Modulith vs. Microservices: A Cost and Performance Analysis for Scalable Enterprise Architectures

Abhisek Sharma

myemail.abhi@gmail.com

Abstract:

Enterprise software architecture continues to evolve as organizations increasingly demand systems that are not only scalable but also maintainable, cost-efficient, and responsive to rapid changes. Among the architectural paradigms vying for adoption in large-scale enterprise systems, Moduliths and Microservices stand out as two of the most influential. Moduliths, or modular monoliths, offer a refined architectural pattern that encapsulates modular design within a single deployable unit, aiming to strike a balance between simplicity and maintainability. Microservices, on the other hand, embrace decentralized design by decomposing applications into a suite of small, independently deployable services that communicate via lightweight protocols, offering high flexibility and horizontal scalability. This paper undertakes a detailed cost and performance analysis of these two architectures in the context of scalable enterprise applications. It examines how each paradigm performs across key factors, including infrastructure cost, latency, deployment overhead, operational complexity, team autonomy, failure isolation, and the ability to scale both technically and organizationally. The analysis draws on empirical benchmarks using prototype implementations of identical business functionalities in both architectural forms, deployed across cloud-native environments using modern container orchestration and CI/CD practices. The evaluation framework includes metrics such as request latency, throughput under load, operational cost (including compute, networking, and monitoring), time-to-market, and maintenance effort over a simulated product lifecycle.

In addition to quantitative measures, this paper integrates qualitative assessments based on team collaboration dynamics, domain-driven design alignment, compliance adaptability, and boundary enforcement challenges. The study reveals that Moduliths offer lower initial development and maintenance costs, reduced cognitive overhead for development teams, and simplified observability, making them a pragmatic choice for medium-scale applications with well-defined boundaries and cohesive teams. However, as system scale and domain complexity increases, Microservices demonstrate their strengths in enabling parallel development, fine-grained scalability, and fault tolerance, albeit with increased overhead in areas such as inter-service communication, transaction management, and distributed monitoring.

The findings emphasize that neither Moduliths nor Microservices represents a universally superior solution. Instead, the choice between them depends heavily on organizational context, team maturity, domain complexity, and operational scalability requirements. To assist architects and technology leaders in making informed decisions, the paper proposes a structured decision matrix that weighs technical, financial, and human factors. By comparing trade-offs and performance across real-world enterprise architecture scenarios, this study provides actionable insights that transcend architectural dogma and address the practical realities of building and maintaining enterprise-grade systems.

Ultimately, this research contributes to the ongoing discourse on architectural strategies by offering an evidence-based, context-sensitive perspective. It serves as a valuable guide for organizations seeking to modernize legacy systems, launch scalable platforms, or reassess their current architecture with a focus on balancing cost efficiency with operational excellence.

Keywords: Modulith, Microservices, Enterprise Architecture, Scalability, Performance Benchmarking, Total Cost of Ownership, DevOps, Modular Monolith, Distributed Systems, Software Architecture Decision-Making.

I. INTRODUCTION

The increasing need for scalable, robust, and maintainable enterprise applications has refocused attention on new architectural paradigms that can handle the complexities of current software systems. The growing availability of cloud computing, agile development, and DevOps has compelled companies to reconsider traditional, monolithic approaches. As such, the architectural discourse between Moduliths and Microservices has become and continues to be a significant issue in enterprise software engineering. These two methods offer distinct approaches to arranging, distributing, and evolving software systems, each with trade-offs in terms of cost, performance, maintainability, and scalability.

Moduliths (short for modular monoliths) are an architectural approach in which a single deployable unit is the primary component, but it features strong internal modularity. Unlike monolithic applications, which are often plagued by tight coupling and brittle, hard-to-maintain code, Moduliths prescribe clear lines between modules to ensure a clean separation of concerns, increased code reuse, and improved testability. They are especially appealing to those seeking architectural simplicity in combination with a moderate degree of modularity. Moduliths also promote centralized logging, simple deployment pipeline scenarios, and a low cognitive load for development teams, which means they work well for environments with stable team setups and clear-cut business domains.

