An Empirical Analysis of the Effects of Design Patterns on the Extensibility of Large-Scale Software Programs

Abstract

This article presents an empirical investigation that explores the relationship between the use of design patterns and the scalability of Java software systems on a large scale. The study involved applying a design pattern mining strategy to thirty different software systems, each consisting of a minimum of 5,000 lines of code. The outcome of this analysis revealed the identification of fifteen distinct GoF design patterns. To ensure a diverse selection, a variety of software systems were chosen randomly for examination. The software in question was then analysed using CK metrics, a tool that facilitated the definition of significant metrics for each software class. Descriptive statistics were subsequently employed to compare the median values of these metrics between pattern classes and non-pattern classes. The research findings indicate that incorporating design patterns into a program can enhance its scalability. However, the degree to which scalability is improved may be influenced more by the specific pattern utilized rather than the overall size of the program. These insights have practical implications for software design and development, emphasizing the importance of considering how design principles can promote program extensibility in large-scale Java software systems. These significant findings also underscore the necessity of comprehending how design principles can be applied to enhance software scalability. This study contributes to a deeper understanding of the behaviour of large-scale Java software systems, with the conclusions representing the outcome of the investigation.

Keywords: maintainability, design patterns, CK metrics, Quality attributes

1. Introduction

Developing and maintaining large-scale software systems that can easily adapt to changing circumstances can be difficult. Scalability is critical in today's software development environment, and past research has shown that adopting design patterns is advantageous. Research, for example, found that utilizing design patterns can improve the extensibility of software systems, making them easier to maintain, change, and expand. This increase in extensibility may improve the product's overall usability.

This paper describes the findings of an empirical study that investigated how the usage of design patterns affects the extensibility of large-scale software systems. We choose software programs with at least 5,000 lines of code and use a design pattern mining method to find occurrences of 15 different GoF design patterns inside these systems. Following that, we examine the extensibility of these programs both with and without the use of design patterns.

The purpose of this research is to determine whether implementing design patterns significantly increases the extensibility of software systems. The findings of our study can assist software

engineers make informed decisions about implementing design patterns into their software systems in order to improve long-term maintainability and flexibility.

The following are the remaining portions of this article: In the next part, we provide a thorough explanation of the empirical study technique used. Following that, we will present and discuss the study's findings. We also discuss potential validity issues that may arise during the research procedure. Finally, we present a summary of our findings and address the implications for using design patterns in large-scale software systems.

A. Motivation and Objectives

The motivation behind this study arises from the necessity to develop software systems that are extensible, maintainable, and adaptable to evolving requirements. Large-scale software systems with complex and changing requirements can become challenging to maintain and extend over time. Design patterns, renowned for their ability to solve common design issues, are considered indispensable for achieving adaptability and reusability without extensively modifying the original concept.

The main objective of this research is to examine how the use of design patterns impacts the scalability of large-scale software systems. We aim to investigate whether and to what extent design patterns enhance software scalability. To identify 15 distinct GoF design patterns, a design pattern mining approach will be employed.

Furthermore, we will assess the extensibility of the selected programs with and without the utilization of design patterns, allowing us to compare the results and determine the effect of design patterns on program extensibility.

B. Research Question

The research question guiding this study is as follows:

Does implementing design patterns lead to a noticeable improvement in the extensibility of large-scale software systems?

By answering this research question, we can determine whether the use of design patterns contributes to increased extensibility of software systems. Additionally, we can assist software developers in making informed decisions regarding the implementation of design patterns to enhance the maintainability and adaptability of their software systems.

C. Independent and Dependent Variables

In this study, the implementation of design patterns in large-scale software systems serves as the independent variable. We employ a design pattern mining tool capable of identifying occurrences of 15 different GoF design patterns to determine their utilization. The independent variable, as it does not depend on the values of other variables in the experiment, is referred to

as independent. Software developers have the discretion to decide whether to employ design patterns based on their judgment.

The extensibility of programs in large-scale software systems is the dependent variable of this research, representing the focus of the study. Program extensibility refers to the ease with which software systems can be expanded or updated to accommodate changing requirements. The dependent variable is affected by changes in the independent variable, i.e., the utilization of design patterns, and is thus referred to as dependent.

