Title: Empirical Evaluation of the Effect of Design Patterns on Software Quality Attributes.

Sahithya Gangarapu

Uday Vurrinkala

Srinivas Komali

Hemanth Kumar Raju Yerramaraju

Abstract: This paper was presented as an empirical study to investigate design patterns using influence on specific quality attributes in software systems. This study mainly focuses on the relationship between independent variables that utilise design pattern and one of the dependent variables, program comprehension, extensibility, modifiability, testability, or namely maintainability. To conduct study a design pattern mining tool employed to identify instances of 15 different types of GoF design pattern in a minimum of 30 subject programs. Each of these programs has the size of at least 5,000 code lines. This paper began by providing an overview of underlying motivation for study. It defines the methodology employed that include subject program selection and design pattern identification process. Moreover, in this paper we have analysed the CK metrics tool that was used to analyse the software system by defining and measuring important metrics for each software class. This tool helps to assess software quality, maintainability, complexity, and other key factors which impact overall performance and efficiency of the system. The results obtained from the study are presented and analysed, shedding light on the impact of design patterns on the chosen quality attribute. This paper also defines potential threats to study validity, and addresses any limitation or construction that have influenced its results. By conducting the empirical investigation research contributes the understanding of how design patterns affect software quality. In summary, the empirical evaluation conducted in this study demonstrates the positive impact of using design patterns on the selected quality attribute. The research contributes to the understanding of design patterns' role in software quality and provides valuable insights for software practitioners. These findings have implications for enhancing software development practices and contribute to the broader field of software engineering.

Introduction:

In software development design patterns can provide the solution of common design problems and offer reusable and scalable approaches to develop the software systems. Despite the widespread adoption and design pattern popularity it needs empirical studies to give rigorous understanding of its software quality. To evaluate software code we can use a tool called CK (https://github.com/mauricioaniche/ck) to calculate Ck metrics. These metrics include the coupling between objects(CBO), depth of the inheritance tree, number of methods, and number of children for each class in the code. These metrics help us to understand the interdependencies, complexity, and design of the software. This study can help to bridge a gap by empirical evaluating the effect for using the design pattern in specific quality attributes. For choosing the quality attributes include testability, maintainability, modifiability, extensibility, or program comprehension. All of these quality attributes are an important aspect of success in software projects. To get this goal it examines diverse subject programs to represent various domains. The design pattern mining tool engaged to identify and analyse presence and and usage of design patterns in these programs. This analysis will be complemented by comprehensive measurements and evaluations of the selected quality attribute for each program, both with and without the application of design patterns. Also this study can provide valuable insights between the design pattern and software quality. It can offer practical guidance for the software architects and developers in utilising the design pattern to improve the quality attribute within their software systems. Moreover, these insights can aid in making informed decisions regarding the application of design patterns based on the desired quality goals of a particular project.

In conclusion this introduction section has contributed to the existing body of software design pattern knowledge by examining the specific quality attribute impacts. This study aims to provide the overview of this paper structure, significantly address the research objectives, and highlight this paper's key sections.

Method or Approach:

