2018 25th Asia-Pacific Software Engineering Conference (APSEC)

# An Investigation of the Relationship Between Extract Method and Change Metrics: A Case Study of JEdit

Eunjong Choi*, Daiki Tanaka*, Norihiro Yoshida†, Kenji Fujiwara‡, Daniel Port§ and Hajimu Iida*
*Nara Institute of Science and Technology, {choi, tanaka.daiki.sx4}@is.naist.jp, iida@itc.naist.jp
†Nagoya University, Japan, yoshida@ertl.jp
‡National Institute of Technology, Toyota College, Japan, fujiwara@toyota-ct.ac.jp
§University of Hawaii at Manoa, USA, dport@hawaii.edu

***Abstract*—Extract Method is one of the most widely used refactoring patterns. So far, low quality of source code has been regarded as an indicator for Extract Method opportunities. However, recent studies showed that there is no clear relationship between source code quality and Extract Method. Change metrics can be indicators for Extract Method because the characteristics of software evolution strongly affect software quality. However, there has been no study that investigated the relationship between change metrics and Extract Method. In this study, we conducted two studies investigating the relationship between Extract Method and change metrics. As a result, we found that (1) change metrics have a clear relationship with Extract Method and (2) both product and change metrics are necessary to recommend candidates for Extract Method with high accuracy.**

***Index Terms*—Extract Method, Change Metrics, Refactoring Recommendation**

## I. Introduction

Refactoring is the process of changing a program to improve its internal structure, without changing the external behavior of the program [7]. Extract Method is one of the most widely performed refactoring patterns [14]. It extracts a part of a method that implements a distinct functionality into a new method to remove bad smells (i.e., symptoms of poor design and implementation choices) [7]. So far, many studies have been done on recommending candidates for Extract Method based on the values of product metrics (e.g., lines of code, lack of cohesion of Methods [5]) to eliminate bad smells [6], [20]. However, recent studies have revealed that no clear relationship exists between Extract Method and product metrics [2], [22]. For instance, Bavota et al. found that Extract Method has a distinct tendency to be performed on classes with only a few high product metrics, such as lines of code [2]. Change metrics measure the change history of source code, such as the number of changes and distinct committers [10]. Generally, the characteristics of software evolution strongly affect software quality. For example, high numbers of distinct committers hinder software quality [3]. Therefore, change metrics could also be regarded as indicators for Extract Method opportunities. However, as far as we know, no study has investigated the relationship between Extract Method and change metrics.

To fill this gap, we conducted two studies investigating the relationship between Extract Method and change metrics in *JEdit*[1], a large-scale open source software system written in Java. Specifically, the investigations aim to address the following two Research Questions (RQs):

- **RQ1: Are Extract Method performed on source code with significantly high change metric values?**
- **RQ2: Can change metrics recommend candidates for Extract Method?**

To answer these RQs, at first, we mined refactoring histories to compare values of change metrics between refactored methods/classes (hereafter called **extracted entities**) and methods/classes to which no Extract Method has been performed (hereafter called **not-extracted entities**). Then, we evaluated six approaches using not only change metrics but also product metrics to check whether change metrics can be used for recommending candidates for Extract Method.

This paper provides the following two contributions:

- We have empirically investigated the quantitative differences in change metrics between **extracted entities** and **not-extracted entities**.
- We evaluated six approaches for recommending candidates for Extract Method with different metrics. The results showed that the highest accuracy can be achieved by an approach using both change and product metrics.

The remainder of this article is organized into the following sections. At first, Section II explains the background and related work of this study. Section III then details our design for investigating the relationship between Extract Method and change metrics Next, Section IV discusses a motivation, approach, and result of our investigations. Section V finally conclues this paper with future work.

## II. Background and Related Work

Extract Method improves software quality by extracting a part of a method that implements a distinct functionality into a new method [7]. Figure 1 represents an example of Extract Method [7]. In this figure, a code fragment that prints