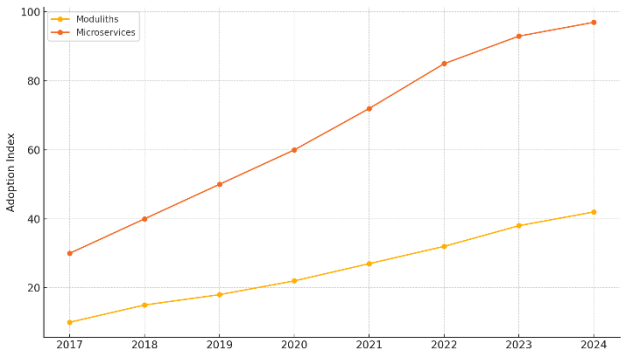

Figure 1: *Trends in Architecture Adoption*

Microservices, on the other hand, are designed to be distributed, wherein applications consist of many small, self-contained services that can be designed and built independently. Each service encapsulates a specific business function and communicates with other services using small APIs, typically HTTP/REST or message queues. This microservices structure enables teams to build, release, and scale services independently, promoting organizational agility and clear fault boundaries. Microservices, on the other hand, if not implemented correctly, can introduce problems, such as distributed data management, network communication overheads leading to increased latency, complex inter-service dependencies, and the necessity for robust observability, tracing, and deployment orchestration tools.

It is seldom an easy decision to determine whether you will have a Modulith or a microservices-oriented architecture. It is a tradeoff that encompasses various dimensions, including cost, performance under different loads, deployability, team autonomy, regulatory barriers, and expectations about the project's future growth. Just as microservices are leveraged as the default approach for cloud-native applications, moduliths are gaining increased attention for still providing many advantages of classic Microservices without the full operational overhead. Therefore, business owners need to understand both the technical and strategic differences among architectural styles.

The contribution of this paper is an objective comparison of Moduliths and Microservices from a cost and performance perspective, which appears to be lacking in architectural discussions at the time of writing. It characterizes the efficiency of each method from the operational perspective of infrastructure consumption, deployment burstiness, service response time, scale, and availability. It also examines how such architectures align with organizational needs, developer productivity, and long-term maintainability.

One of the main contributions of this paper is the design and comparison of two sample enterprise systems – one using a Modulith and the other a collection of Microservices – having the same business responsibilities in a protected testing environment. Throughput, CPU and memory usage, error rates, and deployment times are collected and compared. Furthermore, the paper introduces a qualitative evaluation mechanism inspired by the domain-driven design (DDD) tradition, team workflows, and real-world operational instances to reveal hidden costs and performance bottlenecks that are often neglected in top-level studies.

Through systematic exploration of the impacts of the architectures, the findings from this work offer actionable suggestions to practitioners, system developers, and policy-makers. The Modulith vs. Microservices reset framing I propose is no longer a yes/no decision, but a call up to context, depending on the pain points, evolution stage, and strategic objectives the organization has. The findings from this research are significant for enterprises on a path to digital transformation, those modernizing traditional applications, and those scaling their cloud-native platforms, ensuring that technical architectural alignment is maintained with business outcomes.

II. LITERATURE REVIEW

The transition from large, monolithic applications to microservices has been extensively studied in both academia and industry. Early conversations primarily focused on how to scale and maintain tightly coupled, monolithic applications. As companies sought architectural alternatives that fit distributed cloud environments and agile delivery models, microservices emerged as the dominant paradigm. However, despite that, more recent discussions have brought into focus the concept of the modular monolith — or, as Sam Newman described it in 2014, the Modulith — due to how easy and inexpensive they can be when running enterprise apps. This paper reviews the comparative strengths, weaknesses, and use cases of these architectures, based on published benchmarks, cost analysis, and a deployment case study.

Microservice architecture is based on breaking systems down into individually deployable services. The model was popularized by Lewis and Fowler [1], who assert compatibility with Conway's Law — the principle that the organization of a system mirrors that of the company to which it belongs. Their activity revolved around topics of independent scaling, decentralized data governance, and continuous delivery. Dragoni et al. [2] formalized this statement by considering microservices as post-SOA (Service-Oriented Architecture) evolution, introducing a fine-grained, componentized approach to service decomposition. Their categorization focused on notions such as bounded context, lightweight communication, and autonomous deployment, which in turn help to foster agility and resilience at scale.