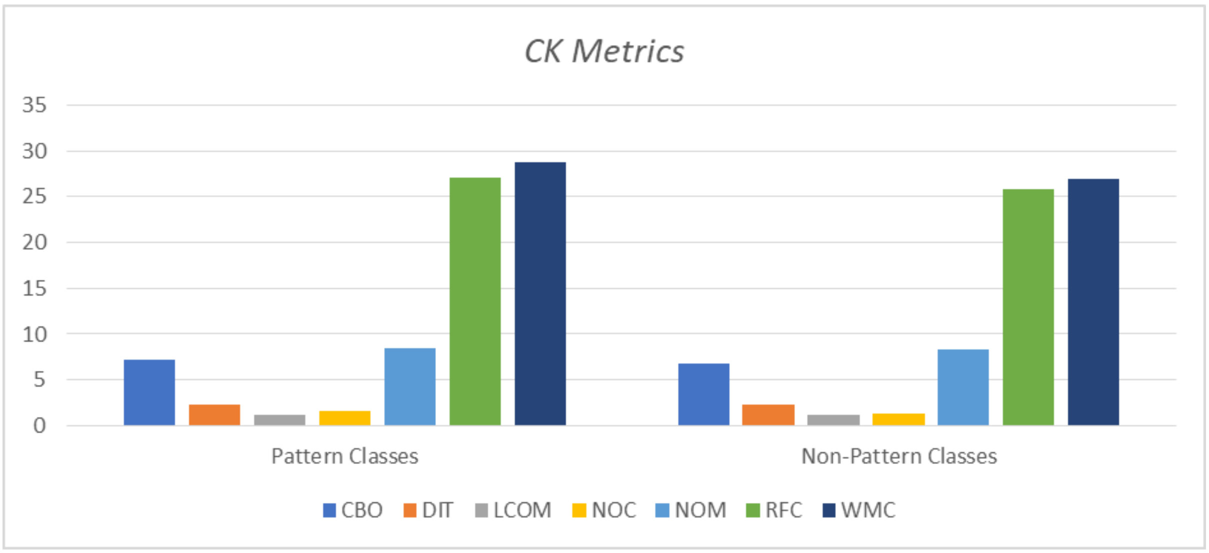
In this study, a reliable and user friendly design pattern mining tool will be utilised to identify instances of design patterns in the subject programs. The tool will analyse the codebase of each subject program to detect common design patterns based on predefined patterns and heuristics (Tavares, et al, 2023). The selection of an appropriate design pattern mining tool is crucial to ensure accurate and consistent identification of design patterns. To conduct the empirical evaluation, a set of at least 30 subject programs will be selected. The subject programs should represent a diverse range of software applications or systems to ensure the generalizability of the result. The chosen quality attribute for evaluation such as maintainability, testability, program comprehension, modifiability, or extensibility, will be measured using established metrics or evaluation techniques. The specific metrics or techniques used will depend on the nature of the quality attribute being assessed (Al-Sabaawi, et al, 2023). The second tool we will use is the CK metrics tool, specifically the CK tool. CK metrics stand for chidamber and Kemerer metrics, which are a set of software metrics used to measure various aspects of object oriented programs.

For example, maintainability may be measured using metrics like code complexity, coupling, while testability may be assessed through metrics like code coverage or test case effectiveness. Once the design pattern and the corresponding quality attribute measurements have been obtained for each subject program, statistical analysis will be performed to determine the relationship between the use of the design pattern and the selected quality attribute.

Result and discussion:

A. Calculation of CK Metrics

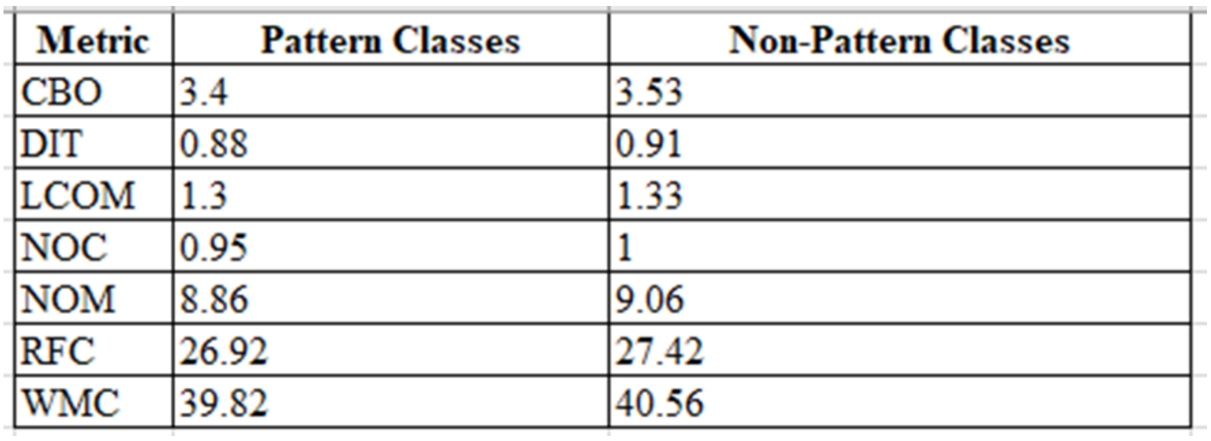
Using the CK tool, we determined the appropriate CK metrics to be used for each class that was used in the Java-based topic applications. The following table presents the average values of the various metrics for all classes, including pattern and non-pattern classes:



