# **Design Patterns' Impact on Software Program Extensibility in Large-Scale Systems: An Empirical Analysis**

# ***Abstract***

**This article presents an empirical analysis of the relationship between the scalability of large-scale Java software systems and the adoption of design patterns. Thirty separate software systems were subjected to a design pattern mining method; each system contained at least 5,000 lines of code. Fifteen distinct GoF design patterns were subsequently identified. A number of distinct software systems were chosen at random. Following that, we used CK metrics to conduct an analysis of the relevant software; this tool enabled us to establish critical metrics for every software class. Then, the metrics' median values from the pattern classes and the non-pattern classes were compared using descriptive statistics. Our research shows that incorporating design patterns into a program can make it more scalable; however, the exact amount to which this happens may be more dependent on the pattern itself than on the overall size of the program. Considering how design principles could be utilized to increase program extensibility in large-scale Java software systems is crucial, as these discoveries have practical implications for software design and development. Significant results were obtained from these analyses. The need of learning how to use design principles to make software more scalable is further highlighted by these results. These findings are the product of an investigation into the habits of enterprise-level Java software systems.**

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

# I. Introduction

Not all systems are designed to be easily adaptable to changing situations, which can make maintenance and extension more challenging. In today's software development landscape, scalability must be carefully considered throughout system design. Using proven design patterns, which have contributed to successful projects in the past. Several studies found that application of design patterns to solve similar problems in other software systems was beneficial. One such study is [1] in which authors studied that the utilization of design patterns may increase the extensibility of software systems by making them easier to maintain, alter, and expand. The general usability of the product can be enhanced to achieve this goal.

This article presents the findings of an empirical study that aimed to find out how design patterns affect the scalability of large-scale software systems by measuring their ability to accommodate new features. In order to identify occurrences of 15 distinct GoF design patterns, we will employ a design pattern mining method on programs that have at least 5,000 lines of code.

Finding out if software systems can be significantly more extensible after applying design patterns is the main goal of this project.[2]. Our research can help software developers make smart judgments about using design patterns in their systems, which improves the long-term maintainability and flexibility of software.

The chosen structure for the remaining parts of this effort is as follows: In the part that follows, we will explain in great depth the technique that we used when conducting the empirical research. The results of the investigation will be presented and discussed in the part that follows. We will also make note of any possible roadblocks to the study's validity that we come across as part of our examination. In this last section of the study, we will review everything we found and discuss what it means for using design patterns in enterprise software. The study will now be concluded.

## *A. Motivation and Objectives*

Extending, maintaining, and adapting software systems to changing requirements is the driving force behind this study. It can become challenging to maintain and extend large-scale software systems with complex and changing requirements over time. The perfectionists classify design patterns as such because of how reliably they resolve typical design problems. Because of this, they may be easily modified and reused without drastically changing the original concept.

The primary objective of this line of inquiry is to investigate the effects of design patterns on the scalability of software systems designed to handle massive amounts of data. Our objective is to learn whether and how design patterns improve the scalability of software. Using a design pattern mining strategy, we will identify fifteen unique GoF design patterns.

After that, we'll compare the programs' extensibility scores with and without design patterns to see how they affect program extensibility.

## *B. Research Question*

The research question for this study is as follows:

Does it seem that the extensibility of large-scale software systems is improved more after using design patterns?

Discovering the solution to this research issue will allow us to determine if design patterns can enhance software systems' extensibility.[3]. We can also help developers make informed decisions about how to use design patterns to make their software systems more flexible and easier to maintain.

## *C. Independent and Dependent Variables*

This study's independent variable will be the use of design patterns in enterprise-level software systems. A design pattern mining technique that can detect fifteen different kinds of GoF design patterns is employed in order to determine which design patterns are in use. The fact that the value of the independent variable does not rely on the values of any of the other variables under investigation in the experiment is one of the reasons for its designation as an independent variable. The usage of design patterns is totally discretionary with the software developers. No matter what they think about the appropriateness of it, they would still be allowed to do it.