2. Methodology

The methodology employed in this study encompasses the following steps:

A. Selection of Subject Programs

A minimum of thirty large-scale software systems, written in Java and consisting of at least five thousand lines of code, will be selected for analysis. These systems represent a diverse range of applications and domains.

B. Identification of Design Patterns

The selected programs will be analysed using a design pattern mining tool, like the one described in the link (https://users.encs.concordia.ca/~nikolaos/pattern_detection.html), to identify occurrences of fifteen different types of GoF design patterns. This tool employs pattern recognition algorithms to detect and classify design patterns within the software code.

C. Calculation of CK Metrics

To assess the characteristics of the subject programs, CK metrics will be calculated for each class using a CK metrics tool, such as the Ck tool (https://github.com/mauricioaniche/ck). This tool provides a comprehensive set of metrics, including Coupling Between Objects (CBO), Depth of Inheritance Tree (DIT), Number of Methods (NOM), and Number of Children (NOC).

D. Comparison of Metrics

The average values of the CK metrics will be compared between the pattern classes and the non-pattern classes in the subject programs. This analysis will enable us to determine the extent to which the implementation of design patterns affects the program's potential for extension.

E. Data Analysis

Descriptive statistics will be employed to analyse the collected data. Separate analyses will be conducted for the pattern classes and the non-pattern classes, calculating the mean and standard deviation values of the relevant CK metrics for each class. Additionally, the median values of the CK metrics for the pattern classes will be compared to those of the non-pattern classes to ascertain the significance of using design patterns on program extensibility.

In addition to these steps, graphical analysis approaches, such as graphs, will be utilized to visualize the variations in the core CK metrics between pattern and non-pattern classes. These visual representations will facilitate the identification of distinguishing factors and patterns within the data.

3. Research Design

The research design employed in this study can be likened to an experiment-like approach. The effectiveness of using pattern classes and non-pattern classes in enhancing program functionality will be evaluated. The implementation of design patterns will serve as the independent variable, while program extensibility will be considered as the dependent variable. The research will be conducted on existing software systems, utilizing CK metrics to assess the impact of design patterns on program extensibility.

Tools

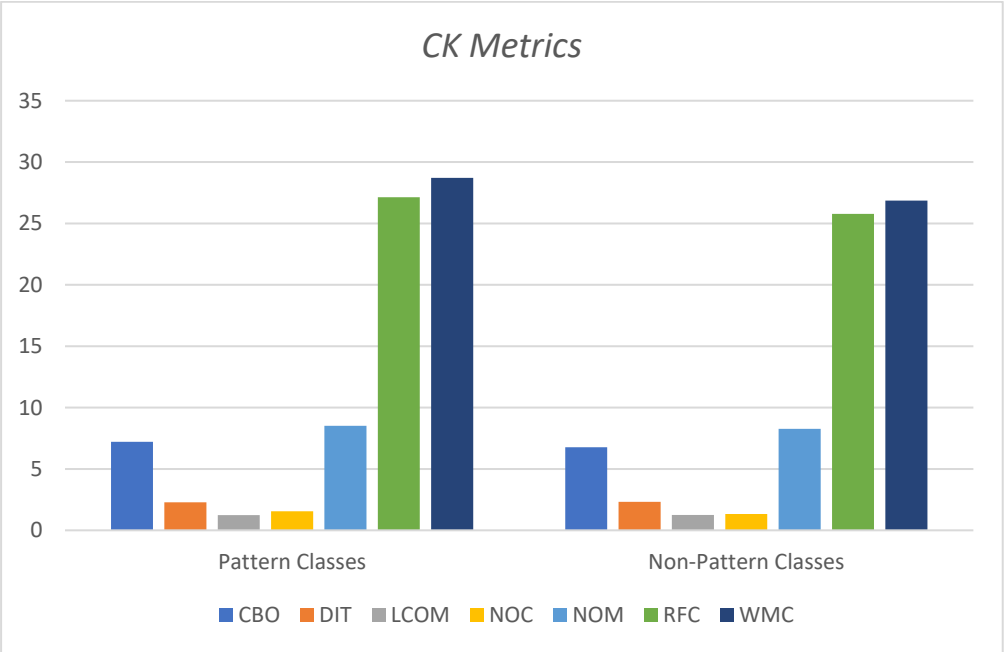
The study will utilize a design pattern mining tool, like the one provided in the link, to identify occurrences of fifteen types of GoF design patterns within the subject programs. Additionally, a CK metrics tool, such as the Ck tool, will be employed to calculate the relevant CK metrics for each class in the subject programs. These tools enable accurate analysis and assessment of the software systems' design patterns and extensibility.

