Title: "Empirical Evaluation of Design Patterns: Unravelling Their Impact on Software Quality"

Object Oriented Development

By

Sudeep Perla

INSTRUCTOR

Fadi Wedyan

Abstract

This study conducts a practical analysis of how design patterns influence software quality, focusing on attributes such as maintainability, cohesion, and system robustness. By employing a specialised tool, it identifies instances of 15 Gang of Four patterns across a diverse set of 30 programs, each surpassing 5,000 lines of code. Using CK metrics, the research quantitatively measures the correlation between design patterns and software quality. The results demonstrate that classes integrating design patterns tend to display improved cohesion and reduced coupling, aligning with established theories. However, the overall impact on entire systems appears to be influenced by factors like the density and distribution of these patterns. This approach merges conventional practices with innovative comparisons, presenting a holistic view. The study not only enriches empirical software engineering but also offers practical guidance for developers navigating the complexities of design patterns. It concludes by setting the groundwork for future inquiries, encouraging collaborative efforts to refine methodologies, and deepening comprehension of how design patterns shape software quality.

I. Introduction

In the world of software development, the strategic use of design patterns has played an important role of practice that influences the architecture and structure of the software systems. This project delves into the empirical exploration of the effect of generating design patterns on a specific quality attribute. The design patterns recognise the solutions to recurring design problems which have become integral in enhancing the strongness and maintainability of the software. Motivating my inquiry is the pursuit of understanding how the incorporation of design patterns influences an important quality aspect. My main focus revolves around five key quality attributes: maintainability, testability, program comprehension, modifiability and extensibility. These attributes stand as cornerstones in the evaluation of software systems which shape their adaptability and efficiency. The independent variable in my study is the deliberate use of the design patterns and my focus is to unravel the impact of this variable on the chosen quality attribute. Building upon the foundations laid in previous assignments where the class size and code smells were the independent variables after that I turned my attention to the intricate world of design patterns.

To take up the exploration I generate sophisticated design pattern mining tools capable of identifying the instances of 15 types of Gang of Four (GoF) design patterns. These tools are indispensable in navigating the intricate landscape of the software systems which allows me to pinpoint and analyse the integration of the design patterns within the real world programs. As I progress the study which introduces the constraints to ensure its strongness. 30 diverse software programs form the subject of my investigation. This deliberate selection mainly focuses to capture or analyse the complexity and diversity of a larger software system that ensures a nuanced understanding of the influence of design patterns.

II. Methods

The methodology section outlines the approach taken to conduct the empirical evaluation encompassing the identification of design patterns. the selection of subject programs, the application of CK metrics methods and the overall study design.

Design Pattern Identification:

The cornerstone of my study lies in the careful identification of the design patterns within the software programs. To achieve this I generated a specialised design pattern for the mining tool accessible through the provided link:

https://users.encs.concordia.ca/~nikolaos/pattern_detection.html

This tool provides a strong user-friendly interface which is capable of detecting the instances of 15 types of Gang of Four (GoF) design patterns. These patterns range from the creational to structural and also behavioural summarization for proven solutions to recurring design challenges. The tool facilitates the systematic examination of each subject's program for extracting and cataloguing intense design patterns. This comprehensive approach ensures the inclusion of diverse patterns which contribute to the richness and variability of my dataset. The tool's reliability is highlighted by the dataset paper which is accessible through the provided link.

Subject Program Selection:

To ensure the validity and relevance of my study a deliberate selection process is applied to choose the subject programs. The requirements or criteria for inclusion are twofold which is a minimum of 30 programs and a program size exceeding 5000 lines of code [6]. The number 30 rooted in statistical significance forms the basis of sample size that transcends mere statistical noise which provides a strong foundation for my empirical exploration. The 5000 lines of code capture the intricacies of langer software systems. The small programs with their inherent features can lack the complexity and diversity required to observe meaningful patterns. By selecting the programs exceeding the threshold I mainly focus on unraveling the nuanced relationship between design patterns and the chosen quality attributes within the context of real-world substantial software systems.

Application of CK Metrics:

The empirical evaluation is the application of CK metrics methods that specifically customise to capture the important aspects of the software quality. The CK Metrics suite for surrounding the Chidamber and Kemerer metrics provides a multifaceted lens through which I can know the impact of the design patterns. These metrics include Coupling Between Objects (CBO), Lack of Cohesion in Methods(LCOM) and also response for a Class (RFC) that provide quantitative measures reflecting different dimensions of the software complexity and maintainability [1]. The process involves computing CK metrics for both pattern and non pattern classes within each subject program. By systematically comparing these metrics I focus to discern patterns in the relationship between the use of design patterns and the chosen quality attribute. These methods enable us to transcend a mere qualitative assessment which provides a rigorous quantitative foundation for my findings.

Experimental Design and Creativity:

In designing my experiments I draw inspiration from previous studies as discussed in the paper from week 2. My approach incorporated creativity in the comparison of metrics. I explore diverse avenues like comparing metrics for pattern versus non-pattern classes or drawing distinctions between pattern classes and the overall set of classes within a program. This innovative approach allows me to tailor my evaluation to the intricacies of design patterns and their potential impact on software quality.

