International Journal of Computer Trends and Technology
ISSN: 2231–2803 / https://doi.org/10.14445/22312803/IJCTT-V71I10P103

Volume 71 Issue 10, 20-27, October 2023
© 2023 Seventh Sense Research Group®

Review Article

Modular Monoliths: Revolutionizing Software Architecture for Efficient Payment Systems in Fintech

Kalpesh Barde

Technical Leader, Fintech, CA, USA.

Corresponding Author : kbarde27@gmail.com

Received: 12 August 2023 Revised: 19 September 2023 Accepted: 03 October 2023 Published: 18 October 2023

Abstract *- The evolution of software architecture has led to the emergence of various paradigms, including monolithic, microservices, and the lesser-explored modular monolith architecture. This paper delves into the historical development of these architectures, assessing their advantages and limitations, with a specific focus on their application in the fintech domain. Through an in-depth literature review and case studies of organizations like Shopify, Root, and Google, the study evaluates the potential of modular monolith architecture as the primary choice for developing efficient payment systems. By addressing the research gaps in existing studies and comparing modular monoliths with traditional monolithic and microservices architectures, this paper provides valuable insights for software developers, architects, and fintech industry professionals.*

Keywords *- Software design architecture, Monolithic architecture, Microservices architecture, Modular monolith software architecture, Fintech domain.*

1. Introduction

In the software development process, it is crucial to consider software design architecture early on, particularly during the planning and requirements-gathering stage. The main reason is that architecture serves as the basis for the entire software system and impacts all subsequent development activities.[6] By prioritizing software design architecture from the beginning and consistently refining it throughout the development cycle, software teams can increase their chances of delivering a high-quality software system that meets the expectations and requirements of stakeholders.[4] Therefore, it is important to study and assess various types of software design architecture, such as Monolithic architecture, Microservices architecture, and Modular Monolithic architecture, to determine which approach is most appropriate for a given project.

A monolithic architecture is a traditional software program model built as a self-contained unified unit and independent from other applications.[2] It is a unified, massive computing network running off of a single code base that connects every facet of the enterprise. One of the notable benefits of this approach is its ability to deliver enhanced performance compared to the microservices design, particularly for applications characterized by low to moderate traffic levels.[7] In addition, the application logic and data access are all contained within a single process, reducing the latency and network overhead associated with microservices architecture.[7] However, at the same time, making a change to this sort of architecture requires updating the entire stack of code by accessing the code base and building and deploying an updated version of the service-side interface. This makes updates restrictive and time-consuming.

In comparison, the microservice architectural style [1] is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and are independently deployable by fully automated deployment machinery. There is a bare minimum of centralized management of these services, which may be written in different programming languages and use different data storage technologies. This architectural style offers the benefit of easy horizontal scaling of individual services, which enhances the system's ability to handle increased loads and demands. Scaling specific services enables greater resilience and flexibility, improving the system's overall performance. It enables developers to develop individual services using different technologies and programming languages. This makes it easier to adopt new technologies, update the system and take advantage of new capabilities.

Finally, modular monolith architecture refers to a software design approach where all the code resides within a single codebase. This approach allows for quicker debugging of issues while minimizing the complexities that often arise

This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)

from using a microservices architecture. With modular monolith architecture, the functionality is broken down into multiple modules, making it easier to manage each module individually. The approach also establishes clear rules for accessing each module through strict interfaces, which helps resolve many of the complexities that can arise in software development. At present, several organizations such as Shopify[8], Eurocommercial Properties (Estatio) [36], Google [37] and Root [38], among others, are utilizing modular monolith architecture for its benefits such as simplified testing, streamlined deployment, and improved system performance.

Currently, both monolithic and microservices designs exhibit some limits in the domain of software development. For instance, the former presents challenges in terms of scalability, whereas the latter is plagued by concerns related to latency and network congestion. Despite the existence of a third technique known as modular monolithic architecture, it has garnered relatively limited attention in the academic research literature. The identified research gap pertains to the limited availability of literature on the subject of modular monolith. Consequently, this study aims to assess the advantages and drawbacks associated with employing a modular monolithic architecture as the predominant approach for software application development within the fintech sector.

