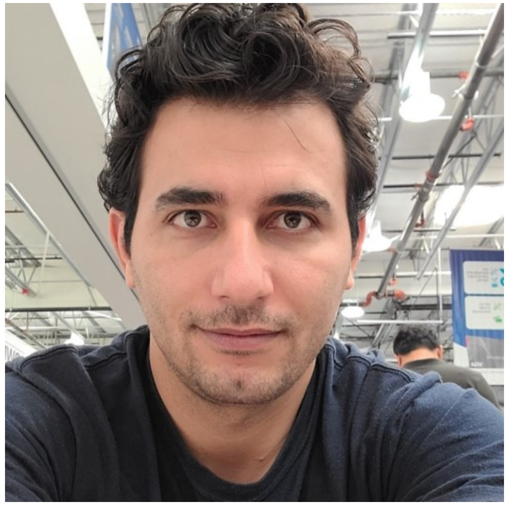
One thousand and one stories: a large-scale survey of software Refactoring

Mohamed Wiem Mkaouer
Department of Software Engineering
Rochester Institute of Technology
mwmvse@rit.edu

Origin of the Study

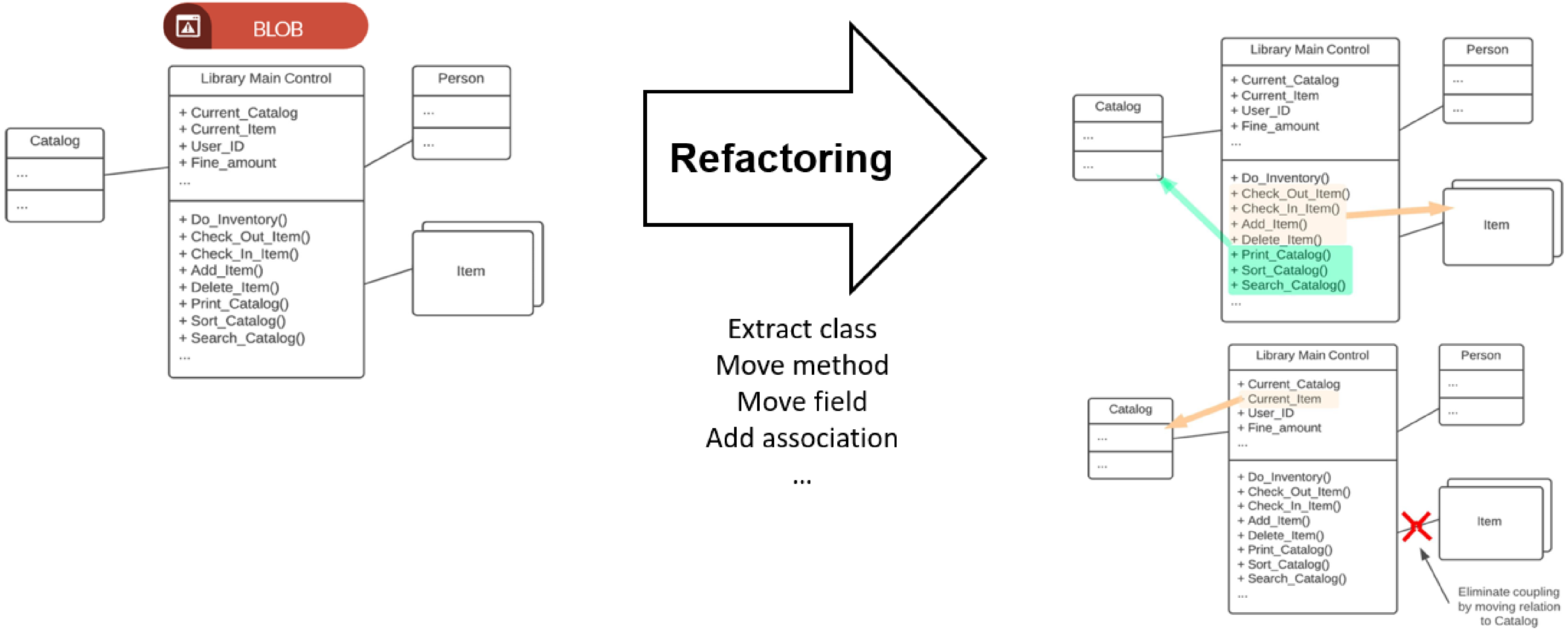
"I noticed that the review of my refactoring changes takes longer than usual to be approved"



