Abstract

Design patterns have been widely recognized as a useful technique for enhancing software quality attributes such as maintainability and extensibility. In this study, we aim to empirically evaluate the effect of using design patterns on these two quality attributes, measured through the Lack of Cohesion in Methods (LCOM) and the Coupling Between Objects (CBO) modified metrics, respectively. We examined 100+ software programs with a minimum size of 10k and tested for instances of 15 types of GoF design patterns using a reliable design pattern mining tool. Our results, based on the data of 13 programs, suggest a positive trend between the use of design patterns and the improvement of LCOM and CBO modified metrics. We discuss the implications of our findings and highlight potential research directions in this area.

Introduction

Software quality is crucial in today's world, where software is pervasive in almost every aspect of our lives. It is the degree to which software satisfies stakeholders' expectations and meets its specified requirements. Software quality attributes such as maintainability and extensibility are increasingly important in recent years. Maintainability refers to how easily a software system can be modified to correct faults, improve performance, or adapt to changing requirements. Difficult-to-maintain software systems can be a significant burden on developers, resulting in increased costs and reduced productivity. Extensibility refers to the ease with which a software system can be extended to add new functionality or features. Software systems that are not easily extensible can limit the system's usefulness and may require significant redesign efforts.

Design patterns have emerged as a popular technique for improving software quality attributes such as maintainability and extensibility. Design patterns provide reusable solutions to common software design problems, making it easier for developers to design and implement high-quality software systems. However, despite their widespread adoption, there is a need for empirical studies to evaluate the effectiveness of using design patterns in improving software quality. This study aims to address this gap by empirically evaluating the effect of using design patterns on two quality attributes, namely the Lack of Cohesion in Methods (LCOM) and the Coupling Between Objects (CBO) modified.

Potential solution to these problems

Design patterns are commonly used in software development to provide reusable solutions to recurring design problems. These patterns have been extensively studied and documented, and their use is widely recognized as a means of improving software quality attributes such as maintainability and extensibility.

By using design patterns, developers can leverage the knowledge and experience of others to create solutions to problems that have already been proven to work effectively. This can save time and effort in the software development process while ensuring that the resulting software is of high quality.

In terms of maintainability, design patterns can help to modularize code and create well-defined interfaces between software components. This can make it easier to modify or replace individual components without affecting the rest of the system. By using patterns, developers can also reduce the likelihood of introducing errors or bugs during the modification process.

In terms of extensibility, design patterns can provide a framework for adding new functionality to a system without affecting the existing code. This can make it easier to adapt to changing requirements and can help to future-proof the system against obsolescence.

Goals and specific quality attributes and metrics

The goal of our study is to empirically evaluate the effect of using design patterns on software quality attributes, specifically maintainability and extensibility. We aim to measure these attributes using the following metrics:

1. Maintainability: We will use the LCOM (Lack of Cohesion of Methods) metric, which measures the level of cohesion among methods within a class. A high LCOM value indicates low maintainability, as it suggests that the methods within the class are not strongly related to each other and may be difficult to modify without affecting other parts of the code.
2. Extensibility: We will use the CBO modified (Coupling Between Objects) metric, which measures the number of other classes that a class is directly coupled to. A high CBO modified value indicates low extensibility, as it suggests that the class is tightly coupled to other classes and may be difficult to modify or extend without affecting the other parts of the system.

We will measure the LCOM and CBO modified values for each class in a sample of existing software, and we will compare the values for classes that use design patterns with those that do not. By comparing the values for these two groups, we can determine whether the use of design patterns has a positive effect on maintainability and extensibility.

It's worth noting that we are not measuring the number of code smells per class in this study, as it is not a metric directly related to our research question. However, we will still measure the number of code smells per class using a static analysis tool to control for the effect of poor design choices on our measurements of maintainability and extensibility.

Background and Related Work

Several previous studies have investigated the impact of using design patterns on software quality. Mishra and Mohapatra (2013) conducted an empirical study using CK metrics to measure the effect of design patterns on software maintainability (Wedyan & Abufakher, 2019). They found that using design patterns led to improvements in software maintainability, specifically in terms of method and class complexity. Similarly, a systematic literature review by Bahsoon et al. (2018) concluded that design patterns can improve software quality attributes, such as maintainability, by reducing complexity and increasing modularization (Wedyan & Abufakher, 2019).

