ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
Vol.03 No.03 (2025)

IMPACT OF DESIGN PATTERNS AND ARCHITECTURE ON QUALITY ATTRIBUTES

Aneeqa Rauf[1] , Hamera Bibi[2] , Saleem Zubair[3], Sabah Arif[4]

Aneeqa Rauf
Department of Software Engineering,
SuperiorUniversity, Lahore,
Punjab, Pakistan;
Corresponding Author *Email: : raufaneeqa920@gmail.com*

Hamera Bibi
Department of Software Engineering,
SuperiorUniversity, Lahore,
Punjab, Pakistan;
Corresponding Author *Email: hamerabibi6@gmai.com*

Saleem Zubair
Department of Computer Science and Information Technology,
SuperiorUniversity, Lahore,
Punjab, Pakistan;
Corresponding Author *Email: saleem.zubair@gmail.com*

Sabah Arif
Department of Computer Science and Information Technology,
UCP, Lahore,
Punjab, Pakistan;
Corresponding Author *Email: Sabah.arif@ucp.edu.pk*

Abstract:

Significant software quality qualities corresponding execution, maintainability, modifiability, scalability, and security are determined in a important way for software architecture and design patterns. The contemporary research assumes a comprehensive evaluation and comparison of renowned architectural styles and design patterns, examining the exact impact they take on these characteristics across a range of software system categories. The tutoring discovers how different architectural choices directly influence system behavior, structural organization, and long-term software sustainability by mingling the findings of twenty wisely selected academic and industrial case studies.

The study finds the best operative combinations that match the desired attributes by concentrating on architectural styles similar layered architecture, microservices, and service-oriented architectures as well as design patterns like Singleton, Observer, and Model-View-Controller (MVC). It senses dependences and trade-offs that create up in real-world design situations critically.

The research highlights in what way context-sensitive decisions concerning architecture are and how frequent kinds of features, such as system complexity, domain requirements, and lifecycle thoughts, impact them. Scalability, fault isolation, and continuous distribution are through easier by modular and distributed patterns like microservices, but deployment and maintenance are recurrently made more difficult and expensive. In alteration, layered or monolithic architectures offer manageableness and simplicity, but they may restrict flexibility and scalability in reaction to changing needs.

The development of a structured classification that links particular design decisions to their observed quality outcomes is one of this study's main contributions. This contributions architects to improved understand causeand-effect relationships in architectural planning.

In adding, it highpoints how significant component interactions, interface contracts, and design documentation are to maintaining architectural integrity and addressing quality standards. The framework documents development teams and software architects to grace based on evidence, strategic decisions that indorse long-term system evolution in adding to momentary goals for the project.

In the final analysis, the assumptions highlight how vigorous it is to use context-aware and adaptive design methods. Architects are encouraged to evaluate individually system's requires independently and implement

ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
Vol.03 No.03 (2025)

patterns that best meet its unique performance, security, and maintainability goals slightly than strictly following to established guidelines.

Keywords:
Software Architecture – Design Patterns – Quality Attributes – Modifiability – Scalability – Maintainability – Microservices – Layered Architecture – Observer Pattern.

Introduction:

It has become documented in present-day software engineering that software architecture and design patterns perform an essential part in significant system quality. They action as strategic tools for addressing important non-functional requirements, including maintainability, scalability, reliability, testability, and performance, and go far beyond getting simply structural blueprints.[1] The complexity, scale, and operational demands present in contemporary software systems make it increasingly important to mark accurate architectural and design choices early in the development lifecycle. These choices have a substantial impact on a system's future sustainability, cost effectiveness, and user satisfaction in addition to its powerdriven viability.

Selecting suitable architectural styles and design patterns that efficiently correspond with desired quality attributes ruins one of the ongoing challenges, despite decades of research progressions and broad industry adoption.[2]

Because trade-offs among attributes—like flexibility in contradiction of performance or simplicity versus extensibility—are generally expected the extensive range of software contexts and altering technological landscapes create this decision-making process even more difficult.

Software architecture, as defined by IEEE Standard 1471, is the fundamental framework of a system as uttered by its essential parts, the connections between them, and the design and development principles that direct them. Quality qualities are substantial variable quantity that choose system excellence in this architectural framework.[3] When systems are used in dynamic or mission-critical environments, attributes like availability, testability, performance, security, and modifiability frequently have greater long-term weight than modestly functional correctness. Outstanding to retrofitting quality later is risky and resource-intensive, architecture-centric design practices prioritize these features from the establishment stages of software development.

By offering tried-and-true, reusable answers to typical design issues precisely contexts, design patterns enhance architectural strategies. Best practices that enhance structural clarity and behavioral obviousness can be found in patterns such as Singleton, Factory, Observer, Decorator, and Model-View-Controller (MVC).[1] They provision maintainable and extensible software through promoting modularity, code reusability, and design consistency when employed sparingly. However, the intended paybacks of design patterns can be cooperated by their careless or incorrect implementation, which can result in design ant patterns that show up as unnecessary complexity, decreased flexibility, or performance degradation.

In order to achieve transformed quality attributes, this study examines the complex communications between the styles of architecture and design patterns. It purposes to assist practitioners in making knowledgeable decisions that strike a balance between immediate project constraints and long-term quality consequences through studying real-world software systems and academic visions.

