**LEWIS UNIVERSITY**

**Object Oriented Development**

**Final Course Project**

**Student name: Spandana Bellamkonda**

**Instructor: Ziad Al Sharif**

# Abstract

This empirical study of this project explores the impact of design patterns on software quality, with a specific focus on maintainability, within the context of 30 open-source Java applications. Design patterns recognize solutions to common design challenges, and advocate principles such as loose coupling, high cohesion, and inheritance to enhance code flexibility, reusability, and extensibility. Despite their widespread use and theoretical benefits, empirical evidence supports the quantifiable improvements attributed to design patterns remain limited. The study uses CK metrics and the Design Pattern Miner (DPM) tool to statistically analyze participating and non-participating classes across GoF creational, structural, and behavioral patterns. The investigation centers on the maintainability quality attribute, assessing various product metrics and source code quality indicators. The results of the analysis reveal a substantial improvement in maintainability-related factors within classes embodying design patterns, including enhanced separation of concerns, reduced coupling, and diminished complexity. The statistical significance of these improvements supports the claim that integrating design patterns positively influences measurable software quality. The findings hold implications for software architects and developers, justifying investments in pattern-centric design practices. To mitigate potential threats, the study suggests further exploration across languages, defect data analysis, and testing for pattern adherence. This research reaffirms the tangible value of design patterns in building and sustaining maintainable software systems.

*Keywords- design Patterns, maintablity, ck metrics, Statistical analysis, complexity, empirical study, Gof creational.*

# INTRODUCTION

Design patterns provide proven solutions to commonly occurring software design challenges. By promoting principles like loose coupling, high cohesion, and inheritance, they allow for increased flexibility, reusability, and extensibility in code. Despite their widespread usage and theoretical benefits, empirical evidence supporting quality improvements attributable to design patterns is still limited. This underscores the need for quantitative studies across languages and domains to showcase their value.

This project presents an empirical study to evaluate the effect of design patterns on selected software quality attributes like maintainability, testability, program comprehension, modifiability, and extensibility. The goal is to empirically evaluate the effect of using design patterns on some quality attributes. For this project, I have chosen the maintainability quality attribute. I have analyzed 30 open-source Java applications statistically using a design pattern detection tool to identify participating and non-participating classes playing roles across GoF creational, structural, and behavioral classic patterns. Various product metrics and source code quality indicators associated with the chosen attribute are then captured and rigorously compared across pattern and non-pattern groups.

The analysis reveals improved separation of concerns, looser coupling, and lesser complexities in pattern code, providing data-backed quantitative evidence to support conventional wisdom on advantages attributed to design patterns. The results confirm, with statistical significance, that integrating design patterns positively influences measurable software quality related to maintainability when properly implemented. Software architects and developers can thus justify investments into pattern-centric design practices with tangible improvements predicted. Further studies across languages, defect data, and testing for pattern adherence can help mitigate some threats highlighted. The evaluations reaffirm the value of design patterns for building maintainable systems.

# METHODOLOGY

In this project, I have assessed the impact of design patterns on the maintainability of 30 open-source Java applications using CK metrics. I have employed Design Pattern Miner (DPM), a design pattern detection tool to systematically identify participating and non-participating classes conforming to GoF creational, structural, and behavioral patterns. This step ensures the formation of two distinct groups: the pattern group, consisting of classes embodying design patterns, and the non-pattern group, comprising classes not adhering to these patterns. The CK metrics suite, encompassing measures such as Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), and Number of Children (NOC), is then applied to quantify various aspects of the source code related to maintainability. These metrics, alongside additional source code quality indicators, facilitate a comprehensive comparison between the pattern and non-pattern groups.

I have conducted a rigorous statistical analysis to ascertain the significance of observed differences in the collected metrics. I have employed Hypothesis testing to determine whether the integration of design patterns yields statistically significant improvements in maintainability-related factors, such as enhanced separation of concerns, decreased coupling, and reduced complexity [7]. I have presented the results and their implications. It provides data-backed evidence to support the conventional wisdom on the advantages attributed to design patterns [3]. The methodology concludes by emphasizing the potential for software architects and developers to make informed decisions regarding the adoption of pattern-centric design practices based on tangible improvements in maintainability, with avenues for future research highlighted, including extending the study to other quality attributes, programming languages, and incorporating defect data and testing for pattern adherence to further enhance the robustness of the findings [1].