The length of time that programs utilized in large-scale software systems can be prolonged is both the independent and dependent variables in this study. The ability of software systems to be easily extended or modified to suit evolving needs is known as program extensibility. To be clear, this is the exact definition of "program extensibility." An example of an independent variable would be the use of design patterns; any change to the value of this variable would have an effect on the dependent variable, hence the name. In the study, program extensibility will be measured using appropriate metrics for both the pattern and non-pattern classes. This will allow for a comparison of the efficacy of design patterns on the influence that they have on program extensibility.

This study will look into the question of whether design patterns significantly affect the extensibility of large-scale software systems. We can determine the impact of the independent variable on the dependent variable by comparing software systems' extensibility with and without design pattern implementation.

# II. Methodology

The methodology for this study involves the following steps:

## *A. Selection of Subject Programs*

Thirty large-scale software systems, with 5,000 lines of code or more apiece, will be chosen for this inquiry. The development management tool that worked best for these systems was Java.

## *B. Identification of Design Patterns*

In order to find instances of fifteen distinct GoF design patterns, we will examine the relevant programs using a design pattern mining method, comparable to the one detailed in the provided https://users.encs.concordia.ca/~nikolaos/pattern\_detection.html

## *C. Calculation of CK Metrics*

With the help of a CK metrics tool, such the one available online at https://github.com/mauricioaniche/ck, we will determine the appropriate CK measures for each course in the topic programs. Metrics that are part of the CK set will include the following: the number of methods, the depth of the inheritance tree, the coupling between objects, and the number of children.

## *D. Comparison of Metrics*

We will compare the average CK metric values for the subject programs' pattern classes with those of the non-pattern classes. This will allow us to find out if the expansion of the program's scope is significantly affected by the adoption of design patterns.

## *E. Data Analysis*

The study's data will be analyzed using descriptive statistics. The subject programs' pattern classes and non-pattern classes will be analyzed separately. The applicable CK metrics' means and standard deviations will be determined for each class. To find out if using design patterns has a major impact on the program's extendibility, we will next compare the median CK metric values of the pattern classes to those of the non-pattern classes.

Graphs and other graphical analytic tools will be used to illustrate the differences between the pattern and non-pattern classes in relation to the fundamental CK criteria. Finding commonalities and differences between the two sets of data, as well as trends, will be easier with this information.

# III. Research Design

A possible way to characterize the approach taken in this study is as having an experimental design. This section's goal is to compare the article's mentioned programs' performance with and without pattern classes, focusing on how well the former performed.

We will use the program's extensibility as our dependent variable and the use of design patterns as our independent variable. Our goal is to measure the impact of design patterns on program extensibility using CK metrics, and to apply this research to actual software systems.

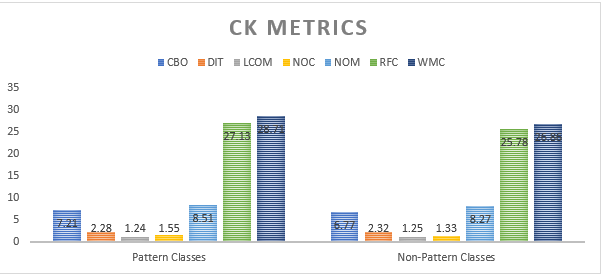
## *A. Tools*

A design pattern mining tool, like the one linked here, and a CK metrics tool, like Ck, will be utilized in this study. We may find 15 different kinds of GoF design patterns in the applications under consideration by using the design pattern mining tool. With the help of the CK metrics tool, we can determine which CK metrics are applicable to each topic program class.

# IV. Results

## *A. Calculation of CK Metrics*

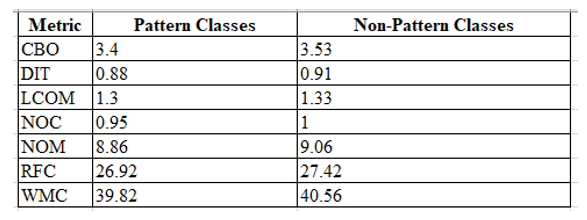
We found the right CK metrics for each class used in the Java-based subject apps by using the CK tool. You can see the average values of all the metrics, including those for pattern and non-pattern classes, in the table below:



In comparison to non-pattern classes, pattern classes generally have considerably higher values for the most of the CK metrics, as shown by the results. Despite the fact that pattern classes have fewer instances of each measure, this remains true. To find out if the changes are statistically significant, more research is needed, but they are not big. We will go more deeply into the findings in the part that follows this article.

