The Journal of Systems and Software 107 (2015) 1–14

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

An experimental investigation on the innate relationship between quality and refactoring

Gabriele Bavota[a,*], Andrea De Lucia[b], Massimiliano Di Penta[c], Rocco Oliveto[d], Fabio Palomba[b]

[a] Free University of Bozen-Bolzano, Bolzano, Italy
[b] University of Salerno, Fisciano (SA), Italy
[c] University of Sannio, Benevento, Italy
[d] University of Molise, Pesche (IS), Italy

ARTICLE INFO

Article history:
Received 8 April 2015
Revised 8 May 2015
Accepted 12 May 2015
Available online 21 May 2015

Keywords:
Refactoring
Code smells
Empirical study

ABSTRACT

Previous studies have investigated the reasons behind refactoring operations performed by developers, and proposed methods and tools to recommend refactorings based on quality metric profiles, or on the presence of poor design and implementation choices, i.e., code smells. Nevertheless, the existing literature lacks observations about the relations between metrics/code smells and refactoring activities performed by developers. In other words, the characteristics of code components increasing/decreasing their chances of being object of refactoring operations are still unknown. This paper aims at bridging this gap. Specifically, we mined the evolution history of three Java open source projects to investigate whether refactoring activities occur on code components for which certain indicators—such as quality metrics or the presence of smells as detected by tools—suggest there might be need for refactoring operations. Results indicate that, more often than not, quality metrics do not show a clear relationship with refactoring. In other words, refactoring operations are generally focused on code components for which quality metrics do not suggest there might be need for refactoring operations. Finally, 42% of refactoring operations are performed on code entities affected by code smells. However, only 7% of the performed operations actually remove the code smells from the affected class.

1. Introduction

Refactoring has been defined by Fowler as "*the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure*" (Fowler et al., 1999). This definition entails a strong relationship between refactoring and internal software quality, i.e., refactoring improves software quality (*improves the software internal structure*). This has motivated research on bad smell and antipattern detection and on the identification of refactoring oppotunities (Bavota et al., 2013a; Boussaa et al., 2013; Fokaefs et al., 2011; Kessentini et al., 2010; Moha et al., 2010; Palomba et al., 2015; Tsantalis and Chatzigeorgiou, 2009).

However, whether refactoring is actually guided by poor design has not been empirically evaluated enough. Thus, this assumption still remains—for some aspects—a common wisdom that has generated controversial positions (Kim et al., 2012). Specifically, there are no studies that **quantitatively** analyze which are the quality characteristics of the source code increasing their likelihood of being subject of refactoring operations. To the best of our knowledge, the available empirical evidence is based on two surveys performed with developers trying to understand the reasons why developers perform refactoring operations (Kim et al., 2012; Wang, 2009).

In addition, concerning the improvement of the internal quality of software, empirical studies have only shown that generally refactoring operations improve the values of quality metrics (Kataoka et al., 2002; Leitch and Stroulia, 2003; Moser et al., 2006; Ratzinger et al., 2005; Shatnawi and Li, 2011), while the effectiveness of refactoring in removing design flaws (such as code smells) is still unknown.

In order to fill this gap, we use an existing tool, namely Ref-Finder (Prete et al., 2010), to automatically detect refactoring operations of 52 different types on 63 releases of three Java software systems, namely Apache Ant,[1] ArgoUML,[2] and Xerces-J.[3] Since Ref-Finder can identify some false positives, we manually analyzed the 15,008 refactoring operations detected by the tool. Among them, 2086 were

[*] Corresponding author. Tel.: +39 333 594 4151.
E-mail address: gabriele.bavota@unibz.it (G. Bavota).

[1] http://ant.apache.org/.
[2] http://argouml.tigris.org/.
[3] http://xerces.apache.org/xerces-j/.

http://dx.doi.org/10.1016/j.jss.2015.05.024
0164-1212/© 2015 Elsevier Inc. All rights reserved.

Table 1
Characteristics of the analyzed projects.

Project	Period	Analyzed	#Releases	Classes	KLOC
Apache Ant	Jan 2000-Dec 2010	1.2–1.8.2	17	87-1,191	8-255
ArgoUML	Oct 2002-Dec 2011	0.12–0.34	13	777-1,519	362-918
Xerces-J	Nov 1999-Nov 2010	1.0.4–2.9.1	33	181–776	56-179
Overall	–	–	63	–	–

classified as false positives. Thus, in the context of our study we analyzed 12,922 refactoring operations.

