Information and Software Technology 95 (2018) 313–327

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Empirical evaluation of software maintainability based on a manually validated refactoring dataset

Péter Hegedűs[a], István Kádár[b], Rudolf Ferenc[*,b], Tibor Gyimóthy[b]

[a] MTA-SZTE Research Group on Artificial Intelligence, Szeged, Hungary
[b] University of Szeged, Hungary

ARTICLE INFO

Keywords:
Code refactoring
Manually validated empirical dataset
Source code metrics
Software maintainability
Empirical study

ABSTRACT

Context: Refactoring is a technique for improving the internal structure of software systems. It has a solid theoretical background while being used in development practice also. However, we lack empirical research results on the real effect of code refactoring and its application.
Objective: This paper presents a manually validated subset of a previously published dataset containing the refactorings extracted by the RefFinder tool, code metrics, and maintainability of 7 open-source systems. We found that RefFinder had around 27% overall average precision on the subject systems, thus our manually validated subset has substantial added value. Using the dataset, we studied several aspects of the refactored and non-refactored source code elements (classes and methods), like the differences in their maintainability and source code metrics.
Method: We divided the source code elements into a group containing the refactored elements and a group with non-refactored elements. We analyzed the elements' characteristics in these groups using correlation analysis, Mann–Whitney U test and effect size measures.
Results: Source code elements subjected to refactorings had significantly lower maintainability than elements not affected by refactorings. Moreover, refactored elements had significantly higher size related metrics, complexity, and coupling. Also these metrics changed more significantly in the refactored elements. The results are mostly in line with our previous findings on the not validated dataset, with the difference that clone metrics had no strong connection with refactoring.
Conclusions: Compared to the preliminary analysis using a not validated dataset, the manually validated dataset led to more significant results, which suggests that developers find targets for refactorings based on some internal quality properties of the source code, like their size, complexity or coupling, but not clone related metrics as reported in our previous studies. They do not just use these properties for identifying targets, but also control them with refactorings.

1. Introduction

Source code refactoring is a popular and powerful technique for improving the internal structure of software systems. The concept of refactoring was introduced by Fowler [1] and nowadays IT practitioners think of it as an essential part of the development process. Despite the high acceptance of refactoring techniques by the software industry, it has been shown that practitioners apply code refactoring differently than Fowler originally suggested. He proposed that code smells should be the primary technique for identifying refactoring opportunities in the code and a lot of research effort [2–5] has been put into examining them. However, there are statements in the literature [6–8] that engineers are aware of code smells, but are not really concerned on their impact as refactoring activity is not focused on them. A similar counter intuitive result by Bavota et al. [9] suggests that only 7% of the refactoring operations actually remove the code smells from the affected class. Besides exploring how, when and why refactoring is used in the everyday software development, their effects on short and long-term maintainability and costs are vaguely supported by empirical results.

To help addressing the further empirical investigations of code refactoring, we proposed a publicly available refactoring dataset [10] that we assembled using the *RefFinder* [11,12] tool for refactoring extraction and the *SourceMeter*[1] static source code analyzer tool for source code

* Corresponding author.
E-mail addresses: hpeter@inf.u-szeged.hu (P. Hegedűs), ikadar@inf.u-szeged.hu (I. Kádár), ferenc@inf.u-szeged.hu (R. Ferenc), gyimothy@inf.u-szeged.hu (T. Gyimóthy).
[1] http://www.sourcemeter.com/.

https://doi.org/10.1016/j.infsof.2017.11.012
Received 14 January 2017; Received in revised form 11 November 2017; Accepted 19 November 2017
Available online 21 November 2017
0950-5849/

metric calculation. The dataset consists of refactorings and source code metrics for 37 releases of 7 open-source Java systems. We applied the dataset for a preliminary analysis on the effects of code refactoring on source code metrics and maintainability [10,13]. After the analysis, however, it turned out that the quality of the refactoring data is quite low due to the false positive instances extracted by RefFinder, thus in this paper[2] we propose an improved dataset that is a manually validated subset of our original dataset. It contains one manually validated release for each of the 7 systems. Besides the list of true positive refactoring instances in the dataset every refactoring is also mapped to the source code elements at the level of methods and classes on which the refactoring was performed. We also store exact version and line information in the dataset to supports reproducibility. Additionally to the source code metrics, the dataset includes the relative maintainability indices of source code elements, calculated by the *QualityGate*[3] tool, an implementation of the *ColumbusQM quality model* [15]. Being a direct measure of maintainability, it allows the analysis of the connection between source code maintainability and code refactoring as well.

