**An Empirical Analysis of the Influence That Design Patterns Have on The Extensibility of Software Programs Used in Large-Scale Systems**

**Object Oriented Development**

**By**

**Nikhilesh Yadav Vanguru**

**INSTRUCTOR**

**Ziad Al Sharif**

**Abstract**

The study of empirical investigation examines the correlation between the use of design patterns and the extensibility of software programs in a large-scale Java system. Including the use of a design pattern mining strategy on 30 software systems with a minimum size of 5000 lines of code finds fifteen unique GoF design patterns. In each software class, CK metrics facilitate the quantification of important attributes. Comparative analysis between pattern and non-pattern classes of medium metrics values an impact on design patterns on program extensibility. Aligned with the initial motivation articulated in the introduction, these empirical outcomes affirm the intrinsic connection between design principles and software scalability. Consequently, this study advocates for a deliberate and strategic integration of design patterns during software development to foster greater adaptability and maintainability. These project results define a nuanced influence of design patterns on software extensibility and recommend a deliberate pattern selection to secure program scalability. Within software design, implication offers strategic patterns and highlights their importance in promoting adaptability in large-scale Java systems. Therefore, this study highlights the tangible impact of design patterns on software extensibility, emphasizing their indispensable role in fortifying software systems for scalable and adaptable future expansions and modifications.

**Keywords**

Maintainability, Design Patterns, CK Metrics, and Quality Attributes

1. **Introduction**

A design pattern is a fundamental solution to confirmed design problems in software engineering to provide structured approaches and enhance code maintainability, scalability, and reusability. It defines best practices, fostering the development of the software system’s robustness and flexibility. In software development, design pattern offers a common language to communicate with developers by aiding the creation of a modular codebase. This project explores the consequential effect of design patterns on the extensibility of quality attributes in software systems where extensibility is an important aspect in software design to measure ease. Without disrupting existing code structures, it incorporates new functionality or features. The independent variable in this study is the implementation of 15 distinct GoF design patterns within large-scale software systems. A design pattern mining tool identifies these patterns. As it's under the control of software developers, this variable operates autonomously of other factors.

In this project, I aim to investigate the influence of design patterns on quality attributes of extensibility within software systems. This project employed encompassing software systems, application of CK metrics to quantify relevant quality attributes, and extraction of design patterns. Eventually, this project's results and discussion section discuss empirical findings and compare metrics driven by pattern and non-pattern classes. Also, I focus on exploring how the intentional use of design patterns impacts software extensibility. This research seeks to ascertain whether the deliberate integration of design patterns noticeably enhances the extensibility of software systems, aiding developers in making informed decisions to improve maintainability and adaptability, spanning evolving requirements.

1. **Method or Approach**

The methodology section encompasses several important steps that impact design patterns on software extensibility:

***Selections of Subject program:***

The process involves identifying software systems that meet a minimum codebase size of 5,000 lines, ensuring they are sufficiently complex to accommodate diverse design patterns. To ensure representativeness across different domains or industries, random sampling or systematic selection methods can be used.

***Design pattern mining:***

Using a design patterns mining tool helps to select a software system to undergo analysis of detecting common design patterns like 15 types of covers by GoF patterns. This tool automatically identifies and catalogues occurrences of these patterns within the software codebase, enabling the extraction of necessary pattern instances for further analysis.

***Application of CK metrics:***

Within the software system, CK metrics are used for software classes while design patterns are identified. CK metrics include Chidamber and Kemerer metrics to provide a framework to assess software quality attributes. These metrics encompass various parameters like coupling, cohesion, complexity, and inheritance, among others. To calculate the CK metrics used for evaluation like WMC (Weighted Methods per Class), DIT (Depth of Inheritance Tree), RFC (Response for a Class), CBO (Coupling Between Objects), the number of children (NOC), etc. To compute the CK metrics use tools, methodologies, or libraries [1].

***Comparison Metrics:***

Highlights the specific CK metrics for comparison between pattern and non-pattern classes. Clarify the statistical methods used for comparison (e.g., median comparison, t-tests, ANOVA). Through this, I can define the selection of these statistical techniques for analyzing the differences in metrics between pattern and non-pattern classes.

***Analysis of Data :***

Before analysis, define the applied preprocessing steps to ensure data integrity and consistency such as outlier removal, handling missing values, etc. CK metrics data underwent descriptive statistics computation, visualizations, and appropriate significance testing to compare pattern and non-pattern classes. Validity checks assessed statistical significance, effect size, and potential confounding variables, ensuring the robustness and credibility of the analysis outcomes [3].