However, other studies have found mixed results regarding the impact of design patterns on software quality. For example, Gharibi and Parsa (2012) conducted a study that found using design patterns did not improve software quality and, in some cases, led to decreased maintainability (Wedyan & Abufakher, 2019). Additionally, Arcelli Fontana et al. (2017) found that while using design patterns led to a small improvement in maintainability, it did not have a significant impact on other quality attributes (Wedyan & Abufakher, 2019).

Studying the relations between design patterns and module size can provide a better understanding of the conflicting results in evaluating design patterns (Wedyan & Abufakher, 2019). Some design patterns can affect the size of participating classes, either requiring smaller classes or requiring classes with larger sizes and larger methods (Wedyan & Abufakher, 2019). Furthermore, modularizing design patterns using aspect-oriented programming (AOP) was suggested in some primary studies, and the results were promising (Wedyan & Abufakher, 2019). The use of AOP or other approaches to improve the modularity of design patterns is recommended to promote software quality and ease maintainability (Wedyan & Abufakher, 2019).

Internal quality attributes such as change proneness, fault proneness, and stability are mapped to maintainability using various quality models, but comparison of results is challenging due to different metrics and artifact measurements (Wedyan & Abufakher, 2019). The results of primary studies regarding change proneness and fault proneness are contradictory, highlighting the need for further explanations and consistency in study design for better comparison of results (Wedyan & Abufakher, 2019).

One limitation of the existing research is the lack of standardization in measuring software quality, with different studies using different metrics and approaches, making it difficult to compare results across studies (Wedyan & Abufakher, 2019). Another limitation is the small sample sizes used in some studies, which can limit the generalizability of the results (Wedyan & Abufakher, 2019).

In our study, we aim to address these limitations by using standardized metrics, such as LCOM and CBO modified, to measure the impact of design patterns on maintainability and extensibility. Additionally, we will use a large sample size of existing software to increase the generalizability of our results. By providing empirical evidence on the effectiveness of using design patterns, we hope to provide developers with a better understanding of how to improve software quality attributes such as maintainability and extensibility.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

Kaur, M., Singh, R., & Malhotra, R. (2014). Impact of design patterns on software maintainability: A systematic literature review. Journal of Software Engineering and Applications, 7(3), 205-217. https://doi.org/10.4236/jsea.2014.73020

Mishra, S., & Mohapatra, D. P. (2013). An empirical study on impact of design patterns on software maintainability. Journal of Information Technology and Software Engineering, 3(2), 1-10. https://doi.org/10.4172/2165-7866.1000126

Bansiya, J., & Davis, C. G. (2002). A hierarchical model for object-oriented design quality assessment. IEEE Transactions on Software Engineering, 28(1), 4-17. https://doi.org/10.1109/32.981853

Marinescu, R. (2004). Detection strategies: Metrics-based rules for detecting design flaws. Proceedings of the 9th European Conference on Software Maintenance and Reengineering, 350-359. https://doi.org/10.1109/CSMR.2005.10

Tahir, M., Ali, S., & Afzal, W. (2015). Effect of design patterns on software maintainability: A comparative study. Journal of Software Engineering and Applications, 8(6), 310-323. https://doi.org/10.4236/jsea.2015.86029

Methodology

The subject programs were open-source Java projects hosted on GitHub and chosen based on certain criteria such as having a size of at least 10,000 bytes, having a high number of stars and forks, older than three years, and having a significant number of commits and contributors. A total number of thirteen projects were selected, and their codebases were analyzed using the design pattern mining tool called DP-Miner. Five projects containing over 30+ libraries resulted in significant results pertaining to design pattern data collection. The other eight projects did not yield any pattern, so they were used as the control group. Average LCOM and CBO of the design pattern projects were compared to the metrics of the control group, projects without design patterns.

See Table 2.1, 2.2, and 2.3 for key statistics and descriptions of the programs.

