Design Patterns and Design Quality: Theoretical Analysis, Empirical Study, and User Experience

Liguo Yu, Indiana University South Bend, South Bend, IN, USA
Yingmei Li, Harbin Normal University, Harbin, China
Srini Ramaswamy, ABB Inc., Churchville, MD, USA

ABSTRACT

Design patterns are reusable software design solutions to object-oriented programs. Since the initial introduction of the 23 well-known design patterns in 1995, more and more patterns have been identified and utilized in the software industry. The benefits of applying design patterns include reducing development cost, improving code quality, and standardizing the integration and maintenance processes. Therefore, using design patterns is becoming a common practice to build both commercial software and open-source products. Although most design patterns are considered creative solutions to some difficult design problems, not all of them are necessarily the best with respect to all different software quality measures, such as program complexity. This paper studies 13 commonly employed design patterns in software industry. First, these 13 patterns are analyzed theoretically about their design complexity in comparison to the conventional solutions. Second, empirical studies are performed on five open-source Java projects to investigate the correlations between design patterns and class structural quality. Finally, these 13 design patterns are evaluated by software programmers who have experience of using all of them. Overall, this study finds that although some design patterns are considered useful, creative, and significant compared to conventional solutions based on user experience, pattern-involved classes are more complex than pattern-free classes, both theoretically and empirically. The authors accordingly recommend a balanced approach to using design patterns: design quality, cost, development time, and product quality should all be considered, when design patterns are utilized.

KEYWORDS

Design Patterns, Java, Program Complexity, Software Industry

1. INTRODUCTION

One of the challenging tasks in software industry is to produce high quality products that are maintainable and reusable. To achieve this goal, software engineering principles are proposed, best practices are summarized, software development tools are built, formal methods are utilized, and so on and so forth. Among these efforts to achieving high quality standard and reducing development cost is the adoption of design patterns (McNatt & Bieman, 2001; Shalloway & Trott, 2005). Design patterns are generalized solutions for common design problems in object-oriented systems (Mor et
The idea that supports design patterns is that although the problems of different programs we are dealing with could belong to different domains, the internal data relations or functional procedures of these applications could be similar. Therefore, similar solutions could be reused to tackle these similar problems.

The use of design patterns can provide several advantages, such as increasing reusability, modularity, and product quality of a program. Design patterns can also improve the consistency between program design and program implementation and help achieve better coordination between the design and the implementation teams (Hasheminejad & Jalili, 2012; Hussain et al., 2017). Since the introduction of design patterns (Gamma et al., 1995), software developers and researchers have identified over 20 patterns, many of which have been widely used in object-oriented programs. Recent studies found that design patterns have also been widely used in open-source programs (Ampatzoglou et al., 2011a; Sfetsos et al., 2014), which greatly facilitated the rapid growth and distribution of open-source projects.

Although design patterns can provide tested, proven solutions to representative and often-occurring software design problems, they are not necessarily the best design solutions with respect to software quality. First, while design patterns could be used to solve certain design problems, they might introduce new problems and reduce the system quality (Ampatzoglou et al., 2011a). Second, the use of design patterns should be context-based; an inappropriate use of design patterns could increase the system complexity and accordingly reduce software quality (Ampatzoglou et al., 2011b). To address this issue, extensive research has been performed in this area to study how design patterns can affect software quality, including reusability and maintainability (Ampatzoglou et al., 2011a; Ampatzoglou et al., 2011b). For example, theoretical analysis has been performed on several design patterns to see how they might affect complexity measurements (Huston, 2001). However, little work has been reported to investigate the relationship between design patterns and software design quality both theoretically and empirically.

This paper presents a comprehensive study to investigate the relationship between design patterns and design quality. The study is carried out in three stages. In Stage 1, we theoretically analyze the complexity and design quality of some representative design patterns; In Stage 2, we empirically study the correlation between software complexity and the usage of design patterns in real world projects; In Stage 3, we collect and analyze user evaluations of design patterns.

The remainder of the paper is organized as follows. Section 2 reviews related work in the quality analysis of design patterns. Section 3 presents the background knowledge of this study. Section 4 describes the theoretical analysis of the complexity of design patterns. Section 5 presents the empirical study of the relationship between design patterns and software structural complexity. Section 6 presents user evaluations of design patterns. Section 7 compares the results of theoretical analysis, empirical evaluation, and user experience about design pattern quality. Threats to validity are discussed in Section 8. Conclusions appear in Section 9.

This paper extends our earlier book chapter on the same topic (Yu & Ramaswamy, 2014). Below listed the major differences between our original book chapter and this paper.

- Section 4 (theoretical analysis) and Section 5 (user experience) are new; they contain unpublished research results.
- Section 6 is also new, it combines and compares the results of theoretical analysis (Section 4), empirical study (Section 5), and user experiences (Section 6).
- Section 1 (introduction), Section 2 (literature review), Section 3 (background), and references are also updated to reflect the latest development.
- Title, abstract, and conclusions are also rewritten to match the new research objective of this paper.
Service Orientation and Software-Intensive Systems
[www.igi-global.com/chapter/service-orientation-software-intensive-systems/8233?camid=4v1a](www.igi-global.com/chapter/service-orientation-software-intensive-systems/8233?camid=4v1a)

On Parallel Online Learning for Adaptive Embedded Systems
[www.igi-global.com/chapter/on-parallel-online-learning-for-adaptive-embedded-systems/108447?camid=4v1a](www.igi-global.com/chapter/on-parallel-online-learning-for-adaptive-embedded-systems/108447?camid=4v1a)