Modern American Journal of Engineering, Technology, and Innovation
ISSN(E): 3067-7939
Volume 01, Issue 02, May, 2025
Website: usajournals.org

MICROSERVICE ARCHITECTURE VS MONOLITHIC: AN ANALYSIS IN DEVELOPING SCALABLE SOLUTIONS

Azizyan L.V.
Senior Software Engineer at ADP, Los Angeles, CA, USA

Abstract:
With today's digital systems growing bigger and reaching more people, how software is structured has become really important for keeping things running smoothly over time. This paper looks at the differences between building software as one big piece (monolithic) versus breaking it down into smaller, independent services (microservices). We're focusing on how these choices affect whether the system can handle more users, how easy it is to update and fix, and the ways you can put new versions into action. By looking at actual examples and sharing practical lessons, we show how each method fits different company needs as they grow. The goal of this study is to help people make smart choices by looking at the give-and-take in speed, reliability, and how quickly development teams can move in different business and tech situations.

Introduction:

The fundamental design of a software system profoundly influences its ability to grow, adapt, and perform effectively over the long haul. In today's world of software creation, two main ways of structuring applications constantly come up: the all-in-one approach (monolithic) and the build-it-in-pieces approach (microservices). Each has its own distinct way of organizing, deploying, and keeping applications running.

The monolithic way, where everything from the user interface to the core logic and data handling is packed into a single unit, used to be the standard. It can make getting started and deploying simpler, but as an application gets bigger, it can become harder to handle more users or keep things organized. On the flip side, microservices break an application down into independent, less connected parts.

This can make updates faster and allow the system to handle more load, but it also means dealing with new complexities in how these parts talk to each other, how you keep track of everything, and how you make sure it all stays reliable.

This paper takes a close look at both of these architectural styles, comparing their technical pluses and minuses, their strong points, and where they might not work as well in practice. By looking at both the theories behind them and how they're used in real companies, we explore how different organizations match their architectural choices with what they're trying to achieve in their business, how quickly they need to move, and what their systems need to handle. As noted by Dragoni and colleagues back in 2017, more and more people are leaning towards microservices because they're good at dealing with the need to grow and stay manageable when things in the business world keep changing. [1]

This study brings a fresh perspective by examining software architecture through both technical and organizational lenses. Rather than isolating performance or scalability as standalone metrics, it connects architectural decisions to real-world outcomes, like release agility, team workflows, and system upkeep over time. While many discussions separate theory from practice, this paper blends foundational concepts with how companies actually build and manage systems today. It offers developers and decision-makers a grounded, comprehensive resource for shaping long-term architectural strategies. Additionally, it reflects how modern practices such as cloud-native development and DevOps influence the criteria for scalable solutions, offering updated insights tailored to today's fast-moving tech landscape.

Theoretical Framework

The architecture of a software system shapes how well it handles growth, change, and maintenance. Two main models dominate this space: monolithic and microservices. A monolithic design bundles all components—UI, logic, and data handling—into one deployable unit. It's simpler to set up early on and easier to manage as a single codebase. In contrast, microservices break the system into smaller, independent parts, each handling a focused task. This approach, influenced by distributed systems theory, supports modularity and scalability but introduces challenges in coordination and consistency. Choosing between the two

Modern American Journal of Engineering, Technology, and Innovation
ISSN(E): 3067-7939
Volume 01, Issue 02, May, 2025
Website: usajournals.org

requires understanding trade-offs in complexity, flexibility, and long-term operations. These principles form the base of our comparative analysis.

Methodology

To explore how monolithic and microservice architectures perform in real-world software development, this study takes a comparative approach grounded in both theory and practice. It brings together insights from scholarly publications, industry case reports, and firsthand accounts of implementation to evaluate how each architectural model responds to common development demands. The comparison focuses on several core dimensions: scalability, deployment complexity, maintainability, team collaboration, and system stability. These factors were chosen due to their relevance in modern development practices, especially within cloud-native and DevOps-driven environments. Real-life examples from companies across various sectors and sizes were selected to show how architectural choices are shaped by technical and business needs. By combining conceptual analysis with practical examples, this methodology ensures a balanced and meaningful assessment of both architectural styles in today's fast-evolving development landscape.