[1]http://jedit.org/

978-1-7281-1970-0/18/$31.00 ©2018 IEEE
DOI 10.1109/APSEC.2018.00084

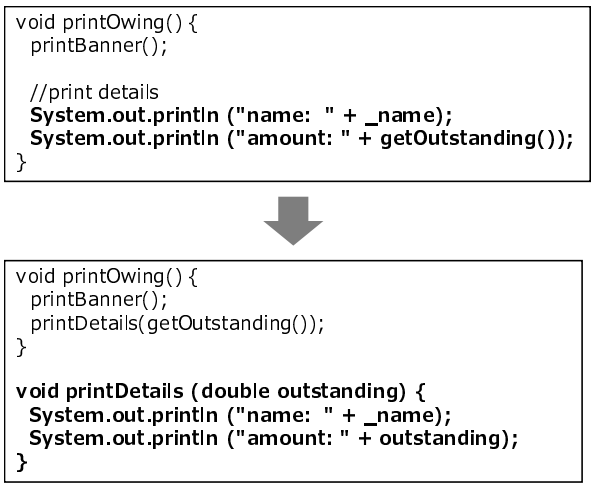


Fig. 1. Example of Extract Method [7]

details in a *PrintOwing()* method was extracted into a new method namely *PrintDetails()* (shown in bold face) by Extract Method. Moreover, the old code fragment was replaced with a call to a new method (i.e., *PrintDetails()* method).

Up to now, several tools have been developed for recommending candidates for Extract Method. For example, Murphy-Hill and Black developed a tool that supports the code selection for Extract Method in Eclipse by analyzing abstract syntax tree of source code [13]. Tsantalis and Chatzigeorgiou developed a tool for supporting Extract Method by identifying cohesive regions from a Java method based on program slicing [20]. Silva *et al.* also proposed a tool that recommends candidates for Extract Method by ranking candidates according to the static dependencies [16]

Moreover, a number of studies have investigated refactoring histories with the aim of identifying the factors that motivate practitioners to perform refactoring. Wang interviewed 10 practitioners about their motivation for performing refactoring and revealed several motivations including contextual and individual factors [21]. Silva *et al.* detected recently performed refactoring from 124 Java projects on the *GitHub* repository, and asked the practitioners to explain the reasons behind their refactoring [17]. As a result, they found out 44 distinct motivations for 12 refactoring patterns. In particular, they identified 11 different motivations for Extract Method including "Extract reusable method". Bavota *et al.* investigated the extent to whether refactoring was performed to classes exhibiting bad smells [2]. The results of their investigation show that only 42% of refactoring was performed on code entities affected by bad smells.

However, these above-mentioned studies focus only on the product metrics for Extract Method. Meanwhile, our study puts emphasis on change metrics because the characteristics of software evolution strongly affect software quality.

TABLE I
STATISTICS OF SELECTED SYSTEM

| Project | #revisions | Time period | LOC |
|---|---|---|---|
| JEdit | 7,707 | 02 Sep.2001 - 13 Feb.2016 | 97,116 - 313,706 |

## III. INVESTIGATION DESIGN

In this section, we describe our investigation design to answer *RQ1: Are Extract Method performed on source code with significantly high change metric values?* and *RQ2: Can change metrics recommend candidates for Extract Method?*. In Section III-A, we describe the selected system and metrics for the investigation in details. In Section III-B, we explain a tool that used to detect Extract Method.

### A. Selected System and Metrics

**Selected System.** We investigated 7,707 revisions of a large-scale open source software written in Java, *JEdit*, a text editor. We selected *JEdit* as a target of our investigations because it has been widely used to investigate the characteristics of refactoring [2], [12], [19] . Table I provides statistic data on investigated system.

**Change metrics.** In this study, we selected six change metrics to calculate change history in *JEdit*. They are NC, NDC, AG, ADD, DEL, and CHURN metrics. These metrics were chosen because (1) we assumed that they highly influence the software quality and (2) they are frequently used in the previous studies [3], [11], [15]. The details of the selected metrics are as follows:

- **NC:** constitutes the number of changes. The NC metric is calculated by $\sum_{i=1}^{r} isChg(M_{r-i})$, where $M_r$ is a program entity (i.e., method/class) at the $r$th revision and $isChg()$ returns 1 if the program entity was changed and 0 otherwise.
- **NDC:** returns the number of distinct committers. The NDC metric is calculated by $\left|\bigcup_{i \in chgRevs(M_r)} au_i\right|$, where $M_r$ is a program entity at the $r$th revision, $chgRevs()$ returns a set of changed revisions until the $r$th revision, and $au_i$ returns a commiter ID at the $i$th revision.
- **AG:** calculates age of the revisions. The AG metric is calculated by $ts(r) - ts(firstRev(M_r))$, where $M_r$ is a program entity at the $r$th revision, $firstRev()$ returns a revision that the program entity was the first committed, and $ts$(r) returns the time in seconds when the revision $r$ was committed.
- **ADD:** quantifies the sum of the lines of added code The ADD metric is calculated by $\sum_{i=1}^{r}(isChg(M_{r-i}) \cdot addLines(M_{r-i}))$, where $M_r$ is a program entity at the $r$th revision, $isChg()$ returns 1 if the program entity was changed and 0 otherwise, and $addLines()$ returns the number of added lines of code of the parameter.
- **DEL:** means the sum of the lines of deleted code The DEL metric is calculated by $\sum_{i=1}^{r}(isChg(M_{r-i}) \cdot delLines(M_{r-i}))$, where $M_r$ is a program entity at the $r$th revision, $isChg()$ returns 1 if the program entity

