Journal of
Innovation, Technology and Sustainability

Modernising legacy enterprise resource planning systems using the microservices architecture: A review

Antony Irungu Njina[1]

Samson Wanjala Munialo[1]

[1]*School of Computing and Informatics, Meru University of Science and Technology*

Abstract

Enterprise Resource Planning (ERP) systems are centralised data management software, tools, and technologies that enhance collaboration and communication between departments while reducing duplication, errors, and inconsistencies, leading to better accuracy and productivity. However, over time, legacy ERP systems have become cumbersome and difficult to maintain and adapt to new technologies. This paper presents a review of microservices architecture (MSA) as a strategy for modernizing legacy ERP systems. It synthesises literature to highlight the core principles, design patterns, challenges, and best practices associated with MSA. The analysis reveals that MSA significantly enhances scalability, flexibility, and maintainability by decomposing monolithic ERP systems into modular, independent services. Key findings include the benefits of service autonomy, decentralised data management, and API-driven communication in overcoming the limitations of traditional ERP architectures. The paper concludes by highlighting the advantages of MSA in ERP modernisation.

Keywords: *Microservices architecture, enterprise resource planning, scalability, modularity, deployment, design patterns, application programming interface*

1. Introduction

The monolithic design of enterprise systems yields a single, unified software platform in which all business processes and functions are tightly integrated within one cohesive application. All modules share the same codebase and database, which means that any change made to one part of the system can potentially affect the entire platform. In the past, this design was initially attractive, as it provided an all-in-one solution for businesses. However, over time, the rigid nature of these systems has increasingly become a liability given the rapid evolution of technology and business environments. Knoche and Hasselbring (2018) and Hasan et al. (2023) observe that many legacy applications, often built as monolithic architectures, face significant challenges in terms of scalability, flexibility, and maintainability. Modifications or updates to one module often require extensive rework or testing across the entire system, leading to long deployment cycles and operational disruptions. Integration of new features, like cloud-based services or third-party tools, can also be particularly difficult due to the tightly coupled nature of the components, which hampers the system's ability to evolve alongside emerging technologies (Oyeniran et al., 2024).

As the system grows and new business needs emerge, users often find it difficult to modify or extend the system without affecting other parts of the platform. Customisations, which are a common requirement for adapting ERP systems to specific industry or organisational needs, often lead to further entanglement in the system's architecture, making future upgrades or maintenance even more difficult. These customisations can also create versioning conflicts, as updates to the core system might overwrite or break user-specific modifications. This design affects the scalability of the systems, since expanding or modifying the system to accommodate increased data or more users generally requires scaling the entire platform, which can be resource-intensive and inefficient.

According to Knoche and Hasselbring (2018) Microservices offer a modular and scalable approach that can address the limitations inherent in monolithic ERP systems. They allow for incremental refactoring of legacy systems, enabling organisations to modernize their IT infrastructure without completely overhauling existing systems. Microservice Architecture (MSA) involves breaking down a large application into smaller, independent services, each of which is focused on a specific business function or process. Each service communicates with other services through lightweight protocols, typically using an Application Programming Interface (API) or messaging queues.

MSA decouples individual components of a large system, offering businesses the flexibility to update, scale, and maintain specific services independently (based on demand) without affecting the entire system (Razzaq & Ghayyur, 2023). For example, if the accounting module of the ERP system faces increased usage during peak periods, only the accounting service needs to be scaled. Similar views are shared by Aljawawdeh et al.

(2023) regarding enhanced modularity, agility, scalability, and easier maintenance offered by MSA.

Integration of legacy systems with other modern technologies, such as cloud platforms, mobile applications, or big data analytics tools, can be a significant challenge (Barua & Kaiser, 2024). Microservices are designed to communicate through APIs and standardised protocols, making them highly compatible with modern systems (Daniel et al., 2023). Modern organisation systems are increasingly relying on external tools and platforms such as collaboration and communication tools, data analytics and Business Intelligence tools, and payment platforms, among others. The compatibility of MSA becomes a critical advantage that allows systems to seamlessly interact with a wide variety of applications and services.

Hasan et al. (2023) argue that microservices enable faster deployment cycles, simpler debugging, and more efficient updates. The flexibility of MSA ensures that an ERP system can evolve alongside technological advancements and changing business needs. MSA supports faster development cycles, since it is compatible with modern development practices such as DevOps, continuous integration, and continuous delivery (CI/CD). This enables businesses to deploy updates more frequently and confidently (Knoche & Hasselbring, 2018).

