

Systematic Analysis of Factors Influencing Modulith Architecture Adoption over Microservices

Chandra Prakash
University of the Cumberland, USA
cprakash@outlook.com
ORCID 0009-0006-6425-7560

Sunil Arora
Dakota State University, USA
Sunil.Arora@trojans.dsu.edu
ORCID 0009-0007-3066-3461

Abstract—As organizations increasingly shift away from traditional monolithic architectures, the requirements for scalable, flexible, and maintainable software systems have led to microservice architecture adoption. However, microservices’ complexity and operational overhead have presented significant challenges, particularly in managing distributed systems, inter-service communication, and deployment processes. In response, modulith architecture has emerged as a middle-ground approach, offering the benefits of modularity and scalability while mitigating some of the drawbacks of monolithic and microservices architectures.

This paper comprehensively reviews the factors influencing the adoption of modulith architecture over microservices. The study identifies key drivers such as dependency management, scalability, deployment efficiency, and system availability through a detailed analysis of existing literature, case studies, and expert opinions. The review reveals that modulith architecture offers a simpler, more maintainable solution that preserves modularity without the complexity of fully distributed systems. The findings offer critical insights for software architects and organizations considering architectural transitions, positioning modulith as a viable alternative in scenarios where microservices may introduce unnecessary complexity. This research contributes to the ongoing discourse on software architecture by providing a nuanced understanding of the trade-offs involved in adopting modulith architecture. It proposes a set of considerations for organizations navigating the evolving architectural landscape.

Index Terms—Modular Monolith, Modulith, Microservices, Software architecture, Modular architecture

I. INTRODUCTION

SOFTWARE architecture has been marked by significant shifts in design paradigms due to the emergence of microservices architectures. Traditionally, monolithic architecture has been the default choice for building applications, characterized by a unified codebase where all components are tightly coupled and deployed as a single unit. While monoliths offer simplicity in development and deployment, they result in scalability, maintainability, and agility challenges as applications grow and complexity increases. These limitations have led many organizations to explore alternative architectures, particularly microservices.

Microservices architecture emphasizes decomposition to break applications into smaller, independent services that communicate over the network [1]. In the microservices architecture, each service is designed to perform a specific business function and can be developed, deployed, and scaled

independently. This approach offers significant benefits, including improved scalability, flexibility, and the ability to adopt different technologies within the same application [1]. However, microservices also introduce new challenges, such as increased operational complexity, inter-service communication overhead, and the need for advanced DevOps practices to manage distributed systems effectively [2]. By consolidating dependencies and internal communication, modulith architectures enable smoother CI/CD pipelines and lower operational overhead. Modulith allows DevOps teams to implement automation and monitoring more effectively, maintain agility, reduce deployment risks, and support the scalable evolution of systems without the high infrastructure costs.

While monolith and microservices are two extremes of software architecture, modulith architecture has emerged as a promising middle ground. A modulith retains the simplicity of a monolithic deployment while focusing on internal modularization. Moduliths provide many organizational and developmental benefits of microservices without the overhead of managing distributed services by clearly separating concerns and establishing well-defined boundaries within the codebase. This approach allows organizations to adopt modular design practices within a single deployable unit, offering a scalable path toward eventual microservices migration if needed. Despite the growing interest in moduliths, there remains a lack of understanding of the factors influencing their adoption. This research paper strives to fill this gap by synthesizing existing literature on modulith architecture and comparing it with monolithic and microservices approaches. By examining the various factors associated with modulith adoption, this research paper provides a comprehensive overview of the decision-making factors driving modulith architecture adoption over microservices architecture.

II. LITERATURE REVIEW

A. Monolith Architecture

The monolithic architecture pattern is one of the oldest and most widely used approaches in software engineering, still favored by many organizations today. In monolithic architecture, the entire application and its components are combined into a single codebase and deployed as a single unit [1]. This architecture offers several advantages, including simplicity in development, testing, deployment, debugging,

and maintenance [2], [3]. Communication within a monolithic application is handled internally via inter-process communication, contributing to its ease of maintenance and making it a recommended choice for initial development phases [1].

However, as applications grow in size and complexity, the inherent limitations of the architecture often outweigh these benefits. Large and complex monolithic systems make code changes difficult and increase the risk of unintended impacts on other parts of the application [4]. Additionally, the tightly coupled nature of monoliths can hinder effective workforce utilization, as maintaining the system often requires comprehensive knowledge of the entire application [1].

B. *Microservices Architecture*

Microservices architectural patterns have gained significant recognition and widespread adoption over the last decade [5]. Microservices are defined as an approach to developing applications composed of small, independent services that run in their own process space and communicate over lightweight protocols such as HTTP [1], [5]. The core principle behind microservices is to decompose an application's functions into business contexts, grouping functions that address common business domain problems. Microservices architecture adheres to several key principles of software engineering: (a) Single Responsibility—each service is responsible for one specific function, ensuring no two services overlap in functionality; (b) Autonomy—microservices are autonomous units that are independently deployable; and (c) Service-First Approach—services expose APIs to consumers, abstracting internal implementation details [1].