Software Engineer at Xerox

Refactoring

- The process of improving a code after it has been written **by changing its internal structure without changing the external behavior**.



M. Fowler, K. Beck, J. Brant, W. Opdyke, and d. Roberts. Refactoring: Improving the Design of Existing Code. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

Pilot Study at Xerox (Design)

171 Refactoring
Pull Requests

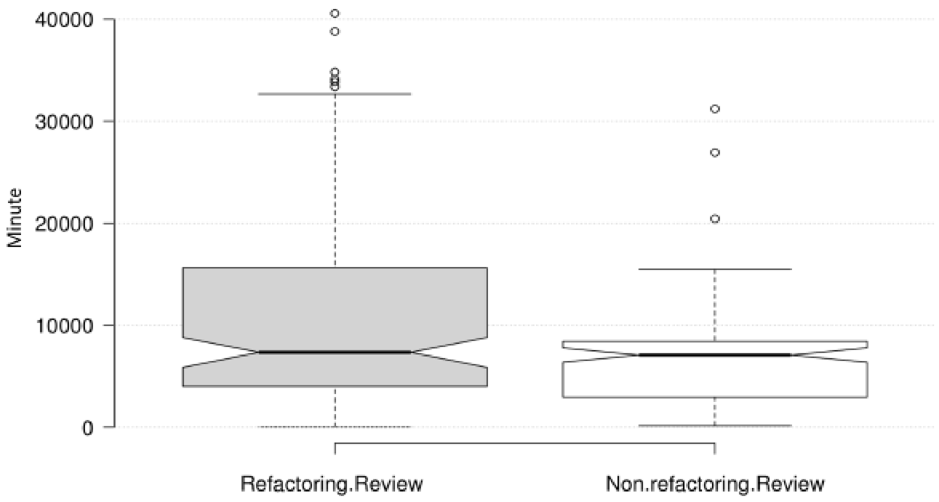
171 non-Refactoring
Pull Requests

Review Duration
exchanged messages

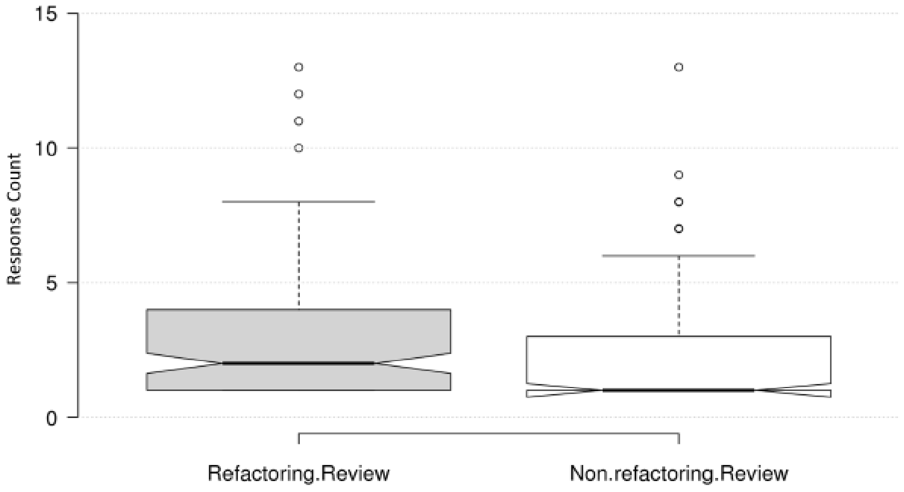
VS

Review Duration
exchanged messages

Challenge 1:



(a) Review duration



(b) Number of exchanged responses

Refactoring code reviews take longer to be completed than the non-refactoring code reviews.

