Infinity Engineering Journal
Vol. 1 No. 3, August (2025), pp. 58-65

ISSN 3090-5621 (online)
Published by Yayasan Cinta Negara Indonesia

The Role of Design Patterns in Improving Code Structure and Efficiency: A Systematic Review of the Literature

Widodo Sihotang[1], Muhammad Irwansyah Putra[2]
[1] Fakultas Teknologi dan Bisnis, Program Studi Bisnis Digital
[2] Fakultas Teknologi dan Bisnis, Program Studi Sistem Informasi
Universitas Putra Abadi Langkat

ARTICLE INFO

Article history:

Received: Jun 20, 2025
Revised: July 12, 2025
Accepted: July 24, 2025

Keywords:

Design Patterns;
Code Efficiency;
Maintainability;
Modularity;
Systematic Literature Review.

ABSTRACT

Code structure quality and efficiency are crucial aspects in modern software development, especially as system complexity increases and demands for scalability and maintainability. This research aims to systematically review the literature related to the use of design patterns in improving the quality of code architecture, with a focus on structure, modularity, and development efficiency. Through a Systematic Literature Review (SLR) approach based on Kitchenham's (2007) protocol, 40 articles from reputable databases such as IEEE Xplore, ACM Digital Library, Scopus, and Springer were screened based on specific inclusion and exclusion criteria. The study results show that design patterns such as Factory, Strategy, Observer, and Composite are the most common and effective in increasing modularity, speeding up the development process, and reducing the level of vulnerability to bugs. In addition, there is empirical evidence that the application of design patterns supports more structured and sustainable software engineering practices in both small and large-scale projects. This study also highlights the importance of integrating design patterns with modern approaches such as DevOps, microservices, and AI-assisted coding. The findings are expected to provide practical contributions for developers, academics, and technical decision-makers in improving the overall quality of software systems.

Corresponding Author:

Widodo Sihotang
Fakultas Teknologi dan Bisnis, Program Studi Bisnis Digital
Universitas Putra Abadi Langkat
Jl. Letjen R. Soeprapto No.10, Sumatera Utara 20814. Indonesia
Email: widodosihotang85@gmail.com

1. INTRODUCTION

Modern software development increasingly demands high-quality code structure and efficiency in the development process. As the scale and complexity of applications increase, especially in large-scale and distributed systems, developers are required to produce code that is not only functional, but also readable, maintainable, and extensible (Abdelmoez et al., 2020). Poor code structure can lead to increased maintenance costs, the occurrence of bugs, and difficulties in integration or further development.

The complexity of software development has also increased with the adoption of new paradigms such as cloud computing, microservices, and container-based development. These environments demand coding approaches that are not only efficient, but also consistent across teams and software lifecycles (Martínez-Fernández et al., 2021). Without good code structure and organized design principles, software development is at risk of significant technical problems.

In this context, a systematic approach to designing and organizing code becomes very important. One approach that has proven useful is the use of software design patterns. Design patterns are reusable solutions to common problems in software design, which help developers

create stable and modular system architectures (Gamma et al., 1995; updated for relevance by Sharma & Chatterjee, 2020).

The application of design patterns allows developers to improve code quality in aspects such as cohesion, separation of concerns, and readability, leading to increased development efficiency. The study by Zhao et al. (2022) shows that projects that implement design patterns tend to have lower levels of code smell and are more maintainable than projects that do not use them.

Design patterns also encourage software development practices based on sound software engineering principles, such as SOLID principles. By applying patterns such as Strategy, Observer, or Factory, developers can avoid duplication of logic, isolate changes, and reinforce the open/closed principle (Zhang & Sun, 2023).

However, inappropriate selection and use of design patterns can also lead to over-engineering or unnecessary additional complexity. Therefore, an in-depth and systematic understanding of the role of design patterns on the quality of code structure and efficiency is needed so that their use is truly targeted (Mourão et al., 2021).

In that context, this research aims to answer two main questions. First, what is the role of design patterns in solving the challenges of code structure and software development efficiency? This includes aspects of how design patterns affect code quality, maintainability, and system scalability. Second, which design patterns are most widely used and proven effective in practice?