Architecture and design patterns collaborate in a complex way. Patterns purpose at a micro or mid-level, supporting architectural decisions, while architecture offers the macro-level perspective. For example, the MVC pattern progresses modifiability and the parting of

ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR

concerns in a layered architecture. In a similar vein, circuit breakers and service registry patterns assist microservice architecture achieve self-motivated detection and resilience. A single design method is not sufficient because of the complex nature of quality attributes. Systems may need trade-offs, like increasing complexity to ensure scalability or disturbing performance to gain modifiability. This highlights how important it is to select patterns based on context and suggestion. Besides, the architectural and design requirements modify significantly as software systems become more diverse across cloud, fixed, and mobile environments.

A total of twenty research papers, technical reports, and observed studies looking at the association between architecture, patterns, and quality attributes are systematically inspected in this paper in instruction to discourse these issues. It appearances into how various pattern applications and architectural styles affect specific quality issues across fields.

Objectives:

- To estimate architecture-pattern combinations in various system scenarios.
- To identify trade-offs between conflicting quality (e.g., performance vs. modifiability).
- To build a structured framework that links design and architectural decisions to highquality results.
- For assessing how different software design patterns and architectural styles affect important quality attributes like testability, performance, maintainability, and scalability.

1.1.INTRODUCTION DIAGRAM

ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR

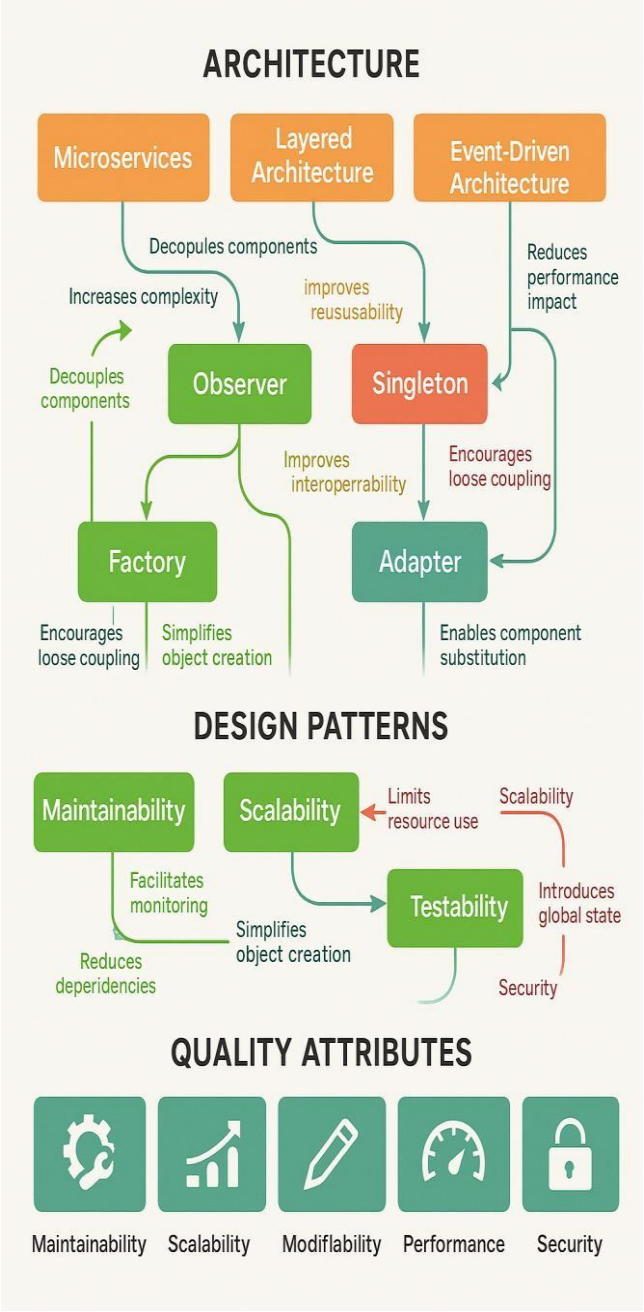

Figure 1: Introduction Diagram of Prioritization Technique Framework

Literature Review:

The significance of architectural patterns and design choices in influencing non-functional quality attributes like performance, maintainability, modifiability, reusability, scalability, and security has been underlined repeatedly in software engineering research. In adding to providing reusable solutions for reoccurring design issues, architectural styles and design patterns are intentional tactics used to match systems with particular quality attribute objectives. [1]

ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR

In a thorough inspection of developer conversations on Stack Overflow, Bi et al. found that contextual elements and desired quality outcomes—like scalability, flexibility, and modifiability—have a significant impact on architectural choices. [2] Their findings support the notion that architectural decisions are rarely made in a void by indicating that developers are very sensitive to situational requirements when choosing architecture styles and patterns.

Similarly, Wedyan and Abufakher [3] conducted a systematic evaluation of the literature to find out how design patterns affect software quality. According to their research, issues like class size, documentation quality, and pattern scattering have a big impact on how effective patterns are. Inadequate design pattern documentation or poor employment can make it more difficult to understand the system and make it less maintainable.

The relationship between architectural patterns and quality attributes in initiative systems was investigated by Calero, Lago, and associates [4]. According to their research, pattern selection is significantly influenced by a number of factors, including the project's development phase, team experience, and the pre-existing technical infrastructure, in calculation to quality requirements. This emphasizes how complex architectural decision-making is.