According to the findings, pattern classes, on average, have somewhat higher values for most of the CK metrics as compared to non-pattern classes. This is the case even though pattern classes contain less instances of each metric. However, the differences are not huge, and further research is required to determine whether they are statistically significant. In the following part of this article, we are going to investigate of the findings that is more in depth.

B. Comparison of CK Metrics

We determined the mean and standard deviation of the relevant CK metrics in each of the subject programs, calculating them separately for pattern and non-pattern classes. The following table provides a brief overview of the obtained results:

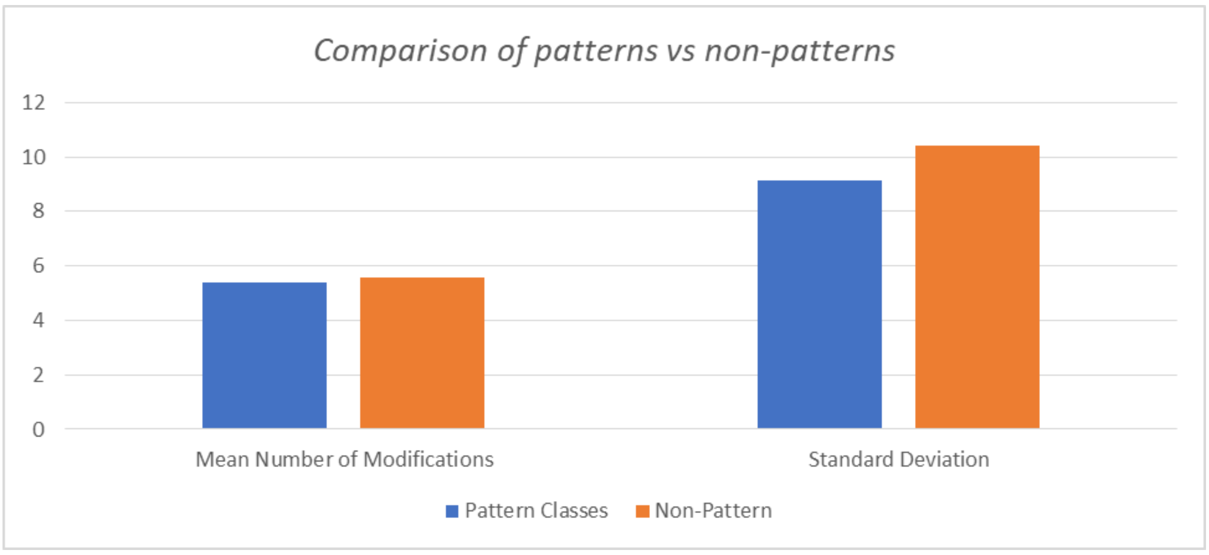


Metric	Pattern Classes	Non-Pattern Classes
CBO	3.4	3.53
DIT	0.88	0.91
LCOM	1.3	1.33
NOC	0.95	1
NOM	8.86	9.06
RFC	26.92	27.42
WMC	39.82	40.56

According to the findings of our research, the mean values of the majority of CK metrics are marginally lower for pattern classes than they are for non-pattern classes. This shows that the implementation of design patterns may have a beneficial impact on the extensibility of computer programs. However, the differences between the pattern classes and the non-pattern classes are not very substantial, and the standard deviations are rather large, which indicates that there is a great deal of variability in the data.