This issue is important to address because in real development practice, not all design patterns are used equally. Some patterns such as Singleton, Factory, and Observer tend to be more popular, but their effectiveness is highly contextual (Pereira et al., 2020). Therefore, empirically identifying the most useful patterns will greatly assist developers in making design decisions.

The main objective of this study is to systematically review the scientific literature that addresses the use of design patterns in improving code structure and efficiency. This review will include academic studies published in the last five years, either from journals, conferences, or technical reports.

In addition, this study also aims to identify commonly used design patterns in object-based software development, and evaluate their effectiveness based on previous findings. By conducting a comparative analysis between studies, it is expected to find patterns that are effective in certain development contexts.

The results of this study will make a significant contribution to software development practice. By providing a deeper understanding of the role and effectiveness of design patterns, this study can serve as a reference for software engineers in making better technical decisions when designing system architectures.

Moreover, in an educational context, the results of this study can inform teaching based on the latest empirical findings. Programming and software engineering curricula can be strengthened with insights into design patterns that are relevant to current industry needs (García-Mireles et al., 2020).

Furthermore, this study will also help organizations in improving the productivity of the development team. By applying the right design patterns, the development process can be more standardized, team collaboration can be improved, and development time can be reduced without sacrificing quality (Nugroho et al., 2021).

On a broader scale, this study also contributes to the development of a more practice-oriented software engineering methodology. By aligning the theory of design patterns with the real needs of the industry, this study can bridge the gap between the theory and practice of software development.

Finally, this study also has strategic value in supporting software architecture decision-making at the technical management level. By understanding effective and contextual patterns, software architects can design more adaptive, modular and sustainable systems (Khomyakov & Borodin, 2022).

2. METHODS

This research uses the Systematic Literature Review (SLR) approach as the main method to collect, evaluate, and synthesize findings from previous studies related to the use of software design patterns in improving code efficiency and structure. The SLR approach was chosen because it provides a systematic and structured framework in identifying relevant literature, thus producing reliable and replicable evidence (Kitchenham & Charters, 2007; Zhang et al., 2020). The SLR protocol in this

study follows Kitchenham's guidelines that have been adapted in various contemporary software engineering studies, which emphasize explicit stages ranging from research question formulation, study selection, data extraction, to analysis and synthesis of results (Kitchenham et al., 2015; Ampatzoglou et al., 2019).

To ensure the quality and relevance of the analyzed studies, strict inclusion and exclusion criteria were set. Studies will be included if they qualify as peer-reviewed journal or conference articles indexed in reputable databases such as Scopus, IEEE Xplore, ACM Digital Library, or SpringerLink. In addition, the study must explicitly address software design patterns, and examine their relationship with code efficiency, code structure, or software maintainability. Studies published within 2013-2023 will be considered to ensure they cover the latest trends in software development (Petersen et al., 2021).

Conversely, studies will be excluded if they are not peer-reviewed, do not provide empirical evaluation, or only focus on developing tools without explicitly discussing the application of design patterns. Articles such as editorials, opinion pieces, or briefs that do not provide data or experimental results will also be eliminated to maintain the quality of the data synthesis (de Oliveira Neto et al., 2020).

The main data sources in this study include four major digital databases that have long been used in software engineering research: IEEE Xplore, Scopus, ACM Digital Library, and Google Scholar. These four sources were chosen because they provide a broad, quality coverage of academic literature relevant to the field of software engineering. To identify relevant studies, a combination of keywords formulated based on the problem formulation and study objectives were used, including: “software design patterns”, “code efficiency”, “code maintainability”, “software architecture”, and “software quality”. The use of Boolean operators such as AND and OR were also applied to optimize the search results (Alsolai & Rilling, 2020).

The search strategy was iterative, with each result evaluated based on title, abstract and keywords to ensure initial suitability. The bibliographies of articles deemed relevant were also further searched as part of the snowballing technique, to expand the scope of literature that may not have appeared in the initial search (da Silva et al., 2019).

The study selection process was conducted in three main stages. The first stage was an initial screening based on titles and abstracts, where articles that were clearly irrelevant or did not meet the inclusion criteria were immediately eliminated. The second stage was a full review of the content of the articles that passed the initial selection, focusing on the methodology, the context in which the design patterns were used, and the empirical data or results presented. At this stage, each article was thoroughly evaluated to determine its eligibility for the final analysis (Kaur et al., 2022).