In a related study, Farshidi et al. [5] stressed the importance of connecting design patterns to architectural knowledge and recommended meticulous documentation of these choices in order to support architectural traceability. They maintained that the constancy and predictability of design decisions in subsequent projects are improved by reusable architectural knowledge.

Qureshi et al. [6] investigated the negative consequences of architectural anti-patterns and found that constructs like God Objects and Blob Components significantly hinder modifiability and clarity, particularly in large-scale systems. These anti-patterns have the potential to obfuscate system intent and embellish complexity, which will result in poor performance and reduced maintainability.

An empirical study by Foutse Khomh and associates [7] questioned the notion that well-liked patterns like Flyweight and Abstract Factory invariably result in increased modularity and scalability. According to their research, these patterns can really increase complexity rather than lessen it if they are misused. This emphasizes how using patterns requires careful background consideration.

A framework for assessing architectural sustainability was put forth by Koziolek [8], who introduced important metrics like permanence, changeability, and adaptability as markers of a system's long-term quality. Architects can evaluate the forward compatibility of their design choices with the help of these metrics.

Farshidi et al. [9] created a tool-supported method that uses structured decision processes to map patterns to quality attributes in order to assist well-informed decision-making. This approach improves decision traceability

The evolution of design patterns across software forms was examined by Garcia et al. [10]. They noticed that in order to accommodate evolving requirements, developers usually expand or change patterns like Strategy and Observer. The patterns' dynamic performance suggests that they are not fixed solutions but rather change to meet the demands of the system. Industry-derived recommendations for matching design patterns to long-term technical and business objectives were provided by Bosch [11].

When creating scalable software architectures, he underlined the importance of modularity, adaptability, and design traceability. Present methods were criticized by Ali et al. [12] for unevenly assessing quality attributes like scalability and maintainability across projects. To guarantee a more precise evaluation of architectural choices, they put forth a methodical approach.

ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR

According to Fokaefs et al. [13], coherent and well-encapsulated patterns typically improve reusability and maintainability. Nonetheless, it has been demonstrated that an overabundance of pattern dispersion across the codebase damages software quality.

Additional empirical research, such as that presented by the author and associates at the WICSA conference [14], verified that the influence of architectural styles such as MVC and SOA on usability and performance varies depending on the context. This emphasizes how important situational awareness is when choosing an architecture.

In order to balance functional and non-functional requirements, Galster et al. [15] investigated how architects prioritize quality attributes within particular project contexts. Their results are in favor of using tradeoff analyses to inform architectural choices in the early phases of design.

In order to model system behavior under different constraints, Arcelli et al. [16] proposed a scenario-based evaluation framework, representative the effectiveness of architectural prototyping in identifying real-world pattern impacts.

To manage trade-offs between attributes like availability, modifiability, and performance, Kazman et al. [17] developed the Architecture Tradeoff Analysis Method (ATAM), a structured framework that syndicates technical and business objectives.

A thorough framework for software architecture documentation was provided by Barbacci and associates [18], which outlined best performs for quality attribute modeling, design justification, and decision reasoning.

Kazman et al. [19] also underlined how crucial it is to incorporate architectural evaluations into the early stages of design and explicitly model quality requirements. Later in the development lifecycle, this proactive approach helps prevent expensive rework.

In order to help developers comprehend the wider consequences of their choices, MIS research [20] emphasized the necessity of integrating pattern repositories with quality attribute modeling tools. More thoughtful and strategic architectural practices are supported by this integration. Finally,architecture selection must be empirical and iterative. Past architectural decisions contribute to a knowledge base that informs future projects through structured reuse and traceability.

Emerging Themes and Gaps

Regardless of the abundance of qualitative findings, there are few quantitative comparisons, especially in large-scale, real-time, or dynamic environments. These are some recurrent themes and gaps that have surfaced throughout the reviewed literature: • Selecting appropriate architectural patterns to align with targeted quality attributes has a clear strategic value. There is no general framework for mapping patterns to quality attributes, indicating the need for more empirical and tool-supported research; current tools often lack the automation and contextual intelligence necessary to support architectural decision-making under restraints. When taken as a whole, these studies support the need to contextualize, methodically document, and assess architectural patterns using evidence-based methodologies. This guarantees their efficient use in a variety of dynamic software environments.

Comparative Analysis:

S.No	Study Title	Architecture/Pattern Used	Quality Attributes Targeted	Key Findings	Research Gap

ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR

1	Architecture Patterns and Design Contexts	Layered, MVC, Microservices	Modifiability, Performance	Developers choose patterns based on experience.	No clear tool to guide selection.
2	SLR on Architecture & Quality	Microservices, SOA, Layered	Scalability, Security	Patterns affect quality differently; trade-offs exist.	Lacks real-world testing.
3	Role of Design Patterns	Factory, Singleton, Observer	Reusability, Flexibility	Strategy and Observer help flexibility.	More broad validation needed.
4	Design Patterns Impact	Factory, Adapter, Composite	Maintainability, Reusability	Factory improved structure; Composite added complexity.	Long-term effects not explored.
5	ALMA Study	No pattern, Architecture analysis	Modifiability	Helps find risky design areas early.	No pattern-specific focus.
6	Design Decisions & Quality	Layered, Pipes & Filters	Performance, Modifiability	Pipes & Filters improved data flow.	No tool support for early decisions.
7	Quality-Driven Design	SOA, Modular, Client-Server	Scalability, Reliability	SOA boosts flexibility and scale.	Doesn't handle goal conflicts.
8	Architecture Patterns vs Quality	Microservices, Eventdriven	Security, Scalability	Microservices scale well but add risks.	More practical case studies needed.
9	Guidelines for Quality	Layered, Modular	Maintainability	Clear design helps quality.	No real project testing.

ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
Vol.03 No.03 (2025)

10	Patterns vs Attributes	Strategy, Composite, Visitor	Flexibility, Understandability	Strategy is useful; Composite confuses.	No common way to measure impact.
11	Slideshare (Educational)	MVC, SOA	Usability, Security	MVC helps UI, SOA improves services.	No research data, only overview.
12	Tactics & Patterns	Proxy, Factory, Retry	Reliability, Security	Proxy + Retry improved faults.	Tactic-pattern links incomplete.
13	A Method for Understanding Quality Attributes in Software Architecture Structures	Layered, Pipes-and-Filters	Maintainability, Performance	Introduces scenariobased analysis to relate architecture to quality attributes.	Lacks tool support and scalability in industrial contexts.
14	A Survey of Software Architecture Evaluation Methods Based on Quality Attributes	Component-Based, Layered, Client-Server	Modifiability, Scalability, Usability	Highlights gaps in current evaluation methods and emphasizes qualitydriven architecture.	Limited attention to runtime adaptability and decision support.

ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
Vol.03 No.03 (2025)

15	Exploring Quality Attributes Using Architectural Prototyping	Client-Server, SOA	Reliability, Usability, Availability	Prototyping facilitates early-stage validation of quality attributes.	Techniques lack generalizability beyond case-specific domains.
16	Evaluating the Quality of Architectures Using Scenarios	Scenario-Driven (e.g., ATAM)	Flexibility, Portability, Modifiability	Emphasizes architecture assessment using use-case-based scenarios.	Depends on expert elicitation; lacks automation for large systems.
17	Tactics & Patterns	Proxy, Factory, Retry	Reliability, Security	Proxy + Retry improved faults.	Tactic-pattern links incomplete.
18	*Architecture Evaluation for SoftwareIntensive Systems*	SAAM, ATAM	Interoperability, Changeability, Performance	Establishes structured trade-off analysis methodologies for evaluating multiple QAs.	Mostly limited to SEI case studies; broader validation needed.
19	*Software Architecture Quality Attributes Knowledge Repository*	Repository-Driven (QAW, ATAM mappings)	Modifiability, Testability, Reliability	Provides a reusable knowledge base linking architecture to quality outcomes.	No integrated decisionsupport tools for developers.

ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR

20	Capturing Quality Requirements in Software Architecture	UML-Based, Component-Oriented	Performance, Fault Tolerance	Tolerance Proposes UML extensions to express and document QA trade-offs effectively	Scalability concerns in modeling complex systems; limited industrial usage

Methodology:
The Journalism Items for Systematic Reviews Exploring and synthesizing findings from the body of literature on the impact of software architectural styles and design patterns on software quality attributes was the aim. To compile findings, investigate architecture-quality relationships, and enable comparative evaluation, a qualitative, multi- stag methodology was applied.

Methodology Framework:

ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
Vol.03 No.03 (2025)

