A Code Refactoring Dataset and Its Assessment Regarding Software Maintainability

István Kádár, Péter Hegedűs, Rudolf Ferenc and Tibor Gyimóthy
University of Szeged, Hungary
Email: {ikadar, hpeter, ferenc, gyimothy}@inf.u-szeged.hu

***Abstract*—It is very common in various fields that there is a gap between theoretical results and their practical applications. This is true for code refactoring as well, which has a solid theoretical background while being used in development practice at the same time. However, more and more studies suggest that developers perform code refactoring entirely differently than the theory would suggest.**

Our paper encourages the further investigation of code refactorings in practice by providing an excessive open dataset of source code metrics and applied refactorings through several releases of 7 open-source systems. As a first step of processing this dataset, we examined the quality attributes of the refactored source code classes and the values of source code metrics improved by those refactorings. Our early results show that lower maintainability indeed triggers more code refactorings in practice and these refactorings significantly decrease complexity, code lines, coupling and clone metrics. However, we observed a decrease in comment related metrics in the refactored code.

***Keywords*-code refactoring; software maintainability; empirical study;**

I. Introduction

The concept of source code refactorings introduced by Fowler [1] has been widely adopted by software engineering practitioners and it has become the part of the everyday processes. However, the application of theoretical results in practice often poses new challenges due to differences in the priorities between research and industry (e.g. time constraint, cost effectiveness, or return on investment). This is true for code refactoring as well. While lots of research effort is spent on identifying refactoring opportunities with the help of code smells and removing them by applying an appropriate refactoring type [2], [3] as Fowler et al. suggest, Peters and Zaidman [4] found that engineers are aware of code smells, but are not very concerned with their impact as refactoring activity is not focused on them. Similarly, Bavota et. al [5] revealed that only 7 % of the performed refactoring operations on three open-source systems actually remove the code smells from the affected class.

These results highlight that more research is needed to find out how developers apply refactoring in practice in order to elaborate new techniques and methods that better suit their needs. Our perception is that empirical investigations on refactoring activities is currently limited by the availability of experimental data. It is very tiresome to collect large amount of refactoring data and map them to low-level source code elements like classes and methods. Additionally, datasets used by researchers are often not shared within the community or contain very coarse-grained information (e.g. only the number of refactorings in a file without exact code diffs or line information as in the work of Bavota et al. [5]).

To overcome this problem, we propose a publicly available refactoring dataset that we assembled using the *Ref-Finder* tool [6] for refactoring extraction and the *SourceMeter*[1] static source code analyzer tool for source code metric calculation. The dataset contains fine-grained refactoring and source code metric information for 37 releases of 7 open-source Java systems at the moment. Each refactoring is mapped to source code elements at the level of methods and classes with exact version and line information that supports reproducible empirical investigations. Besides the source code metrics, the dataset contains the relative maintainability indices of source code elements, calculated by the *QualityGate*[2] tool which implements the *ColumbusQM quality model* [7]. This allows the researcher community to directly analyze the connection between source code maintainability and code refactoring.

We anticipate an increased number of empirical studies on the practical application of code refactorings due to the proposed dataset, similarly to what open bug datasets like PROMISE [8] or Bug Prediction Dataset [9] brought to the area of bug prediction. Although the dataset is in an early stage (e.g. requiring manual validation of refactoring data, extension in terms of systems and releases, addition of other extra information like defects in the source code elements), its value is already observable. With the help of the assembled dataset, we examine the connection between refactorings and practical maintainability of the code in this paper by investigating the following research questions:

RQ1. *Are classes with lower maintainability value subject to more refactorings in practice?*

RQ2. *Which quality attributes (source code metrics) are affected the most by code refactorings and to what extent?*

The main contributions of this work are the following:

- We created an open dataset containing the applied refactorings (40+ kinds) and source code metrics (50+ kinds) for several releases of open-source Java systems, similar to those open bug repositories that stimulated the area of bug prediction.
- Using the collected data we present empirical results on the correlation between source code maintainability and code refactorings.

