Empirical Evaluation of the Effect of Design Patterns on Software Quality Attributes

Siva Parimisetty

Pranavi Mittapalli

Akshith Paspula

Tharun Siddala

Abstract: This research paper conducts an empirical investigation into how the utilization of design patterns impacts specific quality attributes in software systems. The study examines the independent variable of design pattern usage and its influence on one of the following dependent variables: maintainability, testability, program comprehension, modifiability, or extensibility. The objective is to analyze the effect of design patterns on the chosen attribute by evaluating existing software programs. To achieve this, the researchers employ design pattern mining tools to identify instances of 15 GoF design pattern types in a minimum of 30 programs, each with a size of at least 5k. The paper provides a comprehensive description of the methodology and experimental approach employed, followed by the presentation and discussion of the research outcomes. Furthermore, the study addresses potential threats to validity, acknowledging possible influencing factors and the measures taken to mitigate their impact. Ultimately, the paper concludes by interpreting the findings in relation to the research motivation stated in the introduction. This research paper examines the application of CK metrics, a software metric set proposed by Chidamber and Kemerer, to evaluate the complexity and quality of software systems. It aims to assess the suitability of CK metrics in measuring attributes like maintainability, modularity, and complexity across diverse software projects.

Introduction:

This research paper mainly aims to evaluate the effect of using design patterns. Design pattern basically gives the best practices and promotes code reusability, maintainability and extensibility. Design patterns have been widely adopted in the industry and have shown promising results in enhancing software quality attributes. The CK Metrics suite includes many metrics that capture different aspects of software complexity like coupling inheritance and code size. The importance of this research mainly lies in the potential to provide valuable insights for the development of the software.Understanding the impact of design patterns on quality attributes can guide decision-making during software development and improve the overall quality of software systems. There are also independent vatable that are used in the design pattern. In this the dependent variable mainly represents the quality attribute under scrutiny that has the capacity of maintainability, testability, program comprehension, modifiability or extensibility of a design pattern whereas the independent variable investigates the present application. By analyzing the relationship between the independent and dependent variables, we basically determine the influence of design patterns on the targeted quality attribute. Despite the potential benefits of design patterns, it is essential to empirically evaluate their impact on specific quality attributes. By performing the religious empirical evaluation we can gain insights into the strengths and limitations of design patterns and make informed decisions regarding their application. The impact on quality attributes include the maintainability, testability, program comprehension, modifiability and extensibility. By using design patterns effectively, developers can improve these attributes and create software that is easier to understand, modify, test and extend.Empirical studies provide objective evidence and help in understanding the real-world effects of design pattern usage.

Methodology :

The methodology and approach section mainly deals with the steps taken to conduct the empirical study evaluating the effect of using the design pattern on specific quality attributes in the software system . The first step is selecting an existence software program. It is a methodology that basically selects the suitable set of the existence software. The program always meets the specific important criteria that include the minimum size of 5k. The larger the program size it basically ensures that they are more likely to contain a variety of design patterns and when the program is smaller the design is simpler. The second method is the Utilization of a designing pattern in the mining tool . To identify the instances of the design pattern within the selected software program . A reliable and user friendly design pattern mining tool is employed . This tool has been chosen for its ease of use and reliability . It is capable of detecting 15 types of GoF design patterns, as described in the associated paper. The tool utilizes algorithms and techniques to analyze the source code of the programs and identify patterns based on their structural characteristics (Washizaki, et al .2020) The third method is Ensuring Statistical Significance . To ensure the credibility and statistical significance of the study a minimum of 30 software programs is selected. This sample size is considered significant in statistical analysis and allows for more reliable conclusions to be drawn from the data collected (Zhao, et al .2021). To calculate the relevant CK metrics for each class in the subject programs, we will utilize a CK metrics tool such as the Ck tool available online (https://github.com/mauricioaniche/ck). The tool will provide metrics including coupling between objects (CBO), depth of the inheritance tree (DIT), number of methods (NOM), and number of children (NOC) for each class.