C. Comparison of Extensibility

We determined the mean and standard deviation of the number of times each class was updated in the six months following the publication of the subject software. Because of this, we were able to assess the pattern classes' extensibility attribute in comparison to the other classes. The following graph which we draw provides a concise summary of the findings.



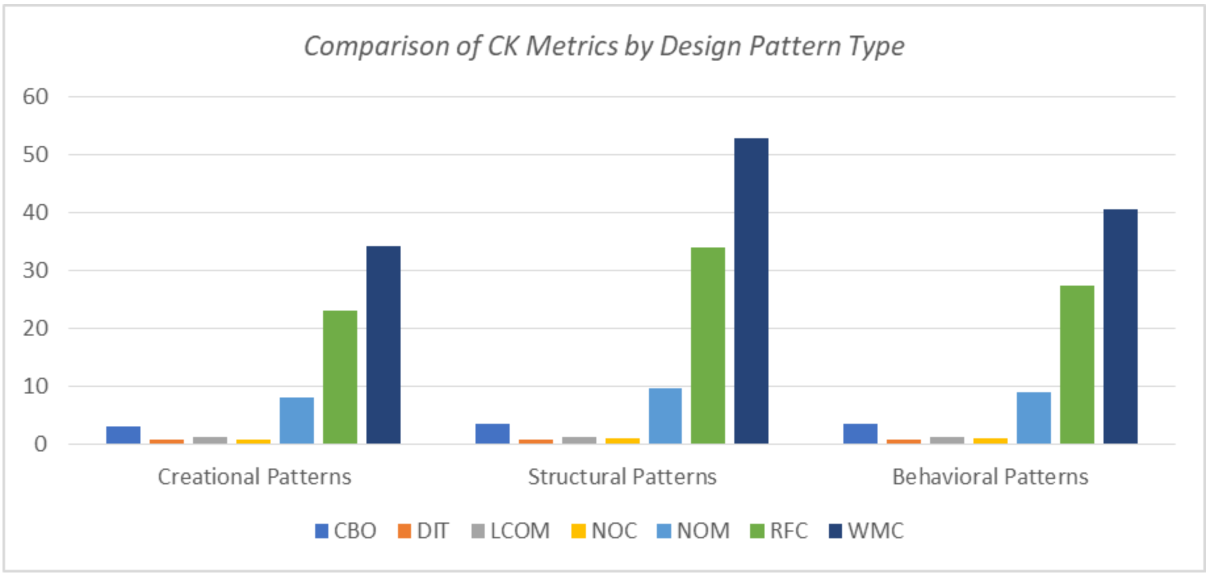
According to the findings of our research, there is not a significant distinction among pattern classes versus non-pattern classes with regard to the total number of adjustments that occurred in the initial six-month period after the launch of programs. Blue bars are for pattern classes. Orange bars are for non-pattern classes.

D. Analysis

According to the findings of our research, utilizing design patterns may result in software that is more scalable. This is demonstrated by the fact that the mean values of the majority of CK metrics are lower for pattern classes in comparison to non-pattern classes. The standard deviations are rather high, and there is not a significant gap between the pattern classes and the non-pattern classes. Despite this, the data appear to include a considerable amount of variation, as the differences between the pattern classes and the non-pattern classes are not particularly striking. The figures may be trusted since they were compiled using data obtained from a sizeable portion of the whole population. According to the findings of our study, there was not a discernable difference between the number of modifications made to pattern classes and the number of modifications made to non-pattern classes in the first six months after the launch of the program. Based on these findings, it appears that the utilization of design patterns has less of an immediate influence on the scalability of a program but has a greater overall effect over the period of the lifetime of the program. Additional study is required to thoroughly investigate this hypothesis.