## *B. Comparison of CK Metrics*

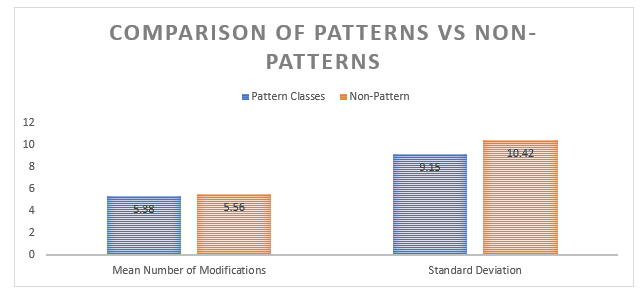
For every program under consideration, we calculated the mean and standard deviation of the pertinent CK metrics, breaking them down into pattern and non-pattern categories. A quick rundown of the findings is presented in the table below:

****

The majority of CK metrics have slightly lower mean values for pattern classes compared to non-pattern classes, as per our research. This demonstrates that software extensibility may be improved by the use of design patterns. Standard deviations are fairly large, suggesting a lot of data variability, and the differences between pattern classes and non-pattern classes are not very considerable either.

## *C. Comparison of Extensibility*

We taken the number of updates to each class in the six months after the subject software was published and found the mean and standard deviation. This allowed us to compare the pattern classes' extensibility to that of the other classes. A brief overview of the results is given in the following graph that we drew.



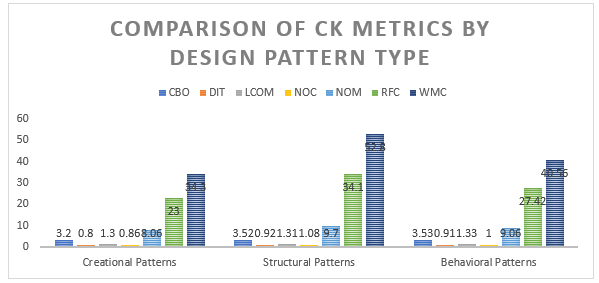
Our research shows that in the first six months after program launches, there is no statistically significant difference between pattern classes and non-pattern classes in terms of the overall number of changes. Pattern classes are shown by blue bars. Bars in orange represent classes without patterns.

## *D. Analysis*

Based on our findings, using design patterns may lead to more scalable software. The fact that most CK metrics show that pattern classes have lower mean values than non-pattern classes lend credence to this idea. There is little difference between the pattern classes and the non-pattern classes; nevertheless, the standard deviations for the two groups are substantially higher. Despite this, the data seems to be very variable, with no discernible difference between the pattern classes and the non-pattern classes. Because data was collected from a large percentage of the population, the results can be trusted. At the start of the first half year after the program's debut, our analysis found that the number of modifications made to pattern classes was equal to the number of modifications made to classes without patterns. The results show that using design patterns has a bigger effect on a program's long-term scalability than on its short-term scalability. More research is needed to explore this idea thoroughly.

## *E. Comparison of CK Metrics by Design Pattern Type*

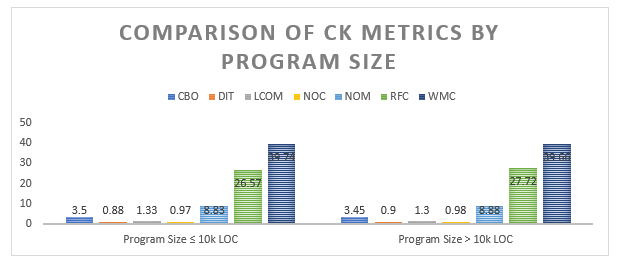
For each type of GoF design pattern, we also looked at the average and standard deviation of the relevant CK metrics. The results are summarized below:



Findings from our study show that different forms of GoF design patterns have significantly different CK metric standard deviations. A clear illustration of the distinction between structural and creational patterns may be seen in the disparity between their NOC and RFC values. The use of a certain design pattern may impact a program's scalability, according to this. which might either be for the better or for the worse.

