The Journal of Systems & Software 144 (2018) 450–477

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

Controversy Corner

A systematic review on the code smell effect

José Amancio M. Santos*[,a], João B. Rocha-Junior[b,c], Luciana Carla Lins Prates[d], Rogeres Santos do Nascimento[d], Mydiã Falcão Freitas[d], Manoel Gomes de Mendonça[c,d]

[a] Technology Department, State University of Feira de Santana, Bahia, Brazil
[b] Department of Exact Science, State University of Feira de Santana, Bahia, Brazil
[c] Fraunhofer Project Center for Software & Systems Engineering, Federal University of Bahia, Bahia, Brazil
[d] Mathematic Institute, Federal University of Bahia, Bahia, Brazil

ARTICLE INFO

Keywords:
Code smell
Systematic review
Thematic synthesis

ABSTRACT

Context: Code smell is a term commonly used to describe potential problems in the design of software. The concept is well accepted by the software engineering community. However, some studies have presented divergent findings about the usefulness of the smell concept as a tool to support software development tasks. The reasons of these divergences have not been considered because the studies are presented independently. *Objective:* To synthesize current knowledge related to the usefulness of the smell concept. We focused on empirical studies investigating how smells impact the software development, the *code smell effect. Method:* A systematic review about the smell effect is carried out. We grouped the primary studies findings in a thematic map. *Result:* The smell concept does not support the evaluation of quality design in practice activities of software development. There is no strong evidence correlating smells and some important software development attributes, such as effort in maintenance. Moreover, the studies point out that human agreement on smell detection is low. *Conclusion:* In order to improve analysis on the subject, the area needs to better outline: (i) factors affecting human evaluation of smells; and (ii) a classification of types of smells, grouping them according to relevant characteristics.

1. Introduction

Since the nineties, Software engineering (SE) researchers have extensively discussed strategies for the systematic evaluation of design problems in object-oriented (OO) systems. In 1996, Riel (1996) presented one of the first books in the area. He presented insights into design improvement and coined the term “design flaw”. In 1999, Fowler (1999) adopted the term “code smell”. His book focused on refactoring and presented a catalog of smells, characterizing and proposing specific actions to remove them. In 2006, Lanza and Marinescu (2006) focused on metrics and heuristics to detect what they called “disharmonies”. Although the terms “design flaw”, “code smell” and “disharmony” have been used to define potential design problems, this work adopts the term code smell, or simply smell, to refer to such problems.

The works of Riel, Fowler, and Lanza and Marinescu mainly outline a theory about code smells. The theory is formed by a wide discussion about heuristics presenting directives “pointing developers in the right direction on smell detection”(Fowler, 1999). The authors build their heuristics from identification of aspects that affect the quality of software design. Lanza and Marinescu (2006), for example, consider that three distinct aspects contribute to the code smell characterization: its size, its interface and its implementation. The works present different types of code smells, characterizing design problems. This characterization is fundamental as it affects common activities of software development, such as diagnostics in code inspection or refactoring and maintenance decisions.

Despite the code smell concept being well defined, it is imperative to observe the effects related to its adoption on software development. The Empirical Software Engineering (ESE) discipline offers a suitable framework for this type of assessment (Wohlin et al., 2012). The discipline supports adoption of scientific empirical methods in SE, such as controlled experiment, case study, survey, ethnography, etc. In fact, code smells have been empirically studied by researchers adopting ESE support, from different perspectives, since the early2000′s. One problem is that, once these studies are presented independently, the SE community lacks a broad knowledge on the issues related to the adoption of code smells. As a consequence, there is not an overall

* Corresponding author:
E-mail addresses: zeamancio@uefs.br (J.A.M. Santos), joao@uefs.br (J.B. Rocha-Junior), luciana@prates.net (L.C.L. Prates), rogeres19@gmail.com (R.S.d. Nascimento), mydiaff@dcc.ufba.br (M.F. Freitas), manoel.mendonca@ufba.br (M.G.d. Mendonça).

https://doi.org/10.1016/j.jss.2018.07.035
Received 15 October 2016; Received in revised form 21 March 2018; Accepted 12 July 2018

comprehension of the studies' findings. The *code smell effect* is the term that we defined to capture the issues being empirically addressed by studies on code smells and the directions where their findings are pointing.

An ad-hoc literature review on the studies addressing the code smell effect evidences an interesting phenomenon. We noted that the empirical results show contradictory findings related to the well-accepted idea that considers code smell as an indicative of potential problems arising from bad design. For example, Sjøberg et al. (2013) investigated the relationship between smells and maintenance effort. They noted that none of the investigated code smells was significantly associated with maintenance effort increase. Macia et al. (2012b) investigated the relationship between code smells and problems that occur with an evolving system's architecture. In their study, they noted that many of the detected smells were not related to architectural problems. Yamashita (2013) summarized a wide analysis based on the same experimental setup used by Sjøberg et al. (2013). One of her findings is that "*aggregated code smells are not so good indicators of system-level maintainability*".