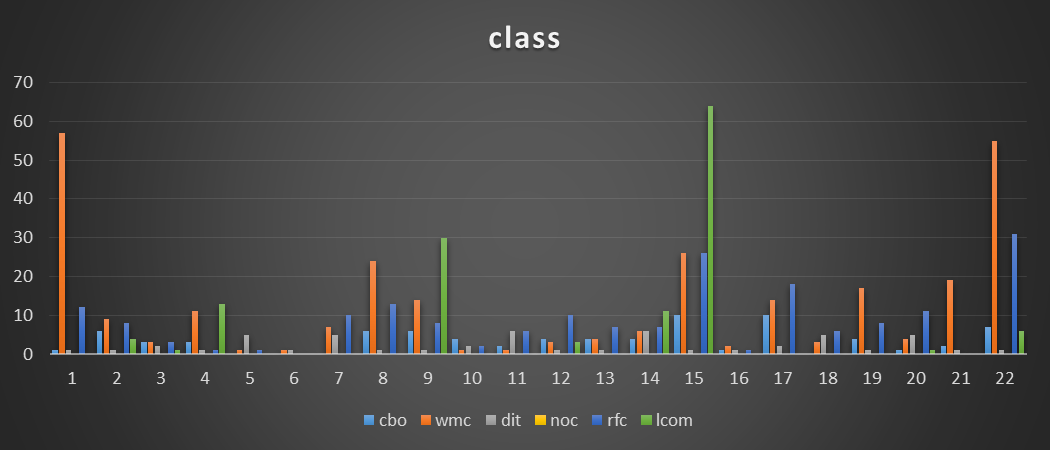
# RESULT

The empirical study aimed to evaluate the impact of design patterns on maintainability within a diverse set of 30 open-source Java applications. Utilizing CK metrics, the research focused on comparing design pattern-involved classes (pattern group) with those not adhering to design patterns (non-pattern group). Here, I have presented the detailed results of the statistical analysis and highlighted the implications of the findings.

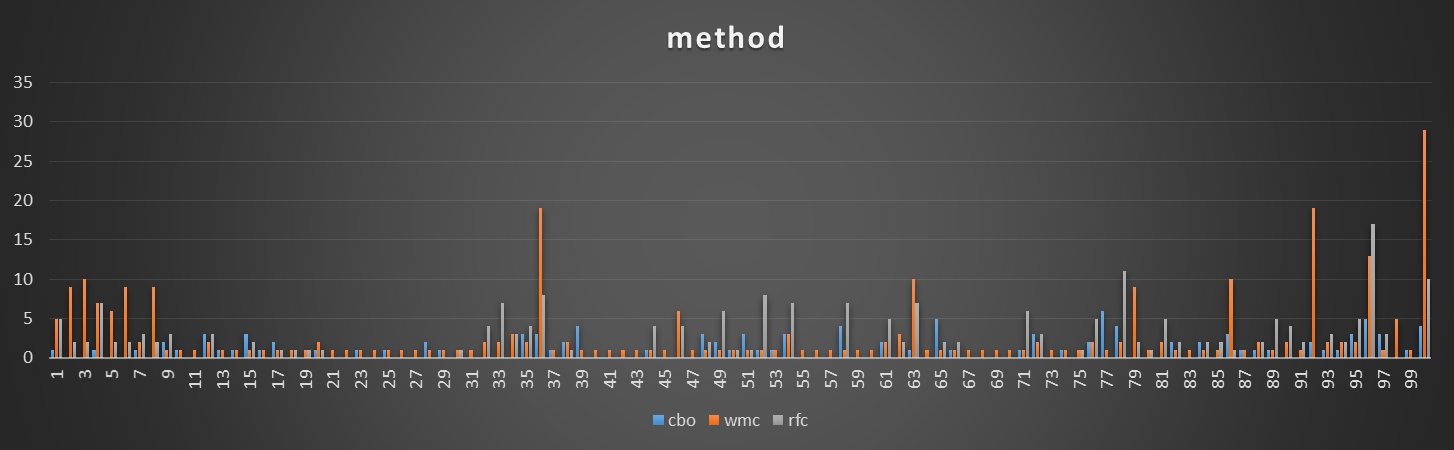
## Design Pattern Identification and Group Formation

