Applied Soft Computing 26 (2015) 357–367

Contents lists available at ScienceDirect

Applied Soft Computing

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

# Source code and design conformance, design pattern detection from source code by classification approach

Abdullah Chihada, Saeed Jalili*, Seyed Mohammad Hossein Hasheminejad, Mohammad Hossein Zangooei

*SCS Lab., Computer Engineering Department, Electrical and Computer Engineering Faculty, Tarbiat Modares University, Tehran, Iran*

ARTICLE INFO

*Article history:*
Received 16 August 2012
Received in revised form 9 September 2014
Accepted 17 October 2014
Available online 27 October 2014

*Keywords:*
Design pattern detection
Machine learning
Support vector machine
Object-oriented metrics

ABSTRACT

Nowadays, software designers attempt to employ design patterns in software design phase, but design patterns may be not used in the implementation phase. Therefore, one of the challenging issues is conformance checking of source code and design, i.e., design patterns. In addition, after developing a system, usually its documents are not maintained, so, identifying design pattern from source code can help to achieve the design of an existing system as a reverse engineering task. The variant implementations (i.e., different source codes) of a design pattern make hard to detect the design pattern instances from the source code. To address this issue, in this paper, we propose a new method which aims to map the design pattern detection problem into a learning problem. The proposed design pattern detector is made by learning from the information extracted from design pattern instances which normally include variant implementations. To evaluate the proposed method, we applied it on open source codes to detect six different design patterns. The experimental results show that the proposed method is promising and effective.



## 1. Introduction

A design pattern encapsulates a proven solution to a recurring design problem [1]. In fact, each design pattern includes some classes with relations like inheritances, aggregations and delegations. Over many years, software developers suggest solutions for satisfying design problems. Then, these experience-based solutions are standardized and have been organized in the form of design patterns. The use of design patterns in software development can provide several advantages, such as increasing reusability, modularization, quality, consistency between the design and the implementation, and relationship between the design and the implementation teams [1–4].

Design pattern detection from source code is an important task in reverse engineering and can provide several advantages, such as understanding the code of program when the documentation is inadequate or absence, providing the design information to help to reconstruct software architecture, and providing ability to conformance checking of source code and design [5–7].

Several methods for design pattern detection from source codes have been proposed and we divided these methods into five categories: *Logical reasoning*, *Similarity scoring*, *FCA-based methods*, *Ontology-based methods*, and *Learning-based methods*. In the following, a brief description of each category is presented.

### 1.1. Logical reasoning

Kramer and Prechelt [8] developed the Pat system where both design patterns and designs are represented in Prolog and the Prolog engine do the actual search. The basic design information itself is extracted from source code by a structural analysis mechanism of commercial object-oriented CASE tools. Similar method is proposed in [9] developing the SOUL environment in which design patterns are described as Prolog predicates and program constituents (classes, methods, fields, etc.) as facts. Smith and Stotts [10] presented a *System for Pattern Query and Recognition* (SPQR) which uses *Elemental Design Patterns* (EDPs) and matches formalizations in a logical calculus. Fabry and Mens [11] proposed a language-independent meta-level interface to extract complex information about a structure of source code, and then used the SOUL environment, to specify and identify design motifs.

* Corresponding author. Tel.: +98 021 8288 3374.
*E-mail addresses:* A.Chihada@Modares.ac.ir (A. Chihada), SJalili@Modares.ac.ir (S. Jalili), SMH.Hasheminejad@Modares.ac.ir (S.M.H. Hasheminejad), MH.Zangooei@Modares.ac.ir (M.H. Zangooei).

http://dx.doi.org/10.1016/j.asoc.2014.10.027
1568-4946/

The prolog facts [12] present both static and dynamic information of source code to achieve the detection of design patterns with high accuracy. In [13,14], the advantage of fuzzy reasoning in dealing with incomplete knowledge to identify variant implementations of the same design pattern are used.

### 1.2. Similarity scoring

Tsantalis et al. [7] proposed a method based on similarity scoring between vertices of design pattern graphs and a graph correspond to a piece of program that reside in one or more inheritance hierarchies. This method has an ability to detect modified versions of same design pattern. In [15], a similarity is calculated between whole two graphs rather than their vertices, this approach adopts a template matching method from computer vision by calculating the normalized cross correlation between pattern graph matrix and system graph matrix. Kaczor et al. [16] proposed a method to identify classes whose structure and organization match exactly or approximately the structure and organization of design pattern classes. The authors used two classical approximate string matching algorithms based on automata simulation and bit-vector processing. In [17–19], dynamic analysis is integrated with static analysis to achieve high accuracy in design pattern detection. Heuzeroth et al. [17] used dynamic analysis results of a given system to decrease false positives of results obtained by static analysis. Dong et al. [18] presented a method in which design pattern detection applied similar to [15] and then false positive results are eliminated by behavioral and semantic analyses. Ng and Guéhéneuc [19] introduced a trace analysis technique to identify occurrences of creational and behavioral design patterns.