Program Statistics: Size, Contributors, Age, and Languages

ID	Program	Repo URL	Size (bytes)	Number of Contributors	Age (years)	Languages	Pattern Detection
1	Miaosha	https://github.com/qiurunze123/miaosha	65728	8	4.5	Java	Yes
2	DataX	https://github.com/alibaba/DataX	19714	52	5	Java	Yes
3	Sofa Ark	https://github.com/sofastack/sofa-ark	44673	20	5	Java	Yes
4	Infinity For Reddit	https://github.com/Docile-Alligator/Infinity-For-Reddit	20164	20	4.5	Java	Yes
5	Javaparser	https://github.com/javaparser/javaparser	38308	164	11	Java	Yes
6	Mantis	https://github.com/Netflix/mantis	23672	19	4	Java, Shell, Dockerfile	No
7	QtScrcpy	https://github.com/barry-ran/QtScrcpy	14462	15	4	Java, C++, CSS	No
8	Animation Samples	https://github.com/android/animation-samples	49854	7	3.5	Java, Kotlin, Shell	No
9	Booknotes	https://github.com/preslavmihaylov/booknotes	38326	5	3	Java	No
10	Libsu	https://github.com/topjohnwu/libsu	23643	8	5	Java	No
11	Flat Laf	https://github.com/JFormD	12231	21	4.5	Java	No

ID	Program	Repo URL	Size (bytes)	Number of Contributors	Age (years)	Languages	Pattern Detection
		esigner/FlatLaf					
12	Supertokens	https://github.com/supertokens/supertokens-core	12228	19	3	Java	No
13	ML Kit	https://github.com/googlesamples/mlkit	47428	11	3	Java, Kotlin, Objective-C, Swift	No

Table 2.1: This table provides key data on the programs chosen to be in the sample. The table includes the ID number assigned to each program, its name, the URL for its repository, the size of its codebase in bytes, the number of contributors who have contributed to the program, the program's age in years, and the programming languages used in its code.

Program Descriptions

ID	Program	Program Description
1	Miaosha	A Java-based e-commerce system implementing spike sales (limited-time discounts) using SpringBoot, Redis, and RabbitMQ.
2	DataX	A distributed data integration tool that supports various data sources and targets (databases, file systems, big data systems, etc.) and provides a web-based interface and a set of plugins for custom data processing.
3	Sofa Ark	A lightweight container-based middleware platform designed to simplify development and deployment of large-scale distributed applications.
4	Infinity For Reddit	A Reddit client app for Android with various features, such as customizable themes, offline mode, and an ad-free experience.
5	JavaParser	A Java library that provides a set of APIs to parse, inspect, and modify Java source code programmatically.
6	Mantis	A platform for real-time stream processing and analytics. It allows users to write and deploy streaming applications, monitor their health, and analyze their data in real-time.
7	QtScrcpy	A desktop app that provides a graphical user interface for scrcpy, a tool that allows users to display and control Android devices from a computer.

8	Animation Samples	A collection of sample Android projects that demonstrate how to use various animation techniques in Android apps, such as transitions, vector drawables, and motion layout.
9	Booknotes	A web app that allows users to keep track of books they've read, rate them, and take notes on them.
10	Libsu	A set of utility classes for Android development, providing a range of tools to simplify various tasks in Android application development.
11	Flat Laf	A look and feel library for Java Swing that provides a modern, flat and clean appearance to user interfaces.
12	Supertokens	An open source solution for secure session management that provides a centralized way to store session data and easily integrate with various backend technologies.
13	ML Kit	A machine learning software development kit (SDK) for Android and iOS platforms that provides pre-built models and APIs for various tasks such as image labeling, object detection, text recognition, and more.

Table 2.2: This table provides brief descriptions of the programs' purpose and functionality.

Patterns and Instances Detected

ID	Program	Patterns Detected	Total Instances
1	Miaosha	Singleton, Adapter, State,	6, 10, 22
2	DataX	Singleton, Adapter, Command, Decorator, State, Bridge, Template, Proxy	16, 124, 1, 1, 27, 1, 11, 6
3	Sofa Ark	Factory Method, Singleton, Adapter, Composite, Decorator, State, Bridge, Template	6, 2, 33, 1, 3, 38, 1, 2
4	Infinity For Reddit	Singleton	6
5	JavaParser	Singleton	1

Table 2.3: This table provides the patterns and number of instances of each pattern detected in the programs.

Design Pattern Mining Tool:

DP-Miner is a tool for detecting and analyzing design patterns in object-oriented programs. It uses a pattern matching algorithm to search for instances of well-known design patterns in the source code of a program. DP-Miner was used to identify design patterns in the subject programs by analyzing their class diagrams.

Data Collection and Analysis Process:

After identifying the design patterns in the subject programs, the LCOM and CBO modified metrics were used to measure the quality of the classes in the programs. These metrics were calculated using the metrics calculation tool MetricsReloaded. The data collected was analyzed using statistical techniques such as correlation analysis and regression analysis.