[1] http://www.sourcemeter.com/
[2] http://www.quality-gate.com/

- Applying statistical methods we investigate the effect of code refactorings on the internal code quality attributes (i.e. source code metrics).

II. Related Work

There are several studies that have investigated the relationship between practical refactoring activities and the software quality through different quality attributes. Many of them used the Ref-Finder tool [6] to extract refactorings from real-life open-source systems, similar to us.

Bavota et al. [5] made observations on the relations between metrics/code smells and refactoring activities. They mined the evolution history of 2 open-source Java projects and revealed that refactoring operations are generally focused on code components for which quality metrics do not suggest there might be a need for refactoring operations. In contrast to this work, by considering maintainability instead of code smells, we found significant, but not very strong relationship with refactoring activities. Bavota et al. also provide a large refactoring dataset with 15,008 refactoring operations, but it contains file level data only without exact line information. Our open dataset contains method level information as well and refactoring instances are completely traceable.

In a similar work to ours Murgia et al. [10] studied whether highly coupled classes are more likely to be targets of refactoring than less coupled ones. Classes with high fan-out (and relatively low fan-in) metric consistently showed to be targets of refactoring, implying that developers may prefer to refactor classes with high outgoing rather than high incoming coupling.

Kataoka et al. [11] also focused on the coupling metrics to evaluate the impact of refactorings and showed that their method is effective in quantifying the impact of refactoring and helped them to choose the appropriate refactoring types.

Contrary to the previous two works, we did not select a particular metric to assess the effect of refactorings, but rather used statistical tests to find those metrics that change meaningfully upon refactorings. This way we could identify that complexity, size, and clone metrics also play an important role in connection with refactorings.

The approach presented by Tsantalis et al. [12] investigates the refactoring activity as part of the software engineering process and not its effect on code quality. They have identified that the refactoring decision making and application is often performed by individual refactoring "managers". They found a strong alignment between refactoring activity and release dates and revealed that the development teams apply a considerable amount of refactorings during testing periods.

Measuring clones and investigating how refactoring affects them has also attracted a lot of research efforts. Our dataset also includes clone metrics, thus clone oriented refactoring examinations can also be performed.

Choi et al. identified [13] that merged code clone token sequences and differences in token sequence lengths vary for each refactoring pattern. They found that extract method and replace method with method object refactorings are the most popular when developers perform clone refactoring. Again Choi et al. [14] present an investigation of actual clone refactorings performed in open-source development. The characteristics of refactored clone pairs were also measured. From the results, they again confirmed that clone refactorings are mostly achieved by replace method with method object and extract method. We also found that refactoring activities improve clone metrics, but we did not distinguish the effect of different refactoring types in this work. Nonetheless, the required data is available in the assembled dataset to conduct similar investigations.

Similarly to us, Murphy-Hill et al. [15] empirically analyzed how developers refactor in practice. They found that refactoring tools are rarely used: 11% by Eclipse developers and 9% by Mylyn developers. Unlike the above paper, we do not focus on how refactorings are introduced (i.e. manually or using a tool), rather on their effect on source code.

III. Approach

In order to aid research on source code refactorings in practice, we constructed an excessive public dataset of source code metrics and applied refactorings. As a first step in utilizing the dataset, we investigate the connection between the number of refactorings affecting the classes of the programs and their various quality properties.

A. Dataset Construction

The dataset contains data of release versions of 7 open-source Java systems available on GitHub. Table I provides details about the projects, their names, Git URLs, number of releases and the covered time interval by the releases.