### 1.3. FCA-based methods

Tonella and Antoniol [20] proposed a method in which *Formal Concept Analysis* (FCA) algorithm is used to infer the presence of class groups which instantiate a common, repeated pattern, without assumption on availability of any predefined design pattern library. In [21,22], the concepts that have been calculated using FCA algorithm are filtered by removing unconnected patterns and merging equivalent ones. Then, the filtered concepts are compared with a reference library of well-known patterns to be assigned as one of the matched patterns. Tripathi et al. [23] proposed a model which solves the problem of performance by using an efficient algorithm called *Concept-Matrix based Concepts Generation* (CMCG) for construction of concepts. In [6], just patterns consisted of four classes or less have been detected because the detection of relatively simple structures in relatively small pieces of source code requires a lot of calculations.

### 1.4. Ontology-based methods

These methods use the *Web Ontology Language* (OWL) to structure source code knowledge. Dietrich and Elgar [24] proposed a Web of Pattern system which formally defines design patterns by using OWL, then, they used a prototype of a Java client that scans the pattern definitions and detects patterns in Java software. Kirasi and Basch [25] proposed a system that has three subsystems: parser, OWL ontologies and analyzer. The parser subsystem translates the input code to an XML tree. The OWL ontologies define patterns and general programming concepts. The analyzer subsystem constructs instances of the input code as ontology individuals and asks the reasoner to classify them.

### 1.5. Learning-based methods

Guéhéneuc et al. [26,27] used machine learning for obtaining rules set called signatures for participant classes (roles) of the design patterns. Then, they integrated these signatures with their constraint-based tools suite to reduce the search space. The authors learned just some individual classes of design patterns. One of the important challenges in learning-based design pattern detection is what piece of code will be given to the learned model as a candidate design pattern. Ferenc et al. [28] used another design pattern tool to candidate some probable design patterns. They used machine learning after structural matching phase for filtering out as many as possible of false hits per pattern rather than classifying them. In [29], the authors instead of analyzing source code directly, they used *Metrics and Architecture Reconstruction Plug-in for Eclipse* (MARPLE) tool to summarize the code into micro architecture that are EDP and *Design Pattern Clues*. After that, they used neural networks and WEKA data mining algorithms [30] to classify design patterns.

In our previous work [4], we proposed a novel learning-based method to select the right design pattern for each design problem. The goal of this paper is to improve limitations of learning based methods, specially [26,27], for detecting design patterns. Therefore, in this paper, we map the design pattern detection problem into a learning problem, without using another design pattern detection tool for preprocessing.

Compared with other learning-based methods [26–29], the proposed method has a number of distinguishing characteristics: (1) It has ability to detect multiple versions of design patterns (i.e., different implementations), (2) as opposed to the other learning-based methods [26–29], it does not use a tool for preprocessing but it proposes a novel, simple and low-cost preprocessing, (3) it learns simultaneously all design pattern elements (i.e., all classes playing role) of a design pattern sample instead of learning each role separately [26,27], and (4) it uses a novel and high precision learning method (i.e., classification method) called SVM-PHGS [31] and customizes it to detect design patterns.

For evaluation, we use samples of design patterns gathered from nine case studies which are manually detected by experts. We apply some different data representation methods and machine learning algorithms on the samples. In our experiments, we use six design patterns, *Adapter*, *Builder*, *Composite*, *Factory Method*, *Iterator*, and *Observer*. The results reveal that for these patterns, the proposed method can identify corresponding source code samples with acceptable Precision, Recall and small False Positive rate.

The paper is organized as follows: Section 2 presents the background needed to understand the proposed method. The proposed design pattern detection method is described in Section 3. The experimental work and results are reported in Section 4 and Section 5 presents a discussion on results and compares the proposed method with the related works. Conclusion and future works are given in Section 6.

## 2. Background

This section describes three categories which are necessary for understanding the proposed method. At first, the definition of design patterns is presented. Then, object-oriented metrics which are used in the proposed method are introduced. Finally, an overview of the classification methods is presented.

### 2.1. Design pattern definition

In general, a design pattern is described in a template consisting of two sections, *Problem Domain* and *Solution Domain*. The *Problem Domain* describes the problem context where the pattern can be applied. Analogously, the *Solution Domain* describes the structure and collaborations of the pattern solution being applied to