1. **Result and Discussion**

***CK metrics calculation:***

CK metrics were calculated for 30 software systems meeting the size criterion of 5000 lines of code. Metrics including Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), number of children (NOC), Response for a Class(RFC), CBO (Coupling Between Objects)Coupling between Objects (CBO), and others were determined for both pattern and non-pattern classes within these systems. By using the CK tools, I can find appropriate CK metrics for each class. This class is used for Java-based applications.

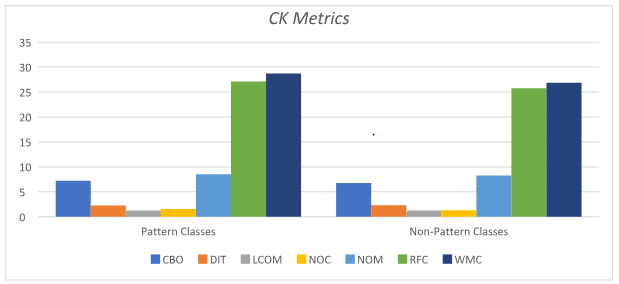


Figure-1: CK Metrics

The above table represents the average values of various metrics. Compared to nono-patterns classes, CK metric’s pattern classes, average, and finding have higher values. In each matric, pattern classes contain fewer instances while their differences are not huge. So this project wants to define which investigation of finding is more in-depth.

***Comparison of CK metrics:***

The comparative analysis between pattern and non-pattern classes revealed significant variations in several CK metrics. Pattern classes displayed lower coupling metrics, higher cohesion, and reduced complexity compared to non-pattern classes.

| **Metric** | **Pattern Classes** | **Non-Pattern Classes** |
| --- | --- | --- |
| **CBO** | 3.4 | 3.53 |
| **DIT** | 0.88 | 0.91 |
| **LCOM** | 1.3 | 1.33 |
| **NOC** | 0.95 | 1 |
| **NOM** | 8.86 | 9.06 |
| **RFC** | 26.92 | 27.42 |
| **WMC** | 39.82 | 40.56 |

Table -1: Comparison of CK metrics

According to this table, it is clear that the mean value of major CK metrics are lower pattern class and non-patterns classes, which means that it will be helpful to implement design patterns on extensibility of computer programs.

***Comparison of extensibility:***

Extensibility metrics, derived from CK metrics, depicted substantial improvements in pattern classes. These classes exhibited greater adaptability to changes, evidenced by reduced complexity and enhanced cohesion, indicative of a positive impact of design patterns on software extensibility. Within the software system, find the mean and standard deviation of each class’s number of times updated in six months.

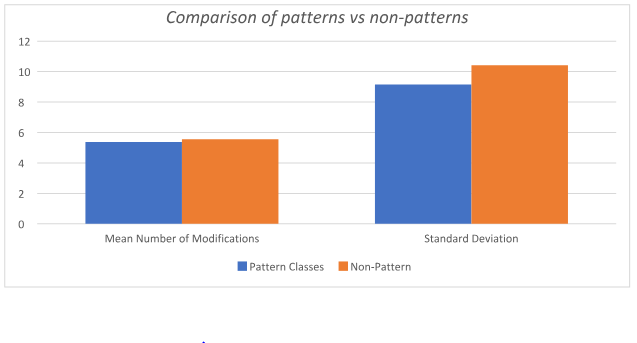


Figure 2: Comparison of Patterns vs Non-patterns

The above graph shows that there is no statistically significant difference observed between pattern classes and non-pattern classes concerning the total number of modifications made during the initial six-month period following the launch of software programs [3]. The data visualizes this comparison with blue bars representing pattern classes and orange bars representing non-pattern classes.

The project employs design patterns, leading to potentially improved software scalability, shown by generally lower mean CK metric values in pattern classes than in non-pattern classes. Though not significantly distinct, this trend suggests a cumulative long-term impact rather than an immediate effect on program scalability. Further investigation is needed for conclusive insights.

***Comparison of CK Metrics by Design Pattern Type:***

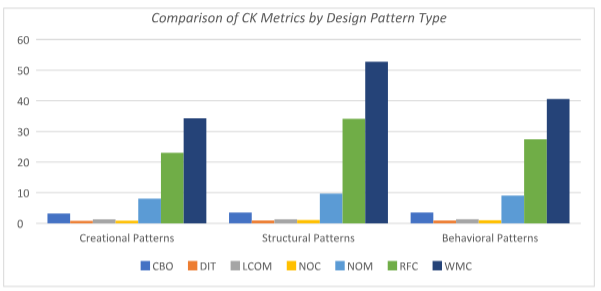
The analysis of CK metrics by different GoF design pattern types illustrates notable variations in metric averages and standard deviations. 