was changed and 0 otherwise, and $delLines()$ returns the number of deleted lines of code of the parameter.

- **CHURN:** indicates the sum of changed lines of code. The CHURN metric is calculated by $ADD(M_r) + DEL(M_r)$, where $M_r$ is a program entity at the $r$th revision, $ADD$ returns the value of the ADD metric, and $DEL$ returns the value of the DEL metric.

### B. Extract Method detector

To detect Extract Method, we adopted a tool called Kenja, which detects Extract Method in multiple revisions. Kenja was selected because its high accuracy in detecting Extract Method[2] [8]. The overview process of Kenja is as follows:

1) Extract syntax information on classes and methods in the software repository using Historage, a fine-grained version control system based on Git [9].
2) Find a method $m_{new}$ which did not exist at revision $i$ but was newly appeared at revision $i+1$.
3) When a method $m$ at revision $i$ and $i+1$ is defined as $m_i$ and $m_{i+1}$, respectively, identify a method $m$ which (1) exists between $i$ and $i+1$ revisions, (2) $m_i$ contains statements $S_d$ that were deleted between $i$ and $i+1$ revisions and (3) $m_{i+1}$ includes statements $S_a$ that are newly added between $i$ and $m_{i+1}$ revisions. Note that $m_{i+1} \equiv (m_i - S_d) \cup S_a$.
4) Check whether $S_a$ contains a call statement to $m_{new}$.
5) Calculate the token-based 2-shingles similarity [4] between $S_d$ and the body of $m_{new}$. If the similarity is higher than a threshold value, the tool determines that the Extract Method has been performed to $m_i$ and it has been transformed into $m_{i+1}$ and $m_{new}$.

## IV. Investigation

In this section, we present overview, motivation, approach, and results of our investigations with respect to two RQs, mentioned in Section I.

### A. RQ1: Are Extract Method performed on source code with significantly high change metric values?

**Overview.** We compared change metrics between **extracted entities** and **not-extracted entities** in the investigated system, *JEdit*.

**Motivation.** Software changes during software development and maintenance affect software quality. Therefore, software evolution can be indicators for Extract Method opportunities. However, as far as we know, no previous literature has investigated the relationship between Extract Method and change metrics.

**Approach.** To compare change metrics between **extracted entities** and **not-extracted entities**, at first, we extracted 7,707 revisions from 2nd Sep. 2001 until 13th Feb. 2016 in *JEdit*. We then separated the extracted revisions into two periods for measuring change metrics (i.e., *the metrics calculation period*) and detecting Extract Method (i.e., *the refactoring detection period*). To explicitly separate *the metrics calculation period* and *refactoring detection period*, we set an arbitrary revision split point *r*. To ensure the generality of our investigation results, we used several revision split points: 1:9, 2:8, 3:7, 4:6, and 5:5, as shown in the first column in Table II. In *the refactoring detection period*, we searched for Extract Method performed between *r+1*th and the final revisions using Kenja, explained in Section III-B. The EM and None-EM columns in Table II show the number of **extracted entities** and **not-extracted entities** at each revision split point, respectively. Based on this data, we then created a **dataset** by selecting all detected Extracted Method data (i.e., **extracted entities**) and the same numbers of randomly selected data from the **not-extracted entities**. In *the metrics calculation period*, we applied selected six change metrics, explained in Section III-A, at the class-level and method-level to the created dataset.

Furthermore, we performed a *Mann-Whitney test*, a nonparametric significance test to analyze the statistical significance of the differences in the median values of the change metrics between **extracted entities** and **not-extracted entities**. We conducted it with a confidence level of 0.05 (i.e., $p-value < 0.05$).

**Results & Discussion.** Table III shows the p-value obtained by the *Mann-Whitney test* from comparing the median values

[2]The precision was 0.96 and the recall was 0.86 with the similarity threshold 0.3. for detecting Extract Method from 4,475 revisions of JEdit.