Microservices offer numerous benefits, such as decomposing complex functionality into modular, domain-centric components, allowing for easier development, change impact identification, deployment, and scalability. The independence of each service enables fault tolerance; if one service fails, the rest of the application can continue to function [1], [5]. However, despite these advantages, microservices also present several challenges. Their distributed nature often leads to increased communication overhead between services, raising infrastructure consumption. The proliferation of microservices can introduce complexities in deployment, monitoring, and management [5]. Furthermore, data management poses significant challenges; When multiple microservices require access to the same data, this can lead to data duplication and consistency issues [1] [5].

C. *Modulith Architecture*

In response to the challenges associated with microservice architecture, a new approach known as modulith architecture has emerged, blending the principles of monolithic and microservices architectures. This hybrid architecture aims to balance the benefits and drawbacks of both approaches, combining their strengths while addressing inherent challenges. Modulith architecture promotes the creation of modular components with loose coupling and high cohesion, encapsulating

components based on distinct business domains. Unlike microservices, these modular components are deployed within a single hosting environment, simplifying deployment and reducing operational complexity [6]. Communication between components occurs through well-defined API interfaces but remains confined within the same application and database, minimizing latency and avoiding the complexities of distributed resources.

The adoption of modulith architecture has gained traction after being successfully implemented by industry leaders such as Shopify and Amazon, which reported significant performance and cost benefits. The architecture has demonstrated improved scalability and availability, garnering further recognition from other major players, including Google with its Service Weaver project, and the SpringOne framework, both of which support building modular monolith applications that achieve microservices-like benefits.

Despite the growing industry acceptance of the challenges inherent in monolithic and microservices architectures and modulith's potential to address them, academic literature on the subject remains limited. Influential industry experts, including [2] [3], have advocated for the modular monolith or modulith approach as a preferable starting point for architectural decomposition before moving to full microservices. This study draws on the limited available literature, industry case studies, and insights from reputable experts to build a comprehensive argument for modulith architecture. It focuses on the key factors influencing software architecture decisions—such as dependency management, deployment, hosting, scalability, and availability—and examines how modulith architecture addresses these concerns. This research paper aims to demonstrate that modulith can be an effective solution for organizations transitioning from monolithic architectures, offering a balanced approach that mitigates the complexities of microservices while enhancing performance and maintainability.

III. RESEARCH QUESTION

The study has identified the research questions below based on fundamental factors behind software architecture decision-making and how those factors offer the benefits of both worlds via modulith architecture.

RQ1: What are the key factors influencing the adoption of modulith over microservices architecture?

RQ2: How does modulith architecture address the performance and complexity challenges typically associated with microservices architecture?

RQ3: What are the scalability and availability implications of adopting modulith architecture in large-scale systems compared to microservices?

IV. METHODOLOGY

To ensure transparency and objectivity while reporting recent progress in the literature review on Modulith architecture, this systematic literature review has utilized the methodology

presented by Tranfield et al. [7]. The objectives of this systematic literature review are: first, to identify the proposals that compared the architectural benefits of modulith and microservices architecture; second, to identify the impact of those benefits on performance, scalability, and maintainability.

This study follows the Preferred Reporting Items for Systematic Reviews and Meta-Analysis (PRISMA). PRISMA model allows for a comprehensive and transparent approach to systematic reviews and inclusion of relevant studies. Table I highlights the inclusion and exclusion criteria for the paper selection. Fig. 1 describes the PRISMA approach for selecting system literature reviews.

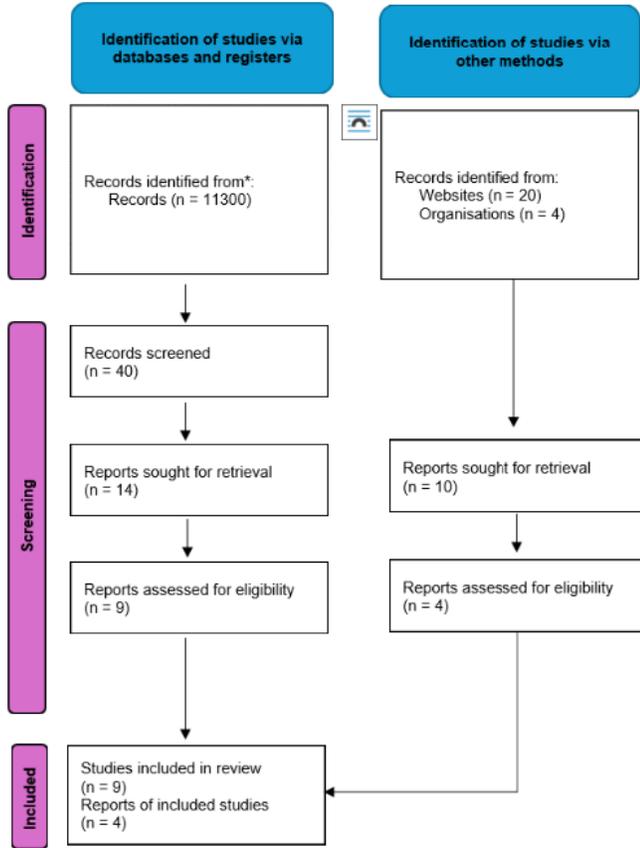


Fig. 1. PRISMA Flow Chart. Adoption from Page et al. [8]