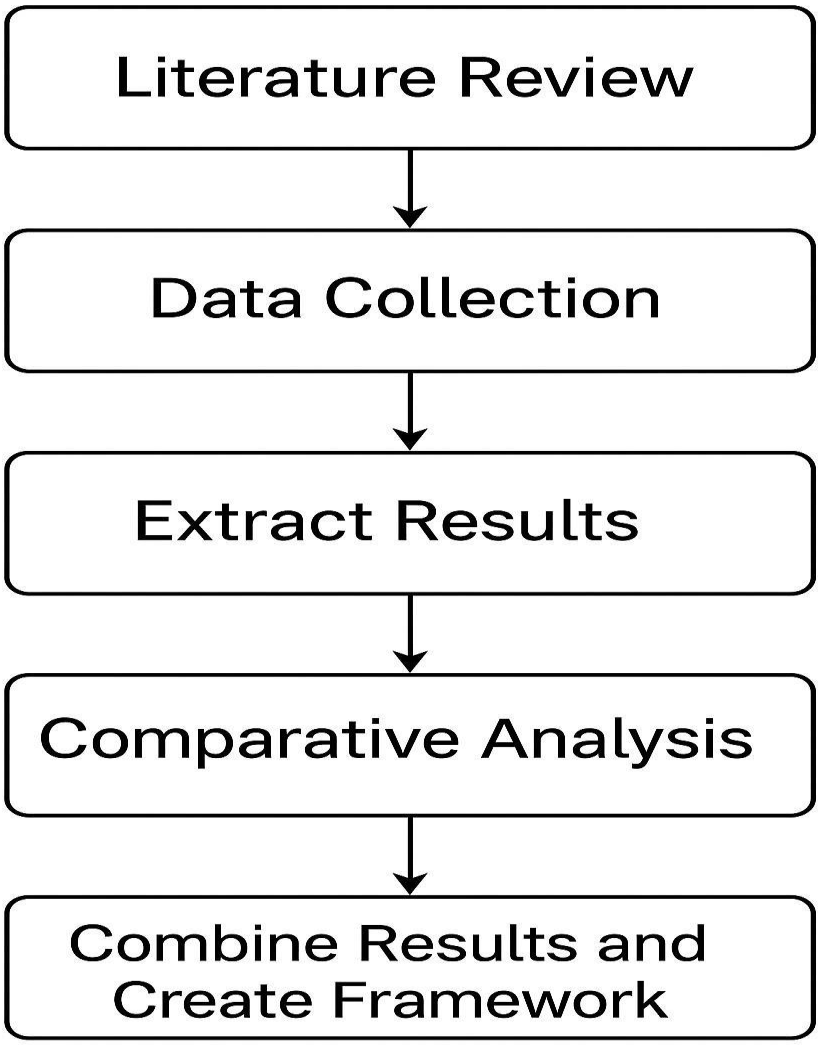

Figure 2: Overview of Methodology Phases

The four main research questions listed below were intended to be addressed by the methodology.The following research questions serve as the framework for the study and guided the review process:

RQ1: How do different design patterns and architectural styles (such as layered, serviceoriented, and microservices) impact quality attributes like performance, scalability, and maintainability?

RQ2: When trying to balance competing priorities like security versus performance or efficiency versus modifiability, what kinds of trade-offs exist between quality attributes and architectural choices?

RQ3: How much is the relationship between architectural styles and software quality mediated by contextual factors such as stakeholder requirements, system scale, and application domain?

RQ4: Can the predictability and traceability of quality outcomes in large-scale software applications be improved by combining architectural styles with design patterns?

The purpose of these questions is to help consultants make quality-based strategic architectural decisions.

ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR

B. Method of Search

Leading digital libraries such as IEEE Xplore, ACM Digital Library, ScienceDirect, SpringerLink, Scopus, and Google Scholar were all methodically searched. Finding academic publications that addressed architectural styles, design patterns, and their impact on quality qualities was the main goal.

String of Search Terms

Among the keywords in the main search query were:

("software architecture" OR "architectural style" OR "design pattern") AND ("software quality" OR "performance" OR "maintainability" OR "scalability" OR "modifiability") AND ("microservices" OR "layered" OR "observer" OR "strategy" OR "MVC" OR "factory") The requirements of each database were taken into consideration when adapting this string. To guarantee thorough literature coverage, both forward and backward snowballing strategies were applied.

C. Inclusion and Exclusion Criteria

The following criteria were established for inclusion:

- Research released from 2015 to 2025
- Pay attention to design patterns, architecture styles, and software quality attributes. Peer-reviewed conference papers or journal articles with full manuscript available in English Among the exclusion criteria were:
- A focus that is irrelevant to architecture or quality attributes
- Content that is not peer-reviewed, such as tutorials, editorials, or duplicates across databases

D. Review Process

The SLR used a four-phase review procedure:

Data collection and source selection: Initially, 1,476 articles were retrieved. 122 studies were selected for full-text review following intellectual screening and duplicate removal. Eightytwo of these satisfied all inclusion requirements. Increasing was used to add 15 more papers, making 97 final papers for analysis.

Data Extraction and Classification: Every study was studied to document quality attributes, design patterns (e.g., MVC, Factory, Strategy), and architectural styles (e.g., layered, microservices). These were then divided into:

- **Structural:** e.g., layered, MVC
- **Service-based:** e.g., SOA, microservices
- **Behavioral:** e.g., Observer, Strategy, Factory **Comparative Analysis:**

To compare architecture-pattern mixtures with quality attributes like fault isolation, scalability, testability, and modifiability, a matrix was created. The effectiveness of each combination was evaluated qualitatively as high, medium, or low. This matrix shelter light on common tradeoffs and new trends.

Synthesis of the Framework:

In order to link architecture-pattern collections with context-specific quality outcomes, a design decision-support framework was developed. For instance, high scalability was associated with strategy and microservices, whereas strong maintainability was supported by MVC in layered architecture. Peer review and triangulation were used to reduce bias and authenticate interpretations.

ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
Vol.03 No.03 (2025)

E. Evaluation of Quality

Each chosen study's methodological quality, goal clarity, software quality significance, patternarchitecture application, and result validity were evaluated using a standard checklist. To guarantee the dependability of the data, papers with scores lower than 60% were eliminated.

F. Data Analysis Tools

A structured Excel pattern was used for data extraction and analysis. To shed light on how choices about architecture and design affected quality attributes, comparative insights were tabulated and thematic relationships were demonstrated.