2. Literature Review

2.1. Importance of Architecture in Software Design

The term' software architecture' (SA) was first coined in the late 1960s, but one of the early pioneers of SA, Fritz Bauer, argued as early as 1968 that SA should be given more importance, even before the software was developed. In the early days of software development, software systems were relatively simple, and their architectures were correspondingly straightforward. However, as software systems have grown in size and complexity, so too has the importance of software architecture. Today, software architecture is recognized as a critical component of software engineering, and many institutions have invested heavily in its development and study.

One of the reasons for this is that software architecture supports early design decisions that impact a system's development, deployment, and maintenance life. [4] Moreover, it gives a basis for analysis of software systems' behavior before the system has been built.[5] Getting the early, high-impact decisions right is important to prevent schedule and budget overruns.[4]

Software architecture has numerous benefits for software design. First, it gives a clear vision of the system and the components involved in it. It provides a solid foundation for identifying gaps and areas that require improvement in a given software. In addition, software architecture provides stakeholders, developers, project managers, user experience designers, quality assurance/testers, and DevOps engineers with a clear comprehension of the software's capabilities and its intended purpose. It helps to discern the distinction between software that has reached the production-ready stage and software still in the prototyping phase.

SA is also used as a blueprint or a foundation for a software system. Organizations choose to implement it on their own methodologies and styles, considering factors such as business needs, cost, scalability, performance, maintenance and support. Some examples of how organizations use software architecture are microservices, cloud computing, event-driven architecture, serverless architecture, monoliths, and modular monoliths.

2.2. Microservices Architecture

Microservices is one of the software architectural patterns that involves breaking down a single application into multiple services or smaller applications.[1]

Each of these services can then be developed in any programming language, which gives the software engineering teams flexibility to develop independently using their preferred programming language. Moreover, each of these services can be deployed independently, offering flexibility to the teams to scale up and scale down as needed.[9] Despite this independence, all these services communicate harmoniously to attain the application's purpose using well-defined API contracts.

The adoption of microservices architecture offers numerous advantages, such as the smooth integration of various technologies into a unified system, enhanced scalability, increased operational efficiency, and simpler deployment procedures [10]. Since it has emerged as a highly appreciated architectural framework, there has been a growing adoption of the microservices paradigm by esteemed software consulting organizations and product design enterprises. This particular methodology has been empirically shown to significantly increase overall productivity and enable the creation of highly successful software products. Moreover, a multitude of non-traditional software companies have both utilized and assessed this architectural approach, resulting in significant benefits.

Microservices are commonly recognized as a suitable architectural decision for systems deployed on cloud infrastructures due to their ability to leverage the flexibility and on-demand provisioning capabilities inherent in the cloud paradigm. Significantly, several prominent companies, including Netflix and SoundCloud, have effectively used the microservices architectural methodology in cloud computing settings, leading to numerous benefits [10].