TABLE I
INCLUSION AND EXCLUSION CRITERIA

Inclusion Criteria	Exclusion Criteria
Papers published in the last 18 months	Papers unrelated to the research question.
Peer-reviewed papers published in journals or presented in conference papers.	Paper written in languages other than English
Papers discussing key adoption factors of modulith architecture	Papers unrelated to the architectural decision-making process
Case studies discussed in the last 5 years by the organization on their technology blogs	Case studies discussed through blog posts other than implementing organization.

A. Search strategy

To answer the research questions, the search strategy was used to identify the relevant studies addressing the key factors influencing the modulith architectural design. The search terms used included “modulith architecture,” “microservices vs. modulith,” “modulith architecture benefits,” “modulith adoption factors,” and “modular monolith” with the combination of and/or criteria to search for better results. The search criteria were applied to Google Scholar and the IEEE website to search for relevant articles, and articles published in the last 18 months were selected.

B. Data Extraction and analysis

Identifying and obtaining relevant articles based on study questions was part of the data extraction and analysis procedure. The characteristics of these studies were then extracted: author(s), publication year, and research technique. Key findings and methodological details were also collected. Throughout the data analysis phase, the retrieved data was classified into themes or subjects related to the study questions.

C. Validity

The study has performed a literature review to answer the identified research questions. While answering the research questions, it is equally important to improve the validity and depth of the review paper. Shaheen et al. [9] have recommended using multiple databases, screening articles, and corroborating the findings with other well-established sources. This study utilized two other methods: 1) case studies directly from the organizations that have implemented modulith architecture and 2) expert opinion from the leading industry experts.

V. RESULT AND DISCUSSION

The results and discussion section provides an in-depth exploration of the findings related to the research questions posed in the study. It is organized into three distinct parts: a) Review: In the Review section, we examine the literature review, summarizing the key discoveries and trends that emerge from previous research. We identify the essential factors influencing the successful adoption of modulith architecture, shedding light on what motivates organizations to transition to this model. b) Case Study: The case study section comprehensively examines the real-world application of modulith architecture within well-respected organizations. We analyze specific case studies to illustrate how these organizations have successfully implemented the architecture and the tangible benefits they have experienced as a result, such as enhanced performance, operational efficiency, and cost savings. c) Expert Opinion: Here, we gather insights from industry experts who share their advice on the process of moving from a traditional monolithic architecture to a more flexible modular architecture. This section discusses the practical considerations and strategic guidance these experts provide, emphasizing how modulith architecture serves as a compelling solution to common challenges, including performance issues, increased complexity, scalability constraints, and availability concerns. Overall, this

section aims to create a comprehensive understanding of how modulith architecture can be effectively adopted and the broader implications for organizations in today’s fast-paced technological landscape.

A. Review

Based on the literature review, four different factors have emerged as decision-making factors for software architecture [1], [4], [10]–[14], [22] have advocated for dependency management. Several authors [4], [10], [11], [13], [14], [23] have demonstrated how centralized deployment management is critical to decision-making. Several studies provided scalability as the foundation for architecture to scale resources at critical times [4], [10], [11], [13], [14]. Software availability and reliability have been identified as another factor that drives the organization’s ability to serve their consumer and play an essential part in reputation management [4], [10], [11], [13], [14], [23], [24].

TABLE II
REVIEW PAPERS

Feature	Authors
Dependency	Faustino, Gonçalves, Portela, & Rito Silva (2024) [4]; Johnson, Kharel, Mannamplackal, Abdelfattah, & Cerny (2024) [11]; Lopes & Silva (2023) [12]; Su & Li (2024) [14]; Su, Li, & Taibi (2023) [13]; Tsecheliadis, Nikolaidis, Maikantis, & Ampatzoglou (2023); M. Felisberto (2024) [15]
Deployment	Barde (2023) [10]; Faustino, Gonçalves, Portela, & Rito Silva (2024) [4]; Johnson, Kharel, Mannamplackal, Abdelfattah, & Cerny, 2024 [11]; Su & Li (2024) [14]; Su, Li, & Taibi (2023); Olariu (2023) [13]
Scalability	Barde (2023) [10]; Faustino, Gonçalves, Portela, & Rito Silva (2024) [4]; Johnson, Kharel, Mannamplackal, Abdelfattah, & Cerny, 2024 [11]; Su & Li (2024) [14]; Su, Li, & Taibi (2023) [13]
Availability	Barde (2023) [10]; Faustino, Gonçalves, Portela, & Rito Silva (2024) [4]; Johnson, Kharel, Mannamplackal, Abdelfattah, & Cerny, 2024 [11]; Su & Li (2024) [14]; Su, Li, & Taibi (2023) [13]; Olariu (2023) [23]; Øvrelid, Moseng, & LO Vinje (2023) [24]