The third stage was mapping the studies based on topic, type of design patterns covered, evaluation methods, and findings. This mapping process helped in identifying common patterns, research gaps, and correlations between the use of design patterns and the outcomes achieved in the studies. All selection decisions were transparently recorded, and if there was disagreement between researchers in the selection process, discussions were held to reach consensus (Dybå et al., 2021).

In the data analysis process, a combined approach of qualitative and quantitative analysis was used. The qualitative approach was conducted through thematic coding technique, where each study was coded based on key themes such as the type of design pattern used, the context in which it was developed, and its impact on the structure and efficiency of the code. This technique allows for systematic and in-depth clustering of information (Cruzes & Dybå, 2019).

Meanwhile, the quantitative approach is applied through tabulations and descriptive statistics, such as the frequency of use of certain design patterns, the distribution of publication years, and the type of evaluation used in each study. In addition, trends in the use of design patterns over time are also identified to provide insight into the evolution of software design practices in the last decade (Wahyudin et al., 2023). With this combination approach, it is expected that the SLR results are not only descriptive, but also able to provide a comprehensive mapping of the effectiveness of design patterns in various software development contexts.

3. RESULTS AND DISCUSSION

Study Description

The studies analyzed in this Systematic Literature Review show the diversity of methodological approaches and implementation focus of design patterns in various software development contexts.

For example, the study by Zhao et al. (2022) titled "Design Pattern Use and Code Smell Reduction: An Empirical Study" used a quantitative empirical study approach to analyze 200 open-source Java projects. The main result of this study shows that the use of design patterns, especially Observer, Strategy, and Factory, significantly reduces the frequency of code smells, especially in the aspects of long methods and god classes. This study indicates that design patterns are not only architectural, but also have a direct impact on code quality at the implementation level.

Meanwhile, Pereira et al. (2020) in their study entitled "Evaluating the Usefulness of Design Patterns: A Controlled Experiment" implemented a laboratory experiment to evaluate developers' understanding and productivity in using design patterns. The study involved two groups of final-year students and junior developers, with one group given training on design patterns before developing a simple application. The results showed that the group trained with patterns such as Singleton, Decorator, and Composite produced code that was more modular and faster at completing tasks. This supports the hypothesis that understanding design patterns can speed up the development process and improve code structure.

Another study by Mourão et al. (2021) entitled "Overuse of Design Patterns: An Empirical Investigation" took a case study approach to five industrial-scale software projects. The study found that although design patterns contribute to the regularity of system architecture, the overuse of patterns such as Abstract Factory and Facade causes unnecessary complexity and increases the learning curve for new developers. Therefore, it is important to consider the context of requirements when deciding to apply a pattern.

Zhang and Sun (2023) in a study entitled “Enhancing Object-Oriented Software Design with Pattern-Based Refactoring” used an experimental method that focused on the pattern-based refactoring process. This study compares the effectiveness of manual refactoring with pattern-based refactoring on legacy systems. The main findings showed that refactoring using patterns such as Template Method and Command accelerated the code modernization process and improved readability and separation of responsibilities between classes. This study confirms the importance of design patterns in the refactoring process of legacy systems to keep them relevant to modern development practices.

In the realm of software education, García-Mireles et al. (2020) examined the effectiveness of teaching design patterns in a study titled "Teaching Design Patterns in Software Engineering Courses: A Review". Using a systematic mapping study method, they reviewed 45 educational articles and found that the integration of teaching design patterns, especially Model-View-Controller (MVC) and Observer, contributed to better understanding of software architecture structures in students. This study supports the use of design patterns not only in professional practice but also in formal education as a basis for systematic design thinking.

Alsolai and Rilling (2020) in their article “Systematic Literature Review of the Impact of Design Patterns on Software Maintainability” conducted an SLR of 38 empirical studies. They concluded that design patterns such as Decorator, Adapter, and Composite are most often associated with increased code maintainability. The study showed that there is a positive correlation between pattern implementation and software maintainability metrics such as change proneness and modularity.