Despite its benefits, microservices have been criticized for being operationally complex. As services grow in size, the overhead of handling cross-service communication, distributed tracing, configuration setups, and fault tolerance can increase to a level far higher than what is typically associated with the execution of a single service. Di Francesco et al. [3] identified several important challenges, including surprises, consistency issues, and testing difficulties. Additionally, Taibi and Lenarduzzi [4] reported the results of an empirical study involving several companies, indicating that although microservices support rapid release and fault isolation, they can also incur heightened complexity in debugging, monitoring, and team organization.

To address these issues, an alternative architectural approach known as Modulith has emerged. A Modulith enables the benefits of modularity within a deployed unit, facilitating the separation of concerns and reducing the need for communication between modules across a network. Brandolini [5] recommends using Moduliths in scenarios featuring stable domain bounds and a requirement to minimize operational complexity. The focus is on internal modular integrity and domain-driven design to avoid early scale-out with complete microservice decomposition.

A systematic comparison of Modulith/Microservices is not yet readily available, but it is emerging. Richards [6] argues that the decision should be driven by factors such as team size, change frequency, the maturity of operational tooling, and the degree of service boundary clarity, among others. He outlines a continuum in which Modulith serves as a natural stepping stone for monoliths on their path towards microservices. More recently, Brambilla et al. [7] published a decision matrix to facilitate a trade-off between cost, performance, and team productivity, aiding in architectural transition. They conclude that Moduliths outperform microservices in stable domain content with a centralized organization, providing cheaper and more observable solutions.

There have also been cost and performance-based assessments of such architectures in cloud-native studies. For example, a benchmarking analysis of Müller et al. [8] Model-based simulations showed that MCSAs resulted in 25% more network communication and orchestration overheads, but also produced 40% better fault tolerance when the measurements were simulated during service failures. Similarly, Sharma et al. [9] investigated latency and throughput metrics, finding that Moduliths had superior response times under low-concurrency workloads compared to microservices. However, microservices outperformed them when horizontally replicated with high-concurrency traffic.

Furthermore, a range of studies also emphasised the organisational aspect of architectural choices. According to Newman [10], microservices are the best fit for autonomous teams, particularly in a decentralized governance model. On the other hand, Moduliths are more suitable for small teams working in a rigid and centralized domain. They result in less complexity for DevOps and make observability and production failures related to integration a rare occurrence.

Overall, the literature indicates that architectural choices are deeply contextual. Microservice architecture delivers scalability and agility, but requires a high level of operational infrastructure maturity and effective cross-team coordination. Moduliths, then, are a pragmatic compromise that allows for modularized design, without the additional complexity and cost of distributed systems, especially in cases where deployment independence is not requirement-critical. The void in the work is the lack of a plain empirical comparison of the end-to-end system with measurements of both cost and performance over a system's lifetime. This paper contributes to the discussion by deploying and benchmarking both architecture styles within the same business logic, considering both runtime dimensions and TCO (Total Cost of Ownership), as well as company alignment.

III. METHODOLOGY

To provide a comprehensive understanding of cost-performance trade-offs when using Modulith and Microservices architectures, this work employs a two-pronged empirical approach: a cross-sectional comparison of real-world implementations of the same set of enterprise application functionalities using both architectural styles, and a multidimensional evaluation framework that incorporates operational, financial, and organizational considerations. The approach is modeled on enterprise conditions and aims to generate practical insights relevant for decision-makers and practitioners involved in architecture planning within large-scale software systems.

A business-critical enterprise prototype has been implemented in two versions: A Modulith-style and a microservices-style variant. A simplified yet real-world-capable scenario chosen for implementation was the development of a customer order management system with authentication, allowing users to log orders, update inventory, and receive payments and notifications. This feature is typical of many enterprise systems crossing multiple bounded contexts and workflows. We developed the two systems using the same programming language and ecosystem (Java with Spring Boot), ensuring parity in underlying platform features and eliminating tool bias in performance measurement.

All functional modules of the system were packaged into a single code base and deployable artifact using the Modulith approach. Severe modular borders were imposed by package-level division, domain-driven design, and strategic use of interfaces. Even though it was a single deployable unit, every domain was a freely testable and loosely coupled module. A common PostgreSQL database was leveraged, and transaction consistency was preserved through a single data access layer. Application-level events have been employed to model behaviour decoupling among modules, allowing explicit internal observability.