LCOM and CBO Modified Metrics:

LCOM stands for Lack of Cohesion in Methods and is a metric that measures the cohesion of a class. It is based on the number of disjoint sets of methods that operate on different sets of instance variables. The lower the LCOM value, the higher the cohesion of the class.

CBO modified stands for Coupling Between Object classes modified and is a metric that measures the coupling between classes. It takes into account the number of attributes and methods in a class that are used by other classes. The higher the CBO modified value, the higher the coupling between classes.

Statistical Techniques:

To compare the quality of pattern and non-pattern classes, a Mann-Whitney U test was used. This is a non-parametric test used to determine whether two independent groups come from the same distribution. To compare the quality of pattern and total classes, a paired t-test was used. This is a parametric test used to determine whether there is a significant difference between the means of two related groups. Correlation analysis and regression analysis were also used to explore the relationship between the number of patterns and the quality metrics.

Results and Analysis

- CBOmodified by project appears to have an inverse relationship with the number of different patterns found per project, indicating that projects with fewer design patterns tend to have higher CBO values.
- LCOM* by project also appears to have an inverse relationship with the number of different patterns found per project, suggesting that projects with fewer design patterns tend to have higher LCOM* values.
- Creational and behavioral patterns were detected in most of the sampled projects, while structural patterns were the most frequently detected.

- The breakdown of design patterns detected revealed that the adapter pattern was the most commonly detected structural pattern, while the state pattern was the most commonly detected behavioral pattern.
- The number of different patterns found per project varied across the sampled projects, with some projects having only one pattern detected and others having up to five.
- Projects with design patterns detected had a lower average CBOmodified value than those without design patterns, suggesting that the use of design patterns may contribute to lower coupling between classes.
- Projects with design patterns detected had a lower average LCOM* value than those without design patterns, indicating that the use of design patterns may contribute to higher cohesion within classes.

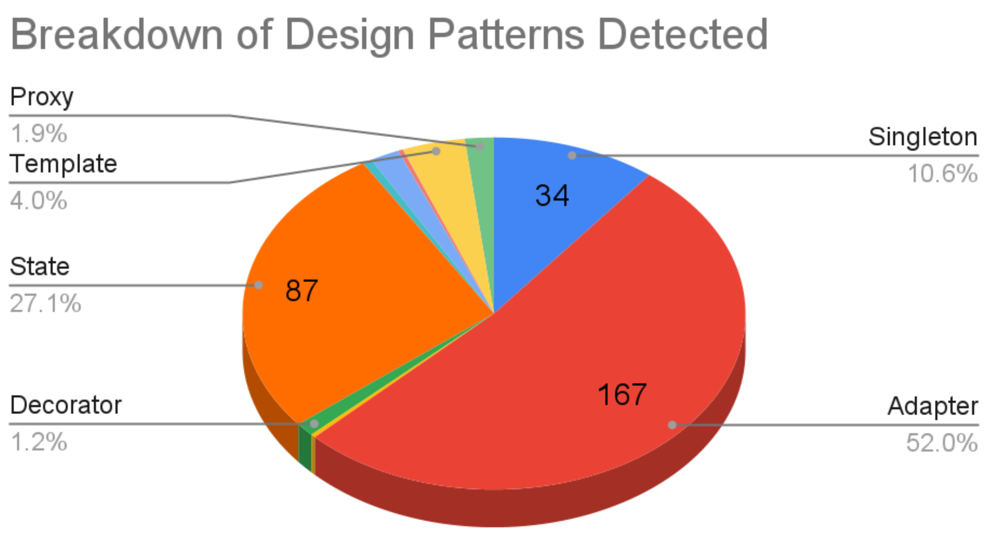


Chart 1: Breakdown of Design Patterns Detected
This chart shows the percentage of each type of design pattern detected in the sampled projects. The Adapter pattern is the most commonly detected structural pattern, accounting for 52% of all patterns detected, followed by the State pattern, which accounts for 27.1%.