Table I
The systems included in the refactoring dataset

System	Git URL	# Rel.	Time interval
antlr4	https://github.com/antlr/antlr4	5	21/01/13-22/01/15
junit	https://github.com/junit-team/junit	8	13/04/12-28/12/14
mapdb	https://github.com/jankotek/MapDB	6	01/04/13-20/06/15
mcMMO	https://github.com/mcMMO-Dev/mcMMO	5	24/06/12-29/03/14
mct	https://github.com/nasa/mct	3	30/06/12-27/09/13
oryx	https://github.com/cloudera/oryx	4	11/11/13-10/06/15
titan	https://github.com/thinkaurelius/titan	6	07/09/12-13/02/15

These projects were found ideal for our research purposes because of the adequate number of release versions and the amount of the code modifications between two adjacent releases. We investigated 3 to 8 releases of each project. For every release version of every project, class and method-level metrics and the number of refactorings grouped by refactoring types (e.g. pull up method, add parameter) are provided; 23 refactoring types on class-level, and 19 on method-level. Table II provides an overview of the total number of classes, methods and refactorings contained in the current dataset.

The release versions of the projects were selected bearing in mind the amount of code modifications performed from a release to the next one. As long as the selected versions do not differ enough, the number of refactorings mined between them are rather low, in most of the cases zero, which do not carry any additional information and e.g. machine learning algorithms cannot be performed efficiently. On the other hand, there has to be a considerable number of release

Table II
TOTAL NUMBER OF CLASSES, METHODS AND REFACTORINGS

System	# Classes	# Methods	# Refactorings
antlr4	622	5,280	248
junit	1,267	4,124	553
mapdb	850	6,180	2,973
mcMMO	505	4,767	62
mct	2,175	11,765	763
oryx	551	2,592	121
titan	2,429	14,214	3,152
Total	**8,399**	**48,922**	**7,872**

versions to make it possible to investigate how the metrics and refactoring activities are varying over time. We found that an approximately half-year interval between release dates is an acceptable heuristic that provides a sufficient amount of code changes which we consider appropriate for most of the research goals. Thus, in case of every project, we dropped those release versions that were too close to each other in time. In the end, we left 3 to 8 release versions in the dataset depending on the considered project.

We used the Ref-Finder refactoring reconstruction tool [6] to reveal refactorings between two adjacent release versions. We note that the precision of this tool is not perfect, 79% according to the Ref-Finder authors [16]. A Ref-Finder analysis on two versions of the source code can be started manually and the results are displayed in the Eclipse IDE. In order to run the analysis over the release versions we have selected, we improved Ref-Finder to be able to perform an automatic batch analysis. To make the further examinations possible, we also implemented an export feature in Ref-Finder that writes the revealed refactorings and all of their attributes into CSV files for each refactoring type. To set up the final dataset we mapped the refactorings to the affected code elements (class and method) and counted their numbers. More specifically, if a code element was touched by a refactoring, the refactoring was counted to that code element. If a method was affected by a refactoring it was counted to the class of this method too. In every release version the accounted refactoring numbers indicate how many refactorings from various types were performed that affected the considered code element between the current release and the previous one.

Besides code refactorings, more than 50 types of static source code metrics were also extracted for every class and method of the systems with the help of the SourceMeter static code analysis tool. As an additional metric, we assigned the so-called *relative maintainability index* (RMI) for each code element as measured by the QualityGate SourceAudit tool [17]. The RMI of a source code element reflects its level of maintainability compared to the other code elements of the system. It is similar to the well-known maintainability index [18], but it is calculated using dynamic thresholds from a benchmark database instead of a fixed formula. The technical details of the RMI can be found in our earlier work [19].

The dataset is part of the *tera-PROMISE* repository [8]: http://openscience.us/repo/refactoring/refact.html,

and available online at the following location as well: http://www.inf.u-szeged.hu/~ferenc/papers/RefactDataSet/

B. Data Analysis Methodology

To answer our two research questions we used the new dataset in the following way. For RQ1, we performed a correlation analysis on the RMI values of the classes and the number of refactorings affecting these classes. We took the RMI values from release x_i, and the number of refactorings from release x_{i+1}. This way we assessed whether poor quality classes got refactored more intensively than other classes or not. We note that in the current study we deal only with classes, but the dataset contains RMI and refactoring values for methods as well. Since we cannot assume anything about the distribution of the maintainability indices nor the number of refactorings, we performed a Spearman rank correlation analysis.