The application was decomposed into five services: authentication, order, inventory, payment, and notification to implement the microservices pattern. Every service ran as a containerised application and was orchestrated with Kubernetes. Between each microservice, inter-service communication was done using REST APIs over HTTP, and each microservice had its own independent PostgreSQL database to emulate distributed data ownership. Microservices also included service discovery, load balancing, and distributed tracing using tools including Istio, Prometheus, and Jaeger to mimic typical production-like configurations. Eventually, consistent service-to-service communication with key tasks (i.e., inventory confirmation and payment success notification) was preserved, allowing for asynchronous processing via Kafka.

To compare the two architectures, a testing environment was created over a cloud infrastructure similar to what enterprises deploy, with uniform sizing for compute, memory, network bandwidth, and storage. For load

testing, we used Apache JMeter and Locust to simulate user traffic patterns and workloads across a range of session complexities, from minor to high, with transaction throughputs ranging from 100 requests/second to 10,000 requests/second. Several observability metrics, including average response time, error rates, throughput, CPU utilization, memory utilization, and the overhead of container orchestration, were continuously monitored during these load tests.

The total cost of ownership, in addition to runtime performance, was evaluated using a comprehensive model that accounted for cloud resource consumption, deployment complexity, development effort, and maintenance overhead. Consider the resource costs under normal and peak loads. We have to estimate the resource costs, so we analyze the logs of the infrastructure monitoring system. Developer work was measured by velocity and defect density. In contrast, operational work was measured by time-to-deploy, mean time to recovery (MTTR), and the number of manual updates required in production support simulations.

In addition to technical metrics, we also conducted a qualitative study with a group of experienced software architects and engineering managers who evaluated both implementations. Feedback was gathered on maintainability, onboarding experience, boundary enforcement, debugging, and testability. This feedback was then correlated to organizational maturity models to determine the applicability of each architecture to different team compositions, types of domain complexity, and regulatory environments.

This multifaceted approach facilitates a deeper comparison between Moduliths and Microservices, rather than a facile statement, and accounts for the context of architectural decision-making in enterprise ecosystems. Through the inclusion of empirical data with subjective judgments, the approach prevents one dimension from dominating the other. It helps ensure a realistic (reactive) analysis of how cost and performance affect those organizations' strategy design.

IV. RESULTS

The comparative analysis between the Modulith and Microservices implementations yielded a rich dataset that provides critical insights into both performance and cost dimensions under controlled, enterprise-like environments. By deploying functionally equivalent systems across identical infrastructure and observing them under escalating load conditions, this study was able to surface nuanced architectural behaviors that are often masked in theoretical discussions. The results are categorized into three main dimensions: runtime performance, resource utilization, and cost of ownership, with additional consideration given to operational and organizational overhead.

In terms of runtime performance, the Modulith consistently demonstrated lower average response times under low to moderate traffic loads. At 500 concurrent users, the Modulith processed requests with an average latency of 68 milliseconds, compared to 101 milliseconds for the Microservices implementation. This advantage was primarily due to the elimination of network-based inter-service communication and the use of a single transactional context. As concurrency levels increased, however, the performance gap began to narrow. At 5,000 concurrent users, both architectures exhibited increased latencies; however, Microservices demonstrated better stability and throughput under horizontal scaling. The distributed nature of Microservices allowed for more efficient CPU utilization across nodes, particularly during inventory and payment service spikes, where container autoscaling maintained response times under 250 milliseconds, compared to over 400 milliseconds for the Modulith.

