Refactoring Articles

Refactoring--Does It Improve Software Quality?

Authors: Konstantinos Stroggylos; Diomidis Spinellis

Journal: Fifth International Workshop on Software Quality (WoSQ'07: ICSE Workshops 2007)

Published: July 2007

URL: https://ieeexplore.ieee.org/abstract/document/4273477/authors#authors

Summary: This article examines the practical impact of refactoring on software quality by analyzing popular open-source projects. It discusses the theoretical goal of refactoring – improving internal structure without altering external behavior – and the use of software metrics (like coupling, cohesion, complexity) to measure quality. The authors studied commit logs, specifically identifying changes developers labeled as "refactoring," and measured metrics before and after these changes in systems like Apache httpd, Log4J, and Hibernate. Their key finding, contrary to expectations, was that these refactorings often resulted in *worse* metric values (e.g., increased coupling and complexity, decreased cohesion). This research is highly relevant because it questions the assumption that refactoring always leads to measurable quality improvements in real-world scenarios, suggesting either that current metrics don't fully capture the benefits or that developers may not be applying refactoring effectively to achieve better metric scores.

When Does a Refactoring Induce Bugs? An Empirical Study

Authors: Gabriele Bavota; Bernardino De Carluccio; Andrea De Lucia; Massimiliano Di Penta; Rocco Oliveto; Orazio Strollo

Journal: 2012 IEEE 12th International Working Conference on Source Code Analysis and Manipulation

Published: September 2012

URL: https://ieeexplore.ieee.org/abstract/document/6392107

Summary: This second article investigates the potential downside of refactoring: the risk of introducing new bugs (faults). The authors used an automated tool (Ref-Finder) to detect a wide variety of refactoring operations across the history of three Java projects (Apache Ant, ArgoUML, Xerces-J), manually validating the findings. They then employed the SZZ algorithm to trace bug fixes back to the changes that likely introduced them, specifically looking at whether refactorings were responsible. Their main findings indicate that while the overall percentage of refactorings linked to bug introduction is relatively low (around 15%), certain *types* of refactoring, particularly those manipulating class hierarchies like “Pull Up Method” and “Extract Subclass”, have a significantly higher probability (around 40%) of inducing faults. The study also explored whether bugs appeared more often in the source or target component of a refactoring operation. This article is relevant because it provides empirical evidence contradicting the theoretical ideal that refactoring is purely

behavior-preserving and risk-free, highlighting that specific refactoring operations can be inherently more dangerous than others and may require more rigorous testing or code review.