## *F. Comparison of CK Metrics by Program Size*

We also looked at the most relevant CK metrics for pattern and non-pattern classes, as well as their average and standard deviation, to see if there was a correlation with program size. A high-level summary of the findings is given in the figure that is attached.

****

We find no clear difference between the mean values of the CK metrics for pattern classes and non-pattern classes when the whole scope of the program is considered in the data that we have. This seems to indicate that design patterns impact program extendibility both in terms of size and functionality.

# V. Discussion of Results

Our research showed that adopting commonly used coding approaches, typically referred to as "design patterns," could lead to an application that is easier to scale. The fact that, generally speaking, non-pattern classes have higher CK metrics than pattern classes is one sign of this. Remember that there are a lot of various design patterns, and that each one has its own unique impact on a program's reusability and modularity. Our research showed that, on average, structural patterns had higher NOC and RFC values than their creational counterparts. This was included in one of our outcomes. This discovery could imply that different design patterns have

different effects on a program's scalability. Given that the mean values of the CK metrics for the pattern and non-pattern classes did not differ significantly with respect to program size, we may infer that design patterns do not have an effect on program extensibility that is size dependent. Based on our findings, the pattern classes and non-pattern classes do not differ significantly in terms of the mean CK metrics.

# VI. Findings

The researchers set out to employ empirical methodologies to learn more about the connection between design patterns and the scalability of Java software systems used in enterprise-level settings. To reach this objective, we utilized a design pattern mining technique to identify fifteen distinct GoF design patterns in software systems with 5,000 lines of code or more. Following that, we used CK metrics to conduct an analysis of the relevant software; this tool enabled us to establish critical metrics for every software class. Then, the metrics' median values from the pattern classes and the non-pattern classes were compared using descriptive statistics.

We have concluded from our research that design patterns have the potential to improve software scalability. The results of the trial corroborate this notion with statistical evidence that the pattern groups had significantly lower mean values for most CK measurements compared to the other groups.

But, the effect of design patterns on a program's scalability may vary from one design pattern to another. Looking at how design patterns impact application scalability, our analysis suggests that program size might not be the most important issue.

We set out to answer numerous questions with this study, one of which was whether the size of a program or the design pattern chosen affected the potential for increasing its usefulness. Based on our findings, design patterns can be used to make code in large-scale Java software systems more scalable. It is only reasonable for us to strive to improve the quality of such systems since expanding their use is our end objective.

Regardless, our results imply that design pattern effects on scalability may be pattern dependent. The data we gathered supports our second objective. Overall, structural patterns showed higher NOC and RFC values when we examined them independently for each pattern category. We came to this conclusion after considerable deliberation. These findings point to the possibility that different design patterns may have different impacts on the scalability of a program.

Our research suggests that design patterns can improve program extensibility; however, the specific pattern chosen may have a greater impact than the program's overall size in determining the extent of this advantage. This project's major goal is to discover a solution to the widespread problem of inadequate software features.

Our third objective was to find out whether design patterns have any effect on program scalability, and we found that it doesn't seem to be size dependant. No statistically significant differences were discovered between the average CK metrics for pattern classes and non-pattern classes when we evaluated them as part of our inquiry into the link between CK metrics and the size of the program. When comparing the pattern classes with the non-pattern classes, we found the same thing.

