On the Relationship Between Developer Experience and Refactoring: An Exploratory Study and Preliminary Results

Eman Abdullah AlOmar  
*Rochester Institute of Technology*

Anthony Peruma  
*Rochester Institute of Technology*

Christian D. Newman  
*Rochester Institute of Technology*

Mohamed Wiem Mkaouer  
*Rochester Institute of Technology*

Ali Ouni  
*University of Quebec at Montreal*

Follow this and additional works at: [https://scholarworks.rit.edu/article](https://scholarworks.rit.edu/article)

Part of the *Software Engineering Commons*

**Recommended Citation**
https://doi.org/10.1145/3387940.3392193
On the Relationship Between Developer Experience and Refactoring: An Exploratory Study and Preliminary Results

Eman Abdullah AlOmar  
eman.alomar@mail.rit.edu  
Rochester Institute of Technology  
Rochester, New York, USA

Anthony Peruma  
anthony.peruma@mail.rit.edu  
Rochester Institute of Technology  
Rochester, New York, USA

Christian D. Newman  
cnewman@se.rit.edu  
Rochester Institute of Technology  
Rochester, New York, USA

Mohamed Wiem Mkaouer  
mwnvse@rit.edu  
Rochester Institute of Technology  
Rochester, New York, USA

Ali Ouni  
ali.ouni@etsmtl.ca  
ETS Montreal, University of Quebec  
Montreal, Quebec, Canada

ABSTRACT
Refactoring is one of the means of managing technical debt and maintaining a healthy software structure through enforcing best design practices, or coping with design defects. Previous refactoring surveys have shown that these code restructurings are mainly executed by developers who have sufficient knowledge of the system’s design, and disposing of leadership roles in their development teams. However, these surveys were mainly limited to specific projects and companies. In this paper, we explore the generalizability of the previous results through analyzing 800 open-source projects. We mine their refactoring activities, and we identify their corresponding contributors. Then, we associate an expertise score to each contributor in order to test the hypothesis of whether developers with higher scores tend to perform a higher number of refactoring operations. We found that (1) although refactoring is not restricted to a subset of developers, those with higher experiences score tend to perform more refactorings than others; (2) our qualitative analysis of three randomly sampled projects show that the developers who are responsible for the majority of refactoring activities are typically on advanced positions in their development teams, demonstrating their extensive knowledge of the design of the systems they contribute to.

CCS CONCEPTS
- Software and its engineering → Software evolution; Maintaining software.

KEYWORDS
Software maintenance and evolution, Mining software repositories, Software refactoring, Developer experience, Quality

1 INTRODUCTION
Refactoring has been considered, along with code reviews, as the main quality safeguard and the backbone of managing technical debt [8]. Therefore, understanding the best refactoring practices is highly important. The spectrum of research exploring the practice of refactoring covers a wide variety of dimensions, such as the identification of refactoring opportunities [11, 23, 24], recommendation of adequate refactoring operations [7, 9, 16–19, 30, 35], detection of applied refactorings [12, 32, 40], studying the impact of refactoring on quality [6, 29, 41, 42], the reasons as to why developers refactor their code [25, 26, 31], etc. However, little is known about how the level of experience influences developer refactoring activities. Nevertheless, developer experience directly impacts their ability to estimate software quality, and therefore, their ability to determine the appropriate refactoring strategy that needs to be deployed. Moreover, developers’ knowledge of the system’s structure and sub-components varies, and so is their privilege to access and modify them. This paper aims to start the discussion around the importance of considering the developer’s experience as part of proposing solutions related to refactoring, since their applicability depends on the perception and privilege of the developers in charge.

A couple of refactoring studies have pointed out that refactoring is typically performed by experienced developers: Tsantalis et al. [36] performed a multidimensional empirical study on refactoring activities that included: the proportion of refactoring operations performed on production and test code, the most active refactoring contributors, the relationship between refactoring with releases and testing activity, and the purpose of the applied refactorings. With regard to developer experience, the authors found that the top refactoring contributors had a management role within the project. In another study, Kim et al. [13] surveyed 328 professional software engineers at Microsoft to investigate when and how they do refactoring. They found that developers with different expertise levels experienced five risk factors involved in refactoring, namely, regression bugs, code churns, merge conflicts, time taken from other tasks, the difficulty of performing code reviews after refactoring, and the testing cost. They also investigated the relationship between the refactoring effort and reduction of the number of inter-module dependencies and after release defects. They reported that other factors, such as developer experience, need to be examined as the changes to the number of module dependencies and post-release defects might be caused by such factors other than refactoring. The findings of these studies [13, 22, 36] indicate that experience plays a significant role in the execution of refactoring, yet they were both limited to developer surveys without any concrete evidence from the source code, and they were also limited to a few projects.