TABLE II
Investigated revisions

| Revision split point | # EM | # None-EM |
|---|---|---|
| 11 Jan. 2003 (1:9) | 5 | 4,102 |
| 28 Mar. 2003 (2:8) | 7 | 5,073 |
| 08 Jun. 2005 (3:7) | 7 | 3,753 |
| 22 Apr. 2006 (4:6) | 14 | 4,960 |
| 05 Feb. 2007 (5:5) | 10 | 5,639 |

TABLE III
Investigated results for change metrics

| Metrics | Level | Category | Median | p value |
|---|---|---|---|---|
| NC | Class | EM | 48.000 | $\boldsymbol{p < 0.05}$ |
| | | None-EM | 4.000 | |
| | Method | EM | 3.093 | $0.05 < p$ |
| | | None-EM | 1.000 | |
| NDC | Class | EM | 1.000 | $\boldsymbol{p < 0.05}$ |
| | | None-EM | 1.000 | |
| | Method | EM | 1.000 | $0.05 < p$ |
| | | None-EM | 1.000 | |
| AG | Class | EM | 118,900,000.000 | $0.05 < p$ |
| | | None-EM | 73,170,000.000 | |
| | Method | EM | 86,750,000.000 | $0.05 < p$ |
| | | None-EM | 73,170,000.000 | |
| ADD | Class | EM | 812.000 | $\boldsymbol{p < 0.05}$ |
| | | None-EM | 9.000 | |
| | Method | EM | 5.000 | $\boldsymbol{p < 0.05}$ |
| | | None-EM | 0.000 | |
| DEL | Class | EM | 951.000 | $\boldsymbol{p < 0.05}$ |
| | | None-EM | 23.000 | |
| | Method | EM | 1.000 | $0.05 < p$ |
| | | None-EM | 0.000 | |
| CHURN | Class | EM | 1,690.000 | $\boldsymbol{p < 0.05}$ |
| | | None-EM | 60.000 | |
| | Method | EM | 5.000 | $\boldsymbol{p < 0.05}$ |
| | | None-EM | 0.000 | |

TABLE IV
LIST OF PRODUCT METRICS USED

| Metrics | Level | Explanation |
|---|---|---|
| PAR | Method | Number of Parameters |
| VG | Method | McCabe Cyclomatic Complexity |
| NBD | Method | Nested Block Depth |
| MLOC | Method | Method Lines of Code |
| WMC | Class | Weighted methods per Class |
| DIT | Class | Depth of Inheritance Tree |
| LCOM | Class | Lack of Cohesion of Methods |
| NSC | Class | Number of Children |

TABLE V
INVESTIGATED RESULTS FOR PRODUCT METRICS

| Metrics | Level | Category | Median | p value |
|---|---|---|---|---|
| PAR | Method | EM | 2.000 | $\boldsymbol{p < 0.05}$ |
| | | None-EM | 1.000 | |
| VG | Method | EM | 6.000 | $\boldsymbol{p < 0.05}$ |
| | | None-EM | 2.000 | |
| NBD | Method | EM | 4.000 | $\boldsymbol{p < 0.05}$ |
| | | None-EM | 1.000 | |
| MLOC | Method | EM | 30.000 | $\boldsymbol{p < 0.05}$ |
| | | None-EM | 3.000 | |
| WMC | Class | EM | 144.000 | $0.05 < p$ |
| | | None-EM | 57.500 | |
| DIT | Class | EM | 2.000 | $0.05 < p$ |
| | | None-EM | 2.000 | |
| LCOM | Class | EM | 0.761 | $\boldsymbol{p < 0.05}$ |
| | | None-EM | 0.694 | |
| NSC | Class | EM | 0.000 | $\boldsymbol{p < 0.05}$ |
| | | None-EM | 0.000 | |

of the change metrics between **extracted entities** and **not-extracted entities**. In this table, the Level column represents the levels of the metrics (i.e., class-level or method-level), the Category column represents the target categories of the dataset (i.e., EM or None-EM), and the Median column represents the median values of the change metrics. Moreover, the p value column indicates the test results with statistically significant values shown in bold face. From this table, we can immediately see that several change metrics at the class-level, as well as the method-level, have a higher chance of being involved in Extract Method. In particular, NC, NDC, ADD, DEL and CHURN metrics involving changes in the source code have a clearer relationship than other metrics. This result concurs with a finding by Silva et al. that Extract Method is performed with various motivations for source code changes [18]. These results indicate that NC, NDC, ADD, DEL, and CHURN metrics at the class-level as well as ADD and CHURN metrics at the method-level show a clear relationship with Extract Method.