Furthermore, a study by Wahyudin et al. (2023) entitled "Recent Trends in Design Pattern Application and Software Modularity: A Meta-Analysis" conducted a meta-analysis of design pattern usage trends over a decade. Using quantitative analysis of 80 publications, they concluded that Strategy, Factory, and Observer remain the three most commonly used patterns in various domains such as mobile apps, web development, and distributed systems. These patterns are considered effective in increasing modularity, flexibility, and enabling high code reusability.

Finally, a study from Kaur et al. (2022) entitled "A Systematic Review on Software Design Patterns: Classification and Impact on Software Quality" presents a classification of design patterns based on functional categories (creational, structural, and behavioral) and their relationship with software quality aspects such as readability, testability, and scalability. They found that behavioral patterns such as State and Command are more often associated with system flexibility, while structural patterns such as Facade and Composite play an important role in managing complexity between modules.

Overall, these studies show that the use of design patterns has a significant impact on code structure and efficiency. However, their effectiveness is highly dependent on the context in which they are used, the developer's understanding, and the balance between the functional and

architectural requirements of the system. Therefore, the selection of design patterns cannot be done generically, but should be based on a proper analysis of system requirements and structure.

Key Findings

The results of this Systematic Literature Review reveal that the most widely used design patterns in software development over the past decade are Factory, Strategy, Observer, and Singleton. These four patterns stand out for their flexibility in various development contexts, ranging from enterprise systems to mobile applications and IoT devices. For example, the Factory pattern is often used in complex and varied object creation scenarios, helping in reducing duplication of instantiation logic as well as improving encapsulation (Wahyudin et al., 2023). Meanwhile, Strategy is widely used in application development that requires the ability to dynamically switch algorithms without changing the main structure of the code, especially in plugin-based or modular systems (Zhang & Sun, 2023). The Observer pattern proves to be very useful in event-driven systems such as user interfaces or real-time applications, as it supports the separation between subjects and observers in an efficient way (García-Mireles et al., 2020). On the other hand, Singleton is still widely used, although it comes with a warning regarding the risk of hidden dependencies in large-scale systems.

In terms of development time efficiency and system modularity, many studies have shown that the use of design patterns significantly speeds up the implementation process by providing a framework of proven solutions. For example, an experimental study by Pereira et al. (2020) showed that a group of developers using design patterns could complete development tasks up to 25% faster than a control group, with fewer code revisions. This increase in efficiency is largely due to the clear initial structure and ease of distributing responsibilities between code components. In addition, patterns such as Composite, Decorator, and Adapter have been shown to increase modularity as they allow developers to build systems from small components that can be replaced or extended without changing other components (Alsolai & Rilling, 2020). This high modularity also facilitates unit testing in isolation and supports continuous integration-based development practices.

In terms of system performance, the findings are contextual. Some studies such as Mourão et al. (2021) note that the use of certain design patterns-such as Decorator or Proxy-can introduce additional runtime overhead due to the use of additional layers in execution. However, if used proportionately and appropriately, such overheads can be compensated by other benefits such as flexibility and ease of maintenance. In general, the trade-off between performance and good structure is an important consideration in the implementation of design patterns. Recent studies emphasize the importance of performance profiling and evaluation when integrating design patterns into large-scale systems (Zhao et al., 2022).

One of the most significant contributions of design patterns found in this study is their impact on bug reduction and improved maintainability. The study by Zhao et al. (2022) revealed that systems using design patterns experienced an average 18% reduction in the number of code smells, including god classes, long methods, and feature envy. This reduction directly contributes to higher internal code quality and makes refactoring easier. In addition, a study by Alsolai and Rilling (2020) showed that systems that utilize patterns such as Composite and Template Method have higher maintainability scores based on metrics such as cyclomatic complexity, change proneness, and depth of inheritance tree. In the context of development teams, García-Mireles et al. (2020) found that documentation and communication between teams became more consistent when design patterns were used as design references because the terminology and structure were standardized.

Other empirical studies also support that developers who understand and apply design patterns are faster at finding and fixing bugs. Kaur et al. (2022) showed that teams using design patterns had an average of 30% less time in debugging compared to non-pattern teams, due to higher code readability and clearer separation of responsibilities. In addition, design patterns facilitate the integration of automated tests due to their modular structure and consistency, thus supporting test-driven development (TDD) and behavior-driven development (BDD) practices.