2. Methodology

This narrative review on MSA designs employed a systematic and structured approach to identify, analyse and synthesize relevant literature from academic databases and case studies. Studies included in this review were required to address Microservices Architecture (MSA) designs, principles, patterns, challenges, and best practices for modernising legacy ERP systems. Exclusion criteria filtered out outdated, irrelevant, or redundant studies. The literature search was conducted across multiple academic databases, including IEEE Xplore, ACM Digital Library, and Google Scholar. A structured search strategy was employed using keywords such as “microservices architecture,” “design patterns,” “scalability,” “modularity,” and “deployment.” Boolean operators and filters were applied to refine the search results. The selection process involved screening titles and abstracts for relevance, followed by a full-text review of potentially eligible studies. The PRISMA flow diagram was used to map out the number of records identified, included, and excluded, along with reasons for exclusions.

Data extraction was performed using a standardised form to capture key information from each study, including study design, sample size, MSA principles, design patterns, challenges, and best practices. Thematic analysis was used to categorize the extracted data into themes such as design principles, design patterns, transition strategies, and best practices. Findings were synthesised into a coherent narrative. The risk of bias in included studies was assessed using the ROBIS tool, ensuring the reliability and validity

of the findings. Subgroup analyses were conducted to explore variations in findings based on study characteristics, such as year and geographic location.

3. Microservices for Modernising Legacy ERP Systems

Core Principles

Velepucha and Flores (2023) present a comprehensive survey on MSA, focussing on its core principles, design patterns, and the challenges organisations face when migrating from monolithic systems. They begin by outlining the foundational principles of microservices, such as decentralisation, service autonomy, scalability, and the need for services to communicate via lightweight protocols like RESTful APIs. According to Hippchen et al. (2017), the principles guide developers in creating microservices that are more modular, cohesive, and closely aligned with the business logic, which enhances maintainability and scalability. Key concepts such as bounded contexts, aggregates, and entities can guide the design of microservices that are not only functionally coherent but also decoupled from other services, making them easier to develop, test, and deploy independently.

Design Patterns

In the early 1980s, distributed systems used Remote Procedure Calls (RPC), which allowed programmes to execute code on remote systems as if they were local. This was followed by technologies such as DCE (1988) and CORBA (1991), which aimed to make remote calls transparent to developers. In the late 1990s and early 2000s, Service Oriented Architecture (SOA) emerged as a means of designing software using loosely coupled services. SOA allowed different services to communicate over a network using standardised protocols, but it often resulted in complex and heavy-weight implementations.

The term “microservices” was popularized in the early 2010s. It represented a shift towards building applications as a collection of small, autonomous services that could be developed, deployed, and scaled independently. This approach was influenced by earlier concepts such as Unix philosophy, which advocates for building simple, modular tools. Earlier writings by Fowler (2003) and Hohpe and Woolf (2004), and experiences of companies like Netflix and Amazon helped shape the microservices landscape. Over the years, several design patterns have emerged and continue to evolve, addressing new challenges and leveraging advances in cloud computing and containerization.

Design patterns are strategies that are employed to solve common problems in the development and maintenance of MSA, allowing efficient development and management of independent services (Shaikh & Agaskar, 2022). They are like templates that can be used to create microservices applications that are rooted in the need to address the limitations of monolithic architecture. Although microservices offer significant scalability

benefits, maintainability and flexibility, effective design and careful implementation are key to maximising the potential of cloud-native applications.

Hippchen et al. (2017) explore the application of Domain-Driven Design (DDD) in the context of designing microservice-based applications. They argue that DDD provides a structured approach to decompose complex systems into smaller, more manageable services by aligning microservices with business domains. Velepucha & Flores (2023) explore the various design patterns used in MSAs, including API gateway, service registry, and circuit breaker patterns, which help manage complexity, enhance fault tolerance, and facilitate service discovery in distributed systems. They also provide a detailed analysis of the challenges and benefits of using DDD in microservice architecture. While DDD helps reduce complexity by fostering a clear separation of concerns, it also presents challenges in terms of integrating disparate services and managing interservice communication. Hippchen et al. (2017) highlights the importance of defining clear service boundaries and ensuring that each microservice has its own data store, which aligns with the DDD principle of encapsulating business logic and data. They also underscore the role of collaboration between domain experts and development teams in successfully implementing DDD for microservices.

Söylemez et al. (2022) identified the use of design patterns such as service discovery, API gateways, and event-driven architectures as effective approaches for managing inter-service communication and ensuring fault tolerance. For data management, they recommend adopting strategies such as eventual consistency, database per service, and Command Query Responsibility Segregation (CQRS) to handle the complexities of distributed data. They advocate for a service mesh architecture and robust authentication mechanisms to safeguard communication between services. However, they advise the need for further research on how to integrate these solutions into comprehensive frameworks and develop more effective tools for managing MSAs at scale.

