The Crown Oracle and the Thirty Code Change Patterns: An Empirical Study of Method-Level Refactorings

Emanuela Calota[1], Lina Ochoa[1], Önder Babur[2,1], Nan Yang[3,1], Görkem Giray[1] and Mark van den Brand[1]

[1]Eindhoven University of Technology, Eindhoven, The Netherlands
[2]Wageningen University and Research, The Netherlands
[3]TNO-ESI, The Netherlands

Abstract

Refactoring is recognised as an important software maintenance task that supports software improvement and development productivity. Given its relevance, this practice has been the subject of empirical study for decades: research has focused on understanding developers' intent behind refactoring, as well as the impact of refactoring on software quality. However, there remains a gap in connecting different refactoring types to the code changes that are actually introduced when they are applied in practice. To address this gap, our work is directed towards analysing code change patterns related to refactoring types in open-source repositories. We, therefore, have two main research objectives: First, we aim to construct a balanced dataset of method-level refactoring instances mined from Java open-source repositories. The result is the Crown Oracle, an automatically collected and manually validated Java dataset of 2,681 refactoring instances from 1,528 GitHub repositories, containing seven method-level refactoring types. Second, we aim to analyse the previously collected refactoring instances and identify code change patterns associated with each method-level refactoring type. For that, we followed a mixed-method approach, clustering recurring patterns of code changes and performing open coding on the obtained clusters to connect them back to code change patterns. This produces a catalogue of 30 code-change patterns spanning seven method-level refactoring types, giving practitioners insight into how refactorings help realise particular programming practices, and providing tool developers with a dataset of real-world refactorings and the code changes that co-occur with common refactorings. This work has been submitted to the Empirical Software Engineering journal, and it is currently under review.

Keywords

refactorings, code change patterns, mining software repositories, empirical software engineering, dataset

1. Motivation

Refactoring improves internal software structure while preserving external behaviour [1]. Empirical research has focused on understanding the refactoring process, the reasons behind refactoring, the impact of refactorings on software quality, among others. However, to the best of our knowledge, little attention has been paid to how these refactoring types co-occur with concrete code changes practitioners perform. We therefore conducted a study addressing this gap, which has been submitted to the *Empirical Software Engineering* journal and it is currently under review.[1] The overarching question of this study is as follows: *what are recurring patterns of code changes in refactorings?* The question is operationalised by considering two research objectives: **RO1** to construct a balanced dataset of method-level refactoring instances mined from Java open-source repositories; and **RO2** to analyse those instances and identify the code change patterns associated with each type. **RO1** is addressed by the creation of the Crown Oracle, while **RO2** yields a catalogue of 30 code change patterns. Both contributions are described below.

BENEVOL'26: The 25th Belgium-Netherlands Software Evolution Workshop, November 26–27, 2026, Madrid, Spain
m.e.calota@tue.nl (E. Calota); l.m.ochoa.venegas@tue.nl (L. Ochoa); onder.babur@wur.nl (Ö. Babur); nan.yang@tno.nl (N. Yang); g.giray@tue.nl (G. Giray); m.g.j.v.d.brand@tue.nl (M. v. d. Brand)
© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

[1]Preprint available at https://research.tue.nl/nl/publications/an-empirical-study-of-code-change-patterns-in-method-level-refact/

2. The Crown Oracle

To construct the dataset, we first select repositories following prior mining guidelines [2]. We then apply RefactoringMiner 3 to detect and label refactorings, keeping commits with a single refactoring instance. We also performed stratified sampling to obtain a balanced and diverse dataset. Finally, the whole dataset is manually validated. In the end, the Crown Oracle contains 2,681 manually validated refactoring instances spanning seven method-level types (i.e., *add parameter*, *change method access modifier*, *change method return type*, *extract method*, *remove parameter*, *rename method*, and *rename parameter*). The refactoring instances are drawn from 1,528 Java GitHub repositories.

The dataset addresses three known limitations of prior refactoring datasets, namely entanglement of multiple refactorings per commit making testing convoluted; imbalance across refactoring types; and unreliable validation. We believe that the Crown Oracle can serve both as a basis for studying refactoring practice and as an oracle for evaluating refactoring detection tools.

3. The Thirty Code Change Patterns

Building atop the Crown Oracle, we follow a mixed-method design. We compute AST edit scripts between the pre- and post-refactored code using GumTree. We then perform consensus clustering using diverse methods. These artefacts then inform the reviewers, who inductively derive patterns per refactoring type. Each instance is reviewed by at least two reviewers and labels are kept stable after alignment discussions. A final manual labelling step assigns every instance to one or more patterns. The analysis yields 30 code change patterns: 28 specific to individual refactoring types and two that recur across types. For example, *add parameter* commonly co-occurs with introducing a new field, establishing a new interaction, or adding guards, suggesting that parameter additions propagate through the surrounding code.

The contributions of the catalogue are twofold: First, it informs context-aware refactoring detection tools either by (1) providing information of co-occurring changes in floss refactoring instances per refactoring type—e.g., *add parameter* can be accompanied by the *insertion of a new class field* whose state is directly modified by the parameter; or by (2) illustrating concrete instantiations of the refactoring mechanics that are to be considered during detection—e.g., *change method return type* can appear when developers *modify type annotations* of the returned type. Second, the catalogue can also inform practitioners about the recurrent scenarios to apply refactorings. This extends the rationale provided by Fowler in his experience-based catalogue [1]. E.g., the *extract method* refactoring is often used with the *guards and guarding* pattern; that is, part of the logic of the refactored method is extracted to be used later in the context of a Boolean expression of an existing or newly created control-flow statement. The extracted method acts directly as a guard.

Declaration on Generative AI

During preparation, the author(s) used Claude Opus 4.8 and Writefull for grammar, spelling, and editing support. The authors take full responsibility for the publication's content.

References

[1] M. Fowler, K. Beck, J. Brant, W. Opdyke, D. Roberts, Refactoring: Improving the design of existing code, Addison-Wesley, 2012.

[2] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, D. Damian, The promises and perils of mining GitHub, in: Proceedings of the 11th Working Conference on Mining Software Repositories, Association for Computing Machinery, New York, NY, USA, 2014, pp. 92–101. doi:10.1145/2597073.2597074.