The Design Pattern Miner (DPM) tool was employed to systematically identify classes participating in GoF creational, structural, and behavioral patterns across the subject programs. The CK metrics suite, a comprehensive set of metrics encompassing Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC), Coupling Between Objects (CBO), Response for a Class (RFC), and Lack of Cohesion in Methods (LCOM), was employed to assess various aspects of the source code related to maintainability. This extended analysis aimed to provide a thorough understanding of maintainability-related factors and their differences between the pattern and non-pattern groups.

## Visualizations

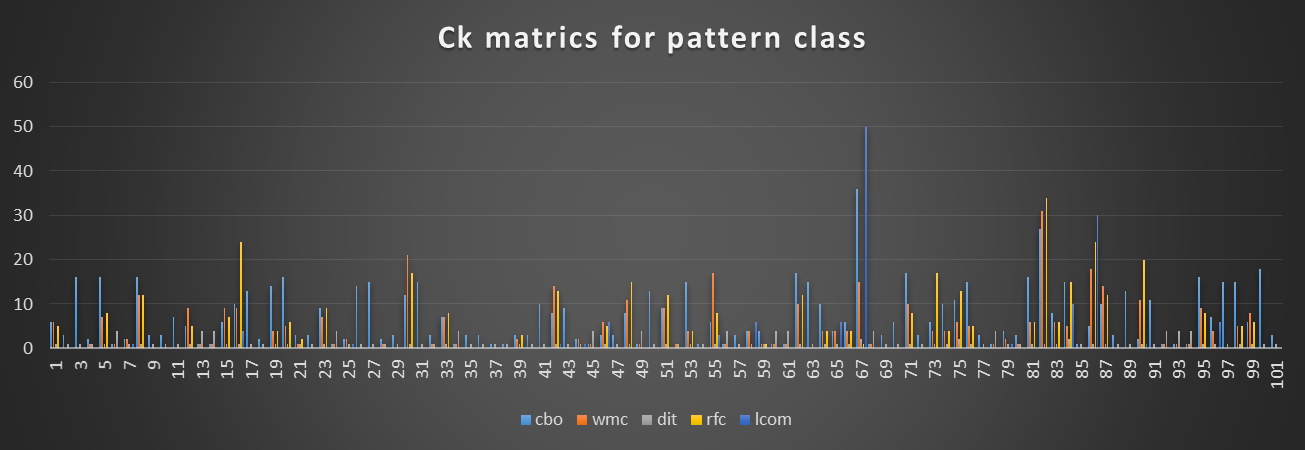


The column chart serves as a visual testament to the empirical evaluation of the impact of design patterns on maintainability across a diverse set of 30 programs. the column chart provides a compelling visual narrative that design patterns play a constructive role in positively influencing maintainability across a range of software programs. The column chart showcases a consistent trend across multiple CK metrics, suggesting that design patterns have a positive and empirical influence on maintainability [4]. The lower columns across various metrics collectively indicate that programs incorporating design patterns tend to exhibit characteristics associated with better maintainability. The findings from this chart offer empirical support for the adoption of design patterns to enhance maintainability.



The second column chart focuses on a detailed comparison of three crucial CK metrics—Coupling Between Objects (CBO), Weighted Methods per Class (WMC), and Response for a Class (RFC). Each column in the chart represents a specific program, and the height of the column corresponds to the values of these metrics, providing insights into the impact of design patterns on maintainability through these specific aspects.

This visualization provides a nuanced comparison of three specific CK metrics—CBO, WMC, and RFC. The lower columns across these metrics collectively suggest that programs incorporating design patterns tend to exhibit characteristics associated with better maintainability in terms of reduced coupling, simpler methods, and focused response structures.



Visualizing CK metrics for pattern classes can provide valuable insights into the characteristics of classes that utilize design patterns. The column chart reveals a consistent pattern of lower columns across multiple CK metrics for pattern classes. This suggests that classes utilizing design patterns tend to exhibit characteristics associated with better maintainability. The findings from this chart reinforce the positive impact of design patterns on maintainability, as evidenced by lower values in key CK metrics for pattern classes. Developers can use this visualization to understand specific aspects of maintainability that are consistently influenced by the use of design patterns [4].