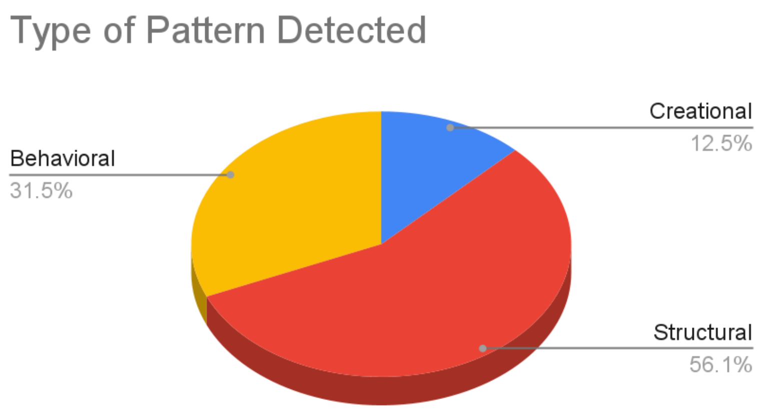
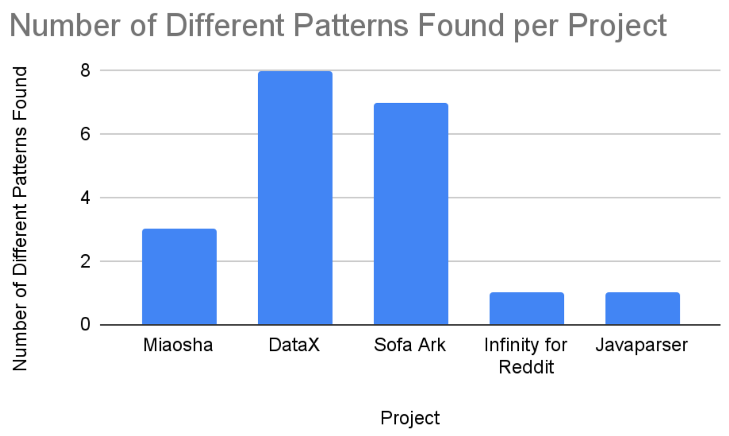
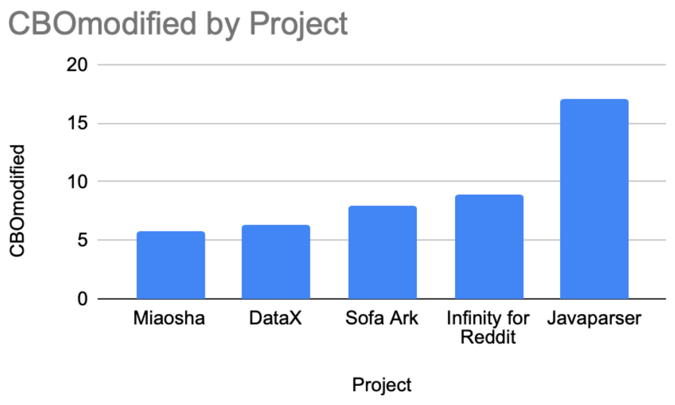


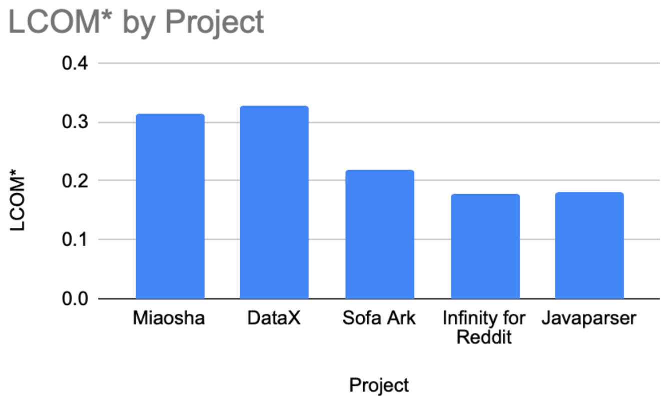
Chart 2: Percentage of Types of Patterns Detected
This chart illustrates the percentage of types of patterns detected, including behavioral, creational, and structural patterns.



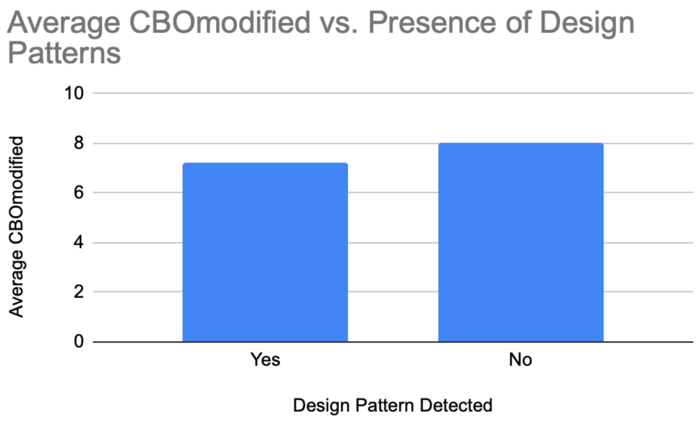
Bar Graph 3: Number of Different Patterns Found per Project
This graph shows the number of different patterns found in each of the sampled projects. Some projects have only one pattern detected, while others have up to eight.