**Answer for RQ1. Yes**, Extract Method has a tendency to be performed on methods with high values of ADD and/or CHURN metrics. It also has a tendency to be performed on classes with high values of NC/NDC/ADD/DEL, and/or CHURN metrics.

TABLE VI
PRECISION, RECALL AND F-SCORE OF APPROACHES

| Models | Precision | Recall | f-score |
|---|---|---|---|
| ChgAll | 0.743 | 0.486 | 0.588 |
| ChgSig | 0.778 | 0.749 | 0.763 |
| ProdAll | 0.757 | **0.842** | 0.797 |
| ProdSig | 0.792 | 0.808 | 0.800 |
| All | 0.770 | 0.530 | 0.628 |
| Sig | **0.825** | 0.828 | **0.827** |

TABLE VII
ODDS RATIO OF THE SIG APPROACH

| Metrics | Level | Odds Ratio |
|---|---|---|
| NC | Class | 0.997 |
| NDC | Class | 1.218 |
| ADD | Class | 0.999 |
| ADD | Method | **1.871** |
| DEL | Class | 1.001 |
| CHURN | Class | 1.000 |
| CHURN | Method | **0.727** |
| PAR | Method | 1.374 |
| VG | Method | **0.553** |
| NBD | Method | **1.885** |
| MLOC | Method | 1.146 |
| LCOM | Class | **1.893** |
| NSC | Class | **0.533** |

### *B. RQ2: Can change metrics recommend candidates for Extract Method?*

**Overview.** We investigated whether change metrics can be used for recommending candidates for Extract Method by comparing recommendation approaches with different metrics.

**Motivation.** The answer for RQ1 implies that change metrics can be indicators for Extract Method opportunities. However, it is still unclear that change metrics can be used for recommending candidates for Extract Method. This research question aims to investigate whether change metrics can be used as indicators for Extract Method opportunities.

**Approach.** To check whether change metrics can be used for recommending candidates for Extract Method, we further analyzed differences in product metrics between **extracted entities** and **not-extracted entities** using the same dataset adopted in Section IV-A. Table IV shows the list of metrics used, and Table V shows the results of the *Mann-Whitney test* with $p - value < 0.05$. Table V is structured the same way as Table III. We used a *logistic regression model* using Weka [1], a collection of machine learning algorithms for the data mining tasks model, with different metrics to construct different recommendation approaches as follows:

- **ChgAll:** uses all the change metrics used in Section IV-A.
- **ChgSig:** uses the change metrics with statistically significant values shown in Table III
- **ProdAll:** uses all the product metrics shown in Table IV
- **ProdSig:** uses the product metrics with statistically significant values shown in Table V
- **All:** uses all the change and product metrics
- **Sig:** uses the change and product metrics with statistically significant values

**Results & Discussion.** We evaluated these different recommendation approaches with a *10 fold cross validation*. To this end, we used three measures of accuracy called *precision*, *recall*, and *f-score* by using information in the dataset created in Section IV-A. The *precision* is the ratio of the accurately recommended methods to the results. The *recall* is the ratio of the accurately recommended methods to **extracted entities**. The *f-score* is the harmonic mean of the *precision* and *recall*. Table VI shows the evaluation results. In this table, the bold face shows the highest values among others. The table reveals that the Sig approach, which used not only product but also change metrics with statistically significant values, has the highest *precision* and *f-score*.

In addition, we calculated the *Odds Ratio (OR)*, measuring how strong the effect size of each metric, of the metrics used in the Sig approach. $OR > 1$ shows that the metric is more likely to affect the recommendation, whereas $OR < 1$ indicates that the metric is less likely to contribute to the recommendation. Table VII shows the *OR* results. In this table, all the metrics with the top three highest or lowest odds ratios are in bold. As can be seen, the top three highest odds ratios are LCOM, NBD, and ADD (method-level) metrics, whereas the bottom three lowest odds ratios are NSC, VG, and CHURN (method-level). Among them, ADD and CHURN are change metrics. These results suggest that not only product metrics but also change metrics are necessary to recommend candidates for Extract Method with high accuracy.

**Answer for RQ2. Yes**, change metrics can recommend candidates for Extract Method. However, the highest accuracy can be achieved by using not only change metrics but also product metrics.