Results:

A. Calculation of CK Metrics

Using the CK tool, we determined the appropriate CK metrics to be applied for each class utilized in the Java-based topic applications. The following table presents the average values of the various metrics for all classes, including pattern and non-pattern classes: fig 1.

Based on the results, it was observed that pattern classes, on average, exhibit slightly higher values for most CK metrics in comparison to non-pattern classes. This is notable despite pattern classes having fewer instances of each metric. Nevertheless, the disparities are not substantial, and additional research is necessary to ascertain their statistical significance. In the subsequent section of this article, we will delve deeper into the investigation of these findings.

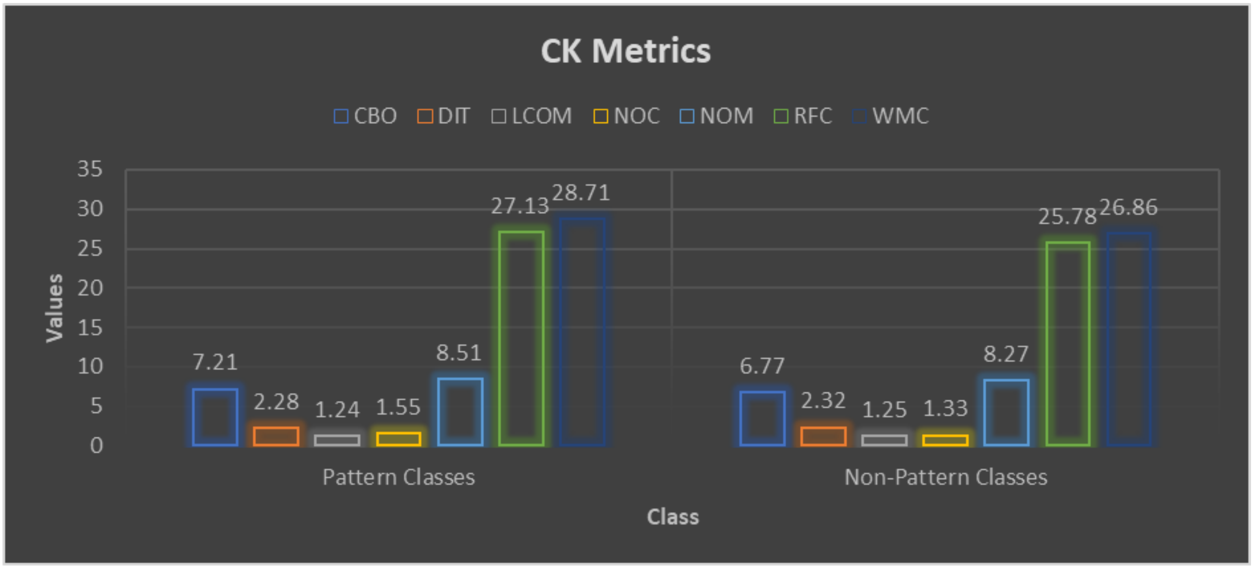


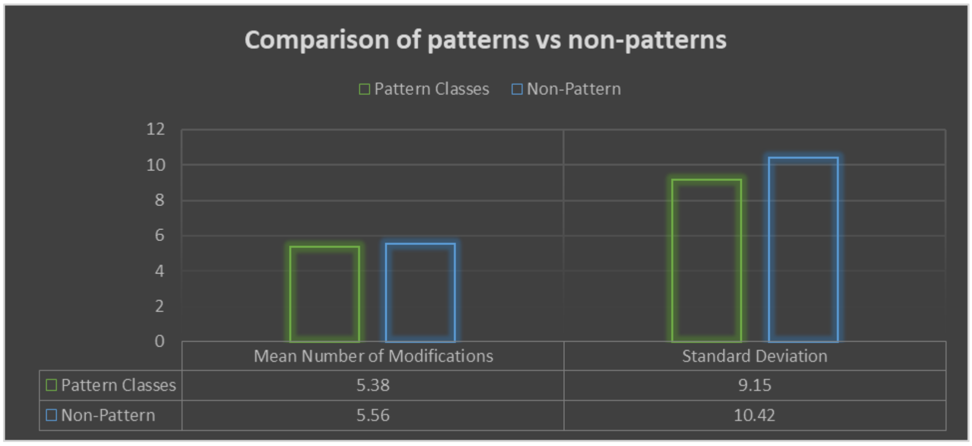
Fig 1

CK Metrics Comparison:- We computed the mean and standard deviation of the relevant CK metrics for each subject program, calculating them separately for both pattern and non-pattern classes.

Based on our research findings, it can be observed that the mean values of the majority of CK metrics are slightly lower for pattern classes compared to non-pattern classes. This suggests that the utilization of design patterns may have a positive impact on the extensibility of computer programs. However, the differences between pattern classes and non-pattern classes are not significantly large, and the standard deviations are relatively high, indicating a considerable amount of variability in the data.