Refactoring code reviews trigger longer discussions between code authors and reviewers before reaching a consensus.

Recommendation 1:

Intent

Understanding the purpose of the intended refactoring

“Removed some dead code”

“Refactored duplicate methods”

Instruction

Reporting refactoring operations developers have performed

“Renamed […]”

“Moved […]”

“Extracted […]”

Impact

Understanding the impact of the applied refactoring

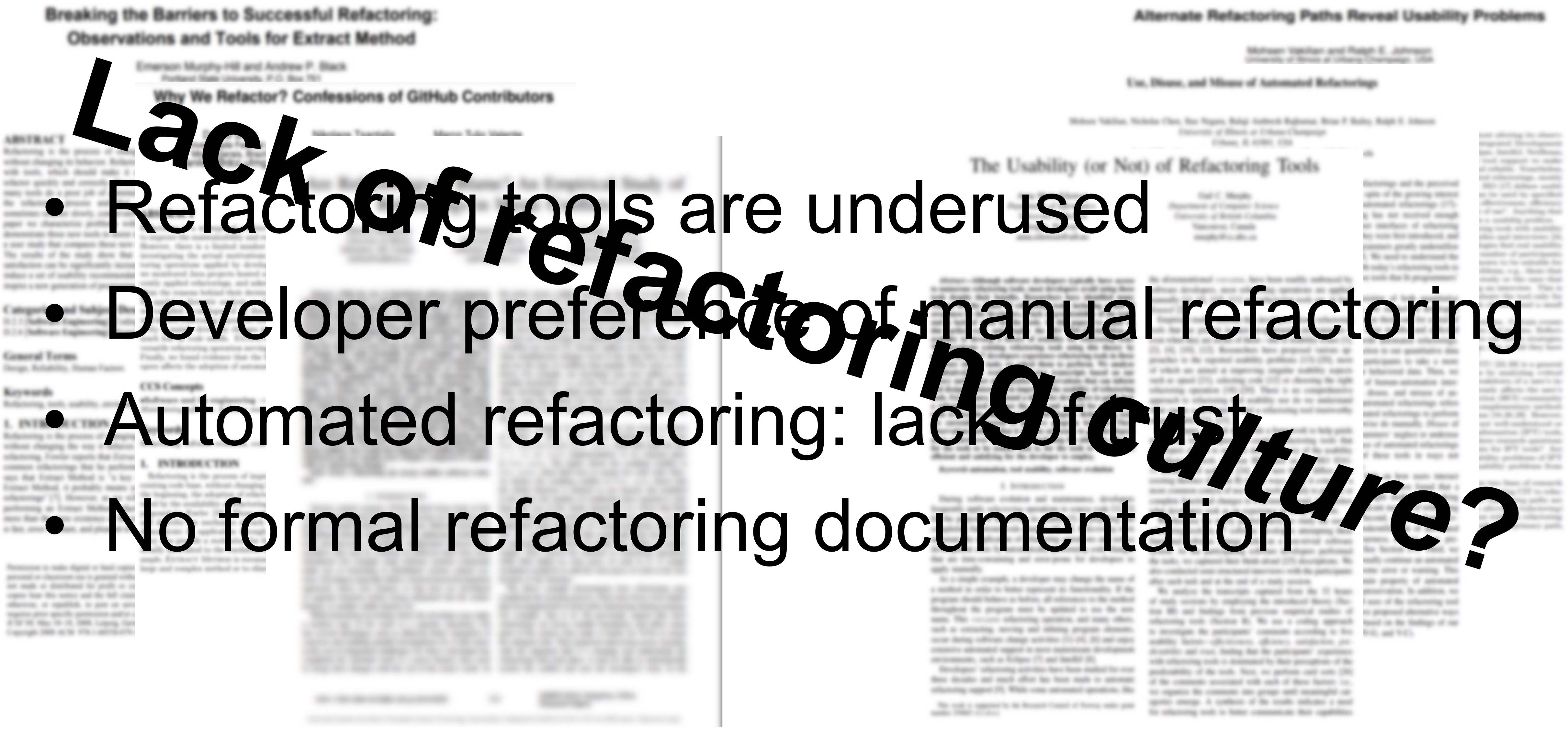
“[…] to improve readability”