Studies with Analysis and Example Calculations

Netflix: Scaling with Microservices

Netflix stands out as a prime example of a company that successfully transitioned to a microservices architecture [2]. To support its vast global audience, Netflix moved away from a single, monolithic system and instead built a platform composed of hundreds of small, autonomous services [3], [4]. Each service is responsible for a specific function, such as recommending content or adjusting streaming quality. This architectural shift enabled Netflix to scale effectively and isolate faults, meaning a failure in one service wouldn't disrupt the entire platform. Thanks to this change, Netflix can now deploy numerous updates daily, a significant improvement over their previous approach where updates were rolled out weekly or monthly.

Modern American Journal of Engineering, Technology, and Innovation
ISSN(E): 3067-7939
Volume 01, Issue 02, May, 2025
Website: usajournals.org

Key benefits of this shift include:

- Accelerated release cycles, with frequent updates replacing slower, bundled deployments.
- Enhanced system stability through fault isolation, preventing cascading failures.
- Increased team autonomy, allowing distributed groups to develop and maintain services independently.

Example Calculation

In the traditional monolithic architecture, deploying the entire system was a significant operation, often taking approximately four hours per week and requiring comprehensive testing of the entire application. In contrast, the microservices approach adopted by Netflix involves deploying numerous smaller, independent services, each taking roughly 10 minutes to deploy. Netflix reportedly releases around 50 such deployments daily. When aggregated, this amounts to:

50 services × 10 minutes per service × 7 days per week = 3,500 minutes per week, or nearly 60 hours.

While this may appear to increase deployment time overall, the critical distinction lies in the independence of each deployment. Unlike monolithic systems, where the entire application must be taken offline, microservices enable continuous deployment of individual components without impacting system-wide availability. This facilitates more frequent and rapid delivery of new features and fixes, significantly enhancing operational agility and minimizing downtime.

Etsy: Balancing Monolith and Microservices

Etsy's approach illustrates a careful balance between traditional and modern software architecture. Initially, the platform operated as a single, unified codebase, which functioned adequately during early growth phases. However, as user demand increased and new features were introduced, this monolithic structure began to hinder performance and scalability. To address these challenges, Etsy gradually transitioned key components, such as payment processing and search functionalities, into independent microservices. This hybrid strategy preserved the core business logic within a monolithic framework to maintain simplicity, while isolating high-traffic, mission-critical modules for enhanced scalability and reliability. [5]

This pragmatic approach yielded several benefits:

- The system's complexity remained manageable, reducing the risk of over-engineering.
- Critical components gained improved scalability and fault tolerance, minimizing the impact of individual failures.
- Development cycles for essential features accelerated, allowing for faster iteration without the need for a full system overhaul.