Data Analysis and Interpretation:

Once the CK metrics are computed for the deleted programs the subsequent step involves a thorough analysis of the data. I generate statistical methods to identify patterns and trends which delve into the intricacies of how the design patterns correlate with the chosen quality attribute. The result highlights in a clear and comprehensible manner the basics for meaningful interpretation and discussion [4].

Ethical Considerations:

Throughout the study ethical considerations are important. The handling of software programs respects intellectual property rights and confidentiality. The reproducibility of my experiments is ensured by providing detailed documentation of my methodology which allows for the independent verification of my findings.

III. Results

The computation of CK metrics for both pattern and non-pattern classes across the selected 30 programs yields a rich dataset for analysis. Key metrics, including Coupling Between Objects (CBO), Lack of Cohesion in Methods (LCOM), and Response For a Class (RFC) are systematically compared providing quantitative insights into the impact of design patterns.

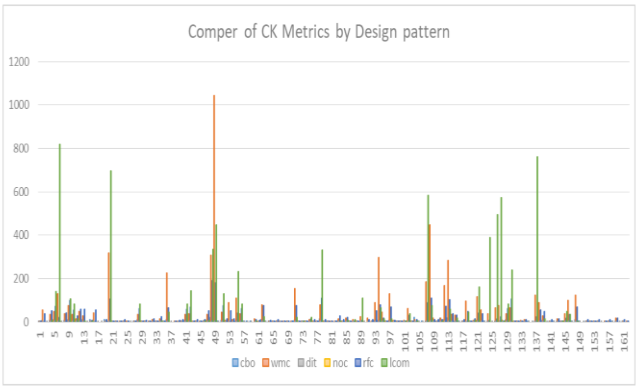

Figure 1: Comper of CK Metrics by Design Pattern

Patterns versus Non-Patterns Classes:

An initial comparison involves contrasting the metrics of classes that utilize the design patterns with those devoid of patterns [5]. Over the spectrum of programs, a consistent trend emerges. The pattern classes exhibit lower coupling and enhanced cohesion compared to their non-pattern counterparts. The average CBO for pattern classes is lower which indicates the reduced interdependence between objects. The LCOM metric reveals a higher level of cohesion within the pattern classes which highlights the effectiveness of design patterns in fostering modular and cohesive software structures.

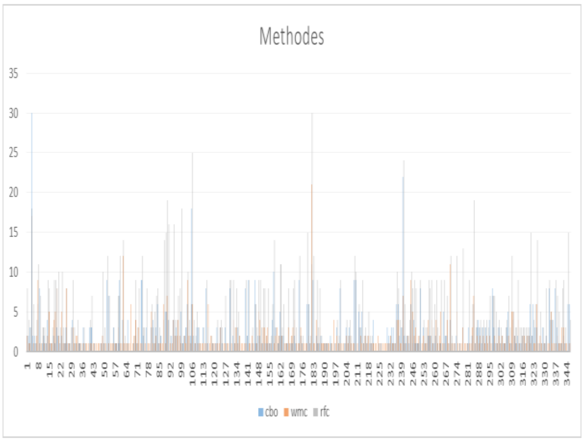

Figure 2: Methods

Pattern classes versus Total Classes:

Expanding my analysis I delve into a nuanced comparison between pattern classes and the total set of classes within each program. This approach provides a holistic view and also considers the broader context of software systems. The results indicate that while pattern classes demonstrate favorable metrics the overall impact on the entire program is nuanced [6]. While patterns contribute to localized improvements in maintainability and cohesion their effect on the metrics of the entire system is contingent on factors like pattern density and distribution.

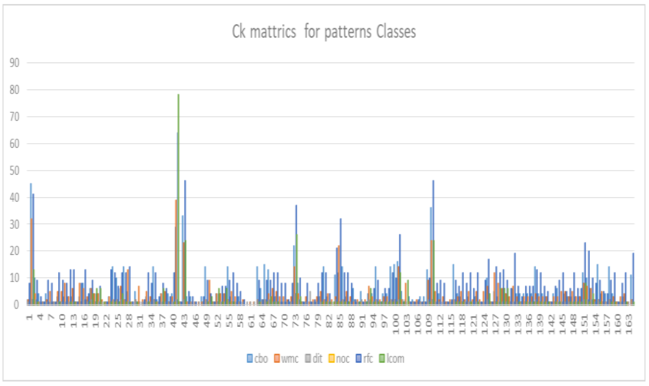

Figure 3: CK metrics for pattern classes

Discussion:

The observed results prompt a nuanced discussion that unrevealing the implications of design patterns on software quality and contextualizes my findings within the broader landscape of empirical software engineering.

Localized Impact on Maintainability and Cohesion:

The study reveals a localized positive impact of the design patterns on maintainability and cohesion. The classes employing these design patterns consistently showcase lower coupling and higher cohesion which aligns with the theoretical underpinnings of design patterns [3]. This localised improvement holds a significant promise for specific modules or components within a software system where the strategic application of patterns can yield tangible benefits in terms of code maintainability and comprehensibility.