For answering RQ2, first we calculated the differences of the metric values between the subsequent releases. In most cases negative differences mean an improvement, as lower metric values (e.g. lower complexity) are better. To decide whether there is a significant difference among the metric decreases in the refactored and non-refactored classes, we run a Mann-Whitney U test. The result of this test gave us a hint on what are those metric values that improve significantly upon refactorings. To estimate the volume of these metric changes, we calculated two effect size measures as well, namely the odds ratio (OR) and the Cliff's delta value (δ).

IV. RESULTS

In this section we summarize the assessment results of the assembled refactoring dataset regarding software maintainability. First, we describe the results of the analysis on the maintainability of refactored classes to answer RQ1. Afterwards, we present the findings on the effect of refactorings on source code metrics to answer RQ2.

A. The Maintainability of Refactored Classes

To answer RQ1, we performed a correlation analysis between the number of refactorings affecting the classes and their maintainability indices in the previous release (as described in Section III). Figure 1 depicts the Spearman correlation coefficients between the RMI values in release x_i and the number of refactorings affecting the corresponding classes in release x_{i+1}.

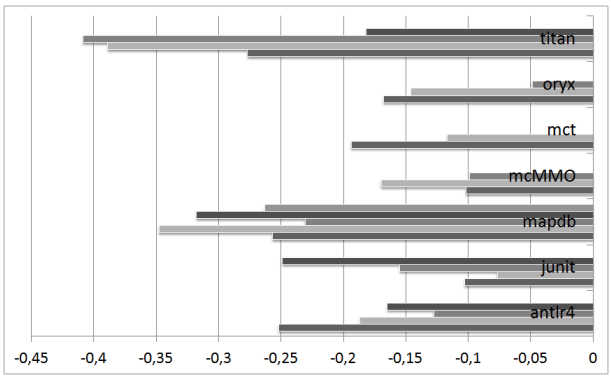

Figure 1. Correlation of maintainability and number of refactorings in classes

System name	CI		WMC		NOI		RFC		TCLOC		TLLOC		TNOS	
	R_{impr}	NR_{impr}	R_{impr}	NR_{impr}	R_{impr}	NR_{impr}	R_{impr}	NR_{impr}	R_{impr}	NR_{impr}	R_{impr}	NR_{impr}	R_{impr}	NR_{impr}
antlr4	3.57%	0.38%	9.52%	0.82%	7.14%	0.65%	8.33%	0.76%	3.57%	0.65%	9.52%	4.57%	8.33%	1.20%
junit	3.17%	0.53%	3.97%	0.20%	4.76%	0.90%	6.35%	0.90%	8.73%	0.74%	7.94%	0.94%	8.73%	0.44%
mapdb	13.46%	1.27%	10.58%	3.70%	16.35%	3.78%	14.42%	5.06%	18.27%	2.17%	11.54%	5.01%	10.58%	4.21%
mcMMO	20.00%	3.95%	20.00%	8.81%	30.00%	8.40%	30.00%	10.73%	30.00%	6.48%	50.00%	13.26%	30.00%	10.63%
mct	2.83%	0.15%	0.94%	0.22%	1.89%	0.32%	0.94%	0.39%	0.94%	0.05%	2.83%	0.49%	3.77%	0.32%
oryx	1.96%	0.32%	7.84%	1.27%	3.92%	0.32%	3.92%	0.25%	5.88%	0.83%	9.80%	2.73%	5.88%	2.35%
titan	2.69%	0.71%	11.98%	2.17%	25.41%	5.56%	25.41%	5.56%	5.79%	1.16%	14.05%	3.17%	14.46%	2.92%
Average	6.81%	1.04%	9.26%	2.45%	12.78%	2.85%	12.77%	3.38%	10.45%	1.73%	15.10%	4.31%	11.68%	3.15%

Figure 2. Metric improvements heat map

As can be seen, all the values are negative, meaning that the worse the maintainability of a class is the more refactorings touch it. Although the coefficients are moderate, they are consistently negative and significant at the level of 0.05 (except for the two lowest values of mcMMO and oryx). There are less correlation coefficients than releases for some systems because we were unable to calculate them when Ref-Finder found no refactorings between two releases, which happened a couple of times.

