Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 116 (2017) 460–467

Procedia Computer Science

www.elsevier.com/locate/procedia

2nd International Conference on Computer Science and Computational Intelligence 2017, ICCSCI 2017, 13-14 October 2017, Bali, Indonesia

Towards a Severity and Activity based Assessment of Code Smells

Harris Kristanto Husien[a], Muhammad Firdaus Harun[b], Horst Lichter[b]

[a] *King Mongkut's University of Technology North Bangkok, The Sirindhorn International Thai-German Graduate School of Engineering, Bangkok, Thailand*

[b]*RWTH Aachen University, Research Group Software Construction, Aachen, Germany*

Abstract

Code smells are the structural weaknesses which reside in a software system. They evolve negatively over time reducing the system quality i.e., maintainability, understandability etc. Therefore, they should be detected and prioritized based on criticality in order to be refactored. Most of the existing approaches are based on severity score, but little works have been done to include the information from changes history. Thus, we introduce a Harmfulness Model that integrates both information: severity and changes history (i.e., code smells activity). This study characterizes a god class activity based on its severity and change frequency of the JHotDraw open source system. The result indicates that there are two main activities of god class that can be assessed as active and passive smells. In fact, an active god class can be differentiated as strong, stable, and ameliorate smells while a passive god class has one type called dormant. Besides that, from severity and activity information, the model can compute the harmfulness score and also indicate the degree of harmfulness level. The harmfulness level may be useful to improve change likelihood estimation and refactoring candidates prioritization.

© 2017 The Authors. Published by Elsevier B.V.
Peer-review under responsibility of the scientific committee of the 2nd International Conference on Computer Science and Computational Intelligence 2017.

Keywords: Code smell activity, Code smell impact, Change-likelihood, Evolution, Harmful code smell

1. Introduction

Code smells indicate a structural weakness either in code or design. Those smells can be detected by a set of dedicated metrics and are prioritized based on their critical score[1]. However, some very critical smells may not be important or urgent to be refactored immediately[2]. For example, although a class exploits some code smells and has a high critical score, it can be used as it is stable and does not change over time. Therefore, this class should not be considered as an important refactoring candidate.

E-mail addresses: harris.h-sse2015@tggs.kmutnb.ac.th (Harris Kristanto Husien)., firdaus.harun@swc.rwth-aachen.de (Muhammad Firdaus Harun)., horst.lichter@swc.rwth-aachen.de (Horst Lichter).

1877-0509 © 2017 The Authors. Published by Elsevier B.V.
Peer-review under responsibility of the scientific committee of the 2nd International Conference on Computer Science and Computational Intelligence 2017.
10.1016/j.procs.2017.10.040

To understand the evolution of smells and hence their impact, an analysis of historical information on smells is required[3], as the historical information conserves the code smell's activities and changes in the past. This information may give valuable hints for future refactoring candidates, in terms of change-proneness and the types of changes. Recently, several studies on code smell activity have been published. For example,[4] introduces the concept of an *active smell* and[5] introduces *stable* and *persistent* code smells, which are further classified as *harmless* or *harmful*.

Unfortunately, the published studies focus on unstable code smells (code smells that were changed frequently in the past), even though these changes might eliminate the code smells. In addition, to classify the stability of code smells, those studies rely only on the changes that increase or decrease the code smells presence or intensity (neglecting other changes). Therefore, the definition and classification of code smell activity can be revised and further extended. Moreover, the relationship between the activity and severity of code smells is not clearly understood.

Therefore, in this paper we explore the characteristics of code smells based on its history and measure its effects. We improve a hint to identify critical smells that subject to change by characterizing its behavior based on its activity and severity. We also introduce a harmfulness level and its four quadrant model. By measuring the degree of harmfulness, we can signify its existence and prioritization for refactoring.

The remainder of this paper is organized as follows: We propose a Harmfulness Code Smell Model in Section 2. Section 3 describes the initial case study. The results of this case study are explained in Section 4 and evaluated in Section 5. Finally, we conclude the paper and suggest some future works in Section 6.

2. Harmfulness Code Smell Model

Rapu et al. classify code smell harmfulness by considering two historical measurements, *stability* and *persistency*[5]. The stability of a class shows how many times it has been classified as *unstable*. A class is *unstable*, if a specified metric value changes after a modification of the class has been performed. If the modification does not change the metric value, it is labeled as *stable*.

The persistency of a class refers to how long it has been *suspected* as a certain type of code smell. It is computed as the fraction of the number of versions in which the class exhibits a code smell over the total number of versions. When the persistency of a class reaches 95%, this class will be classified as *persistent*, otherwise, it will be classified as *not-persistent*. Furthermore, with these two historical measurements, Rapu et al. define *harmful god classes* as unstable and not-persistent, in contrast *harmless god classes* are stable and persistent.

This paper tries to redefine Rapu et al. code smell harmfulness model by refining *stability*, *persistency* and *harmful* classification. Firstly, we refine Rapu's stability according to the performed changes. As they are treated uniformly as long as the class measurements are modified. We argue that this classification can be enhanced since a more refined stability classification can give more meaning to the harmful code smell. For example, a class that tends to increase their measurement over time is expected to have a more harmful effect than classes that tend to decrease their measurement. Therefore, we introduce *activeness* as the degree of class stability to represent how active a class is.

Secondly, we introduce *impact* as a redefinition of Rapu's persistency. The persistency, in Rapu's model, is only based on the number how often a class is being suspected as a certain type of code smell. We argue that the severity of this suspected class and its evolution will give more meaning to the harmful effect of code smell. As an example, when the suspected class is severe and tends to increase its severity, it should be more harmful since this has a higher impact on the maintenance. Therefore, our impact variable extends the class persistency with the class complexity and severity.

By aggregating the activeness and impact, we refine the harmful classification. Rapu et al. specifically classify god classes as either harmful, harmless or not classified. The not classified category requires manual inspection since Rapu et al. argued that the historical information could not improve the code smell detection (metrics-based detection[6]). However, we argue that our activeness and impact variables can outperform the not-classified problem. Through activeness and impact, we can see the types of changes and code smell severity respectively. As an example, when a code smell has high activeness but low impact, it can be classified as quite dangerous (since it has high change-likelihood). And, a code smell that has low activeness but high impact, can be classified as less dangerous since it has low change-likelihood (but it is still dangerous since it has high severity).

This paper also contributes by introducing the harmfulness score of a code smell. When two code smells are classified as harmful, it is hard to determine which code smell that is more harmful using Rapu's model. Therefore,

Download English Version:

https://daneshyari.com/en/article/4960475

Download Persian Version:

https://daneshyari.com/article/4960475

Daneshyari.com