1) *Dependency*: Monolithic software architectures have introduced numerous challenges, with dependency management being a significant concern. In monoliths, shared domain models often increase component coupling, reducing system flexibility and potentially causing cascading failures when changes are not properly anticipated. Lopes and Silva [12] have argued that while shared domain implementations in monoliths simplify development, they also introduce scalability challenges. This increased dependency can lead to inefficiencies in development teams, resulting in slower development cycles, longer time-to-market, and stifled innovation.

Although microservice architecture offers increased flexibility, Lopes and Silva [12] caution that improper decomposition based on business domains can introduce additional complexities through distributed communication, potentially

outweighing the benefits. It is important to note that dependency challenges are not unique to monoliths; microservices can also suffer from poorly defined interfaces, leading to chatty traffic and increased performance overhead [4].

In light of these challenges, modular monolith (or “modulith”) architecture has emerged as a viable alternative. Faustino et al. [4] recommend modulith as a practical approach for agile software development while maintaining system performance. Modulith architecture reduces latency through in-process module interactions, avoiding the overhead associated with inter-service communication in microservices.

To address the complexities of system decomposition and module identification, Lopes and Silva [12] propose a model incorporating various features, including dependency analysis, to guide the creation of modular systems. Barde [10] advocates for modular monoliths as a means to effectively utilize domain models, reusing domain entities to reduce dependencies and increase agility. This approach aligns closely with fundamental software engineering principles.

A qualitative study by Tsecheliadis et al. [15] highlights modulith architecture as a better-integrated yet logically isolated approach, yielding significantly improved results compared to microservice architecture. Johnson et al. [11] further support this view, arguing that modulith architecture enables more effective dependency management compared to microservices. While both monolithic and microservice architectures present dependency challenges, modulith architecture offers a balanced approach. It combines the benefits of monolithic simplicity with improved modularity, potentially addressing many of the dependency-related issues faced in software development today.

2) *Deployment*: Deployment strategies significantly differ across architectural paradigms, each presenting unique challenges and benefits. In monolithic systems, while tight component coupling facilitates efficient deployment, the single-process nature necessitates frequent full-system deployments, increasing risk with each iteration [12]. Conversely, microservices allow for targeted deployments of specific services but introduce complexities in deployment processes, versioning, maintenance of versioned components, and ensuring compatibility across services [4].

Modulith architecture emerges as a balanced approach, allowing organizations to deploy a single artifact, thus simplifying the deployment process compared to the multiple independently deployable units in microservices [4], [12]. Johnson et al. [11] argue that modulith architecture provides superior holistic system visibility compared to other models, facilitating easier monitoring and management of the application as a single unit, thereby reducing the complexity of tracking multiple services and their interactions.

The adoption of modern DevOps practices has been crucial in accelerating software delivery and reducing variability through automation. While microservices architecture has demonstrated effective use of DevOps practices for deployment, becoming a cornerstone of the approach, Barde [10] notes that these practices can be equally beneficial when

applied to modular monolith systems, enhancing deployment flexibility.

Testing and validation during deployment are critical for ensuring software quality. Modulith architecture simplifies and streamlines these processes during releases [10]. Furthermore, cyclomatic complexity, a key measure of software quality, can be more effectively applied to modulith architecture to assess stability and build confidence in the software—a level of measurement that may not be as readily achievable in pure monolithic or microservices architectures [10].

Modulith architecture offers several additional advantages in the deployment context. The first is simplified infrastructure, which reduces the need for complex setups like service meshes or orchestrators often required by microservices. The second is consistent environments, which allow modules to promote consistency across different environments, mitigating issues arising from deploying multiple services across varied settings. The third is efficient resource utilization, where resources are managed within a single runtime, eliminating the need for separate containers or VMs for each service and leading to more efficient use of computing resources.

Modulith architecture presents a compelling middle ground for deployment strategies, combining the simplicity of monolithic deployments with some of the flexibility microservices offer. This approach and modern DevOps practices can lead to more efficient, reliable, and manageable deployment processes in complex software systems.

3) *Scalability*: Scalability remains a critical challenge for monolithic systems and serves as a primary motivator for the adoption of microservices architecture. Monoliths, characterized by their single-process nature, face inherent limitations in scalability, often requiring vertical scaling of both storage and processing resources. This approach can quickly become cost-prohibitive and technically challenging as system demands grow. Faustino et al. [4] emphasize scalability as a key driver for organizations transitioning away from monolithic architectures. In contrast, microservices offer a more flexible approach to scalability through decomposition. This architectural style allows individual components to be scaled independently based on their specific demands, leading to more efficient resource utilization and improved overall system performance.