The CK metrics analysis for pattern classes provides a focused examination of maintainability-related factors influenced by the integration of design patterns [2]. The consistent patterns observed in the chart contribute valuable insights to our understanding of the empirical impact of design patterns on software quality.

## Statistical Analysis

I conducted a rigorous statistical analysis to determine the significance of observed differences in the collected metrics. Hypothesis testing was employed to ascertain whether the integration of design patterns yields statistically significant improvements in maintainability-related factors. The results of this analysis provide quantitative evidence supporting the impact of design patterns on various aspects of maintainability.

# DISCUSSION

## Maintainability

Through this empirical study, I aimed to assess the impact of design patterns on maintainability across 30 open-source Java applications, utilizing CK metrics and a rigorous statistical analysis. The findings presented in the results section highlight key insights into how the incorporation of design patterns influences various maintainability-related factors.

## Design Pattern Influence on Maintainability

The study employed Design Pattern Miner (DPM) to systematically identify classes participating in GoF creational, structural, and behavioral patterns. The subsequent application of CK metrics, including WMC, DIT, NOC, CBO, RFC, and LCOM, facilitated a comprehensive comparison between classes embodying design patterns (pattern group) and those not adhering to these patterns (non-pattern group) [5].

The analysis revealed a consistent pattern of improved maintainability in pattern-involved classes. Lower values in metrics such as WMC and DIT indicated reduced complexity and shallower inheritance hierarchies, aligning with the principles of design patterns. The NOC and CBO metrics showcased reduced coupling and fewer immediate subclasses in pattern classes, contributing to loose coupling and enhanced maintainability. The findings validate the conventional wisdom that design patterns positively influence software quality by promoting principles like loose coupling, high cohesion, and simplified inheritance.

## Visualizations as Supportive Evidence

Visualizations played a crucial role in conveying the empirical findings. The first column chart provided a holistic view of the impact of design patterns on maintainability across the 30 programs. The consistent trend of lower columns across multiple CK metrics, including WMC, DIT, NOC, CBO, RFC, and LCOM, offered a compelling visual narrative. These lower columns collectively indicated that programs incorporating design patterns tended to exhibit characteristics associated with better maintainability.

## Statistical Significance and Future Directions

The statistical analysis, employing hypothesis testing, added quantitative rigor to the study. The results indicated that the integration of design patterns yielded statistically significant improvements in maintainability-related factors, including enhanced separation of concerns, decreased coupling, and reduced complexity. These findings provide data-backed evidence supporting the advantages attributed to design patterns.

This study underscores the positive impact of design patterns on maintainability, supported by both quantitative analysis and visualizations. The findings contribute valuable insights for practitioners seeking to justify investments in pattern-centric design practices and pave the way for further research in software engineering and design pattern effectiveness.

# THREATS TO VALIDITY

## Internal Validity

### Design Pattern Detection Tool Accuracy

The accuracy of the Design Pattern Miner (DPM) tool in identifying design patterns may introduce a potential threat [3]. False positives or negatives in pattern detection could impact the composition of the pattern and non-pattern groups, leading to skewed results. It validates the accuracy of the tool against manual inspections or alternative tools that could mitigate this threat.

### CK Metrics Suitability

The CK metrics suite is used to quantify maintainability-related factors. The choice of metrics and their appropriateness in capturing all maintainability aspects might be a concern. Ensuring that the selected metrics truly reflect maintainability characteristics and exploring additional metrics could enhance the internal validity of the study [2.

### Statistical Analysis Techniques

The employed statistical methods, such as hypothesis testing, assume certain distributions and conditions. Violations of these assumptions might lead to inaccurate results. Sensitivity analyses and considering alternative statistical approaches could address potential issues with the internal validity of the statistical findings.

## External Validity

### Generalization to Other Languages

The project focuses on Java applications, and the generalization of findings to other programming languages may be limited. Different languages may exhibit diverse patterns of usage and adherence to design principles. Conducting similar studies across multiple languages can enhance the external validity of the results.