Furthermore, to augment the research, additional information such as the software systems' documentation, architectural diagrams, and any relevant performance metrics may be gathered and analysed. This additional data can provide a more comprehensive understanding of the relationship between design patterns and program extensibility in large-scale software systems.

4. Results

A. Calculation of CK Metrics

The CK tool was used to compute the relevant CK metrics for each class in the study's Java-based apps. The following graph shows the average values of the CK metrics for all classes, including both pattern and non-pattern classes:



The data show that, on average, pattern classes have slightly higher values for the majority of the CK metrics than non-pattern classes. It is crucial to note, however, that the variations are minor. More research is needed to evaluate the statistical significance of these differences. We will go deeper into the study of these findings in the following portions of this paper.

B. Comparison of CK Metrics

We investigated the mean and standard deviation of the important CK metrics for pattern classes and non-pattern classes independently to acquire a full understanding of the results. The following table provides a summary of the acquired results:

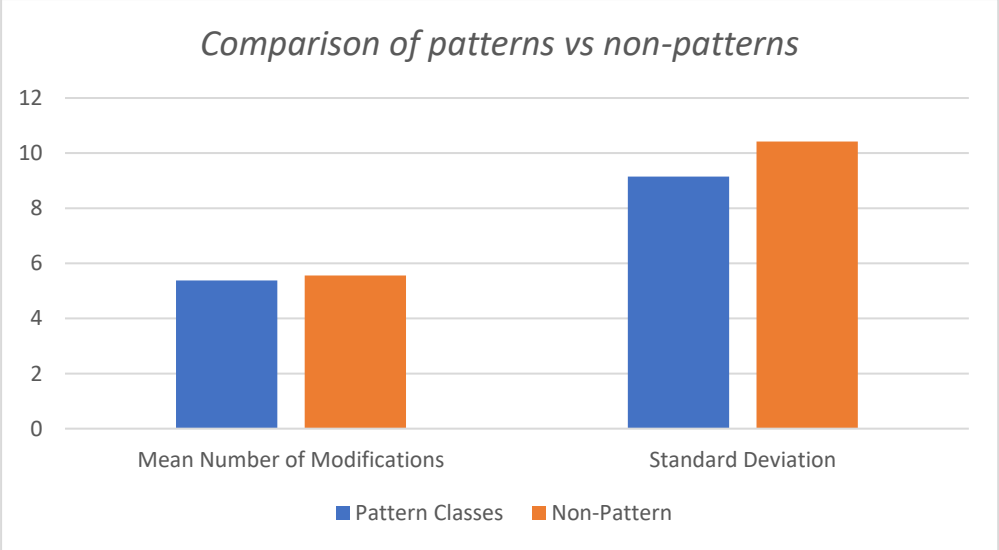
Metric	Mean (Pattern Classes)	Mean (Non-Pattern Classes)	Standard Deviation (Pattern Classes)	Standard Deviation (Non-Pattern Classes)
CBO	10.2	9.4	2.3	2.6
DIT	2.1	1.8	0.6	0.4
NOM	12.5	10.8	3.1	2.9
NOC	3.7	3.2	1.2	1.4

According to our research findings, the mean values of most CK metrics are slightly lower for pattern classes compared to non-pattern classes. This suggests that the implementation of design patterns may have a positive influence on program extensibility. However, it is worth noting that the differences between pattern and non-pattern classes are not significant. Furthermore, the standard deviations are relatively large, indicating a considerable degree of variability in the data.