Answer to RQ1: Based on the findings on our dataset it seems that classes with poor maintainability are subject to higher number of refactorings during their lifetime.

B. The Effect of Refactorings on Source Code Metrics

We found that refactorings affect poorly maintainable code, so the question arises whether applying refactorings really improves the internal quality of the code? And if yes, what are the source code metrics that show the highest improvement (i.e. decrease significantly)?

According to the process described in Section III, we first calculated the metric value differences for every class between the adjacent releases. Then, we grouped these metric difference values into two groups: in the first group we put the metric differences of classes touched by at least one refactoring, and in the second group the metric differences of non-refactored classes. Finally, we analyzed which metrics show significant differences between the values of the two groups with the help of Mann-Whitney U test.

Table III
THE RESULTS OF THE MANN-WHITNEY U TEST (P-VALUES)

System name	CI	WMC	NOI	RFC	TCLOC	TLLOC	TNOS
antlr4	**0.033**	0.428	**0.010**	**0.031**	0.136	**0.002**	0.122
junit	0.728	**0.042**	0.170	N/A	**0.012**	0.101	0.113
mapdb	**0.030**	**0.006**	**0.005**	**0.000**	**0.05**	**0.000**	**0.000**
mcMMO	**0.005**	0.608	**0.003**	**0.013**	0.066	0.257	0.594
mct	0.905	0.200	N/A	0.941	N/A	0.115	0.703
oryx	0.667	0.575	0.381	0.533	0.800	0.743	0.159
titan	**0.022**	**0.016**	**0.000**	**0.000**	0.260	**0.002**	**0.042**

Out of 50+ source code metrics, the ones listed in Table III had the lowest p-values, meaning that the differences in the metric value changes for refactored and non-refactored classes are the most significant for these metrics. We observed that coupling metrics, namely Response Set for Classes (RFC) and Number of Outgoing Invocations (NOI) indeed decrease significantly upon refactorings in accordance with the previous findings of other studies [10], [11]. But besides coupling, we found a significant decrease in size metrics as well, namely in the case of Total Logical Lines of Code (TLLOC) and Total Number of Statements (TNOS). This finding is not really surprising, nor that the complexity metric Weighted Methods per Class (WMC) also decreased significantly. What is more interesting is that the number of Clone Instances (CI) also decreased, thus refactoring activity seems to remove copy-paste code parts in practice. Finally, an interesting result is that the Total Comment Lines of Code (TCLOC) also decreased significantly. This might mean a degradation in maintainability if the developer did not take the time to document the modifications, but it can also mean an improvement if out-of-date comments were removed, or even better, if the developer adhered to the clean code principle.

To get an impression about the magnitude of the differences between the metric value decreases of the refactored and non-refactored classes, we calculated the ratio of classes with metric value decreases within the two groups. The results are depicted on the heat map shown in Figure 2. The left columns contain the proportion of refactored classes having decreased metric value, while right columns show the same ratio for the non-refactored classes. The darker values mark higher ratios. As can be seen, all the dark values are in the left columns, thus metric value decreases are far more frequent in the group of refactored classes than in non-refactored classes.

To quantify what we observed visually in the heatmap, we calculated the odds ratio (OR) and Cliff's delta (δ) effect size measures. The detailed results are presented in Table IV. The average OR values vary between approximately 4-9, which means that on average the chances of a metric value decrease is 4-9 times higher in the classes affected by refactorings than in the non-refactored classes. The Cliff's δ values suggest a similar conclusion, though not as obviously as the OR values. Cliff's δ measures how often the values in one distribution are larger than the values in a second distribution. It ranges from -1 to 1 and is linearly related to the Mann-Whitney U statistic, however it captures the direction of the difference in its sign as well. Simply speaking, if Cliff's δ is a positive number, the metric value differences (thus the metric value decreases) are higher in the refactored classes, while negative value means that the metric value differences are higher in the non-refactored classes. The closer the δ is to $|1|$, the more values are larger in one group than the values in the other group. Generally, most of the Cliff's δ values are positive (i.e. the average δ values are positive for every metric). Nonetheless, there are several large negative δ values for the CI and WMC metrics. This might suggest that cloned code and complexity is decreased by other targeted changes, while refactorings often