Having identified the refactoring operations, for each class in the analyzed systems' releases we (i) measured a set of eleven quality metrics, and (ii) detected if it is affected by any instance of eleven code smells. Using these data we verify whether refactoring operations occur on code components for which the factors above (i.e., quality metrics, presence of code smells) suggest there might be need for refactoring operations. In addition, we also measure the effectiveness of refactoring operations in terms of their ability to remove code smells.

The results achieved can be summarized as follows:

1. More often than not, quality metrics do not show a clear relationship with refactoring. In other words quality metrics might suggest classes as good candidates to be refactored that are generally not involved in developers' refactoring operations.
2. Among the 12,922 refactoring operations analyzed, 5425 are performed by developers on code smells (42%). However, of these 5425 only 933 actually remove the code smell from the affected class (7% of total operations) and 895 are attributable to only four code smells (i.e., Blob, Long Method, Spaghetti Code, and Feature Envy). Thus, not all code smells are likely to trigger refactoring activities.

In summary, such results suggest that (i) more often than not refactoring actions are not a direct consequence of worrisome metric profiles or of the presence of code smells, but rather driven by a general need for improving maintainability, and (ii) refactorings are mainly attributable to a subset of known smells. For all these reasons, the refactoring recommendation tools should not only base their suggestions on code characteristics, but they should consider the developer's point-of-view in order to propose meaningful suggestions of classes to be refactored.

The paper is organized as follows. Section 2 describes the design of our empirical study, while Section 3 reports and discusses the obtained results. Section 4 analyzes and discusses the threats that could affect the results of our study. After a discussion of the related literature (Section 5), Section 6 concludes the paper.

2. Empirical study design

The *goal* of the study is to analyze refactoring operations occurring over the history of a software project, with the *purpose* of understanding (i) if quality metrics and code smells presence provide indications on which code components are more/less likely of being refactored; and (ii) as a consequence, to what extent are refactoring operations effective in removing code smells from source code. The object systems, the tools, and the raw data are available for replication in our online appendix.[4]

2.1. Context and research questions

The study aims at addressing the following research questions:

- **RQ$_1$**: Are refactoring operations performed on classes having a low-level of maintainability as indicated by quality metrics?
- **RQ$_2$**: To what extent are refactoring operations (i) executed on classes exhibiting code smells and (ii) able to remove code smells?

The *context* of the study consists of 63 releases of three Java open source projects, namely Apache Ant, ArgoUML, and Xerces-J. Apache Ant is a build tool and library specifically conceived for Java applications (though it can be used for other purposes). ArgoUML is an open source UML modeler, while Xerces-J is a XML parser for Java. Although this looks a relatively small context (three projects only), such a choice has been necessary to allow us manually validating the detected refactoring and code smells, as detailed below. Table 1 reports characteristics of the analyzed systems, namely analyzed releases, number of analyzed releases, and size range (in terms of KLOC and # of classes).

2.2. Study variables and data extraction

The **dependent variables** considered in our study, for all the research questions, are the refactoring operations (of different types) being observed across releases of different programs. The **independent variables** are the factors we relate to such observed refactoring and namely:

1. For **RQ$_1$**, a series of quality metrics (described below).
2. For **RQ$_2$**, the presence of code smells (of different types) in software releases.

To answer our research questions, we first need to detect refactorings over the evolution history of the studied systems. To this aim we use an existing tool, Ref-Finder (Prete et al., 2010), to detect refactoring operations performed between each subsequent couples of releases of each system. Ref-Finder has been implemented as an Eclipse plug-in and it is able to detect 63 different kinds of refactoring operations. In a case study conducted on three open source systems, Ref-Finder was able to detect refactoring operations with an average recall of 95% and an average precision of 79% (Prete et al., 2010). Even if the accuracy of such a tool is quite high, we tried to (at least) mitigate problems related to false positives (precision) through manual validation of the refactoring operations identified by Ref-Finder. Specifically, each refactoring operation identified by the tool was manually analyzed through source code inspection by two Master's students from the University of Salerno. The students individually validated each of the proposed refactoring operations.

Once students validated the refactoring operations, they performed an open discussion with two of the authors of this paper to solve conflicts and reach a consensus on the refactoring operations analyzed, classifying them as *true positive* or *false positive*. Of the 15,008 refactoring operations detected by Ref-Finder, 12,922 operations have been manually classified as actual refactoring operations, producing as output a set of triples (rel_j, ref_k, C), where rel_j indicates the release number, ref_k the kind of refactoring occurred, and C is the set of refactored classes. Table 2 reports the number of refactoring operations (as well as the number of different types of refactorings) identified on the three systems after the manual validation. While the extracted refactoring operations are needed to answer all our research questions, in the following we detail on data collection activities made to specifically answer each research question.

[4] http://dx.doi.org/10.6084/m9.figshare.1207916 .