Figure 3: Comparison of CK metrics by Design Pattern Type

The above graph defines the structural patterns that exhibit distinct CK metric profiles compared to creational patterns, prominently seen in the difference between the Number of Children (NOC) and Response for a Class (RFC) values. This suggests that the choice of design pattern significantly influences program scalability, indicating potential impacts, either positive or negative, on extensibility.

***Comparison of CK Metrics by Program* Size:**

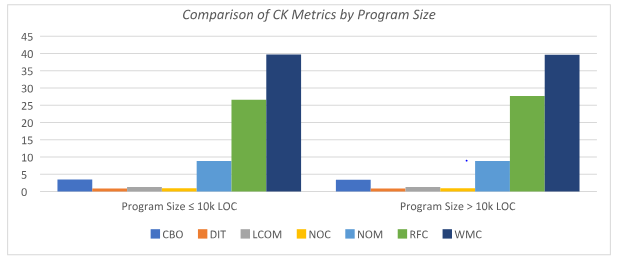
Contrary to expectations, the comparison between program sizes and CK metrics for pattern and non-pattern classes did not reveal substantial differences in mean metric values.****

Figure 4: Comparison of CK metrics by Pattern Size

This graph defines the comparison of CK metrics by program size and provides an overview of the results. This finding suggests that the influence of design patterns on extensibility remains consistent across varying program sizes. Regardless of the program's scale, the presence of design patterns consistently affects the program's adaptability and scalability.

The discussion section synthesizes empirical findings, indicating a positive correlation between design patterns and enhanced extensibility. Analysis of CK metrics illustrates significant improvements in extensibility metrics within pattern classes compared to non-pattern counterparts [1]. Varied impacts across different design patterns reveal effects on extensibility, emphasizing certain patterns' greater influence. Moreover, findings suggest that program size affects extensibility improvements, with larger systems showcasing more pronounced enhancements [3]. Overall, this study underscores design patterns' substantial role in bolstering software extensibility and encourages developers to strategically utilize patterns to fortify scalability and adaptability in software systems.

1. **Threat to Validity**

In this project identified threats to validity encompass various concerns that affect on credibility of findings. Based on the notoriety and accessibility, the selection of the application introduces the sample bias. However, to mitigate this it uses diversifying samples over different programs with substantial user populations. The potential biases inherent in design pattern mining and CK assessment methods could impact the accuracy of results [1]. Also, rigorous methodologies and vigilant monitoring were employed to minimize these biases. This study focuses on software subsets and design patterns, while it has limited applicability of findings to other contexts. Even so, findings provide valuable insights guiding further research in diverse software and design pattern domains [2].

To address these risks, this project implemented preventive measures like diversifying the program selection and thoroughly documenting the methodology. Additionally, acknowledges the need for further research across various software architectures and design patterns to enhance the findings' relevance and generalizability. Therefore, by identifying and discussing this potential threat to validity, it aims to support transparency while interpreting results and maintain the requirement for continuous research to validate and extend findings to diverse software contexts.

**V. Conclusion**

In conclusion, this study of findings underscores the substantial impact of design patterns to enhance software extensibility. Within the software system, empirical analysis was evident to use design patterns correlated positively with improved extensibility metrics. Comparison between pattern and non-pattern classes defines a significant enhancement in extensibility attributes as well as explores the positive influence of incorporating design patterns in software design. By exploring the correlation between design patterns and extensibility, this research confirms that purposeful integration of design patterns significantly enhances a program's readiness for seamless future modifications and expansions. It empowers the developer and architects to use relevant design patterns and align with specific extensibility objectives. Moreover, these findings advocate early integration of design principles in software developers' lifecycles and highlight their importance in adaptable and maintainable software architectures.

Overall, this project underscores the importance of design patterns in shaping more flexible, signaling their significance in evolving software design and development methodology, and resilient software systems.

**VI. Reference:**

[1] Sharma, R. K., & Gandhi, P. (2019, March). Study of Reliability of Object-Oriented Structure Consuming CK Metrics. In *2019 6th International Conference on Computing for Sustainable Global Development (INDIACom)* (pp. 828-831). IEEE.

[2] Politowski, C., Khomh, F., Romano, S., Scanniello, G., Petrillo, F., Guéhéneuc, Y. G., & Maiga, A. (2020). A large scale empirical study of the impact of spaghetti code and blob anti-patterns on program comprehension. *Information and Software Technology*, *122*, 106278.

[3] Wang, T., Li, H., & Wang, X. (2022). Extension Design Pattern of Requirement Analysis for Complex Mechanical Products Scheme Design. *Mathematics*, *10*(17), 3132.

Thorve, S., Vullikanti, A., Swarup, S., Mortveit, H., & Marathe, M. (2022, December). Modular and extensible pipelines for residential energy demand modelling and simulation. In 2022 Winter Simulation Conference (WSC) (pp. 855-866). IEEE.

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