However, the transition to microservices is not without its challenges. Lopes and Silva [12] highlight that improper identification of service boundaries during decomposition can introduce new scalability issues in microservices architectures. This underscores the importance of careful system design and domain-driven decomposition when adopting microservices. Recent research by Faustino et al. [4] acknowledged these challenges and proposed the concept of “modulith” as a viable architectural alternative. A modulith aims to leverage the benefits of modular design while minimizing the complexities associated with inter-service communication in fully distributed microservices systems. This approach can serve as a middle ground, offering improved scalability over traditional monoliths while reducing the operational overhead of managing numerous independent services.

Barde [10] further explores this concept, highlighting how modular monoliths can provide a pathway for gradual migration when designed with appropriate frameworks such as Google’s Service Weaver. This approach allows organizations to design systems that can initially operate as a cohesive unit but have the flexibility to be split and hosted as separate microservices as scalability needs evolve. Additionally, recent studies have shown that the choice between monoliths and microservices is not always binary. Abgaz et al. [16] proposed a “microservice-first” approach, where systems are initially designed with a microservices design approach but deployed as a monolith. This strategy allows for easier future decomposition while avoiding premature optimization and the complexities of distributed systems management.

While microservices offer significant advantages in terms of scalability, organizations must carefully consider their specific needs, technical capabilities, and growth projections when choosing an architectural style. The emergence of hybrid approaches like moduliths and microservice-first designs provides a spectrum of options for balancing scalability, maintainability, and operational complexity.

4) *Availability*: System availability is crucial for organizations to maintain continuous service to their consumers. Key factors contributing to high availability include data consistency, asynchronous behavior, and enhanced performance. Recent research has shed light on how different architectural approaches address these concerns. Faustino et al. [4] highlight that modulith architecture offers several advantages regarding availability and data consistency: the first is improved data consistency by maintaining the data model in a single location. This centralized approach reduces the complexities associated with distributed data management in microservices architectures. Second, modulith architecture minimizes over-the-network remote calls, enhancing data consistency and maintaining asynchronous behavior. This reduced network communication can lead to improved system responsiveness and reduced latency. Third is balanced asynchronous behavior while preserving the benefits of asynchronous operations. Modulith architecture avoids the extreme decoupling that can sometimes lead to data inconsistencies in microservices systems.

Modulith architecture enables system-wide availability through the use of specialized tools to provide a comprehensive view of the entire system’s health and performance. This granular monitoring capability helps identify and address potential issues, ensuring system availability. Domain-specific tracking can be used to enforce quality metrics at the component level. Holistic monitoring ensures that each part of the system meets predefined performance, reliability, and availability standards.

Barde [10] notes that these levels of measurement and enforcement may not be entirely possible in traditional monolithic architectures due to their lack of domain-centric design. Similarly, achieving comprehensive system-wide visibility in highly distributed microservices implementations can be challenging, making it difficult to consistently enforce and measure

quality metrics across all services.

Monoliths often struggle with granular monitoring and domain-specific optimizations. Their tightly coupled nature can make it difficult to isolate and address performance issues that affect availability. Whereas microservices offer high flexibility, microservices can introduce complexities in maintaining data consistency across distributed services. The distributed nature can also make it challenging to get a holistic view of system health and availability. The emergence of moduliths strikes a balance by offering domain-centric design with the ability to monitor and optimize at a modular level while maintaining a unified system view for comprehensive availability management. While each architectural style has its strengths, modulith architecture offers a compelling balance of data consistency, performance, and availability. Its ability to support comprehensive monitoring and quality enforcement makes it an attractive option for organizations seeking to optimize their system's availability without sacrificing the benefits of modular design.

B. Case Study

1) *Shopify*: Shopify operates a Software as a Service (SaaS) platform that provides an e-commerce solution for merchants, enabling them to sell their products through a subscription model. In addition to its core e-commerce platform, Shopify offers various merchant solutions, including payment processing, shipping, and point-of-sale (POS) services. As of 2023, Shopify supports 5.23 million online stores and boasts 2.1 million daily active users, with a gross merchandise volume of \$235 billion and a total revenue of \$7.1 billion [17].

Shopify initially adopted a monolithic architecture without clear boundaries within the codebase. Over time, as changes accumulated, the inherent nature of monolithic architecture led to increased software complexity and high coupling between different processes. Although monolithic systems are advantageous for their simplicity, rapid implementation, and straightforward deployment pipelines, they also present scalability and maintenance challenges as applications grow. Regarding microservices, Westeinde [18] argues that this approach is not universally applicable, as microservices can introduce complexities related to distributed communication and component isolation. Similar findings by Prakash [5] suggest that inadequate domain knowledge and poorly defined service boundaries in microservices can lead to excessive inter-service communication, resulting in performance bottlenecks. While the superior scalability of microservices can mitigate some performance issues, the associated challenges remain significant.

