to achieve a more substantial code improvement. Unlike single-step refactoring, composite refactoring is essential for handling complex code smells and large-scale code restructuring.[6]

Conclusion

Refactoring is a vital practice in software engineering that enhances code quality, security, and maintainability. Various refactoring techniques, such as Extract Method, Move Method, and AI-driven refactoring, contribute to improved software design and performance. Emerging technologies, including deep learning and reinforcement learning, further automate and optimize the refactoring process. While these advancements offer significant benefits, challenges such as computational costs and data quality must be addressed. Overall, effective refactoring strategies are crucial for ensuring scalable, secure, and efficient software systems.

References

[1]. M. Waseem, M. Usman, and A. Ahmad, "A Systematic Literature Review on Software-Refactoring Techniques," VFAST Transactions on Software Engineering, 2023.

[2]. Trovato et al., "An Empirical Study on the Code Refactoring Capability of Large Language Models," arXiv preprint, 2024.

[3]. M. A. Babar et al., "Refactoring Codes to Improve Software Security Requirements," Procedia Computer Science, 2022.

[4]. R. Polu, "AI-Driven Automatic Code Refactoring for Performance Optimization," *International Journal of Science and Research (IJSR)*, vol. 14, no. 1, pp. 1316-1320, Jan. 2025. doi: 10.21275/SR25011114610.

[5]. B. Nyirongo, Y. Jiang, H. Jiang, and H. Liu, "A Survey of Deep Learning Based Software Refactoring," *Beijing Institute of Technology, Peking University, Dalian University of Technology, China*, Oct. 2023.

[6]. A. Brito, A. Hora, and M. T. Valente, "Towards a Catalog of Composite Refactorings," Journal of Software: Evolution and Process, vol. 1, pp. 1–22, 2023.

[7]. K. Tsybulka, "Enhancing Code Quality through Automated Refactoring Techniques," Master's Thesis, European Master in Software Engineering, Universidad Politécnica de Madrid, June 2024.

[8]. O. Tiwari and V. Yadav, "A Review of Methods for Identifying Extract Method Refactoring," in *Proceedings of the 2023 International Conference on Software Engineering and Knowledge Engineering (SEKE)*, 2023, pp. 1-10.