Table IV
EFFECT SIZE MEASURES

System name	CI		WMC		NOI		RFC		TCLOC		TLLOC		TNOS	
	OR	δ	OR	δ	OR	δ	OR	δ	OR	δ	OR	δ	OR	δ
antlr4	9.38	-0.86	11.68	0.21	10.95	0.75	10.95	0.58	5.47	0.61	2.09	0.64	6.97	0.40
junit	6.04	0.13	20.14	0.69	5.31	0.35	7.07	0.06	11.73	0.50	8.43	0.69	19.94	0.35
mapdb	10.56	0.40	2.86	0.50	4.32	0.33	2.85	0.55	8.43	0.30	2.30	0.78	2.51	0.78
mcMMO	5.07	0.96	2.27	-0.22	3.57	0.89	2.80	0.79	4.63	0.63	3.77	0.30	2.82	0.19
mct	19.33	0.06	4.30	-0.89	5.95	0.00	2.42	0.13	19.33	0.00	5.80	0.59	11.90	-0.15
oryx	6.17	-0.40	6.17	0.2	12.34	0.50	15.42	0.50	7.12	0.13	3.59	-0.10	2.50	0.50
titan	3.76	0.39	5.53	0.15	4.92	0.24	4.57	0.30	4.98	0.15	4.43	0.26	4.95	0.17
Average	8.62	0.10	7.56	0.09	6.76	0.44	6.58	0.42	8.81	0.33	4.34	0.45	7.37	0.32

have a side effect to remove code clones or reduce complexity as well. However, this phenomenon needs further investigation.

Answer to RQ2: We found that size (TLLOC, TNOS), coupling (RFC, NOI), clone (CI), complexity (WMC) and comment (TCLOC) related metrics decrease the most in refactored classes. Regarding the volumes of the differences, we can say that for these metrics the average chances of a decrease is 4-9 times higher in the classes affected by refactorings than in the non-refactored classes.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we present a publicly available dataset which is indented to assist the research of refactoring activities in practice. The dataset contains fine-grained refactoring information and more than 50 types of source code metrics for 37 releases of 7 open-source systems at class and method level. By utilizing the dataset, we investigated the relationship between maintainability and refactoring activities, and we also assessed how refactorings affect different source code metrics. We found that classes with poor maintainability are subject to more refactorings in practice than classes with higher technical quality. Considering metrics, number of clone instances, complexity, and coupling have improved, although comment related metrics decreased. We found a significant decrease in size metrics as well. The possible utilization of the assembled dataset goes much beyond the early investigations presented in this paper. We plan to reveal more complex phenomena in connection with practical refactorings, especially the relationship between bugs and refactoring activities.

ACKNOWLEDGMENT

This work was partially supported by the European Union project "REPARA – Reengineering and Enabling Performance And poweR of Applications", project number: 609666.

REFERENCES

[1] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[2] E. van Emden and L. Moonen, "Java Quality Assurance by Detecting Code Smells," in *Proceedings of the 9th Working Conference on Reverse Engineering*, 2002, pp. 97–106.

[3] F. A. Fontana and S. Spinelli, "Impact of Refactoring on Quality Code Evaluation," in *Proceedings of the 4th Workshop on Refactoring Tools*, ser. WRT '11. New York, NY, USA: ACM, 2011, pp. 37–40.

[4] R. Peters and A. Zaidman, "Evaluating the Lifespan of Code Smells using Software Repository Mining," in *Proceedings of the 16th European Conference on Software Maintenance and Reengineering (CSMR)*, March 2012, pp. 411–416.