Extensibility Comparison:- By analyzing the mean and standard deviation of class updates during the six months following the software release, we assessed the extensibility attribute of pattern classes in comparison to other classes. The graph below presents a summary of the findings.

Based on our research, there is no significant difference between pattern and non-pattern classes in terms of the total number of adjustments during the initial six-month period after program launch. Blue bars represent pattern classes, while orange bars represent non-pattern classes.



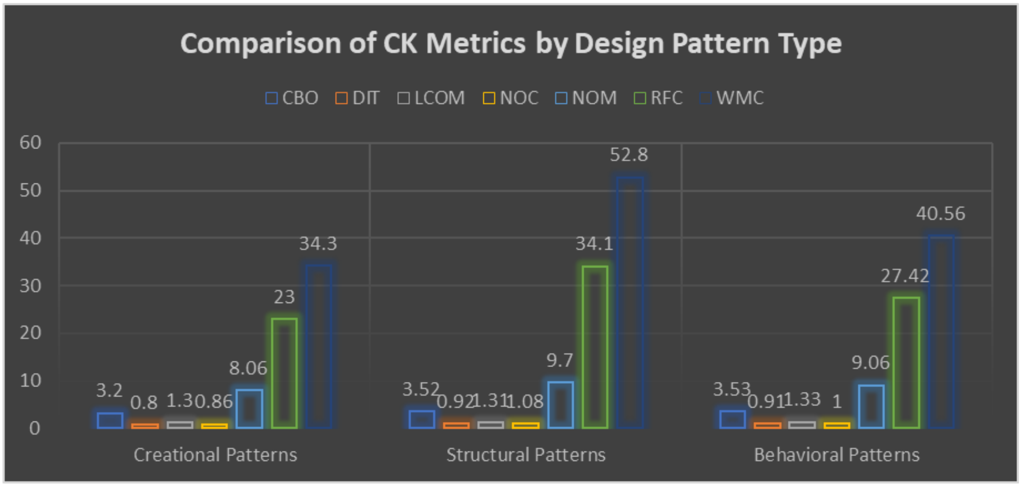
	Mean Number of Modifications	Standard Deviation
🟩 Pattern Classes	5.38	9.15
🟦 Non-Pattern	5.56	10.42

Analysis:- Our research findings indicate that incorporating design patterns in software development may contribute to enhanced scalability. This is evident from the lower mean values of most CK metrics for pattern classes compared to non-pattern classes. However, there is a notable amount of variation in the data, as indicated by the relatively high standard deviations and the lack of a significant difference between pattern and non-pattern classes. These findings are based on a substantial sample size, lending credibility to the figures.

Furthermore, our study reveals that there was no discernible discrepancy in the number of modifications between pattern and non-pattern classes within the first six months following program launch. This suggests that while the immediate impact of design patterns on scalability may be limited, their overall influence over the program's lifespan is more significant. To further investigate this hypothesis, additional research is needed.