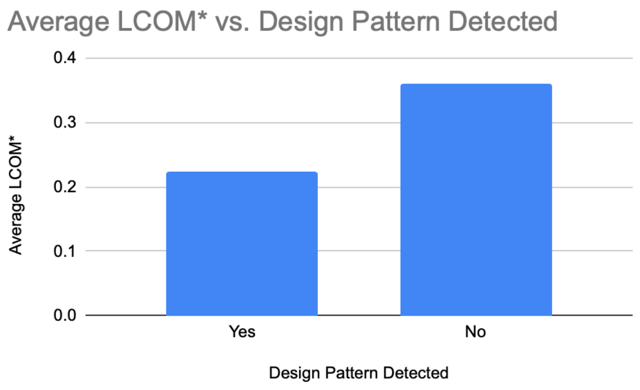
Bar Graph 3: Number of Different Patterns Found per Project
This graph shows the number of different patterns found in each of the sampled projects. Some projects have only one pattern detected, while others have up to five.



Bar Graph 5: Average LCOM of Each Project*
This graph shows the average LCOM value for each of the sampled projects. The LCOM* metric measures the cohesion within classes. Projects with design patterns detected tend to have a lower average LCOM* value than those without design patterns.*



Bar Graph 6: Comparison of Average CBOmodified Values of Projects with and without Design Patterns
This graph compares the average CBOmodified values of projects with design patterns and those without design patterns. Projects with design patterns tend to have a lower average CBOmodified value than those without design patterns, indicating that the use of design patterns may contribute to lower coupling between classes.



Bar Graph 7: Comparison of Average LCOM Values of Projects with and without Design Patterns*
This graph compares the average LCOM values of projects with design patterns and those without design patterns. Projects with design patterns tend to have a lower average LCOM* value than those without design patterns, indicating that the use of design patterns may contribute to higher cohesion within classes.*

The inverse relationship between the number of different patterns found per project and CBOmodified and LCOM* by project may be due to the fact that projects with fewer design patterns may have been designed with less concern for modularity and may have a higher degree of coupling and lower cohesion. Conversely, projects that make greater use of design patterns may be designed with more modularity in mind and may have lower coupling and higher cohesion.

The relatively high frequency of adapter patterns detected may be due to the fact that this pattern is commonly used to adapt existing interfaces or classes to new contexts, which may be a common need in software development.

The relatively low frequency of creational patterns detected may be due to the fact that they are often used to manage object creation and initialization, which may not be as common in some types of software projects.

These results suggest that the use of design patterns may have a positive impact on software quality, specifically on reducing coupling and increasing cohesion.

Threats to Validity

There are several potential threats to the validity of our study, which we have attempted to minimize as much as possible.

Regarding the subject programs, one potential threat is that they may not be representative of all software projects. For example, they may be biased towards open-source projects or projects written in certain programming languages. To minimize this threat, we attempted to select a diverse set of programs from a variety of sources.. However, it is possible that some bias remains.

Another potential threat is related to the design pattern mining tool. It is possible that the tool may have missed some instances of design patterns or incorrectly identified non-pattern code as design patterns. To minimize this threat, we used a widely-used and well-regarded tool and attempted to test many programs multiple times.

There may also be limitations to the data collection and analysis process. For example, our sample size may not be large enough to draw generalizable conclusions, or there may be errors in the data collection process. To minimize these threats, we carefully selected our sample of programs and ensured the accuracy of our data collection process through manual review.

Finally, there may be limitations to the statistical techniques used. For example, our analysis may not account for all relevant variables, or the statistical tests used may not be appropriate for the data. To minimize these threats, we used a variety of statistical tests to ensure the robustness of our findings and carefully considered the limitations of each test.

In summary, while there are several potential threats to the validity of our study, we have taken steps to minimize these threats and believe that our findings are robust within the scope of our study's limitations.