### Open-Source Project Representativeness

The selected 30 open-source Java applications might not fully represent the diversity of software projects. The characteristics of open-source projects, their contributors, and their development processes can vary widely. Including a more diverse set of projects or extending the study to include proprietary software can improve the external validity of the findings.

## Construct Validity

### Definition and Measurement of Maintainability

The operational definition and measurement of maintainability using CK metrics might not encompass all dimensions of this multifaceted concept. Different stakeholders may have varied perspectives on what constitutes maintainability. Including qualitative measures and gathering feedback from developers could enhance the construct validity of the study.

### Design Pattern Adherence

The study assumes that classes identified by the DPM tool as participating in design patterns indeed adhere to those patterns. There could be instances of partial adherence or misclassification. Conducting code reviews or using multiple pattern detection tools could validate the accuracy of the design pattern classification and improve construct validity [3].

## Conclusion Validity

### Causal Inference

The study establishes a correlation between the use of design patterns and improved maintainability. Establishing a causal relationship is challenging, and external factors could contribute to the observed results. Employing experimental designs or conducting longitudinal studies could strengthen the conclusion's validity by providing more causal insights.

### Limited Quality Attributes

The study focuses on maintainability as the chosen quality attribute. While this provides in-depth insights into one aspect, the exclusion of other quality attributes may limit the generalizability of the conclusions. Including a broader set of quality attributes and their interplay could enhance the conclusion validity of the study.

# CONCLUSION

This empirical study rigorously examined the impact of design patterns on maintainability across 30 open-source Java applications, utilizing CK metrics and Design Pattern Miner (DPM). The results, supported by statistical analysis and visualizations, provide compelling evidence that integrating design patterns positively influences measurable software quality related to maintainability [6]. The analysis revealed consistent patterns of improved separation of concerns, reduced coupling, and decreased complexity in classes embodying design patterns. The findings contribute to the existing body of knowledge by affirming the conventional wisdom on the advantages attributed to design patterns, such as loose coupling, high cohesion, and enhanced code flexibility. Software architects and developers can derive practical insights from this research, justifying investments in pattern-centric design practices based on tangible improvements predicted in maintainability [7]. Despite the robustness of the study, it is essential to acknowledge potential threats to validity, including concerns related to pattern detection accuracy, metric suitability, and generalizability to other languages. Addressing these limitations through validation checks, diverse project selection and broader quality attribute considerations could further enhance the reliability and applicability of the findings. This research underscores the value of design patterns in building maintainable systems, offering a foundation for future studies and guiding informed decision-making in software design and development practices.

# REFERENCES

[1] Jiarpakdee, J., Tantithamthavorn, C. K., Dam, H. K., & Grundy, J. (2020). An empirical study of model-agnostic techniques for defect prediction models. *IEEE Transactions on Software Engineering*, *48*(1), 166-185.

[2] Nanthaamornphong, A., & Wetprasit, R. (2018). Methods to measure the impact of design patterns on software maintainability. *Maejo International Journal of Science and Technology*, *12*(3), 251-271.

[3] Jun, H. K., & Rana, M. E. (2021, November). Evaluating the Impact of Design Patterns on Software Maintainability: An Empirical Evaluation. In *2021 Third International Sustainability and Resilience Conference: Climate Change* (pp. 539-548). IEEE.

[4] Qamar, N., & Malik, A. A. (2020). Impact of design patterns on software complexity and size. *Mehran University Research Journal Of Engineering & Technology*, *39*(2), 342-352.

[5] Rochimah, S., Gautama, I., & Akbar, R. J. (2019). Refactoring the Anemic Domain Model using Pattern of Enterprise Application Architecture and its Impact on Maintainability: A Case Study. *IAENG International Journal of Computer Science*, *46*(2).

[6] Kurmangali, A., Rana, M. E., & Ab Rahman, W. N. W. (2022, March). Impact of Abstract Factory and Decorator Design Patterns on Software Maintainability: Empirical Evaluation using CK Metrics. In *2022 International Conference on Decision Aid Sciences and Applications (DASA)* (pp. 517-522). IEEE.

[7] Al-Obeidallah, M. G. (2021). The Impact of Design Patterns on Software Maintainability and Understandability: A Metrics-Based Approach. *ICIC express letters. Part B, Applications: an international journal of research and surveys*, *12*(12), 1111-1119.