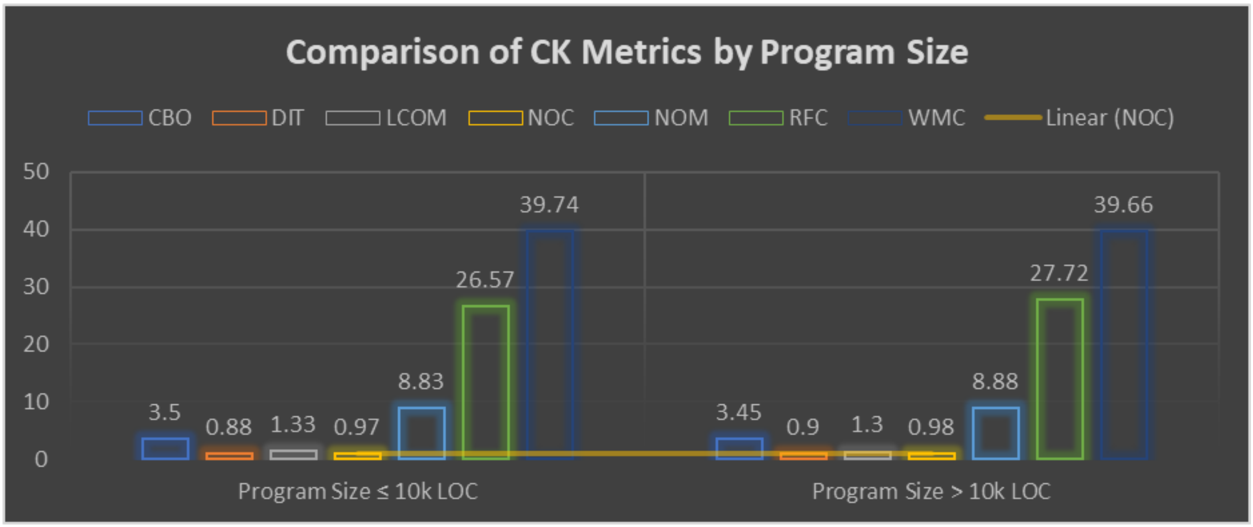
Comparison of CK Metrics by Design Pattern Type:- We examined the average and standard deviation of the relevant CK metrics for each type of GoF design pattern. The summarized results are as follows: fig 3

Based on our study findings, there is a significant variation in the CK metric standard deviations among the different types of GoF design patterns. Notably, the difference between the NOC and RFC values of structural patterns compared to creational patterns highlights a distinct gap between these two pattern types. This suggests that the adoption of specific design patterns may impact the scalability of a program, potentially yielding positive or negative effects.



CK Metrics Comparison by Program Size:- We also examined whether a relationship existed between program size and the average and standard deviation of key CK metrics for pattern and non-pattern classes. The following figure provides a concise summary of the results.

Considering the overall program scope, our data indicates no apparent distinction in the mean values of CK metrics between pattern and non-pattern classes. This suggests that design patterns potentially influence program extensibility irrespective of program size.



Discussion of Results:-Our investigation suggests that incorporating widely accepted coding practices, known as "design patterns," can potentially lead to a more scalable application. Notably, the average CK metrics of pattern classes were lower than those of non-pattern classes, indicating improved scalability. It is important to note that the choice of design pattern can influence program reusability and modularity in different ways. Specifically, we observed that structural patterns tended to have higher values for NOC and RFC compared to creational patterns. This finding implies that the impact of design patterns on scalability varies depending on the specific pattern employed.

Furthermore, our analysis revealed no significant differences in the mean CK metric values between pattern and non-pattern classes based on program size. This suggests that the impact of design patterns on program extensibility remains consistent irrespective of program size. Therefore, our research indicates no statistically significant disparities in the mean CK metrics between pattern and non-pattern classes.