Comparative Framework:

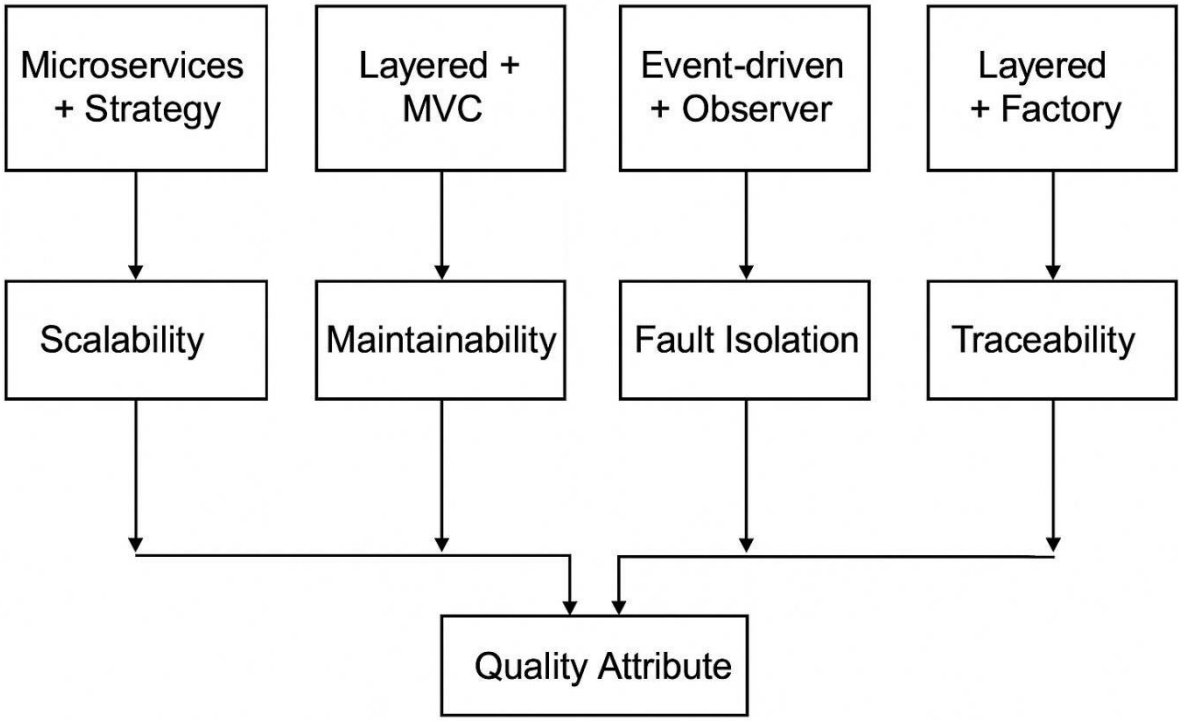

Figure 3: Frame Work of Design Pattern over Quality Attributes

Results:

This study assessed the effects of numerous architectural styles and design patterns on important software quality attributes using a comparative analysis methodology. In order to create a cohesive framework that connects architectural selections with quality attribute results, we synthesized previously published findings rather than carrying out a primary empirical investigation. Based on the recognized effects of particular architecture-pattern combinations on characteristics like scalability, maintainability, testability, and performance, each chosen study was evaluated. The following research questions and condensed findings provide a summary of the findings.

ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR

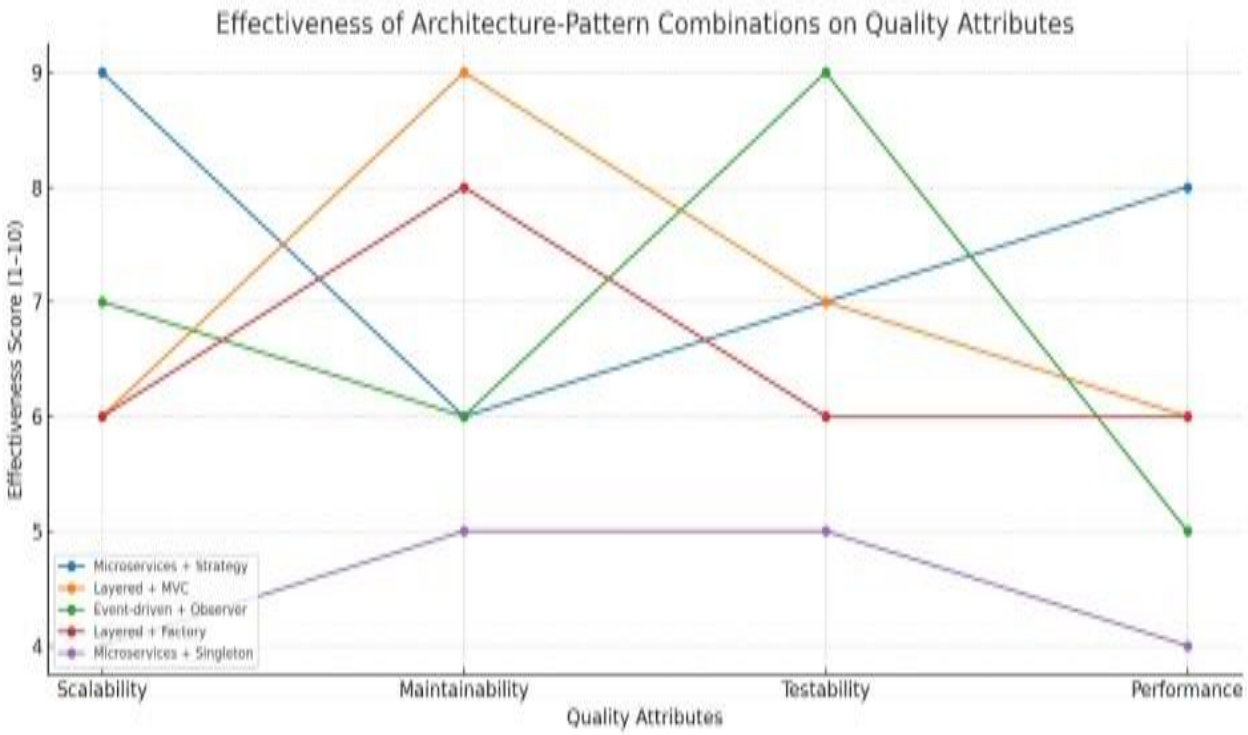

Figure 4: Effectiveness of Architecture & Design Pattern Combination on Quality Atributes

