Science of Computer Programming 95 (2014) 406–425

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

A graph mining approach for detecting identical design structures in object-oriented design models

Umut Tekin [a,*], Feza Buzluca [b]

[a] Informatics and Information Security Research Center, Kocaeli, Turkey
[b] Computer Engineering Department of Istanbul Technical University, Istanbul, Turkey

HIGHLIGHTS

- We propose a graph mining-based approach to detect identical design structures.
- We evaluate our approach by analyzing several open-source and industrial projects.
- Identical designs structures can help in understanding the high-level architecture.
- Identical designs structures can help in discovering reusability possibilities.
- Identical designs structures can help in detecting repeated design flaws.

ARTICLE INFO

Article history:
Received 7 December 2012
Received in revised form 30 April 2013
Accepted 29 September 2013
Available online 10 October 2013

Keywords:
Software design models
Identical design structures
Software motifs
Pattern extraction
Graph mining

ABSTRACT

The object-oriented approach has been the most popular software design methodology for the past twenty-five years. Several design patterns and principles are defined to improve the design quality of object-oriented software systems. In addition, designers can use unique design motifs that are designed for the specific application domains. Another commonly used technique is cloning and modifying some parts of the software while creating new modules. Therefore, object-oriented programs can include many identical design structures. This work proposes a sub-graph mining-based approach for detecting identical design structures in object-oriented systems. By identifying and analyzing these structures, we can obtain useful information about the design, such as commonly-used design patterns, most frequent design defects, domain-specific patterns, and reused design clones, which could help developers to improve their knowledge about the software architecture. Furthermore, problematic parts of frequent identical design structures are appropriate refactoring opportunities because they affect multiple areas of the architecture. Experiments with several open-source and industrial projects show that we can successfully find many identical design structures within a project (intra-project) and between different projects (inter-project). We observe that usually most of the detected identical structures are an implementation of common design patterns; however, we also detect various anti-patterns, domain-specific patterns, reused design parts and design-level clones.

1. Introduction

Many software projects contain a significant number of software clones [1], which are duplicated parts of source code or design models. One reason for design-level cloning is the frequent usage of software design principles and design patterns (e.g., GRASP [2], GoF [3]). Furthermore, there are domain-specific patterns [4] that are optimal for a specific application or

* Corresponding author.
E-mail addresses: umut.tekin@tubitak.gov.tr (U. Tekin), buzluca@itu.edu.tr (F. Buzluca).

0167-6423/$ – see front matter
http://dx.doi.org/10.1016/j.scico.2013.09.015

problem, which allows them to be used repeatedly in a project. In addition, there are unfavorable sources of clones, such as anti-patterns and common design defects. Consequently, a software system can include many identical parts at the design level.

In the article 'Draw Me a Picture' [5], Booch noted that the hidden patterns (domain-specific patterns) in software are crucial to understanding software architecture and to assessing its quality. Domain-specific patterns are the reused design structures in a specific project domain that are specialized to accomplish similar jobs in a common design form. Detection of these structures will give us the opportunity to improve and publish them for all developers. Theoretically, it is also possible to explore currently unnamed design patterns by examining reused design structures in specific project domains.

Studies on software maintenance indicate that more than 2/3 of the total development cost is spent on software maintenance activities [6,7], and more than half of the maintenance cost is spent on comprehension activities [8]. Nevertheless, several real-world evaluations present that the availability of documented design patterns will reduce the cost of program comprehension for object-oriented systems [9,10]. Additionally for these reasons, it is important to discover reused design patterns.

The comprehension of software architecture also plays a key role in refactoring processes, which constitute the reactive part of maintenance tasks. Refactoring improves the internal design structure of software by preventing the production of poor quality products. Identifying these types of identical design structures (i.e., non-standard design patterns and common design defects) could provide a significant advantage in terms of reducing the cost of maintenance; the reason is that the most commonly-used structures in software design are the best places to look for refactoring opportunities because they affect multiple parts of the design. For example, non-standard structures that are similar to design patterns might be modified to conform to standard forms, and common design defects can be quickly identified, which allows them to be fixed in multiple areas at once. In addition, frequently repeated identical design structures are usually the most reusable parts of the design; these parts can provide good candidates for additional use in future designs.

Another source of the clones is the replicated code due to copy–paste activities. Mostly, developers modify these replicated parts separately to allow their source code to change, but the design remains same [11]. The code quality could decrease if developers apply a bug fix to one structure but fail to apply the same correction to its copies. With the help of our approach, copy–paste type design structures can be detected even if their source code is modified. These replicated structures can be combined into a single library entity, to be used efficiently in different parts of the current project or in future projects. This approach will also avoid inconsistent bug fixes.

The area of clone detection is considered to be an important part of several software engineering tasks [12]. Finding repetitive design matches could help developers and architects when evaluating reused design patterns, refactoring duplicated parts, understanding the program architectures and detecting plagiarism.

In this paper, we propose a graph mining-based approach to detecting identical parts in an object-oriented software architecture. The proposed approach contains three main steps. In the first step, the AST (abstract syntax tree) of the source code of the system is analyzed and the UML-based design level of abstraction is created. Based on this abstraction, we construct a software model graph, in which classes, interfaces and templates of software constitute the vertices, and the relations between them form the directed edges. According to the importance of the relation type, we assign weight values to the edges of the graph. In the next step, we apply a graph partitioning algorithm to divide the directed and weighted software model graph into small pieces. Finally, in the last step, a sub-graph mining algorithm is applied to discover identical design structures in the generated software model. Because the scope of our study is primarily focused on the detection stage, analyzing and automatically classifying these structures could be an interesting topic for future studies. However, we also present several evaluations with a manual classification to find useful and meaningful structures from open-source and industrial projects that could inspire future studies.

The remainder of this paper is organized as follows: Background and related work are presented in the next section. The graph representation and definitions are given in Section 3. The identification process is detailed in Section 4. In Section 5, the results obtained from exemplary projects are presented. In Section 6, we discuss critical parts and the efficiency of our approach, and the last section concludes the paper.

2. Related work

There are some graph-based design pattern detection approaches that are proposed in the literature [13,14], but most of them depend heavily on pre-known pattern templates. These approaches usually offer ways to define custom templates that could correspond to design flaws or non-standard pattern structures; however, during the development of the software, the method of using the patterns could vary according to the specific developer, programming language or target environment. In practice, it is not possible to cover all of the types of copy–paste design structures, domain-specific patterns, and non-standard design motifs for pre-known templates that are appropriate for all types of project domains. In contrast to the previous pattern detection studies, our approach does not require that any patterns are known in advance, and we can discover any type of identical design structures even though designers are not aware of them at all.

Koschke provides a survey about the studies in the clone-detection area that attempt to find identical parts of the software [15]. The suffix-tree comparison-based clone detection study in [16] works at the design level and attempts to find identical parts of UML sequence diagrams. Another design-level, pattern-matching-based study [17] operates on the XMI