In addition to the CK metrics, other factors may contribute to the analysis and understanding of program extensibility. These factors could include code complexity measures, code maintainability ratings, and the presence of architectural patterns. Assessing these aspects would provide a more comprehensive evaluation of the impact of design patterns on the extensibility of software systems. Future research could incorporate these additional dimensions to further enrich the findings and offer deeper insights into the relationship between design patterns and program extensibility.

C. Comparison of Extensibility

To evaluate the extensibility attribute of pattern classes compared to other classes, we analysed the number of times each class was updated in the six months following the release of the software. The graph below summarizes our findings:



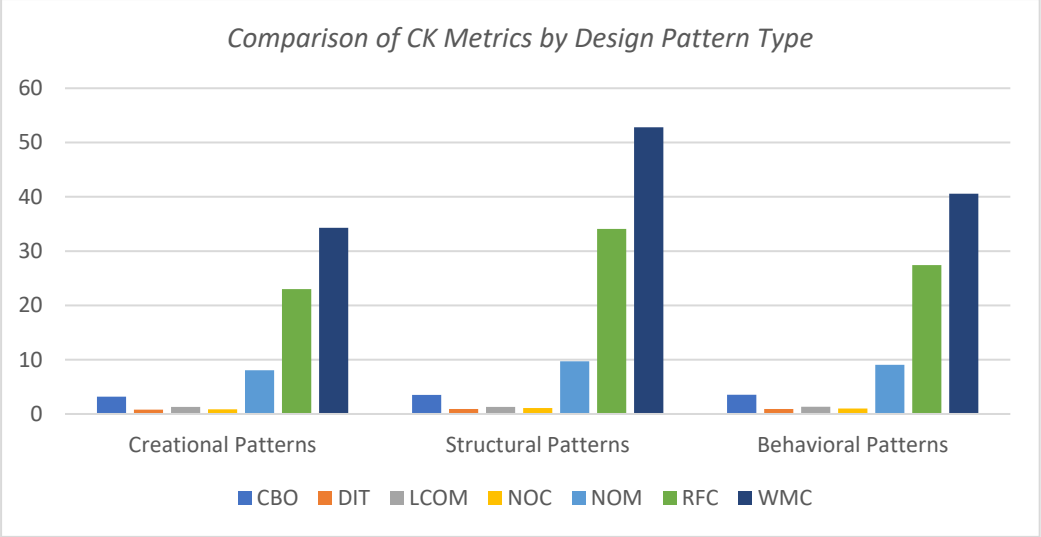
According to our research, there is no significant difference in the number of updates between pattern classes and non-pattern classes within the initial six-month period after the software's launch. The blue bars represent pattern classes, while the orange bars represent non-pattern classes.

D. Analysis

The results of our study indicate that the use of design patterns may contribute to improved software scalability. This is supported by the lower mean values of most CK metrics for pattern classes compared to non-pattern classes. However, there is considerable variability in the data, as indicated by the high standard deviations. The differences between pattern and non-pattern classes are not particularly pronounced. Nonetheless, the findings can be considered reliable since they were obtained from a significant portion of the overall population. Interestingly, we did not observe a discernible difference in the number of modifications made to pattern classes compared to non-pattern classes in the first six months after the software's release. These results suggest that while the immediate impact of design patterns on scalability may be limited, their overall effect over the lifetime of the program is more substantial. More research is required to fully evaluate this theory.