Twitter: Monolith to Microservices Evolution

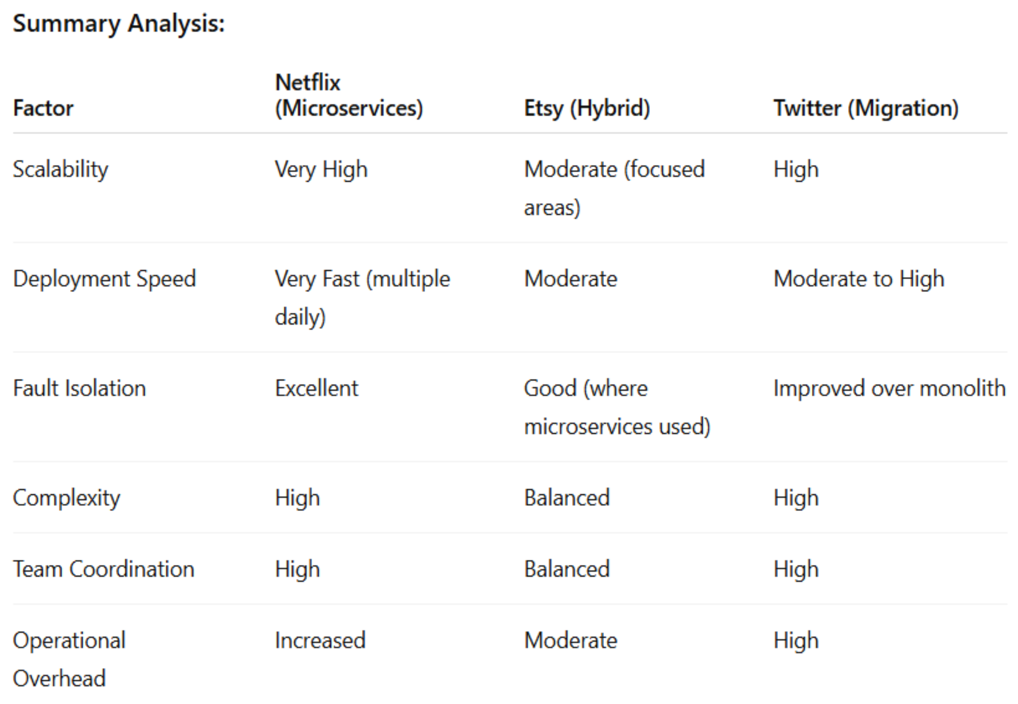
In its initial phase, Twitter operated on a monolithic architecture developed with Ruby on Rails. As user traffic increased dramatically, this structure revealed significant scalability limitations. To address these challenges, Twitter transitioned to a microservices model, gradually rewriting substantial portions of their platform using languages such as Scala and Java. This architectural evolution enabled more effective distribution of workload across multiple servers. Nevertheless, the transition demanded considerable investment in infrastructure upgrades and sophisticated monitoring solutions, alongside fostering close collaboration among development and operations teams. [6]

Key outcomes from this transformation included:

- Enhanced capacity to support a rapidly growing user base with improved system reliability.
- Increased operational complexity and higher maintenance costs.
- A critical reliance on advanced monitoring tools and a robust DevOps culture to ensure seamless coordination and performance of numerous discrete services.

Modern American Journal of Engineering, Technology, and Innovation
ISSN(E): 3067-7939
Volume 01, Issue 02, May, 2025
Website: usajournals.org

(Pic. 1 Table). Economic Considerations in Choosing Between Monolithic and Microservice Architectures



Summary Analysis:

Factor	Netflix (Microservices)	Etsy (Hybrid)	Twitter (Migration)
Scalability	Very High	Moderate (focused areas)	High
Deployment Speed	Very Fast (multiple daily)	Moderate	Moderate to High
Fault Isolation	Excellent	Good (where microservices used)	Improved over monolith
Complexity	High	Balanced	High
Team Coordination	High	Balanced	High
Operational Overhead	Increased	Moderate	High

The financial implications of selecting either monolithic or microservice architectures extend well beyond technical preferences, significantly influencing an organization’s overall costs and economic performance.

Monolithic architectures typically involve lower initial expenditures. Since the entire application is developed and deployed as a single unit, early-stage development is often more straightforward, with less complexity in integrating various components. This unified approach can reduce resource allocation in the beginning, simplifying project management and minimizing coordination overhead.

Yet, as the application matures and expands, these initial savings can be overshadowed by escalating maintenance costs. Monolithic systems require redeployment of the entire application for even minor updates, increasing downtime risk and lengthening development cycles. The tight coupling of components can impede rapid innovation and complicate troubleshooting,