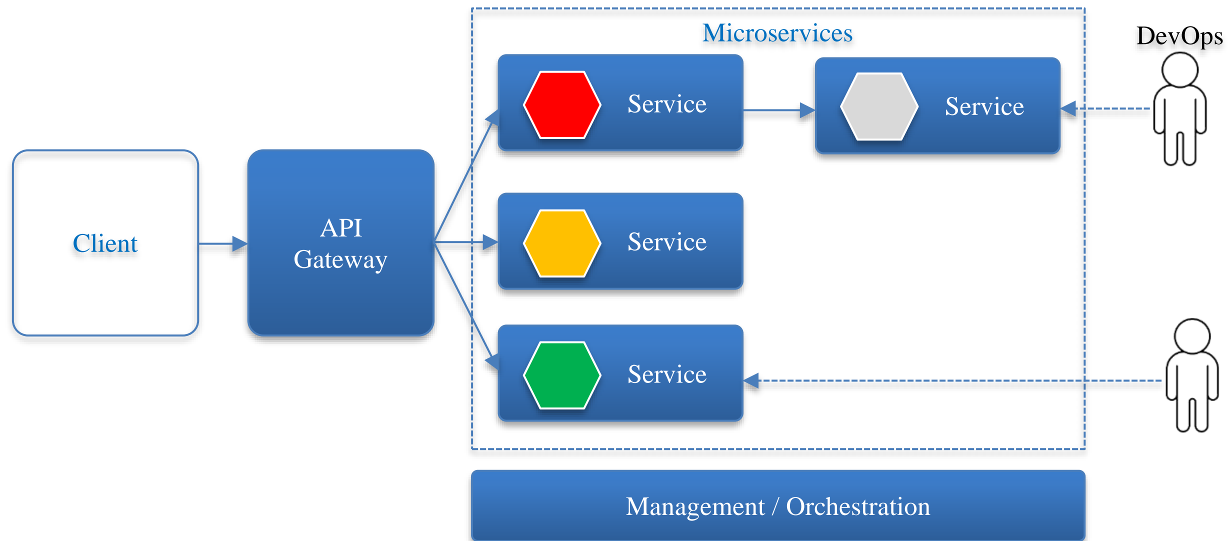


Image Source: docs.microsoft.com

The concept of microservices originated as a reaction to the difficulties and shortcomings encountered in service-oriented architectures (SOA) [11]. Service-Oriented Architecture (SOA), which saw a surge in popularity during the early 2000s, encountered challenges, including excessive hype, inadequate suitability, and inconsistent definitions, resulting in fruitless endeavours to implement it [12]. Microservices, frequently known as "SOA done correctly," present a novel methodology for constructing software applications as collections of self-contained services that can be deployed separately [13]. The utilization of this particular architectural style offers various advantages, including enhanced dynamism, improved modularity, facilitation of distributed development, and seamless integration of diverse systems [14]. The emergence of microservices can be attributed to historical factors such as the growth of software architecture, the widespread adoption of objects and services, and the demand for enhanced agility, scalability, and autonomy in software systems [15].

The microservices architecture enhances large-scale software systems' adaptability, scalability, and fault tolerance. Nevertheless, this architectural paradigm is not without its drawbacks. The migration from legacy systems to microservices presents a significant challenge, involving manual intervention, extensive time commitment, susceptibility to errors, and substantial financial expenditure [20]. Additionally, there are concerns related to escalated security vulnerabilities due to the expanded attack surface resulting from the disintegration of system functionalities into cohesive, small-scale services [16]. Heightened concurrency levels inherent in microservices also raise issues, potentially leading to subtle programming errors like race conditions, deadlocks, and data inconsistencies [17]. Furthermore, microservices pose complexities in performance testing, making establishing a baseline performance intricate and obtaining reliable performance testing outcomes less straightforward [18]. The overall intricacy of loosely connected microservice systems further complicates the testing process, necessitating the utilization of supplementary testing techniques and tools [19].

2.3. Monolith Architecture

Monolithic software architecture refers to a conventional approach where all the different types of foundational architectural elements of an application are integrated together in a single unit.[21] It is considered an older architectural style and is often seen as outdated [22]. It is characterized by a lack of modularization and a tightly coupled structure. The emergence of software architecture as a field of study has brought attention to the need for better organization and design of software systems. Different architectural styles, such as client-server architecture and module interconnection languages, have been developed to address this. The goal is to improve understanding and manage the complexity of software systems. However, challenges still remain in achieving effective software architecture, and there is ongoing research and development in this area [23] [24] [25].

The significant landmarks in the evolution of monolithic software architecture encompass the advent of diverse architectural models, including monolithic architectures and service-oriented architectures. Monolithic designs are widely recognized as traditional and include consolidating all essential application pieces under a singular component or unit. Nevertheless, monolithic designs are frequently regarded as antiquated, requiring architectural restructuring to separate the user interface, business logic, and data layer. Although monolithic systems have certain limits, they

possess several advantages, like reduced complexity in the interaction between components and the convenience of seeing an entire process within a single location. The use of microservices-based architectures is regarded as a contemporary trend in addressing the difficulties associated with interoperability, replacing the traditional monolithic systems.[29] [26]

The utilization of monolith architecture presents several benefits. The strong relationship between domain entities enables rapid initial growth.[3] In addition, the implementation of monolithic architecture guarantees consistent values of software metrics, such as complexity and deployability, irrespective of the quantity of features incorporated.[27] Monolith architecture offers simplicity, reliability, and stability in software development and operation.