In analyzing microservices adoption, Westeinde [18] identifies data access across distributed services as a critical challenge, often involving cross-network communication, latency issues, reliability concerns, and increased change management overhead. In response, Shopify adopted a modulith architecture, which enforces boundaries between components, aiming to strike a balance between the benefits of monoliths and microservices. This approach involved reorganizing

the codebase according to business domains and isolating dependencies between these domains [18]. Recognizing the difficulty in managing these boundaries, Shopify developed a specialized tool to track dependency isolation [18]. According to Müller [19], the modulith architecture enabled Shopify to streamline onboarding, accelerate testing, and simplify feature implementation.

Analyzing Shopify's modulith implementation offers valuable insights into this architectural approach's practical benefits and trade-offs. Following are the insights identified from Shopify's case study. A) *Dependency*- Shopify had a dense cyclical dependency between the components with their monolith system, and adopting modulith helped reduce the circular dependencies, use of inversion of the control mechanism, and efficient management of dependencies between the components. B) *Deployment*- Shopify still uses the single deployment unit with modulith. However, modulith architecture lays the path of component extraction into standalone applications. The organization can also extract certain functions into separate services for better management. C) *Scalability*- After adopting modulith architecture, Shopify utilized horizontal and vertical scaling. Shopify uses vertical scaling within the modulith components, and horizontal scaling is used for extracted components. D) *Availability*- Modulith architecture has helped Shopify achieve better code organization with a modular structure. It has reduced the change impact risk on other parts of the system. Selected extracted components can be scaled for higher availability without scaling the entire system.

Shopify's modulith implementation highlights the benefits of a well-structured monolith compared to microservices. Scalability and maintainability are identified as two major benefits of reducing the complexities of distributed systems. Shopify's modulith adoption addressed the shortcomings of monolith systems while maintaining the benefits of a monolith and avoiding the complexity of microservices architecture.

2) *Amazon Video*: In a case study of Amazon Prime Video, the organization shifted from a distributed microservices architecture to a modulith architecture to enhance performance and reduce costs [20]. This case challenges the common belief that microservices inherently lead to better scalability and performance. Other researchers, such as Fowler [2], Newman [3], and Prakash [5], have highlighted the importance of a use-case-based transition to microservices rather than building them from scratch. Amazon Prime Video, one of the world's largest streaming services with 163 million viewers in the USA and over 335 million globally [21], initially employed a microservices-based architecture for its Video Quality Analysis (VQA) system. The VQA system utilized AWS Step Functions, Lambda, and S3 to host separate components, including media converters, defect detectors, and an orchestration service [20].

However, Amazon Video faced significant challenges with VQA's microservices architecture, particularly in scaling its infrastructure, as the distributed communication between components via AWS Step Functions failed to meet performance expectations, handling only 5% of the expected load capacity.

This architecture also led to increased data transfer and higher operational costs [20]. Amazon re-architected the VQA system to address these issues, consolidating distributed microservices into a single monolithic process with modular components. Internal communication was streamlined using an orchestrator, and infrastructure was shifted to Amazon EC2 and ECS, optimizing data transfer to an efficient in-memory model.

Evaluating Amazon’s VQA across five parameters—dependency, deployment, scalability, availability, and performance—revealed the following:

- **Dependency**—The initial microservices implementation had high interdependencies and relied heavily on external services for data flow and orchestration. The modulith implementation reduced external dependencies, resulting in cost savings and improved performance.
- **Deployment**—Deployment complexity in the microservices architecture stemmed from the numerous components and services. The modulith approach allowed VQA to deploy as a single unit on EC2 and ECS, simplifying the deployment process and enhancing control over various components.
- **Scalability**—Although microservices are generally praised for their scalability, VQA’s microservices setup created bottlenecks, particularly with AWS Step Functions, which capped scaling at 5% of the expected load.
- **Availability**—While microservices theoretically enhance availability through distributed components, VQA’s reliance on multiple services reduced overall availability. By consolidating into a modulith, VQA reduced infrastructure dependencies, minimized points of failure, simplified monitoring, and improved overall availability.

Amazon’s VQA case study highlights the specific challenges of microservices, particularly in managing distributed transactions, and demonstrates how the shift to modulith architecture significantly improved scalability and performance while reducing costs by 90%. The consolidation of components into a single process eliminated costly data transfers, and simplified orchestration allowed for more effective resource utilization and management. Although microservices excel at horizontal scaling, Amazon’s VQA achieved superior scalability through vertical service cloning, showcasing how modulith architecture can outperform microservices in specific contexts.

C. Expert Opinion

This study synthesized expert guidance from leading industry figures to validate the findings on modulith architecture. Notably, the insights of Martin Fowler and Sam Newman—two of the most respected voices in software architecture—underscore the potential of moduliths or similar approaches. Fowler [2] argues that starting with a modulith is a prudent choice for new applications, cautioning that premature adoption of microservices can introduce unnecessary complexity when system boundaries and requirements are not well-defined. Similarly, Newman [3] acknowledges the advantages of microservices but advises that they are not suitable for every

project, particularly in the initial stages or when organizational maturity does not align with the demands of a microservices architecture.