“[…] to reduce complexity”


One train may hide another one

Recent Refactoring Studies

Recent Refactoring Studies



- Refactoring tools are underused
- Developer preference of manual refactoring
- Automated refactoring: lack of trust
- No formal refactoring documentation

Lack of refactoring culture?

JetBrains IDEA

Lack of trust (10 instances) was the most frequent reason. Some developers do not trust refactoring tools for complex operations that involve code manipulation and only use them for renaming or moving:

"I don't trust the IDE for things like this, and usually lose other comments, notation, spacing from adjacent areas."

"I'd say developers are reluctant to let a tool perform anything but trivial refactorings, such as the ones you picked up on my commit."

On the other hand, some developers also think that tool support is unnecessary in simple cases (8 instances). Sometimes the operation may involve only local changes and is trivial to do by hand. Thus, calling a special operation to do it is considered unnecessary, as illustrated by this comment:

"Automated refactoring is overkill for moving some private fields."

Additionally, developers also mentioned: lack of tool support for the specific refactoring they were doing (6 instances), not being familiar with refactoring features of the IDE (3 instances), and not realizing they could use refactoring tools at the moment of the refactoring (2 instances).

5.3 What IDEs developers use for refactoring?

When answering to our emails, 83 developers spontaneously mentioned which IDE they use. Therefore, we decided to investigate these answers, specially because our study is not dependent on any IDE, and thus differs from previous studies which are usually based only on Eclipse data [23, 24]. Table 8 shows the most common IDEs mentioned in these answers and the percentage of refactorings performed automatically in these cases. 139 developers (63%) did not explicitly mention an IDE when answering this question.

Table 8: IDE popularity

IDE	Occurrences	Automated %
Editor not mentioned	139	12%
IntelliJ IDEA	51	71%
Eclipse	18	44%
NetBeans	8	50%
Android Studio	4	25%
Text Editor	2	0%

TRACT METHOD is the "Swiss army knife of refactorings" [36]. It is the refactoring with the most motivations (11 in total). In comparison to [36], there is an overlap in the reported motivation themes for EXTRACT METHOD. We found some new themes, such as *improve testability* and *enable recursion*, but

we did not find any instances of the themes *encapsulate field* and *hide message chain*, reported in [36], which are related to code smell resolution. We assume these different themes are due to the nature of the examined projects, since [36] examined only three libraries and frameworks, while in this study we examined 124 projects from various domains including standalone applications. By comparing to the code symptoms that initiate refactoring reported in the study by Kim et al. [17], we found the *readability*, *reuse*, *testability*, *duplication*, and *dependency* motivation themes in common.

Most of the refactoring motivations we found have the intention to facilitate or even enable the completion of the maintenance task that the developer is working on. For instance, *extract reusable method*, *introduce alternative method signature*, and *facilitate extension* are among the most frequent motivations, and all of them involve enhancing the functionality of the system. Therefore, EXTRACT METHOD is a key operation to complete other maintenance tasks, such as adding a feature or fixing a bug. In contrast, only two out of the 11 motivations we found (*decompose method to improve readability* and *remove duplication*) are targeting code smells. This finding could motivate researchers and tool builders to design refactoring recommendation systems [35, 30, 33, 12, 18, 37] that do not focus solely on detecting refactoring opportunities for the sake of code smell resolution, but can support other refactoring motivations as well.