In this paper, we explore the hypothesis of whether developers with more experience are most likely to be responsible for a higher number of refactoring activities. We aim to challenge the generalizability of the previous findings indicating that only a subset of developers performs major refactoring activities. Our study is driven by the following research questions:
Our research methodology consists of three main phases - Data Collection, Detection & Extraction, and Data Analysis. Figure 1 provides an overview of our methodology. Described below are details of the methodology activities.

To conduct our exploratory study, we used a dataset of well-engineered open-source Java projects. The authors of this dataset [20] curated a set of open-source projects, proven to follow software engineering practices such as documentation, testing, issue and bug tracking, and project management. We chose this dataset as it was also analyzed in previous studies [3, 10, 27] that have been mining refactoring operations, just like our study. In total, our dataset is composed of 800 projects hosted on GitHub. Each project was cloned in order to extract the data needed for our experiments. This data included, but not limited to, each commit author, source files impacted by each committed change, and timestamps etc. The projects in our dataset were cloned in early 2019, and 74.6% of the projects had their most recent commit within the last three years.

Next, we utilized RefactoringMiner [37] to identify refactoring operations occurring in the projects. RefactoringMiner iterates over the commit history of a repository in chronological and compares the changes made to Java source code files in order to detect refactorings. Of the available state-of-the-art set of refactoring detection tools, RefactoringMiner has the highest performance, more specifically, a precision of 98% and a recall of 87% [31, 37]. Running refactoringMiner on the projects under study, resulted scanning a total of 748,001 commits, from which, 111,884 commits contained at least one refactoring operation, and we collected a total of 711,495 refactoring operations. On average, each project contains 732 refactoring commits authored by 19 developers.

Finally, we analyzed the output generated from our detection and extraction activities to answer our research questions. Since our research questions are both quantitative and qualitative, we used tools/scripts along with manual activities to arrive at our findings.

For replication purposes, our dataset and other artifacts are available on our project website [1].

3 RESULTS & DISCUSSION

In this section, we report and discuss our findings for analyzing the identified refactoring-related patterns to answer our two research questions RQ1 and RQ2.

3.1 What is the distribution of experience among developers that perform refactorings?

For our experiment on developer experience, we looked at the project contributions made by the developer. In other words, we utilized the volume of commits made to Java source code files by a developer as a proxy for experience. Introduced by [14], this approach calculates the Developer’s Commit Ratio (DCR) for each developer in the project. This ratio measures the number of individual commits made by the developer against all project commits. Formally, this ratio is defined as:

\[
DCR = \frac{\text{IndividualContributorCommits}}{\text{TotalProjectCommits}}
\]  

(1)

In our experiment, we only consider the author of a commit as its developer. We followed the same approach as in Peruma et al. [27], where the authors looked at the DCR distribution of developers that perform rename refactorings. As shown in Figure 2, we looked at two types of distributions - developers that only performed non-refactoring operations (depicted as ‘Non-Refactoring Commits’ in the chart), and developers that performed a mix of refactoring and non-refactoring operations on the source code (depicted as ‘All Commits’).
The relationship between developer experience and refactoring was explored through an exploratory study and preliminary results. Figure 2 shows the distribution of developer commit ratios (DCR) for different types of commits. The graph illustrates a clear observation: developers involved in implementation and maintenance of a system have more experience compared to those focusing solely on new features. The density plot reveals a concentration of non-refactoring developers on the lower end of the DCR scale. Table 1 summarizes the statistical distribution of DCR scores for non-refactoring and all commits.

A strong correlation was found between developer experience and the type of commit performed. Developers with higher DCR scores are more likely to engage in refactoring activities. This aligns with studies that have highlighted the importance of developer experience in software refactoring. Our findings confirm these studies, and our approach is more scalable as it relies on an automatic, non-subjective method.

3.2 Would higher experience indicate higher refactoring activities?

In this research question, we investigate whether specific developers are contributing significantly to the refactoring process. We approach this question from two perspectives: quantitative and qualitative. The quantitative study uses the DCR associated with developers and commit types to estimate the most experienced developers and identify those suitable for specific tasks. The qualitative approach involves a manual study on a select set of projects.

Summary
Using an alternate approach (i.e., developer contributions), we confirm findings from prior research that more experienced developers are typically involved in refactoring activities. Our approach utilizes existing repository data and is automated, providing a scalable method to estimate the most experienced developers in a project. It helps identify developers suitable for specific tasks.