## V. Conclusion and future works

We conducted two studies investigating the relationship between Extract Method and change metrics in *JEdit*. At first, we mined refactoring histories to compare change metrics between **extracted entities** and **not-extracted entities** and confirmed that change metrics have a clear relationship with Extract Method. We then evaluated different sets of metrics for machine learning approaches for recommending candidates for Extract Method and found the highest accuracy can be achieved by using both change and product metrics to recommend candidates for Extract Method.

As future work, we plan to analyze additional software systems to achieve the generality of our investigation results. In addition, we would like to extend our investigations for more types of change metrics such as the number of fixed defects during software development.

## References


[1] Weka 3: Data Mining Software in Java. http://www.cs.waikato.ac.nz/ml/weka/.

[2] Gabriele Bavota, Andrea De Lucia, Massimiliano Di Penta, Rocco Oliveto, and Fabio Palomba. An experimental investigation on the innate relationship between quality and refactoring. *J. Syst. Softw.*, pages 1–14, 2015.

[3] Christian Bird, Nachiappan Nagappan, Brendan Murphy, Harald Gall, and Premkumar Devanbu. Don't touch my code!: Examining the effects of ownership on software quality. In *Proc. of ESEC/FSE*, pages 4–14, 2011.

[4] A. Z. Broder. On the resemblance and containment of documents. In *Proc. of SEQUENCES*, pages 21–29, 1997.

[5] Shyam R. Chidamber and Chris F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, 20(6):476–493, 1994.

[6] Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. Finding refactorings via change metrics. In *Proc. of OOPSLA*, pages 166–177, 2000.

[7] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison Wesley, 1999.

[8] Kenji Fujiwara, Norihiro Yoshida, and Hajimu Iida. An approach for fine-grained detection of refactoring instances using repository with syntactic information. *IPSJ Journal*, 56(12):2346–2357, 2015. in Japanese.

[9] Hideaki Hata, Osamu Mizuno, and Tohru Kikuno. Historage: fine-grained version control system for java. In *Proc. of IWPSE-EVOL*, pages 96–100, 2011.

[10] Y. Kamei, S. Matsumoto, A. Monden, K. Matsumoto, B. Adams, and A. E. Hassan. Revisiting common bug prediction findings using effort-aware models. In *Proc. of ICSM*, pages 1–10, 2010.

[11] T. Lee, J. Nam, D. Han, S. Kim, and H. Peter In. Developer micro interaction metrics for software defect prediction. *IEEE Trans. Softw. Eng.*, 42(11):1015–1035, 2016.

[12] A. Murgia, R. Tonelli, S. Counsell, G. Concas, and M. Marchesi. An empirical study of refactoring in the context of fanin and fanout coupling. In *Proc. of WCRE*, pages 372–376, 2011.

[13] E. Murphy-Hill and A. P. Black. Breaking the barriers to successful refactoring: observations and tools for extract method. In *Proc. of ICSE*, pages 421–430, 2008.

[14] E. Murphy-Hill, C. Parnin, and A. P. Black. How we refactor, and how we know it. *IEEE Trans. Softw. Eng.*, 38(1):5–18, 2011.

[15] Foyzur Rahman and Premkumar Devanbu. How, and why, process metrics are better. In *Proc. of ICSE*, pages 432–441, 2013.

[16] Danilo Silva, Ricardo Terra, and Marco Tulio Valente. Recommending automated extract method refactorings. In *Proc. of ICPC*, pages 146–156, 2014.

[17] Danilo Silva, Nikolaos Tsantalis, and Marco Tulio Valente. Why we refactor? confessions of github contributors. In *Proc. of FSE*, pages 858–870, 2016.

[18] Danilo Silva, Nikolaos Tsantalis, and Marco Tulio Valente. Why we refactor? confessions of github contributors. In *Proc. of FSE*, pages 858–870, 2016.

[19] Daniela Steidl and Sebastian Eder. Prioritizing maintainability defects based on refactoring recommendations. In *Pro. of ICPC*, pages 168–176, 2014.

[20] N. Tsantalis and A. Chatzigeorgiou. Identification of extract method refactoring opportunities for the decomposition of methods. *J. Syst. Softw.*, 84(10):1757–1782, 2011.

[21] Yi Wang. What motivate software engineers to refactor source code? evidences from professional developers. In *Proc of ICSE*, pages 413–416, 2009.

[22] N. Yoshida, T. Saika, E. Choi, A. Ouni, and K. Inoue. Revisiting the relationship between code smells and refactoring. In *Proc. of ICPC*, pages 1–4, 2016.