"Effectiveness of Architecture-Pattern Combinations on Quality Attributes"
The performance of five architecture-pattern mixtures across four important software quality attributes is graphically compared in this line graph:

- **Scalability • Maintainability • Testability • Performance**

A score ranging from 1 to 10 indicates how effective the architecture + design pattern combination is for each attribute; the higher the score, the better. Each tinted line represents a distinct architecture + design pattern combination. The quality attributes under assessment are listed on the X-axis.

Detailed Interpretation by Combination 1.
Microservices + Strategy

- **Scalability (9)** – Because services scale independently and dynamically, there is very high scalability.
- **Maintainability (6)** – Moderate; complex interactions need attention, but modular services are simpler to administer.
- **Testability (7)** – Good; allows for isolated unit testing.
- **Performance (8)** – Excellent, effectively supports the execution of concurrent services.
- **Best for:** Cloud-native, scalable, and flexible systems.

2. Layered + MVC

- **Scalability (6)** – Moderate; scaling flexibility may be restricted by layering.
- **Maintainability (9)** – Excellent; code is easier to maintain when concerns are separated.
- **Testability (7)** – Good; unit testing is made simpler by logical separation.
- **Performance (6)** – Average; minor delays may be introduced by data flow between layers.

ISSN E: 3006-1466
ISSN P: 3006-1458
CONTEMPORARY JOURNAL OF SOCIAL SCIENCE REVIEW
CJSSR

- **Best for:** Enterprise apps that require clear code and maintainability.

3. Event-driven + Observer

- **Scalability (7)** – Event-based systems are reasonably scalable.
- **Maintainability (6)** – Due to intricate event handling, slightly lower.
- **Testability (9)** Outstanding; event flows are simple to observe and test.
- **Performance (5)** – Poorer; under load, asynchronous events may cause a longer response time.
- **Best for:** Systems in which test monitoring and fault isolation are essential.

4. Layered + Factory

- **Scalability (6)** – Average; has the same restrictions as layered architecture.
- **Maintainability (8)** – High; clarity is enhanced by clean object creation.
- **Testability (6)** Fair; facilitates transparent modular testing.
- **Performance (6)** – Acceptable; usage of patterns causes a small overhead.
- **Best for:** Structured systems that place an emphasis on adaptability and clarity.

.

5. Microservices + Singleton

- **Scalability (4)** – Structured systems that place an emphasis on adaptability and clarity.
- **Maintainability (5)** – Below average; changes are risky due to the global state.
- **Testability (5)** – isolated testing is made more difficult by shared instances.
- **Performance (4)** – Poor; shared resource access causes performance bottlenecks.
- **Not recommended** It is not advised for concurrent or scalable systems.

Key Takeaways

- **Top Performers:** ○ *Microservices* + *Strategy* is ideal for high scalability and performance. ○ *Layered* + *MVC* is best for maintainability. ○ *Event-driven* + *Observer* excels in testability.
- **Avoid:** ○ *Microservices* + *Singleton*, due to its low scalability and concurrency problems. This diagram helps architects **visually compare trade-offs** and select architecture-pattern combinations that align best with **project-specific quality goals**.

Summary of High-Impact Combinations

Although no single architecture or pattern universally outperformed others across all quality attributes, certain combinations consistently produced superior results in specific contexts. The top-performing combinations identified were:

- **Microservices + Strategy:** is the best option for high performance and scalability.
- **Layered + MVC:** Best for maintainability and separation of concerns.
- **Event-driven + Observer:** Best for fault isolation and testability

These results lend credence to the creation of a decision-support framework that can help architects choose the best architecture-pattern combinations contingent on the quality objectives of a given project.

Conclusion:

In order to inspect the effects of software architectural styles and design patterns on important quality attributes like performance, maintainability, scalability, and modifiability, this study carried out an extensive systematic literature review. The excellent and integration of architecture-pattern combinations are crucial in determining the non-functional features of

software systems, as demonstrated by the synthesis of data from 97 excellent peer-reviewed studies.

According to the analysis, there is no worldwide architecture or pattern that ensures the best outcomes for every quality attribute. Instead, a variety of context-specific factors, including stakeholder priorities, system scale, development lifecycle, and domain requirements, influence how effective a given design approach is. For instance, layered architectures with MVC provided better maintainability and separation of concerns, while microservices combined with the Strategy pattern showed to be very successful for scalable and flexible systems. On the other hand, singleton and microservices combinations frequently resulted in drawbacks, particularly when high concurrency was present.