We also observe that developers are seriously concerned about avoiding code duplication, when working on a given maintenance task. They often use refactorings—especially EXTRACT METHOD—to achieve this goal, as illustrated by the following comments:

"I need to add a check to both the then- and the else-part of an if-statement. This resulted in more duplicated code than

bug fix required
riplication. That
up the replicated

ically performed
, the most com-
ATTRIBUTE, and
nts, so that they
levance.

In a field study
that most refac-
n our study, in-
IDEs, we found
rmed. However,
nd to use more
E users. More-
METHOD refac-
% in our study
ntages of manu-
, we should keep
efactorings, like
th tool support.
al. [23], where
ormed manually
(considering also renamings), we detected significantly more automated refactorings. We suspect this difference may be due to two reasons. First, automated refactoring tools may have become more popular and reliable over the last years.

Why We Refactor? Confessions of GitHub Contributors

Danilo Silva
Universidade Federal de Minas Gerais, Brazil
danilofs@dcc.ufmg.br

Nikolaos Tsantalis
Concordia University
Montreal, Canada
tsantalis@cse.concordia.ca

Marco Tulio Valente
Universidade Federal de Minas Gerais, Brazil
mtov@dcc.ufmg.br

ABSTRACT

Refactoring is a widespread practice that helps developers to improve the maintainability and readability of their code. However, there is a limited number of studies empirically investigating the actual motivations behind specific refactoring operations applied by developers. To fill this gap, we monitored Java projects hosted on GitHub to detect recently applied refactorings, and asked the developers to explain the reasons behind their decision to refactor the code. By applying thematic analysis on the collected responses, we compiled a catalogue of 44 distinct motivations for 12 well-known refactoring types. We found that refactoring activity is mainly driven by changes in the requirements and much less by code smells. EXTRACT METHOD is the most versatile refactoring operation serving 11 different purposes. Finally, we found evidence that the IDE used by the developers affects the adoption of automated refactoring tools.

CCS Concepts

•**Software and its engineering → Software evolution;** *Maintaining software; Software maintenance tools;*

Keywords

Refactoring, software evolution, code smells, GitHub

1. INTRODUCTION

Refactoring is the process of improving the design of an existing code base, without changing its behavior [27]. Since the beginning, the adoption of refactoring practices was fostered by the availability of refactoring catalogues, as the one proposed by Fowler [10]. These catalogues define a name and describe the mechanics of each refactoring, as well as demonstrate its application through some code examples. They also provide a *motivation* for the refactoring, which is usually associated to the resolution of a code smell. For example, EXTRACT METHOD is recommended to decompose a large and complex method or to eliminate code duplication. As a second example, MOVE METHOD is associated to smells like Feature Envy and Shotgun Surgery [10].

There is a limited number of studies investigating the real motivations driving the refactoring practice based on interviews and feedback from actual developers. Kim et al. [17] explicitly asked developers "in which situations do you perform refactorings?" and recorded 10 code symptoms that motivate developers to initiate refactoring. Wang [40] interviewed professional software developers about the major factors that motivate their refactoring activities and recorded human and social factors affecting the refactoring practice. However, both studies were based on general-purpose surveys or interviews that were not focusing the discussion on specific refactoring operations applied by the developers, but rather on general opinions about the practice of refactoring.

Contribution: To the best of our knowledge, this is the first study investigating *the motivations behind refactoring based on the actual explanations of developers on specific refactorings they have recently applied*. To fill this gap on the empirical research in this area, we report a large scale study centered on 463 refactorings identified in 222 commits from 124 popular, Java-based projects hosted on GitHub. In this study, we asked the developers who actually performed these refactorings to explain the reasons behind their decision to refactor the code. Next, by applying thematic analysis [6], we categorized their responses into different themes of motivations. Another contribution of this study is that we make publicly available[1] the data collected and the tools used to enable the replication of our findings and facilitate future research on refactoring.

Relevance to existing research: The results of this empirical study are important for two main reasons.