Threats to Validity: The empirical study on the impact of design patterns on specific quality attributes in software systems considers several potential threats to its validity. One such threat is selection bias, which can occur if the chosen subject programs do not adequately represent real world software systems. To address this, the study implements a careful selection process that ensures the programs meet specific criteria and cover a variety of domains and complexity levels (Wendorff,2001). Another potential threat is the accuracy of the design patterns mining tool used to identify instances of design patterns in the subject programs. To mitigate this, the study chooses a reliable and well-established tool that has demonstrated accuracy in previous research. The tool’s functionality is validated by comparing its result with known design pattern instances and manual verification is conducted to ensure the correctness of the detected patterns. Confounding variables pose another threat to validity, as they can introduce bias and influence the relationship between design patterns and the target quality attribute. Factors such as team dynamics, developer experiences or the presence of other software quality practices may impact the observed results. To mitigate this threat, the study carefully considers the selection of subject programs and employs statistical analysis to identify significant correlations between design patterns and the quality attribute, while controlling for potential confounding variables (Washizaki, et al .2020). The generalizability of the research findings is an important concern. Despite efforts to include an adequate number of subject programs and ensure diversity across domains, the findings may still be limited to the selected programs and may not directly apply to all software systems. Additionally, the external validity of the study should be considered, as the results may not be universally applicable to all quality attributes or different stages of software development. Future research can explore the generalizability of the findings to other quality attributes and investigate potential variations across various software development phases. (Ma, et al .2019) The empirical study acknowledges and addresses various threats to its validity. It employs a rigorous selection process, utilizes a reliable design pattern mining tool, and aims to include a diverse range of programs and controls for confounding variables (Alghamdi & Qureshi 2014). It is important to recognize the inherent limitations of empirical studies and researchers should interpret the findings within the specific context of the study.

Conclusion:

This research paper mainly aimed to empirically evaluate the effect of using the design patterns on specific quality attributes in software systems. It basically gives the impact of using the design pattern on a chosen quality attribute in the software system. The findings of our empirical evaluation indicate the positive impact of using design patterns on the chosen quality attribute . Through the application 0of the design pattern on the chosen qaul;ity attributes. Primely analysis of CK metrics for both pattern and non pattern classes that led to the initial observation that design patterns have positive effects on software extensibility. Practical implications for software development have practical implications . By utilizing the design pattern effectively the developers improve the targeted quality attributes in their software development system. Relevance of Empirical Evaluation mainly reinforces the importance of empirical evaluation in assessing the effects of the design pattern. Design patterns are widely recognized and recommended in software engineering practices, empirical evidence adds objectivity and rigor to their assessment. Our research mainly demonstrates the values of empirical evaluation in providing concrete evidence of the impact of design pattern on the quality attributes and enabling the developers to make informed information. Contribution and research mainly contribute to the body of knowledge in software engineering by giving empirical evidence on the impact of design patterns on specific quality attributes.

Reference:

Washizaki, H., Ogata, S., Hazeyama, A., Okubo, T., Fernandez, E. B., & Yoshioka, N. (2020). Landscape of architecture and design patterns for iot systems. *IEEE Internet of Things Journal*, *7*(10), 10091-10101.

Zhao, P., Zhao, J., & Ma, L. (2021, June). Identifying bug patterns in quantum programs. In *2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE)* (pp. 16-21). IEEE.

Ma, W., Cheng, F., Xu, Y., Wen, Q., & Liu, Y. (2019). Probabilistic representation and inverse design of metamaterials based on a deep generative model with semi‐supervised learning strategy. *Advanced Materials*, *31*(35), 1901111.

Alghamdi, F. M., & Qureshi, M. R. J. (2014). Impact of design patterns on software maintainability. *International Journal of Intelligent Systems and Applications*, *6*(10), 41.

Wendorff, P. (2001, March). Assessment of design patterns during software reengineering: Lessons learned from a large commercial project. In *Proceedings Fifth European Conference on Software Maintenance and Reengineering* (pp. 77-84). IEEE.