Information and Software Technology 74 (2016) 127–142

Contents lists available at ScienceDirect

Information and Software Technology

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

The relationship between design patterns and code smells: An exploratory study

Bartosz Walter[a,*], Tarek Alkhaeir[b]

[a] Faculty of Computing, Poznań University of Technology, Poznań, Poland
[b] Poznań Supercomputing and Networking Center, Poznań, Poland

ARTICLE INFO

Article history:
Received 29 April 2015
Revised 14 February 2016
Accepted 14 February 2016
Available online 3 March 2016

Keywords:
Design patterns
Code smells
Software evolution
Empirical study

ABSTRACT

Context—Design patterns represent recommended generic solutions to various design problems, whereas code smells are symptoms of design issues that could hinder further maintenance of a software system. We can intuitively expect that both concepts are mutually exclusive, and the presence of patterns is correlated with the absence of code smells. However, the existing experimental evidence supporting this claim is still insufficient, and studies separately analyzing the impact of smells and patterns on code quality deliver diverse results.

Objective—The aim of the paper is threefold: (1) to determine if and how the presence of the design patterns is linked to the presence of code smells, (2) to investigate if and how these relationships change throughout evolution of code, and (3) to identify the relationships between individual patterns and code smells.

Method—We analyze nine design patterns and seven code smells in two medium-size, long-evolving, open source Java systems. In particular, we explore how the presence of design patterns impacts the presence of code smells, analyze if this link evolves over time, and extract association rules that describe their individual relationships.

Results—Classes participating in design patterns appear to display code smells less frequently than other classes. The observed effect is stronger for some patterns (e.g., Singleton, State-Strategy) and weaker for others (e.g., Composite). The ratio between the relative number of smells in the classes participating in patterns and the relative number of smells in other classes, is approximately stable or slightly decreasing in time.

Conclusion—This observation could be used to anticipate the smell-proneness of individual classes, and improve code smell detectors. Overall, our findings indicate that the presence of design patterns is linked with a lower number of code smell instances. This could support programmers in a context-sensitive analysis of smells in code.

1. Introduction

Design patterns and code smells represent two different approaches to assuring source code quality. The first approach, perfective, is focused on solutions which positively impact some attributes of quality, and which have been empirically validated. The other approach, preventive, concentrates on detecting and removing elements that could be harmful for a software system, or make it insufficiently effective. Moreover, the preventive methods also include mechanisms that can identify symptoms of anomalies before their negative impact on quality grows and could become destructive for the system.

Design patterns represent the perfective group as they describe practically validated solutions to recurring design problems. They can easily be adapted and applied several times without changing the core of the concept. Since their introduction to software engineering by the Gang of Four in 1994 [15], they have been an object of rising interest of programmers and researchers, and practically demonstrated their ability to be implemented in different contexts. Intuitively, it is expected that the use of design patterns

* Corresponding author. Tel.: +48 616652980.
E-mail addresses: bartosz.walter@cs.put.poznan.pl (B. Walter), tarekkh@man.poznan.pl (T. Alkhaeir).

http://dx.doi.org/10.1016/j.infsof.2016.02.003
0950-5849/

positively impacts several significant characteristics of software systems: comprehensibility, readability, reliability, defect proneness etc. These assumptions underwent verification in several research studies, but their outcome did not deliver a definite answer. For example, Vokáč [40] reported that classes participating in design patterns contain fewer defects, and Prechelt and Unger [36] argued that design patterns have a positive impact on maintainability. On the other hand, a number of studies resulted in the opposite conclusions, e.g. Khomh and Guéhéneuc [23], Wendorff [42] or Wydaeghe et al. [45], and results of other studies (e.g. [35]) were inconclusive. Therefore, the question of the patterns' impact on the various attributes of code quality is still open, although there exists a consensus among researchers that the use of patterns is a recommended practice.

Code smells, unlike design patterns, play a protective role in the software quality improvement process. They were first mentioned by Fowler [14] as a metaphor for high-level signs of bad design that can hinder future maintenance of a software system. Code smells are symptoms, which can, but not necessarily have to, point to an actual issue. Therefore, they are not just patterns to be avoided, but rather easy to notice signals that require a more thorough examination.