A monolithic architecture possesses several drawbacks. One such disadvantage pertains to the challenge of staying abreast with novel development methodologies like DevOps, which necessitate frequent deployments [3]. Another drawback lies in the inflexibility and lack of scalability inherent in monolithic architectures, for they are not readily adaptable to evolving requirements and cannot be independently scaled [28]. Monoliths also pose difficulties in terms of code complexity and magnitude, rendering the management and maintenance of the codebase more arduous [27]. Furthermore, monolithic architectures encompass a solitary database, which can give rise to complications when transitioning to a microservice architecture wherein data storage is decentralized [29]. Finally, the close coupling of domain entities in a monolith can impede agility and expedited development, in contrast to a modular or microservice architecture [30].

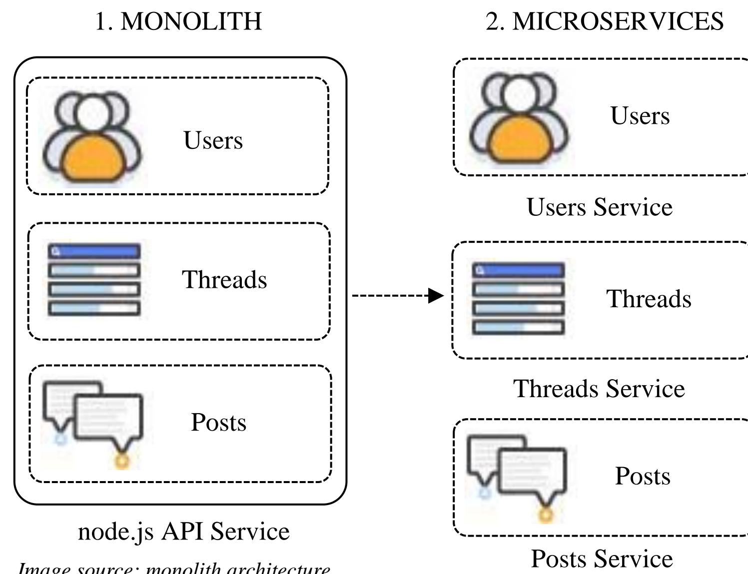


Image source: monolith architecture

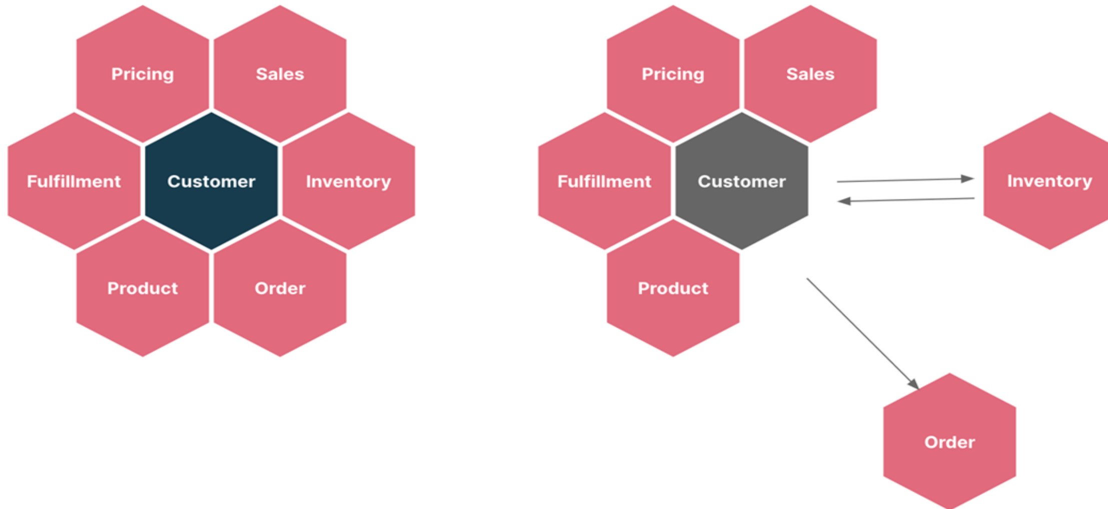
2.3. Modular Monolith Architecture

A modular monolith software architecture entails the division of business entities into separate modules or services while still maintaining a comprehensive domain model. This approach enables the rapid initial development of software but does come with a drawback in terms of performance, as it introduces costs related to the separation of business logic and inter-service communication [3]. Conversely, Monolithic software applications encompass all functionalities within a single deployable unit, rendering them challenging to comprehend and maintain as they mature. On the other hand, microservice architectures advocate for the construction of applications through smaller, loosely coupled functional services [31]. A modular web content architecture presents a separation between renderable content modules and the primary web application, which in turn offers improved flexibility and reusability [32].

The concept of modular monolith software architecture was introduced within the context of transitioning from a monolith to a microservices architecture. The transition to a modular monolith is perceived as an intermediary step in the overall process. This particular approach entails dividing the business logic into separate modules, which are then encapsulated through well-defined interfaces. The benefits of utilizing a modular monolith include the ability to establish a comprehensive domain model that facilitates efficient development and a codebase that is more easily maintainable. However, it is important to note that there are certain trade-offs in performance due to the introduction of inter-service communication. The migration effort and performance challenges associated with transitioning to a modular monolith are already quite significant. Therefore, it is imperative for software architects to carefully consider