Both Fowler [2] and Newman [3] provide compelling arguments for adopting a modular monolith (modulith) architecture, especially during the early stages of a project or when an organization lacks the readiness to manage the complexities associated with microservices. Their collective insights suggest that moduliths offer an effective starting architecture, providing simplicity, ease of management, and a more straightforward path to scalability as the system evolves. Furthermore, moduliths mitigate organizational readiness risks by maintaining a centralized, manageable code structure, which helps avoid the premature complexity inherent in microservices. This expert advice supports the strategic adoption of moduliths as a robust and flexible foundation, making them a pragmatic choice for many organizations.

VI. RESULTS

Our review of software architectures reveals a nuanced landscape where traditional monoliths and microservices each present distinct challenges. Monolithic architectures, while offering simplicity in deployment and development, often struggle with scalability, high coupling, and difficulties maintaining and evolving the system over time. On the other hand, microservices, touted for their scalability and flexibility, introduce their own set of complications, particularly in terms of performance, system complexity, and inter-service communication overhead.

A key finding from our review is the critical importance of domain-driven decomposition in microservices architectures. Without this, organizations risk creating distributed monoliths that inherit the drawbacks of the monolith and microservices architectural paradigms without realizing their respective benefits. In this context, modulith architecture emerges as a promising middle ground. It offers a balanced approach that aims to combine the strengths of monolithic and microservices architectures while mitigating their individual weaknesses. Moduliths are particularly advantageous for smaller teams, legacy system modernization, and environments where microservices’ complexity and operational overhead are prohibitive. In a fast-changing world, green field microservices may not yield the business value desired by organizations, and the study recommends taking a more pragmatic approach to modularizing the legacy system through modulith architecture design.

A. Implications for Practice

The findings from our comprehensive review, case studies, and expert opinion provide valuable insights for practitioners and researchers engaged in the software architecture field. Organizations must take the time to thoroughly evaluate their unique use cases, scaling requirements, and their teams’ skills and competencies when determining the most suitable architecture—monolithic, microservices, or modulith.

Particularly noteworthy is the modulith approach, which merits serious consideration as a practical alternative. This architecture is especially beneficial for systems that demand a level of modularity while not necessarily needing the complexities of complete distribution or the implications of distributed services. Organizations can enhance their operational efficiency and adaptability by selecting the architecture that is appropriately suitable while meeting their specific business needs.

B. Gaps and Future Research

The study primarily relied on the systematic literature review to understand the motivation behind modulith adoption. However, due to the limited availability of prior research, much of the literature review and case study have relied on isolated examples from specific industries and lack a generalization across other business domains.

Future studies can benefit from investigations related to performance, long-term scalability, and maintenance of modulith systems. A quantitative analysis of performance, scalability, and maintainability comparisons between monoliths, microservices, and moduliths across various applications and scales is recommended. Future Longitudinal studies tracking the evolution of systems implemented as moduliths would be beneficial in understanding how well this architecture supports system growth and adaptation over time.

VII. CONCLUSION

This study used a systematic review to analyze the factors influencing the adoption of modulith architecture. Our research findings indicate that key factors driving modulith adoption include simplified dependency management, deployment, and availability. In addressing performance and complexity challenges, modulith architecture demonstrates its strength by reducing the overhead often associated with inter-service communication and cohesive management of shared resources within a unified codebase. The discussed findings contribute valuable insights into the software architectural decision-making process and recommend modulith architecture as a viable alternative to address the complexity of distributed communications of services. This study highlights the importance of aligning architectural choices with organizational needs, the technical maturity of the team, and system requirements over microservices as a default choice. Future studies are encouraged to investigate the longitudinal impact of modulith adoption on operational costs, developer productivity, and long-term scalability as system requirements evolve.

ACKNOWLEDGMENT

Grammarly was utilized in the writing process to assist with editing, spell-checking, and grammar enhancement.