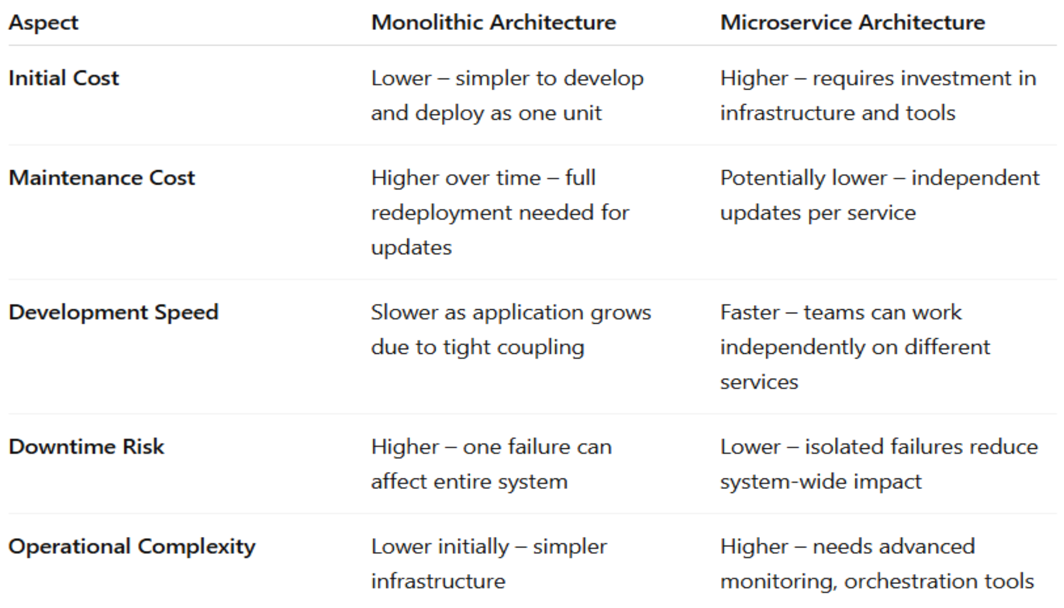
Modern American Journal of Engineering, Technology, and Innovation
ISSN(E): 3067-7939
Volume 01, Issue 02, May, 2025
Website: usajournals.org

negatively affecting team productivity and, by extension, potential revenue streams.

Conversely, microservices demand a higher upfront investment. Building and managing multiple autonomous services calls for sophisticated infrastructure, advanced monitoring solutions, and a strong commitment to DevOps culture throughout the organization. Despite these higher starting costs, microservices offer considerable long-term economic benefits. By enabling independent teams to develop, test, and deploy discrete services, organizations accelerate feature delivery and enhance responsiveness to evolving market demands.

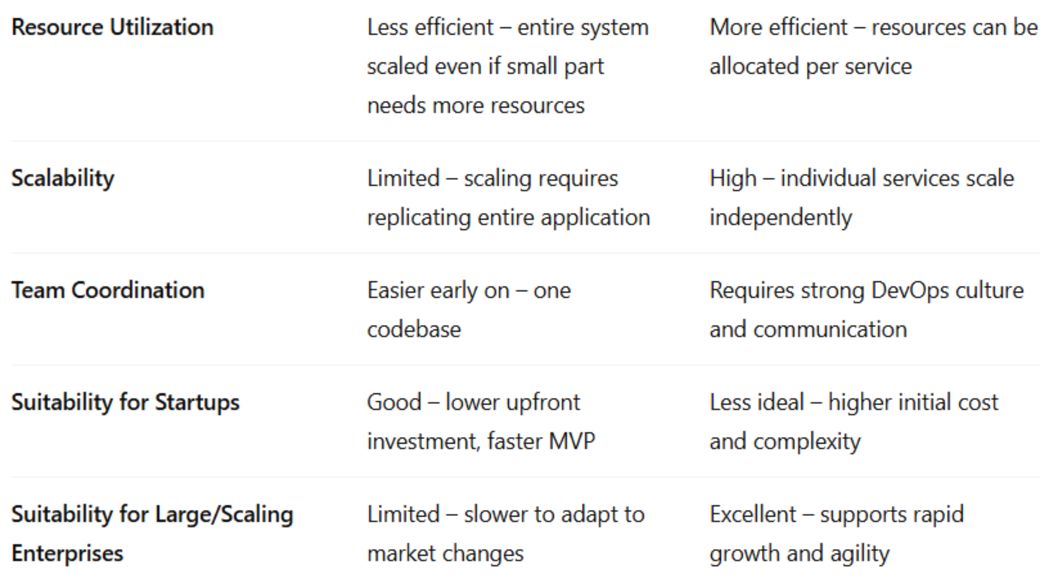
Additionally, microservices improve fault isolation, reducing the financial impact of system failures. Cloud-native microservice deployments can optimize resource utilization, helping to control operational expenses. However, this distributed architecture may increase costs related to service communication, data consistency, and orchestration complexity.