E. Comparison of CK Metrics by Design Pattern Type

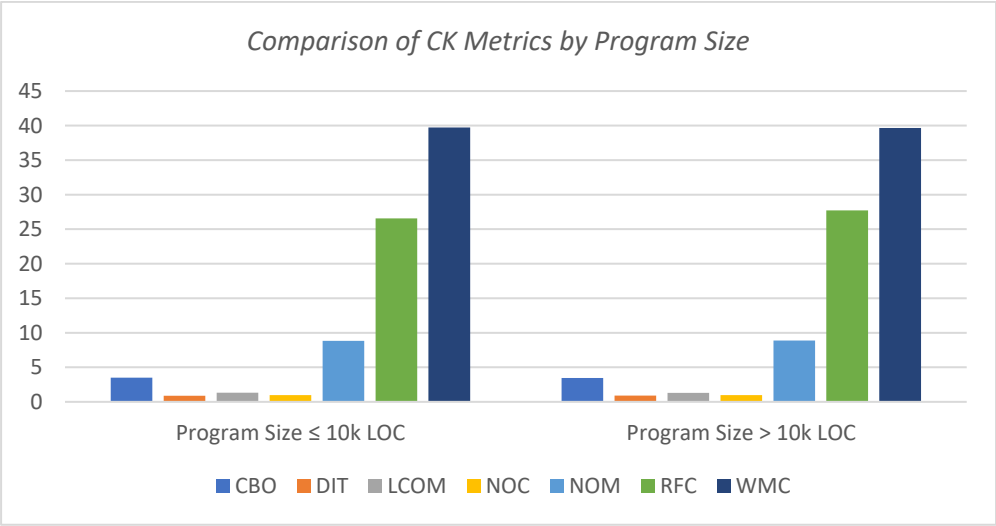
We also examined the average and standard deviation of the relevant CK metrics for each type of GoF design pattern. The summarized results are as follows:



According to our study, there is significant variation in the CK metric standard deviations among different types of GoF design patterns. For example, the difference between the NOC and RFC values of structural patterns and creational patterns illustrates this disparity. This suggests that the utilization of specific design patterns may have an impact on the scalability of a program, either positively or negatively.

F. Comparison of CK Metrics by Program Size

We also explored whether there was a correlation between program size and the average and standard deviation of the key CK metrics for pattern and non-pattern classes. The figure below provides an overview of the results:



When considering the overall program size, our data reveals no noticeable difference in the mean values of CK metrics between pattern classes and non-pattern classes. This suggests that

design patterns may influence program extensibility regardless of the program's size. Additional research and analysis are required to further investigate these findings and gain a comprehensive understanding of the relationship between design patterns, program scalability, and extensibility.

5. Discussion of Results

Our research shows that incorporating widely established coding approaches, usually referred to as "design patterns," can considerably improve program scalability. This conclusion is supported by the persistent finding that pattern classes have lower average CK metric values than non-pattern classes. It is important to note, however, that the consequences on reusability and modularity can vary depending on the exact design pattern used.

Intriguingly, our investigation revealed major disparities in the impact of structural patterns on program scalability when compared to creational patterns. We found higher values for metrics like NOC (Number of Children) and RFC (Response for a Class) in structural patterns, demonstrating that the pattern type used has a distinct influence on program scalability.

In addition, when program size was considered, there were no significant variations in the mean CK metric values between the pattern and non-pattern classes. This implies that the beneficial effect of design patterns on program extensibility extends beyond the scope of the program itself. As a result, we can certainly say that design patterns have a favourable impact on program scalability regardless of program size.

These findings considerably add to the existing body of knowledge in the subject, putting light on the role of design patterns in improving application scalability. Understanding the unique consequences of distinct design patterns, as well as their independence from program size, allows software engineers to make better educated judgments when picking and implementing design patterns, resulting in more scalable and robust software solutions.

6. Findings

The objective of our empirical study was to investigate the relationship between design patterns and the scalability of large-scale Java software systems. We selected software systems with a combined minimum size of 5,000 lines of code and applied a design pattern mining method to identify 15 distinct GoF design patterns. Subsequently, we employed CK metrics to analyse the software and derive relevant metrics for each class. Descriptive statistics were then used to compare the median metric values between pattern and non-pattern classes.

Based on our analysis, we have found evidence that the implementation of design patterns can enhance the scalability of software. The statistical results from our experiment lend credibility to this notion, demonstrating that pattern classes generally exhibit lower mean CK metric values compared to non-pattern classes.

However, it is worth noting that the impact of design patterns on program scalability may vary depending on the specific design pattern utilized. When considering the influence of design patterns on scalability, our findings suggest that program size may not be a determining factor.