[5] G. Bavota, A. De Lucia, M. Di Penta, R. Oliveto, and F. Palomba, "An Experimental Investigation on the Innate Relationship Between Quality and Refactoring," *Journal of Systems and Software*, vol. 107, pp. 1 – 14, 2015.

[6] M. Kim, M. Gee, A. Loh, and N. Rachatasumrit, "Ref-Finder: a Refactoring Reconstruction Tool Based on Logic Query Templates," in *Proceedings of the 18th ACM SIGSOFT international symposium on Foundations of software engineering (FSE'10)*, 2010, pp. 371–372.

[7] T. Bakota, P. Hegedűs, P. Körtvélyesi, R. Ferenc, and T. Gyimóthy, "A Probabilistic Software Quality Model," in *Proceedings of the 27th IEEE International Conference on Software Maintenance (ICSM)*, Sept. 2011, pp. 243–252.

[8] T. Menzies, R. Krishna, and D. Pryor, "The Promise Repository of Empirical Software Engineering Data," 2015. [Online]. Available: http://openscience.us/repo

[9] M. D'Ambros, M. Lanza, and R. Robbes, "An Extensive Comparison of Bug Prediction Approaches," in *Proceedings of 7th IEEE Working Conference on Mining Software Repositories (MSR)*. IEEE CS Press, 2010, pp. 31–41.

[10] A. Murgia, R. Tonelli, M. Marchesi, G. Concas, S. Counsell, J. McFall, and S. Swift, "Refactoring and its Relationship with Fan-in and Fan-out: An Empirical Study," in *Proceedings of the 16th European Conference on Software Maintenance and Reengineering (CSMR)*, March 2012, pp. 63–72.

[11] Y. Kataoka, T. Imai, H. Andou, and T. Fukaya, "A Quantitative Evaluation of Maintainability Enhancement by Refactoring," in *Proceedings of the International Conference on Software Maintenance*, 2002, pp. 576–585.

[12] N. Tsantalis, V. Guana, E. Stroulia, and A. Hindle, "A Multidimensional Empirical Study on Refactoring Activity," in *Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research*, ser. CASCON '13. Riverton, NJ, USA: IBM Corporation, 2013, pp. 132–146.

[13] E. Choi, N. Yoshida, and K. Inoue, "An Investigation into the Characteristics of Merged Code Clones during Software Evolution," *IEICE Transactions on Information and Systems*, vol. 97, no. 5, pp. 1244–1253, 2014.

[14] ——, "What Kind of and How Clones Are Refactored?: A Case Study of Three OSS Projects," in *Proceedings of the 5th Workshop on Refactoring Tools*, ser. WRT '12. New York, NY, USA: ACM, 2012, pp. 1–7.

[15] E. Murphy-Hill, C. Parnin, and A. P. Black, "How We Refactor, and How We Know It," *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 5–18, Jan 2012.

[16] K. Prete, N. Rachatasumrit, N. Sudan, and K. Miryung, "Template-based Reconstruction of Complex Refactorings," in *Proceedings of the 2010 IEEE International Conference on Software Maintenance (ICSM)*, Sept 2010, pp. 1–10.

[17] T. Bakota, P. Hegedüs, I. Siket, G. Ladányi, and R. Ferenc, "QualityGate SourceAudit: A Tool for Assessing the Technical Quality of Software," in *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering, CSMR-WCRE 2014, Antwerp, Belgium, February 3-6, 2014*, 2014, pp. 440–445.

[18] P. Oman and J. Hagemeister, "Metrics for Assessing a Software System's Maintainability," in *Proceedings of the International Conference on Software Maintenance*. IEEE CS Press, 1992, pp. 337–344.

[19] P. Hegedűs, T. Bakota, G. Ladányi, C. Faragó, and R. Ferenc, "A Drill-Down Approach for Measuring Maintainability at Source Code Element Level," *Electronic Communications of the EASST*, vol. 60, 2013.