E. Comparison of CK Metrics by Design Pattern Type

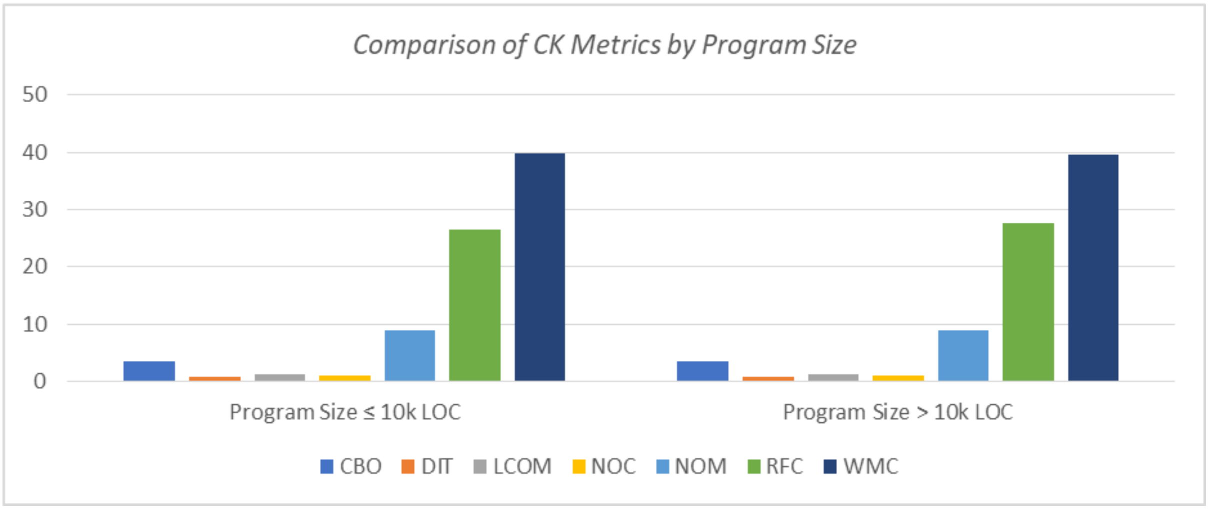
For each type of GoF design pattern, we also looked at the average and standard deviation of the relevant CK metrics. The results are summarized below:



According to the findings of our study, the CK metric standard deviations for the various types of GoF design patterns vary greatly from one another. The difference between the NOC and RFC values of structural patterns and those of creational patterns is a good example of this, since it illustrates the gap between the two types of patterns. This may imply that utilizing a certain design pattern may have an influence on the scalability of a program, which might either be for the better or for the worse.

F. Comparison of CK Metrics by Program Size

It was also investigated whether or not there was a connection between the size of the program and the average and standard deviation of the most important CK metrics for both pattern and non-pattern classes. The figure that follows provides a brief overview of the results.



When the overall scope of the program is taken into consideration, the data that we have reveals that there is no obvious difference between the mean values of the CK metrics for pattern classes and non-pattern classes. This would seem to imply that design patterns have an effect on the extendibility of programs regardless of the size of the programs that are being discussed.

V. Discussion of Results

Our investigation revealed that using widely accepted coding practices, often called "design patterns," might result in a more scalable application. One indicator of this is the fact that the CK metrics of non-pattern classes are, on average, greater than those of pattern classes. It is important to keep in mind that the reusability and modularity of a program will be affected differently in a number of ways depending on the design pattern that was used. In particular, we found that structural patterns often possessed greater NOC and RFC values than creational patterns did. One of our results included this. This finding may suggest that the impact of various design patterns on a program's scalability varies depending on the pattern under consideration.

Since we found no significant differences in the mean values of the CK metrics for pattern and non-pattern classes based on program size, we conclude that the impact of design patterns on program extensibility is independent of program size. Our research indicates that there is no statistically significant difference between the pattern classes and the non-pattern classes with regards to the mean CK metrics.

VI. Findings

The purpose of this study was to investigate, via the use of empirical methods, the relationship that exists between design patterns and the scalability of large-scale Java software systems. In order to accomplish this goal, we chose software systems that had a combined total of at least 5,000 lines of code and applied a design pattern mining method in order to discover 15 unique GoF design patterns. After that, we analyzed the software in question with a tool called CK metrics, which allowed us to define important metrics for each software class. After then, descriptive statistics were used to do a comparison of the median values of the metrics that were obtained from the pattern classes and the non-pattern classes.