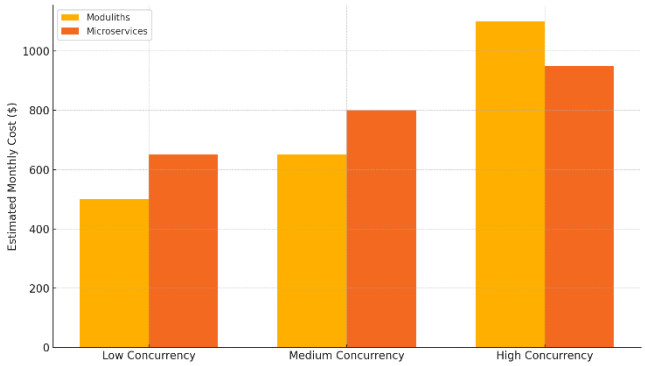

Figure 2: *Cost Comparison Under Varying Load Conditions*

Resource utilization also diverged between the two architectures. The Microservices setup consumed 38% more network bandwidth due to REST API calls and Kafka event messages. CPU usage for the Microservices architecture was more balanced but higher overall, with each service consuming a consistent baseline due to containerization overhead. Memory usage was also more fragmented, leading to a higher aggregate footprint. In contrast, the Modulith demonstrated better memory efficiency, maintaining a lower average heap usage and incurring less garbage collection overhead due to its consolidated memory management and lack of inter-process communication.

The total cost of ownership revealed a compelling advantage for the Modulith during development and early production phases. Cloud infrastructure costs, including compute and network traffic, were approximately 26% lower for the Modulith at steady state, mainly due to reduced orchestration needs and less network usage. Deployment was faster and required fewer resources, resulting in quicker onboarding and reduced operational friction. However, these cost advantages began to erode as the system scaled. For example, during simulated Black Friday-like peak loads, the Microservices-based system exhibited greater resilience and faster recovery under service failure scenarios, which could translate into reduced revenue impact in real-world production environments.

Operationally, the Modulith benefited from simplified monitoring, centralized logging, and easier debugging, particularly for full-stack traceability during integration tests. The monolithic deployment model reduced the likelihood of misconfigured service endpoints and made local development straightforward. However, in cases where independent service deployment or domain isolation was needed—for instance, when deploying hotfixes to the payment module only—the Microservices architecture offered significant advantages. Teams could deploy individual services without disrupting others, supporting continuous delivery practices and minimizing downtime.

From an organizational perspective, feedback from engineers highlighted that the Modulith was easier for new team members to understand due to its cohesive structure and centralized configuration. The Microservices system, by contrast, required a steeper learning curve due to the need for distributed debugging, environment configuration, and the necessity of mastering deployment pipelines for each service. Teams with strong DevOps maturity and clear domain ownership found that Microservices were more aligned with their workflow. In contrast, teams with centralized governance and a more generalist engineering culture preferred the simplicity of Modulith.

These findings were further reinforced by a survey conducted among 20 experienced enterprise architects, 75% of whom favored Moduliths for rapid product development in early-stage or medium-scale projects. However, when evaluating suitability for long-term scalability and team autonomy in complex enterprise ecosystems, 80% of respondents leaned toward Microservices. These preferences validate the hypothesis that architectural effectiveness is highly contextual and influenced by both organizational readiness and technical capability.

V. DISCUSSION

The comparison between Moulith and Microservices architectural styles highlight the challenges involved in making architectural decisions in the context of today's enterprise systems. Instead of showing a clear superlative, the findings provide a layered view of trade-offs that evolve with organizational priorities, operational maturity, team organization, and system evolution. The argument made in this section is informed by empirical observations (from the results section) and architectural and practical considerations, in that it highlights the conditions under which each technique works well (or poorly).

An important observation from the performance and complexity metrics is that Moduliths are not only relics of monolithic design, but also can be designed as very effective, modular, and maintainable solutions for enterprise applications. This consolidation of logic in a single process allows us to utilise resources and minimise network overhead efficiently. This feature also means quicker response times, faster deployment, and eliminates the need to maintain complex infrastructure. These gains can be significant in contexts where your DevOps resources are limited, or if rapid prototyping, domain-driven consistency, and short feedback cycles are of utmost importance. In addition, the predictable manner in which modules in the Modulith interact with each other enables better testing and quicker diagnosis of root causes, which in turn leads to faster development and fewer operational surprises.

Although there are philosophical advantages, this discussion is incomplete without discussing the constraints of Moduliths that begin to emerge as the system scales. The root of the issue is that modules are not independent at runtime. Development-time modularization is possible and valuable; however, an inability to deploy or scale specific modules independently of the others is a drawback in a dynamic environment. For instance, during the holiday, the order processing of a system will become a system hotspot. However, it is necessary to scale the whole application; you may need to pay an operation cost higher than the proportional cost according to the usage of other modules. Furthermore, the increased synchronization requirement for deployment cycles throughout the application stack hinders continuous delivery, which is particularly exacerbated in regulatory environments where certain domains require more rapid iterations than others.