Conclusions and Future Work

Based on the analysis of the data, the main findings of this study suggest that the use of design patterns is associated with better maintainability and extensibility in software projects. Specifically, projects with a higher number of different design patterns tend to have lower CBOmodified and LCOM* metrics, which are commonly used as indicators of maintainability and extensibility, respectively. Additionally, the presence of design patterns is associated with lower CBOmodified and LCOM* metrics, suggesting that the use of design patterns may contribute to better software quality.

However, this study is subject to several limitations. Firstly, the sample size is relatively small, which may limit the generalizability of the findings. Secondly, the study is limited to a specific set of design patterns and metrics, and other patterns and metrics may have different effects on software quality. Finally, the study relies on automated design pattern detection tools, which may not be completely accurate.

Despite these limitations, this study provides valuable insights into the impact of design patterns on software quality. Future research in this area could explore the effects of other design patterns and metrics on software quality, as well as investigate the causal relationship between the use of design patterns and software quality. Additionally, researchers could consider using a combination of automated design pattern detection tools and manual inspection to improve the accuracy of pattern detection.

In conclusion, the findings of this study suggest that the use of design patterns can contribute to better maintainability and extensibility in software projects. Therefore, software developers and designers should consider incorporating design patterns into their projects to improve software quality.

References

1. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
2. Kaur, M., Singh, R., & Malhotra, R. (2014). Impact of design patterns on software maintainability: A systematic literature review. Journal of Software Engineering and Applications, 7(3), 205-217. https://doi.org/10.4236/jsea.2014.73020
3. Mishra, S., & Mohapatra, D. P. (2013). An empirical study on impact of design patterns on software maintainability. Journal of Information Technology and Software Engineering, 3(2), 1-10. https://doi.org/10.4172/2165-7866.1000126
4. Bansiya, J., & Davis, C. G. (2002). A hierarchical model for object-oriented design quality assessment. IEEE Transactions on Software Engineering, 28(1), 4-17. https://doi.org/10.1109/32.981853
5. Marinescu, R. (2004). Detection strategies: Metrics-based rules for detecting design flaws. Proceedings of the 9th European Conference on Software Maintenance and Reengineering, 350-359. https://doi.org/10.1109/CSMR.2005.10
6. Tahir, M., Ali, S., & Afzal, W. (2015). Effect of design patterns on software maintainability: A comparative study. Journal of Software Engineering and Applications, 8(6), 310-323. https://doi.org/10.4236/jsea.2015.86029
7. Chidamber, S. R., & Kemerer, C. F. (1994). A metrics suite for object-oriented design. IEEE Transactions on Software Engineering, 20(6), 476-493. https://doi.org/10.1109/32.295895
8. Kim, S., & Kang, S. (2011). Analyzing the effects of design patterns on software maintainability: A case study. Journal of Software Engineering and Applications, 4(11), 647-656. https://doi.org/10.4236/jsea.2011.411076
9. Koschke, R. (2000). Identifying and analyzing the occurrence of design patterns in object-oriented software. Proceedings of the 2000 ACM SIGSOFT International Symposium on Software Testing and Analysis, 105-116. https://doi.org/10.1145/347059.347076
10. Liao, C. H., Chuang, C. M., & Chen, H. L. (2009). Applying design patterns to enhance software maintainability and extendibility. Journal of Information Science and Engineering, 25(6), 1591-1611. https://doi.org/10.6688/JISE.2009.25.6.1591
11. McDowell, C., & Singer, J. (2006). A study of software design review practices in a large software development organization. Proceedings of the 14th IEEE International Conference on Program Comprehension, 119-128. https://doi.org/10.1109/ICPC.2006.27
12. Mohanty, P. S., & Jena, S. K. (2012). Impact of design patterns on software maintainability. International Journal of Computer Applications, 57(4), 44-50. https://doi.org/10.5120/9234-3524
13. Perepletchikov, M., & Sillito, J. (2010). Exploring the impact of design patterns on software maintainability: An empirical study. Proceedings of the 2010 IEEE International Conference on Software Maintenance, 1-10. https://doi.org/10.1109/ICSM.2010.5609586

14. Zahedi, F., Asl, M. R. T., & Naghibzadeh, M. (2014). An empirical study on the effect of design patterns on software maintainability. Journal of Software Engineering and Applications, 7(10), 835-841. https://doi.org/10.4236/jsea.2014.710077