First, having a list of motivations driving the application of refactorings can help researchers and practitioners to infer rules for the automatic detection of these motivations when analyzing the commit history of a project. Recent research has devised techniques to help in understanding better the practice of code evolution by identifying frequent code change patterns from a fine-grained sequence of code changes [26], isolating non-essential changes in commits [13], and untangling commits with bundled changes (e.g., bug fix and refactoring) [7]. In addition, we have empirical evidence that developers tend to interleave refactoring with other types of programming activity [24], i.e., developers tend to *floss refactor*. Therefore, *knowing the motivation behind a refactoring can help us to understand better other related changes in a commit*. In fact, in this study we found

[1] http://aserg-ufmg.github.io/why-we-refactor

• Silva, Danilo, Nikolaos Tsantalis, and Marco Tulio Valente. "Why we refactor? confessions of GitHub contributors." *Proceedings of the 2016 24th acm sigsoft international symposium on foundations of software engineering*. 2016.

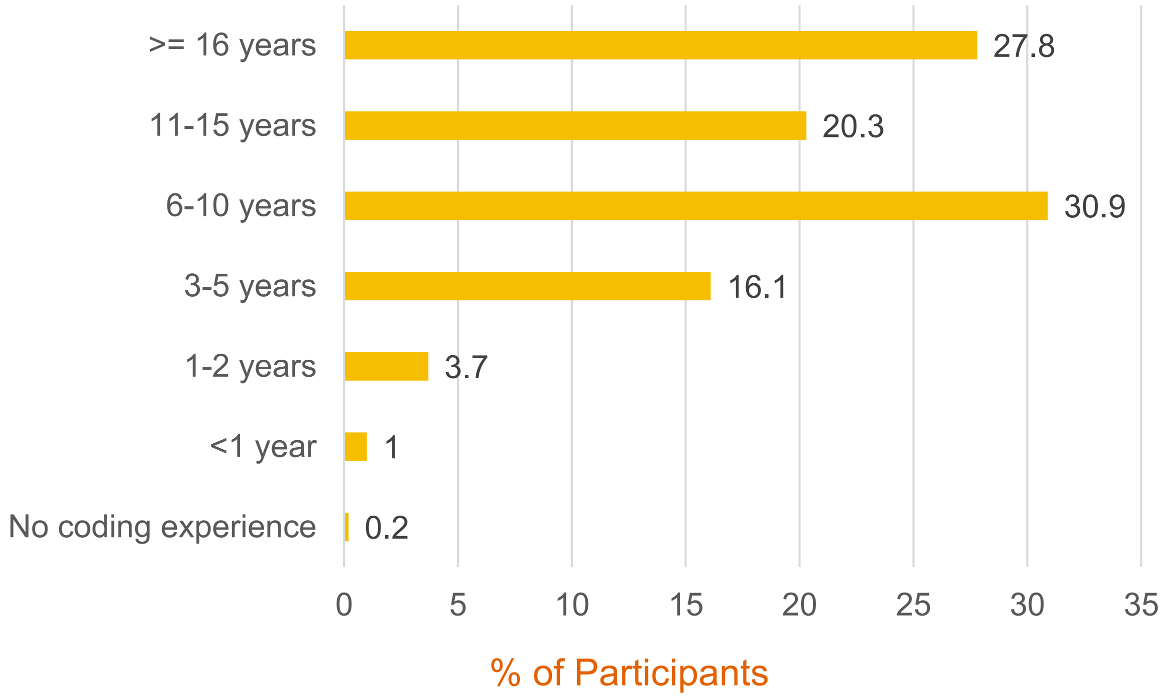
Research Goal

How do developers use IntelliJ to refactor code?

Study Design

1183 Developers

Participants professional development experience



How Code Gets Refactored in IntelliJ?