these trade-offs and meticulously plan the migration process accordingly [21].

The popularity of modular monolith software architecture can be attributed to several factors. Firstly, it facilitates the incorporation of a comprehensive domain model, where domain entities are closely interconnected, thus enabling rapid development [3]. Secondly, it offers a means to encapsulate business logic within modules and/or services, featuring well-defined interfaces, thereby promoting modularity and reusability [31].



Source - modular monolith

Thirdly, it presents a progressive migration pathway from a monolith to a microservice architecture, allowing for a gradual transition and minimizing the impact on migration effort and performance [32]. Lastly, it tackles the challenges associated with the maintenance and comprehension of aging monolithic applications by advocating for smaller, loosely coupled functional services that are easier to maintain [33].

Utilizing a modular monolith software architecture presents various benefits, such as incorporating intricate domain models and the ability to reuse domain entities. However, it also entails a drawback in terms of performance. The current business landscape emphasizes the importance of agility, which facilitates the separation of corporate units into modules and services. This approach enables organizations to achieve flexibility and promote the reuse of resources [34]. Nevertheless, the task of preserving a monolithic design might present difficulties as applications mature, resulting in complexities in comprehension and upkeep. In contrast, microservice architectures promote the development of systems using smaller, functionally independent services that are loosely connected. This approach has been shown to enhance maintainability [31]. The motivation for transitioning from a monolithic to a modular architecture can be attributed to the requirement for enhanced flexibility and scalability, particularly within the framework of emerging development methodologies such as DevOps [35]. In general, the benefits of employing a modular monolith software architecture are rooted in its adaptability and capacity for reusability. Conversely, the drawbacks encompass potential performance drawbacks and the enduring difficulties associated with sustaining a monolithic architecture throughout its lifespan.