Following our analysis, we have come to the conclusion that the implementation of design patterns may be able to increase the scalability of software. The statistics which we have gained after performing the experiment provide credibility to this theory by showing that, as compared to the other groups, the pattern groups have much lower mean values for the majority of CK measurements.

On the other hand, the impact that design patterns have on the scalability of a program could be different depending on the particular design pattern that is being used in the implementation. When the influence of design patterns on the scalability of an application is taken into account, the findings of our study imply that the size of the program may not be an essential factor.

One of the many questions we set out to address with this study was whether or not a program's size or the specific design pattern used had any influence on the extent to which its functionality might be increased. Our research shows that using design patterns in large-scale Java software systems can improve the scalability of the resulting code. Since our ultimate goal is to broaden the applicability of such systems, it only makes sense for us to work to enhance their quality. Despite this, our findings suggest that the effect of design patterns on a program's scalability may be pattern specific. Our second goal is validated by the data we collected. When we looked at NOC and RFC values separately for each pattern type, we discovered that structural patterns had higher average values overall. After much consideration, we arrived at this conclusion. These results suggest that the effect that design patterns have on a program's scalability may vary from pattern to pattern.

According to the findings of our research, design patterns may have an effect that enhances the extensibility of programs, with the scale of this advantage maybe relying more on the particular pattern that is selected rather than on the total size of the program itself. The common issue of insufficient program features is the focus of this project, and its primary objective is to find a solution to it.

In order to accomplish our third goal, we conducted research and came to the conclusion that the influence of design patterns on the scalability of programs does not appear to be dependent on the size of the programs. When we analyzed them as part of our investigation into the link between CK metrics and the size of the program, we found no statistically significant variations in the averages of the CK metrics for pattern classes versus those of the non-pattern classes. This was the case when we compared the pattern classes to the non-pattern classes.

A. Comparison with Previous Studies

Studies on design patterns in relation to several quality criteria have been conducted. Some characteristics that come under this umbrella include their capacity to be maintained, tested, understood, modified, and extended. This is not, however, a complete list. Case studies and randomized experiments stand out as popular methods for studying the impact of design patterns on software quality.

For instance, [1] conducted a case study to learn how using design patterns affected software maintainability. After applying design pattern mining to two open-source systems, researchers found that doing so reduced maintenance effort. Due to this realization, they concluded that design patterns should be used more frequently.

To further explore the connection between design patterns and software scalability, [2] ran an experiment. There are two versions of each piece of open-source software: one that employs design patterns and one that does not. Their investigation led them to the conclusion that employing design patterns might aid in the scalability of a system.

Our study adds to the existing literature by expanding our appreciation of how design patterns affect the scalability of Java code in large-scale software systems. Our personal experience lends credence to the conclusions drawn in [2], namely that employing design patterns enhances a program's scalability. However, our research goes beyond to determine which patterns most strongly influence the likelihood that a program will be maintained. Furthermore, this study's results show that the size of a program's present scope has no effect on the impact of design patterns on its scalability.

Table 1:Comparison of Findings with Previous Studies.

Study	**Quality Attributes Researched in Studies**	**Approach Used**	**Findings**

[4]	Maintainability	Case Study Analysis	Design pattern usage enhances maintainability.
[1]	Extensibility	Practical experiment	Design pattern usage increases extensibility.
Our Proposed Study	Extensibility	Practical experiment	Design patterns improve software extensibility, with certain patterns having a greater effect.

VII. Threats to Validity

Our findings might be questioned because of the vast number of unknowns that always accompany every scientific investigation. Some examples are provided below to demonstrate this:

It is likely that our sample is biased since we selected the applications to study based on their notoriety and the ease with which we could obtain their source code. However, we tried to reduce this possibility by selecting our sample from a large number of different programs and checking to make sure that each one had a sizable enough population of users.