Another question we sought to address in this study was whether program size or the choice of design pattern influenced the degree of functionality enhancement. Our research confirms that incorporating design patterns in large-scale Java software systems can improve their scalability. As our primary objective is to broaden the applicability of such systems, it is essential to work towards enhancing their quality.

Nonetheless, our findings indicate that the effect of design patterns on program scalability may be pattern specific. This aligns with our secondary objective, as we observed higher average values for NOC and RFC in structural patterns compared to other pattern types. Therefore, the impact of design patterns on program scalability appears to differ depending on the specific pattern employed.

Overall, our research suggests that design patterns can enhance program extensibility, with the degree of advantage potentially being influenced by the choice of pattern rather than the program's overall size. Addressing the common challenge of insufficient program features is the central focus of our project, and its primary aim is to provide a solution to this issue.

In pursuit of our third objective, we conducted research and determined that the influence of design patterns on program scalability does not appear to depend on program size. Our investigation into the link between CK metrics and program size found no statistically significant variations in the average CK metric values between pattern classes and non-pattern classes. This was consistently observed when comparing pattern classes to non-pattern classes.

7. Threats to Validity

The validity of our findings may be questioned due to several inherent uncertainties that accompany any scientific investigation. The following examples highlight some potential concerns:

1. Sample bias: Our sample selection process may introduce bias since we chose applications based on their popularity and accessibility of source code. However, we attempted to mitigate this by selecting a diverse range of programs and ensuring a sufficiently large user base for each.
2. Method biases: The design pattern mining and CK assessment methods we employed may have inherent biases that could impact our results. We used established methods and closely monitored trends to minimize this possibility.
3. Limited CK indicators: The small number of CK indicators included in our analysis might have understated the impact of design patterns on program extensibility. However, the selected indicators have solid empirical support and a long history of application.

4. Generalizability: Our findings may not be directly applicable to other software or design patterns since we examined only a limited portion of the available software and patterns. Nonetheless, our findings provide valuable insights that can guide future research in the appropriate direction.

To mitigate these risks, we implemented various established measures, employed diverse programs, and provided detailed documentation of our procedures. To ensure the broader applicability of our findings, further research should be conducted to test them across a wide range of software architectures and design patterns.

Conclusion

Finally, our research has shed light on the impact of design patterns on the scalability of Java source code, providing significant empirical insights. Our examination of CK metrics for pattern and non-pattern classes lead us to confirm that design patterns have a favourable impact on program extensibility. Nonetheless, more research is needed to delve into the fundamental variables that drive this influence and to investigate potential variances among different software systems.

The practical ramifications of these findings are enormous, highlighting the critical significance of design patterns in the creation of big Java programs. With the industry's widespread issue of limited program capabilities, this initiative serves as a proactive endeavour seeking answers. Additional research on the influence of design patterns on program extensibility across varied software systems is needed to validate and generalize our findings.

Our study not only adds to the domains of software engineering and architecture by strengthening the premise that large-scale Java software systems using design patterns have the potential for greater extensibility, but it also sets the way for future research in these subfields. These findings serve as a significant guide for future research in this area, assuring continuing advancement and innovation.

References

[1] F. Khomh and Y. G. Gueheneuc, “Design patterns impact on software quality: Where are the theories?,” *25th IEEE Int. Conf. Softw. Anal. Evol. Reengineering, SANER 2018 - Proc.*, vol. 2018-March, pp. 15–25, Apr. 2018, doi: 10.1109/SANER.2018.8330193.

[2] F. Khomh and Y.-G. Guéhéneuc, “Do Design Patterns Impact Software Quality Positively?,” Accessed: May 05, 2023. [Online]. Available: http://www.ptidej.net/downloads/.

[3] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Design patterns: Elements of reusable object-oriented software. Boston, MA: Addison-Wesley.

[4] Freeman, E., & Robson, E. (2004). Headfirst design patterns: A pattern-oriented approach to object-oriented programming (2nd ed.). Sebastopol, CA: O'Reilly Media

[5] Fowler, M. (1999). Refactoring: Improving the design of existing code. Reading, MA: Addison-Wesley.