REFERENCES

- [1] G. Blinowski, A. Ojdowska, and A. Przybyłek, "Monolithic vs. Microservice Architecture: A performance and scalability evaluation," *IEEE Access*, vol. 10, pp. 20357–20374, 2022. doi:10.1109/access.2022.3152803
- [2] M. Fowler, "Monolith First," [martinfowler.com, https://martinfowler.com/bliki/MonolithFirst.html](https://martinfowler.com/bliki/MonolithFirst.html).
- [3] S. Newman, "Microservices for Greenfield?" <https://samnewman.io/blog/2015/04/07/microservices-for-greenfield/>.
- [4] D. Faustino, N. Gonçalves, M. Portela, and A. Rito Silva, "Stepwise migration of a monolith to a microservice architecture: Performance and migration effort evaluation," *Performance Evaluation*, vol. 164, p. 102411, May 2024. doi:10.1016/j.peva.2024.102411
- [5] C. Prakash, "Zero-Trust Architecture Approach to Secure Microservices for the Healthcare Insurance Industry," dissertation, ProQuest, 2024
- [6] M. AIT SAID, A. EZZATI, S. MIHL, and L. BELOUADDANE, "Microservices adoption: An industrial inquiry into factors influencing decisions and implementation strategies," *International Journal of Computing and Digital Systems*, vol. 15, no. 1, pp. 1417–1432, Mar. 2024. doi:10.12785/ijcads/1501100
- [7] D. Tranfield, D. Denyer, and P. Smart, "Towards a methodology for developing evidence-informed management knowledge by means of systematic review," *British Journal of Management*, vol. 14, no. 3, pp. 207–222, Sep. 2003. doi:10.1111/1467-8551.00375
- [8] M. J. Page et al., "The Prisma 2020 statement: An updated guideline for reporting systematic reviews," *BMJ*, Mar. 2021. doi:10.1136/bmj.n71
- [9] N. Shaheen et al., "Appraising systematic reviews: A comprehensive guide to ensuring validity and reliability," *Frontiers in Research Metrics and Analytics*, vol. 8, Dec. 2023. doi:10.3389/frma.2023.1268045
- [10] K. Barde, "Modular Monoliths: Revolutionizing Software Architecture for efficient payment systems in Fintech," *International Journal of Computer Trends and Technology*, vol. 71, no. 10, pp. 20–27, Oct. 2023. doi:10.14445/22312803/ijctt-v71i10p103
- [11] J. Johnson, S. Kharel, A. Mannamplackal, A. Abdelfattah, and T. Cerny, "Service weaver: A promising direction for cloud-native systems?," *Proceedings of the 14th International Conference on Cloud Computing and Services Science*, 2024. doi:10.5220/0012624500003711
- [12] T. Lopes and A. R. Silva, "Monolith microservices identification: Towards an extensible multiple strategy tool," *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*, pp. 111–115, 2023. doi:10.1109/icsa-c57050.2023.00034
- [13] R. Su, X. Li, and D. Taibi, *Back to the Future: From Microservice to Monolith*, 2023. doi:10.48550/arXiv.2308.15281
- [14] R. Su and X. Li, "Modular Monolith: Is this the trend in software architecture?," *Proceedings of the 1st International Workshop on New Trends in Software Architecture*, vol. 169, pp. 10–13, Apr. 2024. doi:10.1145/3643657.3643911
- [15] M. Tsehelidis, N. Nikolaidis, T. Maikantis, and A. Ampatzoglou, "Modular monoliths the way to standardization," *Proceedings of the 3rd Eclipse Security, AI, Architecture and Modelling Conference on Cloud to Edge Continuum*, Oct. 2023. doi:10.1145/3624486.3624506
- [16] Y. Abgaz et al., "Decomposition of monolith applications into microservices architectures: A systematic review," *IEEE Transactions on Software Engineering*, vol. 49, no. 8, pp. 4213–4242, Aug. 2023. doi:10.1109/tse.2023.3287297
- [17] "Shopify Announces Fourth-Quarter and Full-Year 2023 Financial Results," Document, <https://www.sec.gov/Archives/edgar/data/1594805/000159480524000006/exhibit991press>
- [18] K. Westeinde, "Deconstructing the monolith," [Shopify, https://shopify.engineering/deconstructing-monolith-designing-software-maximizes-developer-productivity](https://shopify.engineering/deconstructing-monolith-designing-software-maximizes-developer-productivity).
- [19] P. Müller, "Under deconstruction: The state of shopify's monolith," [Shopify, https://shopify.engineering/shopify-monolith](https://shopify.engineering/shopify-monolith).
- [20] M. Kolny, "Scaling up the Prime Video audio/video monitoring service and reducing costs by 90%," *Prime Video Tech*, <https://www.primevideotech.com/video-streaming/scaling-up-the-prime-video-audio-video-monitoring-service-and-reducing-costs-by-90>.
- [21] J. Stoll, "Global netflix and prime video viewership 2024," *Statista*, <https://www.statista.com/statistics/1449359/netflix-amazon-prime-video-viewers-worldwide-by-country/>.
- [22] M. Felisberto, "The trade-offs between Monolithic vs. Distributed Architectures," *arXiv preprint arXiv:2405.03619*, May. 2024. doi:10.48550/arXiv.2405.03619
- [23] F. Olariu, "Overcoming Challenges in Migrating Modular Monolith from On-Premises to AWS Cloud." *2023 22nd RoEduNet Conference: Networking in Education and Research (RoEduNet)*. IEEE, 2023.
- [24] E. Øvrelid, OH Moseng, and LO Vinje. "Operational Backbone Work: Modernization Activities in the Migration of Monolith-Oriented IT Architectures." *NIKT: Norsk IKT-konferanse for forskning og utdanning*. Bidsys Open Journal Systems, 2023.