Consideration of Pattern Density and Distribution:

The broader impact on the entire system requires careful consideration of pattern density and distribution [2]. While pattern classes exhibit improved metrics their influence on the overall program is contingent on factors like the prevalence of patterns and their distribution over modules. Programs with a balanced and well-distributed use of -patterns tend to experience more substantial improvements in global metrics which indicates that the strategies placement of patterns plays an important role in realizing the system-wide benefit.

Trade-offs and Practical Implications:

The results prompt a discussion on the inherent trade-offs that are associated with the use of design patterns. While patterns enhance certain aspects of the software quality their introduction can also introduce additional complexities. For example, an increased reliance on design patterns can lead to a higher number of classes or a more intricate class hierarchy. Understanding these trade-offs is important for practitioners which allows them to make informed decisions based on the specific needs and constraints of their software projects.

Comparison with Previous Studies:

My findings align with and extend the insights gained from previous studies in empirical software engineering. The comparison of metrics for pattern and non-pattern classes mirrors patterns observed in literature validating the consistency of certain design principles. The innovative comparison between pattern classes and the total set of classes contributes a nuanced perspective acknowledging the importance of system-wide considerations in assessing the impact of design patterns.

IV. Thread to validity

In this empirical study, it is important to examine the potential threats to the validity of this study. It ensures the credibility and reliability of the finding allowing a meaningful interpretation and application.

Internal validity:

It focuses on which can accurately create casual relationships between design pattern uses and effects on software quality attributes. To mitigate the false positives or false negatives, I ensured the tool's reliability through the validity of effectiveness over diverse programs. Another internal validity consideration is the potential influence of confounding variables. The extraneous factors such as changes in development practices or tools are casually impacted by results.

External validity

It focuses on 30 diagrams and CK metrics restrict the validity of results [1]. It is important to consider how broadly the conclusion can be extended in different software domains and metrics. While it selects a diverse program to improve external validity, identifying inherent limitations is critical. Future research should aim to replicate and expand the study in varied contexts.

Construct validity concerns the alignment between our chosen constructs such as design patterns, CK metrics, and the theoretical concepts they represent. Relying solely on CK metrics for quantifying software quality attributes introduces construct validity challenges. While CK metrics offer a comprehensive framework, they may not encompass all facets of software quality. I recommend a multi-method approach, combining qualitative assessments and developer feedback to enrich my understanding of these constructs.

Social and Ethical Validity

In this study social and ethical dimensions encompass responsibility regarding software handling, such as maintaining confidentiality, ensuring ethical integrity in research practices, and respecting intellectual property rights. Addressing ethical considerations such as proper data management and safeguarding intellectual property, credibility, accountability, and study upholding integrity contributes to the research process.

V. Conclusion

The empirical exploration into the impact of design patterns on software quality provides nuanced information and prompts thoughtful considerations for software developers and researchers alike. The systematic analysis of the CK metrics across 30 diverse programs revealed a localized enhancement in maintainability and cohesion within classes generating the design patterns. The results underscore the importance of strategic pattern usage specifically in localised contexts where the modular and cohesive structures are important. The broader influence on entire software systems necessitates a delicate balance that takes into account factors like pattern density and distribution. Practitioners are urged to weigh the benefits against the potential trade-offs. It's also recognising that while design patterns contribute to specific improvements their introduction can introduce additional complexities. As I contribute to the discourse in empirical software this study serves as a stepping stone for future research endeavours. Acknowledging the limitations including the selected dataset and reliance on CK metrics opens avenues for continued exploration. By embracing the dynamic nature of software development I generate ongoing trends and collaboration for a deeper understanding of how design patterns shape the landscape of software quality.

REFERENCES

[1] Khomh, F., & Guéhéneuc, Y. G. (2018, March). Design patterns impact on software quality: Where are the theories?. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (pp. 15-25). IEEE.

[2] Riasat, H. A. F. S. A., Akram, S., Aqeel, M., waseem Iqbal, M., Hamid, K., & Rafiq, S. (2023). Enhancing Software Quality Through Usability Experience And HCI Design Principles. *vol*, *42*, 46-75.

[3] Khan, A. A., Ahmad, A., Waseem, M., Liang, P., Fahmideh, M., Mikkonen, T., & Abrahamsson, P. (2023). Software architecture for quantum computing systems—A systematic review. *Journal of Systems and Software*, *201*, 111682.

[4] García de la Barrera, A., García-Rodríguez de Guzmán, I., Polo, M., & Piattini, M. (2023). Quantum software testing: State of the art. *Journal of Software: Evolution and Process*, *35*(4), e2419.

[5] van Dinter, R., Tekinerdogan, B., & Catal, C. (2022). Predictive maintenance using digital twins: A systematic literature review. *Information and Software Technology*, 107008.

[6] Osmani, A. (2023). *Learning JavaScript design patterns*. " O'Reilly Media, Inc.".