Although the manually validated refactoring dataset is in itself a major contribution, we also utilized it to replicate and extend our preliminary studies [10,13] and re-examine the connection between maintainability and code refactoring as well as the distribution of the individual source code metrics in the refactored and non-refactored source code elements. The previous studies used the original (i.e. not validated) dataset, thus it is a question how the results change using the manually validated dataset. Our empirical investigation focused on the low-level quality attributes of refactored (and non-refactored) classes and methods, and we tried to find patterns that may explain the motivations of the developers to perform refactoring and how the internal structure of the source code elements change upon refactoring.

To concisely describe our research motivations, we framed the following research questions, which we investigated with the help of the improved dataset:

RQ1. *Are source code elements with lower maintainability subject to more refactorings in practice?*

Since refactoring is by definition a change to improve the internal code structure by preserving its functionality, it is an intuitive assumption that poor code structure is the primary driver behind code refactoring. To verify this, we investigated the maintainability values of the refactored and non-refactored source code elements to see whether there are patterns that support or contradict this assumption. By applying statistical methods on the refactoring data contained in our dataset we found that the low maintainability values of source code entities indeed triggered more code refactorings in practice.

RQ2. *What are the typical values of source code metrics of the refactored and non-refactored elements and how do they change upon refactorings?*

The first research question investigates the maintainability of the refactored and non-refactored source code elements, but we were also interested in the typical source code metric values of these elements and the effects of refactorings on these metrics. Although the RMI itself relies on source code metrics, it uses and combines only a small fraction of the available metrics (i.e. those extracted by SourceMeter). We wanted to analyze each and every metric by itself to get a deeper insight about the effect of refactorings on them. Moreover, besides the sheer metric values we were also interested in their changes throughout the releases.

Therefore, in RQ2 we examined how do the well-known source code metrics, like complexity, lines of code, coupling, etc., shape and change for the refactored and non-refactored source code elements. In general, we found that source code elements that were refactored had significantly different (typically higher) size related metrics (e.g. lines of code, number of statements), complexity (e.g. McCabe's cyclomatic complexity [16], nesting level) and coupling (e.g. coupling between object classes and number of incoming invocations) on average than source code elements not refactored at all.

Moreover, these were the metrics that changed more significantly in the refactored elements than in the non-refactored ones. Additionally, we found no such metric that would be consistently larger in the non-refactored classes and/or would grow much slower in non-refactored classes than in the refactored ones.

We also compared the findings with the previous results obtained on the not validated refactoring dataset and found that most of the metric groups found to be relevant in connection with refactoring was the same for both datasets. However, while previous results displayed 2–4 significant cases out of 7, we obtained 3–6 significant cases with much stronger p-values using the manually validated dataset. We also identified that clone related metrics had no strong connection with refactoring, even though previous results on the not validated dataset suggested so due to the false positive refactoring instances.

The main contributions of the paper can be summarized as follows. In the conference version [14] we already presented:

- A manually validated dataset containing true positive refactoring instances attached to source code elements at method and class level and their source code metrics and maintainability scores.
- An extension of the RefFinder tool that allows batch-style analysis and result reporting attached to the source code elements.
- An empirical investigation of the maintainability scores of the source code classes and methods affected by at least one refactoring and those of not.

On the basis of the achieved positive results so far, in this paper we extend our previous analysis with:

- An empirical evaluation of the main quality properties (i.e. source code metrics) and their changes due to refactoring (an entirely new research question).
- A comparison of the findings with the previous results obtained on the not validated refactoring dataset.
- Detailed information of the existing and new statistical test results and an extended discussion of them.
- We made our data analysis results available online just like the dataset itself.

The rest of the paper is organized as follows. First, we start with a related literature overview in Section 2. Next, Section 3 outlines the data collection and validation process of creating the dataset. We describe the data analysis methodology applied for answering the research questions in Section 4. In Section 5, we display the results of our empirical investigation on the maintainability and source code metrics of refactored and non-refactored source code entities. The threats to the validity of our results are listed in Section 7. Finally, we conclude the paper in Section 8.

2. 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 RefFinder tool [11] to extract refactorings from real-life open-source systems, similarly as we did.

Bavota et al. [9] 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.

[2] This journal paper is an extended version of our conference paper [14].
[3] http://www.quality-gate.com/.