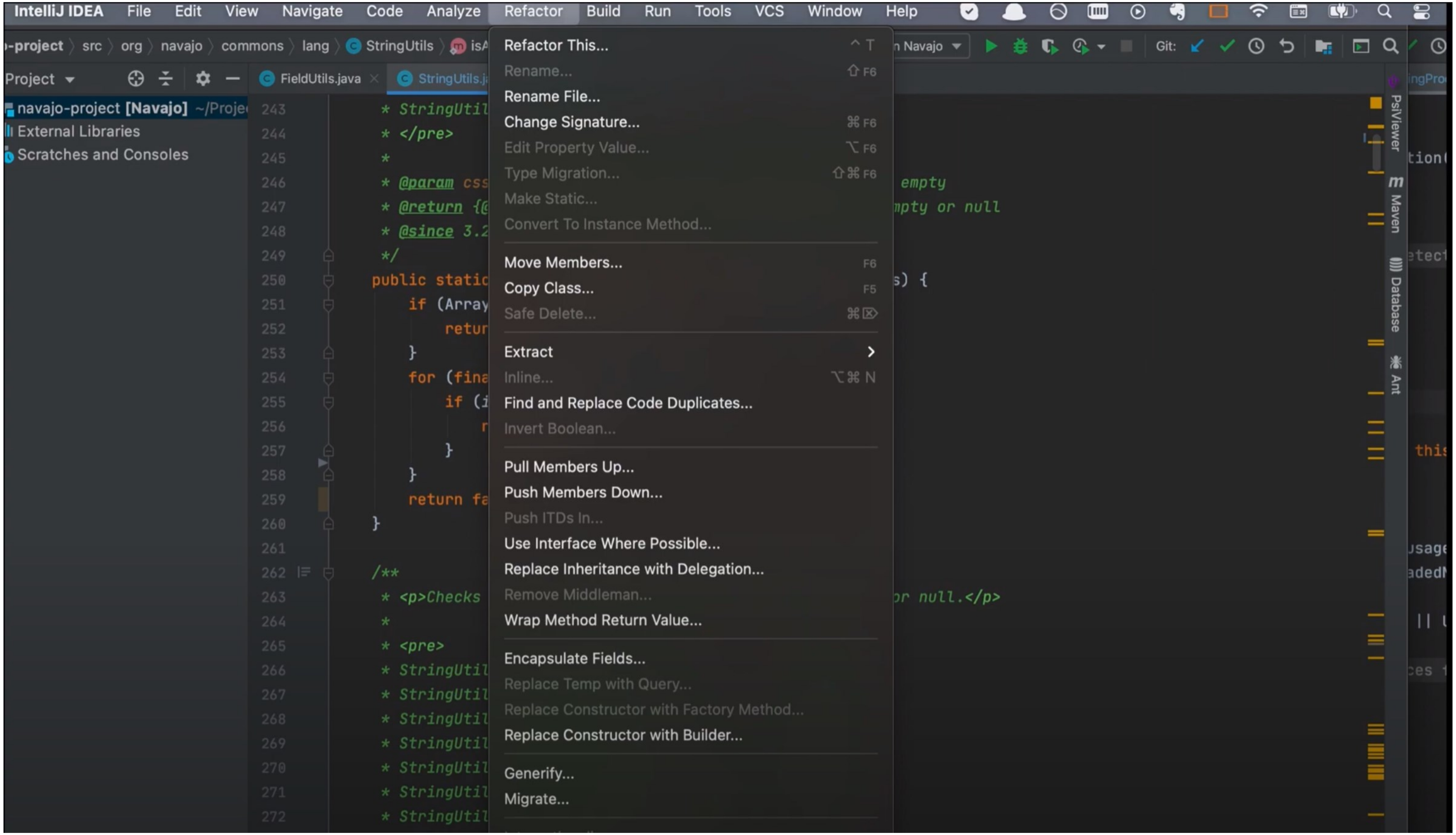
How Code Gets Refactored in IntelliJ?

	Used IDE refactoring	Used Find and Replace	Used Copy and Paste	Edited manually	Didn't have this scenario
Renaming a class, method, variable, or symbol	85.8%	46.2%	21.3%	29.8%	0.3%
Extracting a method or a variable from existing code	54.7%	20.7%	30.4%	33.2%	10.7%
Moving code to another file	38.6%	12.2%	57.5%	30.8%	4.4%

Rename Extract Move

Challenge 2:

Recommendation 2:



Questions?