Overall, the findings from these studies show that design patterns not only help develop cleaner and more structured code, but also have real implications in team productivity, system quality, and development cycle time. However, their effectiveness is greatly influenced by the context in which they are used, including the experience of the team, the complexity of the project, and the discipline of following correct design principles. Therefore, design patterns should not be used mechanistically, but based on careful consideration of architectural needs and software engineering principles.

4. DISCUSSION

Interpretation of Findings

Based on a systematic literature analysis, it can be concluded that consistent use of design patterns helps improve code structure and efficiency in various scales of software projects. In small-scale projects, design patterns such as Singleton, Factory Method, and Strategy provide a more organized code structure and facilitate the separation of responsibilities between modules. This has a positive impact on code readability and makes it easier to maintain and test units independently (Pereira et al., 2020). The study by García-Mireles et al. (2020) also shows that utilizing these patterns in academic projects encourages students to think in a modular and reusable framework, despite the limited scale of the project.

Meanwhile, in large-scale projects, the benefits of design patterns become more significant. Patterns such as Observer, Facade, Composite, and Template Method help address system complexity by simplifying interactions between subsystems, reducing logic duplication, and improving module cohesion (Wahyudin et al., 2023; Zhao et al., 2022). In enterprise systems with multiple developers, design patterns also serve as a common language for conveying system architecture and flow, thus accelerating team understanding and collaboration (Zhang & Sun, 2023). In addition, design patterns support SOLID principles, which are proven to improve code maintainability and scalability in large systems (Alsolai & Rilling, 2020).

Practical Implications

The findings have a number of important practical implications, particularly for software developers and system architects. First, developers are advised to not only understand design patterns theoretically, but also learn the context of their application in real projects. Design patterns should be used to solve specific design problems such as the need for flexibility, reduced duplication, or increased modularity, rather than simply following patterns without justification (Mourão et al., 2021). Software architects can also take advantage of the catalog of design patterns to design an initial system architecture with a stable and easily extensible foundation.

In the context of education, the integration of learning design patterns in the software engineering curriculum in higher education is very important. As pointed out by García-Mireles et al. (2020), teaching design patterns not only introduces students to good software design principles, but also improves problem-solving and technical communication skills. Pattern-based coding practices can be made part of a final project or case study, for students to understand the link between design and implementation. Moreover, with the increasing adoption of Agile and DevOps methodologies, mastering design patterns is becoming increasingly relevant to accelerate development iterations and facilitate continuous refactoring (Kaur et al., 2022).

Study Limitations

While the findings from this study provide significant insights, there are some limitations that need to be noted. First, the number of empirical studies with a robust quantitative methodology on the impact of design patterns on code efficiency is limited. Many studies are exploratory or case study-based, so conclusions drawn are still dependent on project context and individual experiences (Alsolai & Rilling, 2020). Secondly, publication bias is also a challenge, where studies reporting positive results are more likely to be published than studies with negative or neutral results, which can lead to an imbalance in the interpretation of design pattern effectiveness.

Another limitation lies in the diversity of project scales and domains in the literature. Most studies are conducted on small or medium-sized projects, with the participation of students or junior developers, which do not fully reflect the challenges of industrial-scale system development (Pereira et al., 2020). In addition, there are few studies that specifically compare the effectiveness of one pattern with another in the same context. This makes it difficult to generalize which pattern is most efficient in different types of applications. Therefore, more systematic and large-scale follow-up studies, with quasi-experimental or longitudinal approaches, are needed to evaluate the long-term impact of using design patterns on software quality.

5. CONCLUSIONS

This study confirms that the use of design patterns plays a crucial role in improving the quality of structure, modularity, and code efficiency in software development. Based on the results of a Systematic Literature Review of various recent scientific publications, it was found that design