3.2 Would higher experience indicate higher refactoring activities?

In this research question, we investigate whether specific developers are significantly contributing to the overall refactoring of the system, or if it is randomly distributed among all developers. We approach this research question from two fronts - quantitative and qualitative. The quantitative part of our study is based on empirical data and provides a scalable method to identify developers suitable for specific tasks. In the qualitative approach, we perform a manual, case study-like investigation on a select set of projects.

Quantitative
This part of the research question utilizes the DCR values associated with each developer in the project, along with the total number of refactoring and non-refactoring commits made by the developers.

Non-Refactoring Commits

<table>
<thead>
<tr>
<th>Min.</th>
<th>1st Qu.</th>
<th>Median</th>
<th>Mean</th>
<th>3rd Qu.</th>
<th>Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0001</td>
<td>0.0010</td>
<td>0.0019</td>
<td>0.0031</td>
<td>0.0043</td>
<td>0.0130</td>
</tr>
</tbody>
</table>

All Commits

<table>
<thead>
<tr>
<th>Min.</th>
<th>1st Qu.</th>
<th>Median</th>
<th>Mean</th>
<th>3rd Qu.</th>
<th>Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0002</td>
<td>0.0065</td>
<td>0.0197</td>
<td>0.0456</td>
<td>0.0604</td>
<td>0.2632</td>
</tr>
</tbody>
</table>

Figure 2: Distribution of DCR values for developers based on the type of commit performed in their project.

Table 1: Statistical summary of DCR scores based on the type of commit performed in the project.
looking at Figure 3, the first clear observation we see is the volume of commit counts made by the two sets of developers. A violin plot provides an ideal mechanism to represent developers, while chart 'B' shows the same counts for the remaining developers. For developers in each of the two sets, we obtained the count of 372 developers, while the remaining developers amount to 7,066. On average, a TOP-5 developer makes 70.24 and 223.7 refactoring and non-refactoring commits, respectively. On the other hand, the rest of the developers average around 3.21 and 15.69 refactoring and non-refactoring commits, respectively. Furthermore, the TOP-5 violin plot shows a high frequency of developers performing, approximately, 15 to 75 refactoring commits. The same does not hold for non-refactoring commits, where we see a higher density within the range of 75 to 125 commits. Additionally, we observed that our dataset contains some developers that perform at most around 300 refactoring commits while non-refactoring commits go up to around 800. Hence, non-refactoring commit counts have a higher variation than refactoring commits. The refactoring box plot is more condensed than the non-refactoring boxplot; this indicates that the data varies less and hence is more consistent.

Finally, we looked at the number of refactoring operations performed by the two groups of developers. It should be noted that a single refactoring commit can contain one or more refactoring operations. A statistical summary of our findings is presented in Table 2, while a comparative histogram is available in Figure 4. Even though the histogram shows a higher volume of refactoring operations by inexperienced developers, it should be noted that this is the cumulative count across all projects in the dataset. If we were to look at the individual developer contributions, we could see that experienced developers apply refactorings more often than inexperienced developers.

### Qualitative

To better understand the key role of the TOP-5 contributors in the development team, we extract refactorings from a select set of projects - Hadoop\(^1\), OrientDB\(^2\), and Camel\(^3\). These three systems were chosen based on the criteria used in [15] (i.e., had more than 100 stars, had more than 60 forks, had size over 2 MB, these repositories are active and well-used). Next, we cluster production and test files of these projects, by developer ID. Finally, we carefully examine the top contributor’s professional profiles to identify their role in the organization hosting the software project. Our findings are detailed below.

Figure 5 portrays the distribution of the refactoring activities on production code and test code performed by project contributors for each software system we examined. The Hadoop project has a total of 114 developers. Among them are 73 (64%) refactoring

---

1. https://github.com/apache/hadoop
2. https://github.com/orientechnologies/orientdb
3. https://github.com/apache/camel
Our findings show that refactoring activities are mainly performed by a subset of developers who have a management role in the company. Senior developers care more about refactoring the source code to ensure high-quality software and make the software easier for future development. These subsets of developers may perform certain practices when applying code refactoring (e.g., refactoring before and after adding new features, testing frequently to avoid any bugs that may introduce and affect the functionality of the software, and documenting and automating the application of refactoring). One of the reasons that seems not to encourage the other subsets of developers to significantly refactor the code is the technical constraints such as inadequate tool supports or lack of trust of automated support for composite refactorings. A discussion about various barriers to refactoring has been highlighted in Murphy et al. [21].