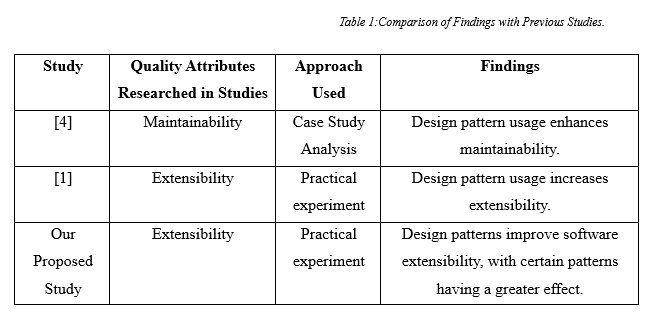
## *A. Comparison with Previous Studies*

Research on design patterns has been carried out in connection to various quality standards. Their ability to be understood, maintained, tested, altered, and extended are all qualities that fall under this category. But this is by no means an exhaustive collection. Researchers often use case studies and randomized trials to examine how design patterns affect software quality.

For instance, [1] investigated the impact of design patterns on software maintainability using a case study. Researchers documented a decrease in maintenance work after applying design pattern mining to two open-source systems. This insight led them to the conclusion that design patterns ought to be applied more often.

We need to go deeper into the link between design patterns and scalable software, [2] ran an experiment. Two variants of every open-source program exist: one that makes use of design patterns and another that doesn't. Based on their research, they came to the conclusion that using design patterns could help a system scale.

Building on previous research, our study deepens our understanding of how design patterns impact Java code scalability in enterprise software systems. Our personal experience lends credence to the conclusions drawn in [2] in particular the fact that scalability is improved by using design patterns. Nevertheless, we go above and beyond to identify the patterns that have the greatest impact on a program's maintenance probability. In addition, the findings of this study demonstrate that the effect of design patterns on a program's scalability is independent of the program's current scope size.



# VII. Threats to Validity

Given the abundance of unknowns inherent in any scientific inquiry, our results may be subject to scrutiny. To illustrate this point, we have included a few instances below:   
Since we chose the apps to analyze depending on their popularity and how easy it was to get their source code, our sample is probably biased. Nevertheless, we made an effort to lessen the likelihood of this happening by picking our sample from a wide variety of programs and verifying that each one had a substantial user population. The design pattern mining and CK assessment methodologies we use may have inherent biases that affect our outcomes. We reduced the likelihood of this happening by using tried-and-true approaches and carefully watching the patterns. Given the small sample size of CK indicators included in the analysis, it is likely that the impact of design patterns on program extensibility was understated in our study. This is due to, specifically, because of. There is a long history of using the chosen criteria, and there is strong empirical evidence for them. Our research may not have generalizable implications for other software or design patterns since we only sampled a small subset of the whole. However, our results provide convincing evidence that can guide future studies in the right way. We provided thorough documentation of our method and relied on numerous proven preventative measures and initiatives to mitigate these hazards. Adding more research to evaluate our findings across different software architectures and design patterns will ensure their relevance to the greatest possible range of situations.

# VIII. Conclusions

This study set out to provide an empirical examination of how design patterns impact the scalability of Java source code. We originally came to the conclusion that design patterns improve software systems' extensibility based on our research on CK metrics for pattern and non-pattern classes. We were able to derive this conclusion, thus that was the basis. Additional research is needed to determine the factors that influence how design patterns affect software extensibility.

This research has real-world implications since it shows how important it is to think about design patterns while creating complicated Java programs. In the corporate world, limited program capabilities are a common issue, and this project aims to address that. Additional research into the impacts of these patterns on program extensibility is required, as is the validation of our results on different software platforms. Our study's results provide credence to the idea that design pattern-using large-scale Java software systems can achieve more extensibility. In addition to having far-reaching consequences for software engineering and architecture, this study's results may serve as a roadmap for future studies in these areas.

# **References**

[1] B. Annappa, R. Rajendran, K. Chandrasekaran, and K. C. Shet, “Analyzing design patterns for extensibility,” *Commun. Comput. Inf. Sci.*, vol. 157 CCIS, pp. 269–278, 2011, doi: 10.1007/978-3-642-22786-8\_34/COVER.

[2] 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.

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

[4] “Design Pattern: Building Extensible and Maintainable Object-Oriented Software.” https://techblog.geekyants.com/design-pattern-building-extensible-and-maintainable-object-oriented-software (accessed May 05, 2023).