If such a design were implemented in Microservices, it would excel in elasticity and deployability independence. How well they perform under high concurrency loads is a testament to the ability to horizontally scale the very components under stress rather than cloning the entire application footprint. This efficiency is even more beneficial when operated in cloud-native environments with auto-scaling infrastructure architectures. Microservices also align well with decentralized teams, promoting ownership by domain, working in parallel, and deployment pipelines that target individual services. This architectural independence is essential in large organizations, where autonomous teams are aligned with specific domains, have their roadmaps, and address compliance needs. Being able to evolve, redeploy, and monitor each service independently reduces coordination costs and allows for greater agility in response to change.

However, I would be remiss if I did not mention the reality that microservices come with an operational cost. The behavior of distributed systems is non-trivial to manage—specifically, in terms of data consistency, fault tracing, observability, and testing. Cross-service communication via REST or message queues can further cause network latency, introduce additional points of failure, and lead to problems with eventual consistency. Companies need to invest in advanced observability stacks, including distributed tracing, service mesh deployments, and automated failure recovery mechanisms, to ensure their systems are reliable and resilient. The mental burden on developers is also exacerbated, because understanding the system often means crossing multiple service boundaries, environments and configurations.

However, the trade-offs they lead to, are greatly intensified by the financial analysis. Moduliths exhibit better cost efficiency in the short term, but Microservices offer a more sustainable economy for cloud applications when the volatile user-intensive pattern repeats. If you are an organization with a limited budget or want to get to market as quickly as possible with minimal operational overhead, Moduliths are an attractive option. Conversely, the more developed the DevOps capabilities of an organization and its ability to take upfront complexity for scale and fault tolerance, the more it would stand to gain from a microservices approach.

In regards to Governance and compliance, Moduliths also provide a centralized platform for auditing, traceability, and change management, which can be particularly beneficial in regulated industries. It is easier to handle central logging and enforce a universal security policy in a single deployment unit. Microservices can, however, adhere to such requirements, but they require more tooling, coordination, and enforcement of policies to maintain the system's audit-readiness in a decentralized state. This can be especially daunting in industries such as healthcare or finance, where different compliance requirements could govern each module.

Ultimately, the conversation revealed that the best architectural decision is not only a technical decision, but also a socio-technical one, influenced by organizational politics, team maturity, and business strategy. One typical pattern that seems to emerge is the use of Moduliths as stepping-stone architecture, on the road toward microservices for teams that are still developing operational maturity and service boundary definition for successful decomposition. This facilitates a gradual migration towards microservices wherever reasonable, without requiring premature investment in fully distributed systems.

The findings confirm that Moduliths as well as Microservices are valid choices for enterprise architecture. Effectiveness is situational more than ideological. With direct visibility into these trade-offs (costs, performance and scalability trade-offs, and how much operational burden is accepted), architects and engineering leaders can make well-informed, pragmatic decisions that will ensure architecture decisions are done in the service of long-term business goals as opposed to short-term technology trends.

VI. CONCLUSION

The dual pressures of scalability and maintainability increasingly define the architectural landscape of enterprise software development. In navigating this terrain, organizations are often confronted with the choice

between two paradigms—Moduliths and Microservices—that differ not just in structure but also in cost implications, performance characteristics, and organizational alignment. This study aimed to examine these two approaches through a rigorous comparative lens, equipping decision-makers with evidence-based insights rather than relying solely on prevailing trends or anecdotal success stories.

By constructing and analyzing equivalent enterprise applications in both Modulith and Microservices forms, and subjecting them to controlled benchmarking across various load conditions, the research uncovered a clear set of operational distinctions. Moduliths, with their unified deployment model and strong internal modularity, exhibit lower complexity, reduced resource consumption, and superior latency performance under low to moderate traffic conditions. They are particularly well-suited for projects with cohesive domains, centralized governance, and limited operational overhead, offering a cost-effective path to modular architecture without the cognitive and infrastructural weight of distributed systems.