Several research reports analyzed the impact of smells on various maintenance-related aspects, but, like in the case of patterns, the published results are not univocal and lead to different conclusions. For example, Li and Shatnawi [25] showed that some smells are related with higher defect density, while reports by Yamashita et al. [38,47,48] challenged a popular belief that smells are intrinsically related with maintainability issues and can be used for explaining them. It has also been demonstrated that various factors, e.g., code size [46], domain of a software system [12], or the age of a programmer [29], play an important role in detecting smells. Hence, it is justified to expect that other factors could also enhance the detection process. Extending the knowledge about these factors could improve our understanding of the role of smells, and their meaning for code quality.

Patterns and smells represent different approaches to software quality assurance and cannot be directly compared with each other. Intuitively, we expect the code with design patterns to display higher quality, and, as a consequence, to expose fewer instances of code smells. However, taking into account diverse results concerning the impact of patterns on quality, and an unclear relationship between code smells and maintainability, the link between patterns and smells is not so obvious anymore. As a result, the relationship deserves a deeper analysis and an experimental verification.

In this report we look for the relationship connecting design patterns with code smells, and examine if and how the presence of selected patterns interacts with the presence of smells in the same classes. Specifically, we present an empirical study which involves seven code smells and nine design patterns identified in two long-evolving, mid-sized Java applications (Apache Maven and JFreeChart), and analyze the associations between selected patterns and smells.

The paper is organized as follows. In Section 2 we discuss the published work related to smells and/or patterns, in particular with respect to the method of detecting them, and in relation to selected quality characteristics. In Section 3 we formulate the research questions and corresponding hypotheses, explain the experimental setup for the study, and briefly present the subject patterns and smells. Section 4 describes the procedure we followed in the study, and the obtained findings. In Section 5 we interpret and discuss the results. Finally, in Section 6, we conclude the study and propose future research directions.

2. Related works

To the best of our knowledge, no analysis of direct relationships between design patterns and code smells was performed. As a result, we provide an overview of smell and pattern detection approaches, and the reports on relationships between patterns or smells on one side, and various aspects of maintainability (like defect- or change-proneness) on the other.

2.1. Design patterns

2.1.1. Approaches to detection of design patterns

The rising popularity of design patterns fostered the development of various approaches of detecting them in code. The most popular ones are based on the analysis of the abstract syntax tree (AST): they look for a complete representation of patterns (e.g., PTIDEJ [16]) or their selected parts (e.g., MARPLE [3]), and a combination of static and dynamic properties of code (e.g., Heuzeroth et al. [7,19]).

Most of pattern detection tools are available for C++, Java and Smalltalk.

2.1.2. Empirical studies on design patterns

The systematic literature review on the effectiveness of patterns, published in 2012 [50], revealed several studies on the relation between design patterns and various quality attributes: defects, effort, and changeability. The general conclusion was that there is no firm support for any of the popular claims made for patterns, referring to software reliability, maintenance effort etc.; they only appear partially useful as a framework that supports maintenance, but on the other hand, they do not help novices to learn about proper software design.

A few reports analyzed the impact of patterns on the defect proneness. Vokáč [40] compared the defect rates for classes participating in selected design patterns to the defect proneness of entire project code. They found also significant differences between individual patterns with respect to that characteristic: application of a pattern does not immediately imply fewer defects. Specifically, Singleton and Observer appeared more defect-prone, whereas Factory-related patterns had lower defect numbers; the results for Template Method were inconclusive.

Prechelt and Unger [36] performed a series of experiments aimed at evaluating the impact of design patterns on the performance. He found that design patterns do not directly contribute to the quality, but programmers who implemented the patterns appeared to document the design more properly and analyzed also alternative solutions, which had an overall positive impact on the quality.

Ng et al. [30] analyzed various factors that impact maintenance effort. They found that the prior exposure of programmers to design patterns reduced time necessary to complete a maintenance task.

However, these observations are not supported univocally. For example, Khomh and Guéhéneuc [23] reported an empirical study concerning the impact of design patterns on code quality. This paper concluded, against expectations, that some design patterns have a negative impact on reusability or understandability of code, so they should be introduced cautiously.

The evolution of code with patterns was analyzed by Aversano et al. [4]. They examined three open source systems, investigating the frequency and scope of modifications of design patterns. The conclusion was that the patterns closely related to the application's purpose are modified more frequently than others.

Di Penta et al. [11] investigated the relationship between roles played by classes participating in design patterns, and their