3. Case Studies and Success Stories

Shopify, a major Ruby on Rails codebase, initially used a monolithic architecture for billing, product updates, and delivery. However, the advantages of monolithic architectures outweighed their disadvantages, leading to a shift to a modular monolith. This approach established and respected component boundaries, expediting development and deployment. Monolithic architectures were easy to create and deploy, but they also had drawbacks, such as increased DevOps time and decreased application resiliency and security. Shopify implemented modular monoliths, also known as “Componentization,” to address these issues. The team divided the codebase to improve code organization and adhere to real-world principles. After extensive stakeholder research and input, this was implemented in a single comprehensive pull request. Shopify developed a tool called Wedge to track component isolation and decouple business domains. The tool monitors the progress of each component’s isolation, identifying violations of domain boundaries and data coupling across boundaries. In the long term, the team aims to programmatically enforce limits to ensure each component only imports its explicitly dependent components and eliminates accidental and circular

dependencies. In conclusion, no initial system architecture is without drawbacks, and the choice of software architecture depends on the scale of the application and is always evolving. [8]

Root's expedition in handling the obstacles of a swiftly expanding startup is delineated in this comprehensive article. Initially, they confronted the intricacies of team and technology expansion by adopting a MonolithFirst approach, constructing a greenfield application with a select few engineers. However, anticipating future growth, they transitioned into a Modular Monolith methodology. This technique entailed organizing their Rails project without a central directory, effectively compartmentalizing their code through the utilization of gems and engines. They successfully eradicated circular dependencies by establishing distinct architectural boundaries, implementing a robust test suite, and harnessing tools such as Bundler and the observer pattern. They enhanced the overall architecture of their application.

This Modular Monolith approach enabled them to efficiently manage modifications and improve code clarity by structuring it around domain concepts. It empowered them to effectively differentiate between stateful and stateless logic. Their experience exemplified the simplicity yet potent scalability of the Modular Monolith concept for managing a burgeoning team and evolving software requirements.[38]

In deciding on the architectural framework for a system, it is crucial to carefully evaluate the factors of scalability and domain complexity. This discussion delves into the comparison between modular monoliths and microservices, elucidating the challenges presented by issues like JAR hell and circular dependencies. The Apache Isis framework has emerged as a potential option that effectively addresses cross-cutting concerns, allowing developers to concentrate their efforts on complex business challenges.

The essay emphasizes the importance of implementing organized database management and ensuring coordinated operations among individual modules. Furthermore, this study addresses the selection of platforms for monolithic and microservices-based systems, using Estatio as a case study. Estatio is an invoicing system developed using Apache Isis. The ultimate choice between a monolithic or microservices architecture is contingent upon finding a delicate equilibrium between the domain's complexities and the system's scalability needs. The aforementioned discoveries provide significant contributions to the scholarly discourse within the domain of system architecture.[36]

Google's Service Weaver framework allows writing applications as a modular monolith and deploying them as a set of microservices.

The binary in Service Weaver is organized as a set of modules or components, with all code residing within a singular binary.

Service Weaver splits up the application by components, enabling it to run independently and on distinct machines. [37]

Therefore, the article discusses how Service Weaver enables the development and deployment of applications as modular monoliths, which can then be split into microservices.

4. Conclusion

The choice of software architecture is a critical decision in developing efficient payment systems within the fintech domain. This study has explored three prominent architectural approaches: Monolithic architecture, Microservices architecture, and Modular Monolithic architecture. Each approach has its advantages and limitations, making it essential to assess their suitability for payment system development.

Monolithic architecture, characterized by a single, self-contained codebase, offers advantages in terms of performance and reduced latency, making it suitable for applications with moderate traffic. However, it suffers from scalability and update flexibility limitations, making it less ideal for rapidly evolving systems.

On the other hand, Microservices architecture provides flexibility, scalability, and independence for individual services. It excels in handling increased loads and offers resilience. Yet, it introduces network communication and security complexities, making it challenging for some applications.

Modular Monolithic architecture emerges as a middle ground between the two. It retains the benefits of a comprehensive domain model and modularization while minimizing the complexities associated with microservices. Organizations like Shopify, Eurocommercial Properties (Estatio), Google and Root have successfully adopted this approach to streamline development and maintainability.

The decision to choose a Modular Monolithic architecture as the first choice for developing efficient payment systems depends on various factors. It offers the advantages of clear domain modeling, ease of maintenance, and a gradual path toward microservices adoption. However, it comes with performance trade-offs due to inter-service communication.