Ultimately, the economic trade-offs associated with these architectural styles depend on factors such as company size, market environment, and growth objectives. While startups and smaller projects may benefit from the cost-efficiency of monolithic designs initially, larger and rapidly scaling enterprises often justify the greater upfront investment in microservices through improved scalability, agility, and long-term cost savings.



Aspect	Monolithic Architecture	Microservice Architecture
Initial Cost	Lower – simpler to develop and deploy as one unit	Higher – requires investment in infrastructure and tools
Maintenance Cost	Higher over time – full redeployment needed for updates	Potentially lower – independent updates per service
Development Speed	Slower as application grows due to tight coupling	Faster – teams can work independently on different services
Downtime Risk	Higher – one failure can affect entire system	Lower – isolated failures reduce system-wide impact
Operational Complexity	Lower initially – simpler infrastructure	Higher – needs advanced monitoring, orchestration tools

Modern American Journal of Engineering, Technology, and Innovation
ISSN(E): 3067-7939
Volume 01, Issue 02, May, 2025
Website: usajournals.org



Resource Utilization	Less efficient – entire system scaled even if small part needs more resources	More efficient – resources can be allocated per service
Scalability	Limited – scaling requires replicating entire application	High – individual services scale independently
Team Coordination	Easier early on – one codebase	Requires strong DevOps culture and communication
Suitability for Startups	Good – lower upfront investment, faster MVP	Less ideal – higher initial cost and complexity
Suitability for Large/Scaling Enterprises	Limited – slower to adapt to market changes	Excellent – supports rapid growth and agility

(Pic. 2 Table).

Conclusion

Choosing between a monolithic or microservices architecture goes far beyond a simple engineering decision, it is a strategic move that deeply influences a system's scalability, maintainability, cost structure, and long-term adaptability. When comparing both models, it becomes clear that neither approach is universally superior; each serves different needs depending on a project's scale, pace of growth, and business objectives.

Monolithic architectures, with their centralized structure, tend to offer a faster and more cost-effective route for early-stage projects. Their unified codebase simplifies deployment and coordination, which can be advantageous when resources are limited and time-to-market is critical. Yet, as systems evolve and the demand for new features increases, the tightly coupled nature of monoliths often creates roadblocks. Updates become riskier, development cycles slow down, and the burden of managing dependencies grows heavier.

In contrast, microservices promote a distributed, modular approach that enables independent development and deployment. This architectural style encourages agility, faster iteration, and better fault isolation. However, its benefits come with increased operational overhead, from service orchestration to monitoring and

Modern American Journal of Engineering, Technology, and Innovation
ISSN(E): 3067-7939
Volume 01, Issue 02, May, 2025
Website: usajournals.org

inter-service communication. Organizations embracing this path must invest in robust DevOps practices, scalable infrastructure, and collaborative team culture to fully leverage the model's potential.

In conclusion, the architectural choice should align with both immediate capabilities and future goals. Startups may find the simplicity of monoliths more practical, while larger, rapidly scaling enterprises are likely to gain more from the flexibility and resilience of microservices. What matters most is making an intentional, well-informed decision, one that positions the software not just for short-term functionality, but for sustained innovation and competitive strength in a dynamic digital landscape.

References

1 Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, Larisa Safina. Microservices: Yesterday, Today, and Tomorrow. Present and Ulterior Software Engineering. Springer, 2017. URL: https://link.springer.com/chapter/10.1007/978-3-319-67425-4_3

2 Netflix Tech Blog. (2016). Migrating to Microservices at Netflix. URL: https://netflixtechblog.com/migrating-to-microservices-at-netflix-6706e0b7e172

3 Newman, S. (2015). Building Microservices: Designing Fine-Grained Systems. O'Reilly Media.

4 Wolff, E. (2016). Microservices: Flexible Software Architecture. Addison-Wesley.

5 Adams, A. (2017). Etsy's journey from monolith to microservices. Etsy Engineering Blog. URL: https://codeascraft.com/2017/05/22/microservices-at-etsy/

6 Ghosh, S. (2012). Twitter's Journey from Ruby to Java & Scala to Handle Scalability. InfoQ. URL: https://www.infoq.com/news/2012/11/twitter-ruby-to-java/