patterns such as Factory, Strategy, Observer, and Composite consistently support the development of more organized and maintainable systems. Design patterns provide standardized solutions to common software engineering problems, allowing developers to save development time, improve code readability, and reduce system complexity. Moreover, empirical studies show a positive correlation between the use of design patterns and a decrease in the number of bugs, improved maintainability, and teamwork efficiency. Overall, the findings reinforce the notion that design patterns are not just theoretical artifacts, but have a real impact in modern software development practices. To enrich the understanding of the role of design patterns in the modern context, future studies are recommended to explore the integration of design patterns with contemporary software development practices such as DevOps, microservices architecture, and AI-assisted coding. In the era of automation and continuity of integration, it is important to know how design patterns support continuous development pipelines and deployment of infrastructure as code. In addition, further research can examine how design patterns are applied in microservices systems, especially in managing inter-service communication, resiliency, and scalability. In addition, the emergence of artificial intelligence-based programming tools-such as code generated by AI coding assistant systems-opens up new research opportunities to understand whether and how design patterns can be automatically recognized, taught, or applied by helper models. It is also important to conduct longitudinal studies and cross-domain studies to see the long-term and cross-context impact of applying design patterns. This study makes meaningful contributions to the software development community, academia, and decision-makers in software engineering. For developers, this study provides a synthesis of empirical knowledge that can be used as a reference for selecting and applying design patterns contextually and effectively in real projects. For academics, the results of this study reinforce the importance of integrating design patterns in the higher education curriculum, and encourage further research that is more in-depth and methodological. Meanwhile, for technical decision-makers in software companies, these findings can be used as a basis for designing software architectures that are oriented towards maintainability, scalability, and long-term efficiency. Thus, this study is not only conceptual, but also has practical relevance in supporting quality and sustainable software development.

REFERENCES

Abdelmoez, W., Moustafa, M., & Elfatatry, A. (2020). Improving Software Quality through Design Patterns: A Systematic Review. Journal of Systems and Software.

Alsolai, H., & Rilling, J. (2020). Systematic Literature Review of the Impact of Design Patterns on Software Maintainability. Empirical Software Engineering, 25(4).

Ampatzoglou, A., Avgeriou, P., & Chatzigeorgiou, A. (2019). A Review of Systematic Literature Review Guidelines in Software Engineering. Information and Software Technology, 102.

Cruzes, D. S., & Dybå, T. (2019). Research Synthesis in Software Engineering: A Tertiary Study. Information and Software Technology, 101.

da Silva, F. Q. B., Santos, A. L. M., & Soares, S. (2019). Snowballing Literature Review in Software Engineering: A Systematic Mapping Study. Information and Software Technology, 104.

de Oliveira Neto, F. M., Pimenta, M. S., & Tedesco, P. C. (2020). How to Improve Software Quality Through Design Patterns: A Review-Based Framework. Journal of Systems and Software, 163.

Dybå, T., Dingsøyr, T., & Moe, N. B. (2021). The Role of Evidence in Software Engineering: Systematic Reviews and Beyond. IEEE Transactions on Software Engineering.

García-Mireles, G. A., Rodríguez, D., & Piattini, M. (2020). Teaching Design Patterns in Software Engineering Courses: A Review. ACM Transactions on Computing Education.

Kaur, R., Singh, H., & Singh, J. (2022). A Systematic Review on Software Design Patterns: Classification and Impact on Software Quality. Computer Science Review, 43.

Khomyakov, A., & Borodin, A. (2022). Strategic Software Architecture Using Design Patterns: A Case-Based Perspective. Software Architecture Review.

Kitchenham, B., & Charters, S. (2007). Guidelines for Performing Systematic Literature Reviews in Software Engineering. EBSE Technical Report.

Martínez-Fernández, S., Ayala, C., & Franch, X. (2021). Software Development Trends and Practices: Insights from Industry. IEEE Software, 38(4).

Mourão, D., et al. (2021). Overuse of Design Patterns: An Empirical Investigation. Journal of Software: Evolution and Process, 33(6).

Nugroho, A., et al. (2021). Patterns for Efficient Team Collaboration in Software Engineering. International Journal of Advanced Computer Science and Applications.

Pereira, R., Silva, J., & Goulão, M. (2020). Evaluating the Usefulness of Design Patterns: A Controlled Experiment. Journal of Software Engineering Research and Development.

Wahyudin, D., et al. (2023). Recent Trends in Design Pattern Application and Software Modularity: A Meta-Analysis. Software Engineering Journal.

Zhang, X., & Sun, Y. (2023). Enhancing Object-Oriented Software Design with Pattern-Based Refactoring. Journal of Systems and Software, 200.

Zhao, Y., et al. (2022). Design Pattern Use and Code Smell Reduction: An Empirical Study. Empirical Software Engineering, 27(1).