### 3.3 Research implications and future directions

While refactoring is being applied by various developers, it would be interesting to evaluate their refactoring practices. We want to capture and better understand the code refactoring best practices and learn from experienced developers so that we can recommend them for other developers. Also, it would be interesting to investigate the difference between the experienced and inexperienced developers in terms of distributions of refactoring operations, i.e., we aim to see if any specific refactoring types are highly solicited by one group compared to the other. As previous studies have already shown, some refactoring operations tend to be more complex than others [21], and so it is interesting for us to validate it in practice.

AlOmar et al. [2] performed an exploratory study on how developers document their refactoring activities in commit messages; this activity is called Self-Affirmed Refactoring (SAR). They found that developers tend to use a variety of textual patterns to document their refactoring activities, such as refactor, move and extract. In follow-up work, AlOmar et al. [3] identified which quality models are more in-line with the developer's vision of quality optimization when they explicitly mention in the commit messages that they refactor to improve these quality attributes. Since we noticed that various developers are responsible for performing refactorings, one potential research direction is to investigate which developers are responsible for the introduction of SARs in order to examine whether or not experience plays a role in the introduction of SARs. Another potential research direction is to study if developer experience is one of the factors that might contribute to the significant

---

Used in previous studies as a source to identify developers skills and experience.
Figure 5: Refactoring contributors in production and test files in Hadoop, OrientDB and Camel
improvement of the quality metrics that are aligned with developer perception tagged in the commit messages. In other words, we would like to evaluate the top contributors refactoring practice against all the rest of refactoring contributors by assessing their contributions on the main internal quality attributes improvement (e.g., cohesion, coupling, and complexity). Furthermore, previous studies analyzed the impact of refactorings on structural metrics and quality attributes [6, 29, 41, 42]. It would be interesting to revisit such analysis while taking into account the degree of expertise of the refactoring contributors. As developers with larger experience and managerial roles have better exposure to the system’s design, it is expected that their restructurings are of better quality, and this can be empirically demonstrated.

Furthermore, with regards to the analysis of refactoring and design quality, previous studies investigated how refactorings can be responsible for introducing code smells, and so hindering the design quality [28, 38]. It would be interesting to verify whether such unexpected results can correlate with the developer’s experience. Along with hindering design quality, the misuse of refactorings can also be responsible for bugs [4, 5], and various studies have proposed testing strategies to make refactoring safer [33, 34]. One of our future directions is to also correlate the bug-proneness of refactorings with the degree of expertise of the contributors. It is assumed that the lack of functional knowledge may facilitate the introduction of bugs, but this is subject to empirical validation as well.

5 CONCLUSION

We present an exploratory study of the level of experience of developers that apply refactorings. Prior studies use smaller samples to study similar questions; however, in our study, we have examined a more extensive and representative set of systems by comparison. Since we can confirm results from prior, manual studies automatically, we have identified a way to obtain similar results automatically. This means that it is possible to now study the impact of developer experience on a larger scale.

In future work, we plan to leverage the results from this study to determine specific types of refactorings made by developers at different experience levels. We would also like to explore ways to leverage this data to help suggest/recommend refactorings or suggest/recommend refactoring methodology based on the developers level of experience.

ACKNOWLEDGMENTS

We would like to thank the authors of Refactoring Miner for publicly providing it.

REFERENCES

[1] [n.d.] Project Website. https://sites.google.com/g.rit.edu/refactoring/.

4 THREATS TO VALIDITY

The first threat is that our analysis is restricted to only open-source, Java, Git-based repositories. However, we were still able to analyze projects that are highly varied in size, contributors, number of commits, and refactorings. Additionally, the representativeness of the dataset can be considered as a threat to this study. However, we mitigate this threat by utilizing 800 engineered projects that have also been part of a prior study on refactoring [27]. Furthermore, the projects are of varying sizes, contributors, and refactoring operations.

The accuracy of the refactoring detection tool also poses a threat to our study. However, previous studies [31, 37] report on high precision and recall scores for RefactoringMiner. However, a drawback to using RefactoringMiner is that the study is limited to Java projects. Our future work includes the use of refactoring mining tools that support other programming languages, such as RefDiff [32], to expand the representativeness of our dataset.

A major threat to validity is related to the calculation of experience. Obtaining the experience of each and every developer is a challenge for our study, given the volume of data in our dataset and also that experience can be subjective. Hence, we adopted a mechanism (i.e., DCR), used by prior research [14, 27], where we utilized project contributions as a proxy for experience. The reasoning behind the measurement assumes that the longer a developer is involved in a project, and the more they contribute to it, the more experience they become. Such an assumption may not hold for some specific scenarios; however, since the projects in our dataset are heterogeneous in nature, our assumption holds.

On the Relationship Between Developer Experience and Refactoring: An Exploratory Study and Preliminary Results  ICSEW’20, May 23–29, 2020, Seoul, Republic of Korea