In fact, there is no strong evidence linking code smells with problems arising from bad design choices. Or at least, the question is not well understood, yet. We highlight some statements reinforcing this idea. Zhang et al. (2011) declared *"... we do not know whether using code bad smells to target code improvement is effective"*. Sjøberg et al. (2013) declared that *"the present focus on bad design as operationalized by code smells may be misdirected"*. Considering each experiment by itself, it is possible to be confident about its findings. However, in order to evaluate how the empirical studies improve the understanding of the code smell effect, it is necessary to consider a large set of studies (Juristo and Vegas, 2009).

This work builds empirical support mitigating the lack of understanding regards the code smell effect. Our aim is to understand how the empirical studies address the subject. To do this, we carried out a systematic literature review, or systematic review (SR), as proposed by Kitchenham and Charters (2007). In order to achieve our aims, we explored the *experimental settings* and *findings* from an extensive set of empirical studies (called primary studies) on the code smell effect. To explore *the experimental settings* of the primary studies, we identified and extracted their relevant attributes, such as the type of software and the type of empirical method adopted. This makes possible to analyze variations in the experimental settings. To explore *the primary studies findings* we had to code findings of the primary studies. Then, we grouped the findings into themes and sub-themes, presenting them in a thematic map (Cruzes et al., 2011) describing how the empirical studies have addressed the smell effect. It is important to highlight that our SR aims to identify how the area has addressed the code smell effect, instead to present an empirical evaluation for each different type of code smell.

One of the main difficulties of this type of SR is that the findings of the empirical studies are frequently presented as textual data. Even studies that focus on quantitative data, such as studies exploring data in software repositories, present their findings as textual information. In our SR, we combined instructions of different methods of synthesis, as recommended by Cruzes et al. (2011). We followed Cruzes and Dybå (2011) and Cruzes et al. (2015) mainly. We detail the synthesis method later on.

From the outcomes of our SR, we present our own findings. We based them on the observation of: (i) the experimental settings variations of the primary studies; (ii) the number of primary studies by themes and sub-themes; and (iii) the convergence/divergence on the primary studies findings. We found, for example, that the smell concept does not support the evaluation of design quality, in practice activities of software development. The evidences for this finding are the divergent results from the primary studies of our SR. Moreover, the studies did not find correlation between smells and effort on maintenance activities, and between smells and architectural quality. Another finding from our SR is that, nowadays, human detection of smells should not be trusted. Two main evidences support this finding. First, the primary studies of our SR show that the agreement among developers detecting smells is low. Second, the studies highlight the relevance of demographic data on smells detection activity, such as developer experience.

We also discuss the difficulties we faced in our SR as some challenges toward smell effect understanding. For sake of simplicity, we highlight two of these challenges in this section. First, the high number of code smells proposed in the literature makes difficult to group studies in order to synthesize the knowledge for the area. However, for some smells, similar problems are expected. We consider that investigations on this topic strengthen syntheses for secondary studies in the area. The two works Fontana et al. (2016) and Mäntylä and Lassenius (2006a) that we found in this direction were not strongly validated by other studies, yet. Another challenge is to deeply understand the relevance of subjectivity in smell evaluation. As previously discussed, we found the agreement among humans evaluating smells is low. Some questions spring up from this outcome: (i) what are the cognitive aspects on smell concept comprehension? or (ii) what are the factors impacting on the human perception of smells? These questions are relevant because they impact the use of smell concept as a tool to support software development tasks.

The structure of the rest of this paper is as follows. Section 2 introduces the code smell as the central concept of our work. Section 3 presents prior empirical studies addressing code smell. Sections 4 and 5 present the protocol and the data extraction process of our SR. Sections 6 and 7 present and discuss the findings. Section 8 presents and discusses some threats concerning the validity of this study. Lastly, Section 9 presents our conclusions and proposes future works.

2. A brief history of the code smell concept

The main concept we address in this work is code smell. As discussed in Section 1, we are using the code smell as a "design flaw" and "disharmony" indistinctly. All these terms are used to refer to design problems (Ahmed et al., 2015; Tufano et al., 2015; Palomba et al., 2015; Bán and Ferenc, 2014; Palomba et al., 2014). Basically, the authors propose strategies for the identification of aspects in the code that break the principles of the object-oriented paradigm. In one of the first books on the topic, Riel (1996) used his experience to discuss common problems observed. He addressed the misuse of relationships between classes, inheritance, the containment relationship from classes and attributes, and others. After each discussion, he presented heuristics to avoid the problems. For example, related to the misuse of multiple inheritance, one of Riel's suggestions is *"if you have an example of multiple inheritance in your design, assume you have made a mistake and then prove otherwise"*.

Another book in the area was written by Fowler (1999). He focused on discussions about refactoring. He defined refactoring 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"*. The main idea is improving the internal structure (design), avoiding future problems, especially for maintenance. He named and presented refactoring techniques for many situations, such as "extract method", "move method", "replace data value with object", and many others. Due to the fact that his presentation was based on a step-by-step format, including pieces of code in examples, in some cases it is possible to apply the techniques automatically.

However, the most interesting discussion proposed by Fowler is related to *when we apply refactoring?* Rather than *how we apply refactoring?* Fowler himself classifies *how to apply refactoring* as a simple problem and *when to apply refactoring* as a "not so cut-and-dried" problem. He suggested that until then, a "vague notion of programming aesthetics" had been commonly proposed and he wanted "something a bit more solid". Then, he presented the term code smell to describe