Contact me:
mwmvse@rit.edu

Refactoring Practices in the Context of Modern Code Review: An Industrial Case Study at Xerox

Eman Abdullah AlOmar*, Hussein AlRubaye†, Mohamed Wiem Mkaouer*, Ali Ouni‡, Marouane Kessentini§
*Rochester Institute of Technology, Rochester, NY, USA
†Xerox Corporation, Rochester, NY, USA
‡ETS Montreal, University of Quebec, Montreal, QC, Canada
§University of Michigan, Dearborn, MI, USA
eman.alomar@mail.rit.edu, hussein.alrubaye@xerox.com, mwmvse@rit.edu, ali.ouni@etsmtl.ca, marouane@umich.edu

***Abstract*—Modern code review is a common and essential practice employed in both industrial and open-source projects to improve software quality, share knowledge, and ensure conformance with coding standards. During code review, developers may inspect and discuss various changes including refactoring activities before merging code changes in the code base. To date, code review has been extensively studied to explore its general challenges, best practices and outcomes, and socio-technical aspects. However, little is known about how refactoring activities are being reviewed, perceived, and practiced.**

This study aims to reveal insights into how reviewers develop a decision about accepting or rejecting a submitted refactoring request, and what makes such review challenging. We present an industrial case study with 24 professional developers at Xerox. Particularly, we study the motivations, documentation practices, challenges, verification, and implications of refactoring activities during code review.

definition, is not intended to alter to the system's behavior, but to improve its structure, so its review may differ from other code changes. Yet, there is not much research investigating how developers review code refactoring. The research on refactoring has been focused on its automation by identifying refactoring opportunities in the source code, and recommending the adequate refactoring operations to perform [6]–[8]. Moreover, the research on code reviews has been focused on automating it by recommending the most appropriate reviewer for a given code change [3]. However, despite the critical role of refactoring and code review, the innate relationship between them is still largely unexplored in practice.

The goal of this paper is to understand how developers review code refactoring, *i.e.,* what criteria developers rely on

One Thousand and One Stories: A Large-Scale Survey of Software Refactoring

Yaroslav Golubev
JetBrains Research
Saint Petersburg, Russia
yaroslav.golubev@jetbrains.com

Zarina Kurbatova
JetBrains Research
Saint Petersburg, Russia
zarina.kurbatova@jetbrains.com

Eman Abdullah AlOmar
Rochester Institute of Technology
Rochester, United States
eman.alomar@mail.rit.edu

Timofey Bryksin
JetBrains Research
Higher School of Economics
Saint Petersburg, Russia
timofey.bryksin@jetbrains.com

Mohamed Wiem Mkaouer
Rochester Institute of Technology
Rochester, United States
mwmvse@rit.edu

ABSTRACT

Despite the availability of refactoring as a feature in popular IDEs, recent studies revealed that developers are reluctant to use them, and still prefer the manual refactoring of their code. At JetBrains, our goal is to fully support refactoring features in IntelliJ-based IDEs and improve their adoption in practice. Therefore, we start by raising the following main questions. How exactly do people refactor code? What refactorings are the most popular? Why do some developers tend not to use convenient refactoring tools provided by modern IDEs?

Stories: A Large-Scale Survey of Software Refactoring. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Refactoring [12] is traditionally defined as the process of improving the internal code structure without altering its external behavior. Since this practice had been introduced to a wide audience of software engineers, it has become a crucial tool to maintain high-quality software and to reduce its technical debt. Several refactoring types,

AlOmar, et al. "Refactoring practices in the context of modern code review: An industrial case study at Xerox." In *ICSE-SEIP*, pp. 348-357. 2021.

Golubev et al. "One thousand and one stories: a large-scale survey of software refactoring." In *FSE*, pp. 1303-1313. 2021.