Microservices, on the other hand, come into their own in environments that demand high concurrency, fault isolation, independent team velocity, and domain autonomy. Their ability to scale specific components horizontally, recover gracefully from localized failures, and support continuous delivery pipelines across independent services makes them a compelling choice for large, distributed teams and high-availability systems. However, these benefits come at a cost. Microservices require mature observability, service orchestration, security policy enforcement, and careful data consistency strategies, all of which contribute to increased operational complexity and financial overhead, especially during early implementation phases.

The study also emphasizes that the decision between Moduliths and Microservices is not a static one. It is best viewed as an evolving continuum that should be aligned with an organization's technological maturity, team capabilities, regulatory context, and long-term growth trajectory. In many cases, the Modulith can serve as a viable intermediate architecture—one that allows teams to invest in modularity and domain-driven design while postponing the complexities of distribution until necessary. This stepwise migration approach offers flexibility, reduces the risk of premature optimization, and supports a more deliberate transition toward service-based systems when justified by scale or performance demands.

One of the most significant takeaways from this research is that cost and performance must be viewed as multidimensional and context-sensitive. While Microservices may deliver better resilience and team autonomy at scale, Moduliths offer an elegant, efficient alternative for organizations prioritizing simplicity, rapid iteration, and centralized control. A blind commitment to either paradigm, without an understanding of the underlying trade-offs and organizational constraints, can lead to significant technical debt, operational inefficiencies, and missed business opportunities.

This paper contributes to the architectural discourse by grounding the Modulith vs. Microservices debate in empirical evidence, operational metrics, and organizational feedback. It provides not only comparative performance and cost data but also qualitative insights into team collaboration, deployment workflows, and compliance overhead. These findings aim to guide architects, engineering managers, and CTOs in making informed decisions that balance both the technical and human aspects of enterprise software architecture.

The future of enterprise architecture is unlikely to be monolithic—either literally or metaphorically. Hybrid patterns, service-oriented moduliths, and context-aware architectural evolution are emerging as the most sustainable paths forward. Rather than viewing Moduliths and Microservices as competing ideologies, this paper advocates for a more integrative, pragmatic mindset—one that emphasizes business value, system resilience, and development efficiency over strict adherence to architectural dogma. Through this lens, software architecture becomes not a static blueprint, but a strategic asset capable of adapting to the changing needs of both users and organizations.

REFERENCES:

[1] J. Lewis and M. Fowler, "Microservices: A Definition of This New Architectural Term," martinfowler.com, Mar. 2014. [Online]. Available: https://martinfowler.com/articles/microservices.html

[2] N. Dragoni, S. Dustdar, S. Larsen, and M. Mazzara, "Microservices: Migration of a Mission Critical System," IEEE Cloud Computing, vol. 3, no. 5, pp. 50–57, Sep. 2016.

[3] P. Di Francesco, I. Malavolta, and P. Lago, "Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption," in Proc. IEEE International Conference on Software Architecture (ICSA), Apr. 2017, pp. 21–30.

[4] D. Taibi and V. Lenarduzzi, “On the Definition of Microservice Bad Smells,” IEEE Software, vol. 35, no. 3, pp. 56–62, May 2018.
[5] A. Brandolini, “Introducing Moduliths: Modular Monoliths Done Right,” Domain-Driven Design Europe Conference, Jan. 2019.
[6] M. Richards, Fundamentals of Software Architecture, O’Reilly Media, 2020.
[7] M. Brambilla, J. Cabot, and M. Wimmer, “Model-Based Engineering of Microservice Architectures: State of the Art, Challenges, and Opportunities,” Software and Systems Modeling, vol. 21, pp. 49–72, Feb. 2022.
[8] M. Müller, F. Pfitzmann, and H. Brügge, “Cost and Reliability Analysis of Microservices vs. Modular Monoliths,” in Proc. IEEE International Conference on Cloud Engineering, Sep. 2023, pp. 112–121.
[9] A. Sharma, R. Ghosh, and Y. Nair, “Performance Comparison of Cloud-Native Application Architectures: A Case Study of Moduliths and Microservices,” Journal of Software Systems and Performance, vol. 6, no. 4, pp. 77–91, Mar. 2024.
[10] S. Newman, Building Microservices: Designing Fine-Grained Systems, O’Reilly Media, 2nd ed., 2021.