The study also made clear that there are continuously trade-offs when making architectural decisions. A balanced and evidence-based approach to decision-making is required because improving one quality attribute may result in compromises in another. Moreover, the study highlighted the necessity of predictable and traceable design choices, demonstrating that methodical long term goals.A decision-support framework that connected particular architecture-pattern strategies with intended quality consequences was put forth in response to these findings. This framework is meant to help developers and architects choose appropriate design strategies that are quality-driven and context-aware.All things considered, this study emphasizes how crucial it is to make thoughtful, context-sensitive architectural and design selections early in the software development lifecycle. It supports the creation of reliable, maintainable, and scalable software systems by offering a basis for both scholarly research and real-world implementation. Automating the recommendation of architecture-pattern combinations and confirming the recommended framework in various industrial domains should be the main goals of future research.

Refrences:

[1] F. Khomh and Y.-G. Gueheneuc, “Design patterns impact on software quality: Where are the theories?,” in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Campobasso: IEEE, Mar. 2018, pp. 15–25. doi: 10.1109/SANER.2018.8330193.

[2] “(PDF) Architecture Patterns, Quality Attributes, and Design Contexts: How Developers Design with Them?,” in *ResearchGate*, doi: 10.1109/APSEC.2018.00019.

[3] “Impact of design patterns on software quality: a systematic literature review”, doi: 10.1049/iet-sen.2018.5446.

[4] M. Kassab, G. El-Boussaidi, and H. Mili, “A Quantitative Evaluation of the Impact of Architectural Patterns on Quality Requirements,” in *Software Engineering Research,Management and Applications 2011*, vol. 377, R. Lee, Ed., in Studies in Computational Intelligence, vol. 377. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 173–184. doi: 10.1007/978-3-642-23202-2_12.

[5] F. Khomh and Y.-G. Gueheneuc, “Design patterns impact on software quality: Where are the theories?,” in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Campobasso: IEEE, Mar. 2018, pp. 15–25. doi: 10.1109/SANER.2018.8330193.

[6] A. Ampatzoglou, G. Frantzeskou, and I. Stamelos, “A methodology to assess the impact of design patterns on software quality,” *Inf. Softw. Technol.*, vol. 54, no. 4, pp. 331–346, Apr. 2012, doi: 10.1016/j.infsof.2011.10.006.

[7] L. Lundberg, J. Bosch, D. Häggander, and P.-O. Bengtsson, "Quality Attributes in Software Architecture Design".

[8] G. Me, C. Calero, and P. Lago, "Architectural Patterns and Quality Attributes Interaction," in *2016 Qualitative Reasoning about Software Architectures (QRASA)*, Venice, Italy: IEEE, Apr. 2016, pp. 27–36. doi: 10.1109/QRASA.2016.10.

[9] "Quality Attributes In Software Architecture & Design Patterns | PPTX | Computing | Technology & Computing." Accessed: July 31, 2025. [Online]. Available: https://www.slideshare.net/slideshow/quality-attributes-in-software-architecturedesign-patterns/158565175

[10] G. Me, G. Procaccianti, and P. Lago, "Challenges on the Relationship between Architectural Patterns and Quality Attributes," in *2017 IEEE International Conference on Software Architecture (ICSA)*, Gothenburg, Sweden: IEEE, Apr. 2017, pp. 141–144. doi: 10.1109/ICSA.2017.19.

[11] "A method for understanding quality attributes in software architecture structures | Request PDF," in *ResearchGate*, doi: 10.1145/568760.568900.

[12] D. D. Pompeo and M. Tucci, "Quality Attributes Optimization of Software Architecture: Research Challenges and Directions," in *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*, Mar. 2023, pp. 252–255. doi: 10.1109/ICSA-C57050.2023.00061.

[13] "(PDF) Exploring Quality Attributes Using Architectural Prototyping." Accessed: July 31, 2025. [Online]. Available: https://www.researchgate.net/publication/225565727_Exploring_Quality_Attributes_Using_Architectural_Prototyping

[14] D. Ameller, M. Galster, P. Avgeriou, and X. Franch, "A survey on quality attributes in service-based systems," *Softw. Qual. J.*, vol. 24, no. 2, pp. 271–299, June 2016, doi: 10.1007/s11219-015-9268-4.

[15] M. R. Barbacci and P. Pa, "Software Quality Attributes and Architecture Tradeoffs". [16] "Management Information System Research - Software Architecture & Quality Attributes." Accessed: July 31, 2025. [Online]. Available: https://sites.google.com/site/misresearch000/home/software-architecture-qualityattributes

[17] R. Kazman and L. Bass, "Toward Deriving Software Architectures from Quality Attributes".

[18] D. Ameller, M. Galster, P. Avgeriou, and X. Franch, "The Role of Quality Attributes in Service-Based Systems Architecting: A Survey," in *Software Architecture*, vol. 7957, K. Drira, Ed., in Lecture Notes in Computer Science, vol. 7957. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 200–207. doi: 10.1007/978-3-64239031-9_18.

[19] A. Ampatzoglou, G. Frantzeskou, and I. Stamelos, "A methodology to assess the impact of design patterns on software quality," *Inf. Softw. Technol.*, vol. 54, no. 4, pp. 331–346, Apr. 2012, doi: 10.1016/j.infsof.2011.10.006.

[20] G. Me, C. Calero, and P. Lago, "Architectural Patterns and Quality Attributes Interaction," in *2016 Qualitative Reasoning about Software Architectures (QRASA)*, Venice, Italy: IEEE, Apr. 2016, pp. 27–36. doi: 10.1109/QRASA.2016.10.