Ann Casey

Title: ***Software refactoring side effects***

Authors: Amjad AbuHassan, Mohammad Alshayeb, Lahouari Ghouti
Journal: ***Electronics***
Publication Date: 28 October 2021
URL: https://onlinelibrary.wiley.com/doi/full/10.1002/smr.2401?casa_token=nP7pJCCK1G0AAAAA%3AF_uJ8Fg8v1NAME0Sa_mkH31IafYB4qTvFwtwePUR0-UCsLfE678zoR4ozpjT2e-42ptvmK5mYXQlyfQ
Summary: This article addresses a critical but often ignored challenge in refactoring: the side effects. When refactoring to eliminate "bad smells" (indicators of poor design), developers can unintentionally introduce new problems into the system. The authors propose three methods for handling these side effects and use a multi-objective optimization algorithm (MO-CMA-ES) to achieve better refactoring outcomes. Their experiments with four open-source Java projects show that proactively fixing emerging "new smells" during refactoring yields significantly better system quality. For a student learning software engineering, this paper underlines that refactoring is not just about making "one good change" but about understanding the complex, evolving impact of those changes. The insight here is the concept of dynamic smell sequencing: while refactoring, students must expect that "new problems" will arise, and learning to update refactoring plans on the fly (rather than rigidly following the first plan) leads to much higher quality codebases.

Title: Test code refactoring unveiled: where and how does it affect test code quality and effectiveness?
Summary:
The study investigates how developers apply refactoring to test code, an area less understood than production code refactoring. By mining 65 open-source Java projects, the authors analyze whether test classes with low quality or effectiveness are more likely to be refactored and if refactoring improves them. Findings show test refactorings often target low-quality test classes (those with test smells and poor structural metrics), but there is no strong link between refactoring and improved code/mutation coverage. Refactorings like Extract Class help improve modularity and cohesion, whereas others like Extract Method may unintentionally introduce complexity or new test smells. Overall, while test refactoring helps code organization, it does not always enhance the ability to detect defects. For a CS student, this highlights that "cleaner code" isn't always about fewer lines or simpler structures—it's about smarter organization that improves *test effectiveness*. Especially in course projects where testing is critical (like software engineering or systems classes), students should prioritize refactoring in a way that increases coverage and reliability, even if it means more, smaller test methods rather than giant ones.

Journal: Empirical Software Engineering

Web address: https://doi.org/10.1007/s10664-024-10577-y

Authors: Luana Martins, Valeria Pontillo, Heitor Costa, Filomena Ferrucci, Fabio Palomba, Ivan Machado

Date of Publication: November 16, 2024 (Volume 30, Article number 27, 2025)

Josephine McPhereson

The paper titled **"Code Refactoring for Software Reusability: An Experimental Study,"** authored by Abdullah Almogahed, Hairulnizam Mahdin, Mazidah Mat Rejab, Abdulwadood Alawadhi, Samera Obaid Barraood, and Manal Othman, was published in the **2024 4th International Conference on Emerging Smart Technologies and Applications (eSmarTA)** and made available through IEEE Xplore on **August 26, 2024**. The study focuses on understanding how specific code refactoring techniques impact **software reusability**, a crucial aspect for enhancing productivity, code quality, and reducing development costs. Using the open-source **jEdit dataset**—which contains 1152 classes—the authors applied five common refactoring techniques: **Extract Interface, Encapsulate Field, Extract Class, Inline Class,** and **Inline Method**. A total of **338 refactorings** were performed, distributed across these techniques based on opportunities available in the dataset.

To evaluate reusability, the study used established software metrics: **Design Size in Classes (DSC), Class Interface Size (CIS), Cohesion Among Methods (CAM),** and **Direct Class Coupling (DCC)**. Reusability was calculated before and after refactoring using a weighted formula combining these metrics. Tools like **Eclipse Metrics Plugin**, **JDeodorant**, and **Eclipse IDE** were employed to apply and measure the refactorings. The results demonstrated that **Extract Interface**, **Encapsulate Field**, and **Extract Class** consistently **improved** software reusability, while **Inline Class** and **Inline Method** techniques **reduced** it. The study emphasizes that techniques focusing on modularity and separation of concerns enhance the ability to reuse code across projects, thus saving development time and improving collaboration. In contrast, inlining techniques, by increasing complexity and tight coupling, harmed reusability. The authors conclude that developers and software architects should prioritize extraction-based refactorings when aiming to create robust and reusable systems. They also suggest that future studies explore additional refactoring techniques like **Extract Method**, **Move Method**, and others across different datasets to broaden the understanding of how refactoring influences reusability.

Summary:
This paper presents an empirical study analyzing how **software refactorings** impact the

software attack surface in open-source projects written in Python, C, and JavaScript. Examining 30 projects and 3,523 refactorings, the study reveals that while refactoring aims to improve code structure, it can unintentionally increase security risks by dispersing vulnerabilities, bugs, and code smells. The findings show that refactorings can have **positive, negative, or neutral effects**: while most have a neutral impact, some introduce new vulnerabilities or bugs. Tools like SonarQube, RefDiff, and PyReff were used to collect data, and the results stress that developers must carefully sequence and combine refactorings to maintain or enhance software security, not just code quality. The paper calls for future research into security-aware refactoring practices.

Reference:

- **Title:** *On the Impact of Refactorings on Software Attack Surface*
- **Journal:** *IEEE Access (Volume 12)*
- **Web address:** https://doi.org/10.1109/ACCESS.2024.3404058
- **Authors:** Estomii Edward, Ally S. Nyamawe, Noe Elisa
- **Date of Publication:** 22 May 2022

Mary Swanson
Title: *A Metrics-Based Approach for Selecting Among Various Refactoring Candidates*
Authors:
Journal: Empirical Software Engineering
Publication Date: December 2023
URL: https://link.springer.com/article/10.1007/s10664-023-10412-w
Summary: This study proposes a metrics-based framework to assist developers in prioritizing refactoring tasks. By analyzing structural characteristics of code and their impact on quality, the authors develop a method to guide the selection of refactoring candidates that would yield the most significant improvements. The approach aims to optimize the refactoring process by focusing efforts on areas of code that would benefit most, thereby enhancing overall software quality. This research is pertinent for teams seeking data-driven strategies to manage technical debt effectively.

Data Title:
Behind the scenes: On the relationship between developer experience and refactoring

Summary:
This study examines how developer experience influences code refactoring practices. By

analyzing 800 open-source Java projects, the authors find that developers with higher contribution levels perform more refactorings but tend to document them less. Refactoring motivations include feature additions, bug fixes, internal code improvements, and external quality enhancements. The work uses automated mining tools and machine learning models to classify refactoring activities, highlighting that refactoring is integral to everyday development, not just design maintenance.

Authors:
Eman Abdullah Alomar, Anthony Peruma, Mohamed Wiem Mkaouer, Christian D. Newman, Ali Ouni

Where Published:
Journal of Software: Evolution and Process (Wiley Online Library), published October 28, 2021.