In conclusion, while Modular Monolithic architecture presents a promising option, there is no one-size-fits-all solution. The choice should be based on the specific requirements, scalability needs, and complexities of the

payment system in question. It is crucial for organizations to carefully evaluate these factors and consider their long-term goals when selecting the most suitable software architecture. Furthermore, ongoing research and industry best practices will continue to shape the landscape of software architecture, making it imperative for fintech companies to stay informed and adaptable in their approach to payment system development.

References

[1] James Lewis, and Martin Fowler, "Microservices," *Martinfowler.com*, 2014. [Google Scholar] [Publisher Link]

[2] Chandler Harris, "Microservices vs. Monolithic Architecture," *Atlassian*, 2023. [Google Scholar] [Publisher Link]

[3] Diogo Faustino et al., "Stepwise Migration of a Monolith to a Microservices Architecture: Performance and Migration Effort Evaluation," *arXiv*, pp. 1-12, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[4] Len Bass, Paul Clements, and Rick Kazman, *Software Architecture in Practice: Software Architect Practice*, Pearson Education, pp. 1-589, 2012. [Google Scholar] [Publisher Link]

[5] Dewayne E. Perry, and Alexander L. Wolf, "Foundations for the Study of Software Architecture," *ACM Sigsoft Software Engineering Notes*, vol. 14, no.4, pp. 40-52, 1992. [CrossRef] [Google Scholar] [Publisher Link]

[6] Humberto Cervantes, and Rick Kazman, "*Designing Software Architectures: A Practical Approach*, Pearson Education, pp. 1-320, 2016. [Google Scholar] [Publisher Link]

[7] Omar Al-Debagy, and Peter Martinek, "A Comparative Review of Microservices and Monolithic Architectures," *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*, pp. 149-154, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[8] Kirsten Westeinde, Deconstructing the Monolith: Designing Software that Maximizes Developer Productivity, Shopify, 2019. [Online]. Available: https://shopify.engineering/deconstructing-monolith-designing-software-maximizes-developer-productivity

[9] Freddy Tapia et al., "From Monolithic Systems to Microservices: A Comparative Study of Performance," *Applied Sciences*, vol. 10, no. 17, pp. 1-35, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[10] Nuha Alshuqayran, Nour Ali, and Roger Evans, "A Systematic Mapping Study in Microservice Architecture," *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, pp. 44-51, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[11] Luciano Baresi, and Martin Garriga, *Microservices: The Evolution and Extinction of Web Services?*, Microservices, pp. 3-28, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[12] Ran Mo et al., "The Existence and Co-Modifications of Code Clones within or Across Microservices," *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, no. 22, pp. 1-11, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[13] Nicola Dragoni et al., Microservices: Yesterday, Today, and Tomorrow, *Present and Ulterior Software Engineering*, pp. 195-216, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[14] Pooyan Jamshidi et al., "Microservices: The Journey So Far and Challenges Ahead," *IEEE Software*, vol. 35, no. 3, pp. 24-35, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[15] Alan Sill, "The Design and Architecture of Microservices," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 76-80, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[16] Sean Whitesell, Rob Richardson, and Matthew D. Groves, *Introducing Microservices*, Pro Microservices in .NET 6, pp. 1-27, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[17] Mohamed Ibrahim Elkholy, and Marwa A. Marzok, "Trusted Microservices: A Security Framework for Users' Interaction with Microservices Applications," *Journal of Information Security and Cybercrimes Research*, vol. 5, no. 2, pp. 135-143, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[18] Jeremy M.R. Martin, "Designing and Verifying Microservices Using CSP," *2021 IEEE Concurrent Processes Architectures and Embedded Systems Virtual Conference (COPA)*, pp. 1-4, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[19] Simon Eismann et al., "Microservices: A Performance Tester's Dream or Nightmare?," *Proceedings of the ACM/SPEC International Conference on Performance Engineering*, pp. 138-149, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[20] Khaled Sellami et al., "Improving Microservices Extraction Using Evolutionary Search," *Information and Software Technology*, vol. 151, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[21] Catalin Strimbei et al., "Software Architectures-Present and Visions," *Informatica Economică*, vol. 19, no. 4, pp. 13-27, 2015. [CrossRef] [Google Scholar] [Publisher Link]