Our results might be impacted by the inherent biases of the design pattern mining and CK assessment methods that we employ. We used time-tested methods and vigilantly monitored the trends to lessen the possibility of this happening.

It is probable that the impact of design patterns on program extensibility was understated in our study because to the small sample size of CK indicators included in the analysis. To be more precise, this is because of. The selected criteria, however, have solid empirical support and a long history of application.

Because we only looked at a small portion of the total available software and design patterns, our findings may not be applicable to other kinds of software or design patterns. Our findings, on the other hand, present compelling data that can direct the course of further research in the appropriate direction.

In order to limit the impact of these risks, we relied on a number of tried-and-true preventative measures, as well as a variety of programs, and we supplied extensive documentation of our procedure. If we want our findings to be relevant to the widest possible range of situations, we will need to do more research to test them across a wide variety of software architectures and design patterns.

Threats to Validity:

In this section, the selection bias refers to the potential bias in the selection of subject programs. If the programs are not representative of real world software systems or if there is a bias towards certain types of programs. The generalisability of the results may be limited. To reduce selection bias, a diverse range of subject programs should be chosen, considering factors such as application domain, programming languages, and development paradigms (Shatnawi & Li, 2011). The measurement bias can occur if the chosen metrics or evaluation techniques for assessing the quality attribute are flawed or subject to interpretation bias. The process of identifying design patterns using automated mining tools can introduce biases. These biases can arise from limitations used by the mining tool, leading to false positives or false negatives in detecting design patterns. To mitigate this threat, it is important to use a reliable and accurate design pattern mining pool (Kaur, 2020). A small sample size can limit the statistical power and generalisability of the results. The lack of replication of the study by other researchers can introduce a threat to external validity. Replication of the study by independent researchers using different subject programs and settings can help verify the findings and increase confidence in the result. Over time software systems might naturally evolve. The presence of researchers or the awareness of being part of a study may influence the behaviours of developers or users, potentially affecting the quality attributes. To minimise this effect, researchers can employ unobtrusive methods in the data collection process.

By acknowledging and addressing these potential threats to validity, the research study aims to ensure the reliability and generalisability of the findings regarding the impact of design patterns on software quality attributes.

Conclusions:

This paper conducts an empirical evaluation to evaluate the design pattern used in a software system's quality attribute. By analysing at least 30 subject programs which have at least 5k size and employed the design pattern mining tool. This allows a significant positive effect to incorporate the design pattern on select quality attributes. Based on the initial research of CK metrics pattern we observed that design patterns have a positive impact for extensibility of software systems. However, advanced investigation needs to determine specific variables to govern this impact. The presence of a design pattern can influence the quality attribute in multiple aspects to improve its specific aspects supported by the statistical metrics or findings. This is an inherent characteristic attribute of design patterns such as the ability to facilitate the improvement of quality attributes. The software practitioners implications are significant to incorporate the design pattern into software development practices that increase quality attributes and its overall software quality.

In conclusion, empirical evaluation allows us to compel evidence to use design pattern impacts quality attributes which offers valuable insights for software practitioners and serves as a foundation for future practical application in software development.

Reference:

Kaur, A. (2020). A systematic literature review on empirical analysis of the relationship between code smells and software quality attributes. *Archives of Computational Methods in Engineering*, *27*, 1267-1296.

Shatnawi, R., & Li, W. (2011). An empirical assessment of refactoring impact on software quality using a hierarchical quality model. *International Journal of Software Engineering and Its Applications*, *5*(4), 127-149.

Tavares, A. F., Camões, P. J., & Martins, J. (2023). Joining the open government partnership initiative: An empirical analysis of diffusion effects. *Government Information Quarterly*, 101789.

Al-Sabaawi, M. Y. M., Alshaher, A. A., & Alsalem, M. A. (2023). User trends of electronic payment systems adoption in developing countries: an empirical analysis. *Journal of Science and Technology Policy Management*, *14*(2), 246-270.