[22] Luís Nunes, Nuno Santos, and António Rito Silva, *From a Monolith to a Microservices Architecture: An Approach Based on Transactional Contexts*, European Conference on Software Architecture, pp. 37-52, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[23] Judith A. Stafford, and Alexander L. Wolf, "Software architecture," *Component-Based Software Engineering: Putting the Pieces Together*, pp. 371-387, 2001. [Google Scholar] [Publisher Link]

[24] David Garlan, and Dewayne Perry, "Introduction to the Special Issue on Software Architecture," *IEEE Transactions on Software Engineering*, vol. 21, no. 4, pp. 269-274, 1995. [Google Scholar] [Publisher Link]

[25] Rikard Land, "A Brief Survey of Software Architecture," *Mälardalen Real-Time Research Center (MRTC) Report*, pp. 1-15, 2002. [Google Scholar] [Publisher Link]

[26] Robert L. Glass, "Silver Bullet" Milestones in Software History," *Communications of the ACM*, vol. 48, no. 8, pp. 15-18, 2005. [CrossRef] [Google Scholar] [Publisher Link]

[27] Nabor C. Mendonça et al., "The Monolith Strikes Back: Why Istio Migrated from Microservices to a Monolithic Architecture," *IEEE Software*, vol. 38, no. 5, pp. 17-22, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[28] Salvatore Augusto Maisto, Beniamino Di Martino, and Stefania Nacchia, *From Monolith to Cloud Architecture Using Semi-Automated Microservices Modernization*, International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, Advances on P2P, Parallel, Grid, Cloud and Internet Computing, vol. 96, pp. 638-647, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[29] Miika Kalske, Niko Mäkitalo, and Tommi Mikkonen, *Challenges when Moving from Monolith to Microservice Architecture*, International Conference on Web Engineering, Current Trends in Web Engineering, volume 10544, pp. 32-47, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[30] Justas Kazanavičius, Dalius Mažeika, and Diana Kalibatienė, "An Approach to Migrate a Monolith Database into Multi-Model Polyglot Persistence Based on Microservice Architecture: A Case Study for Mainframe Database," *Applied Sciences*, vol. 12, no. 12, pp. 1-29, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[31] Alex Mathai et al., "Monolith to Microservices: Representing Application Software through Heterogeneous Graph Neural Network," *arXiv*, pp. 1-15, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[32] Justin Scott Lowery, and Frank Anthony Nuzzi, "Modular Web Content Software Architecture," *United States Patent Application*, pp. 1-13, 2019. [Google Scholar] [Publisher Link]

[33] Cody Allard et al., "Modular Software Architecture for Fully Coupled Spacecraft Simulations," *Journal of Aerospace Information Systems*, vol. 15, no. 12, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[34] Nuno Gonçalves et al., "Monolith Modularization Towards Microservices: Refactoring and Performance Trade-offs," *2021 IEEE 18th International Conference on Software Architecture Companion (ICSA-C)*, pp. 1-8, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[35] Mohammad Raji, and Behzad Montazeri, "On the Relationship between Modularity and Stability in Software Packages," *arXiv*, pp. 1-4, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[36] Dan Haywood, In Defence of the Monolith, Part 2, InfoQ, 2017. [Online]. Available: https://www.infoq.com/articles/monolith-defense-part-2/

[37] Matt Campbell, Google Service Weaver Enables Coding as a Monolith and Deploying as Microservices, InfoQ, 2023. [Online]. Available: https://www.infoq.com/news/2023/03/google-weaver-framework/

[38] Dan Manges, The Modular Monolith: Rails Architecture, Medium, 2018. [Online]. Available: https://medium.com/@dan_manges/the-modular-monolith-rails-architecture-fb1023826fc4

[39] Rahul Garg, When (Modular) Monolith is the Better Way to Build Software, Thoughtworks, 2023. [Online]. Available: https://www.thoughtworks.com/en-us/insights/blog/microservices/modular-monolith-better-way-build-software