Daniel et al. (2023) explore the design of adaptable, future-ready MSAs by analysing various microservice patterns that enhance system flexibility and long-term sustainability. They identify several microservice patterns such as domain-driven design, event sourcing, and API versioning that promote adaptability by decoupling services, ensuring that individual components can evolve without disrupting the entire system. The study stresses the importance of designing microservices with forward compatibility, allowing the system to integrate new features or third-party technologies with minimal disruption.

Ataei and Staegemann (2023) explore the application of microservices architectural patterns to big data systems, focusing on how microservices can address the unique challenges of scalability, flexibility, and manageability in large-scale data processing environments. The authors review several core microservices patterns, such as service decomposition, data management, and inter-service communication, and evaluate how these patterns can be applied to big data architectures. They highlight the synergy

between microservices and big data systems, particularly in enabling modularity and allowing data systems to scale more effectively. The paper emphasises that by adopting microservices, organisations can improve the responsiveness and fault tolerance of their big data systems, reduce complexity, and facilitate easier maintenance and updates. The authors also discuss the role of containerisation and cloud technologies in supporting the deployment and management of microservices within big data infrastructures.

Oyeniran et al. (2024) explore MSA in the development of cloud-native applications, focusing on design patterns and scalability. The authors explain how these applications, which are designed to run in dynamic cloud environments, benefit from MSA by offering scalability, flexibility, and improved fault tolerance. They discuss key design patterns in microservices, such as service discovery, API gateway, Circuit Breaker, and Strangler Fig patterns, which help manage the complexity of distributed systems. These patterns enable efficient service decomposition, dynamic load balancing, and fault isolation, which are critical for handling large-scale applications in the cloud. The authors also stress the importance of containerisation technologies, like Docker, orchestration tools, such as Kubernetes, and integration with cloud platforms like AWS, Azure, and GCP to facilitate the deployment, scaling, and management of microservices in cloud environments.

Esparza-Peidro et al. (2024) explore the methodologies and approaches for effectively modelling microservice architectures, with a focus on creating scalable, maintainable, and well-structured systems. They discuss the importance of modelling as a critical step in the design phase of microservice-based systems, where decisions about service boundaries, inter-service communication, and data management must be carefully considered. They present various modelling techniques, including DDD and service-oriented modelling, that help to decompose complex applications into smaller, manageable microservices. Their paper emphasises that a well-defined model facilitates easier communication between development teams, a clearer understanding of system behaviour, and a more efficient implementation of microservice patterns. The authors also highlight the role of visualisation tools in supporting the modeling process, allowing for a clearer representation of the system architecture and dependencies.

Barua and Kaiser (2024) discuss the technical aspects of a cloud-enabled MSA focussing on key components such as service orchestration, API management, and data consistency across distributed services. They emphasise the use of containerization technologies, such as Docker, and orchestration platforms, such as Kubernetes, to automate service deployment and management. The research also addresses critical challenges, such as maintaining data consistency across distributed services and ensuring seamless communication between microservices in a cloud environment. Through the application of modern DevOps practices and continuous integration/continuous deployment (CI/CD) pipelines, the authors argue that the proposed architecture not only enhances the efficiency of the reservation system but also allows for greater flexibility in terms of system upgrades and feature expansion.

Use Cases

Minakova et al. (2022) presented a design for a location-based microservice architecture aimed at enhancing the management and coordination of university activities. The proposed system organises various university functions, such as event management, room bookings, and campus navigation, based on real-time location data from students and visitors. The system decomposed the university's complex services into independent, location-aware microservices, to allow for greater flexibility, scalability, and maintainability. The paper emphasises how this microservice approach allows for better coordination between different departments and services, allowing students and faculty to easily access information related to on-campus activities, such as lectures, meetings, and university events. The use of location-based services ensures that the system can provide personalised, context-aware information to users, such as directions to specific locations or real-time updates on room availability. Their microservice architecture is divided into four separate components, each handling different aspects of the navigation process, and is integrated with the university's information system to manage schedules, event venues, and equipment availability. The authors also discuss the technical aspects of microservice architecture, including service decomposition, communication protocols, and integration with existing university systems.

Oyeniran et al. (2024) discuss the scalability aspects of microservices in cloud-native applications. They argue that due to their independent and loosely coupled nature, microservices are inherently more scalable than monolithic applications. Cloud-native MSAs enable services to scale horizontally, rendering them efficient in handling varying workloads and traffic spikes. They address challenges in managing data consistency across distributed services and ensuring seamless communication between microservices by using event-driven architectures and eventual consistency models.

In MSA, service visibility is crucial for the effective management, monitoring, and scaling of distributed systems. Tokmak et al. (2024) argue that as MSAs grow in complexity, with many loosely coupled services, it becomes increasingly difficult to ensure that services are discoverable, trackable, and observable. They address the challenge of discovering and routing traffic using service discovery tools to dynamically locate the IP address and port number of microservices, ensuring efficient routing and load balancing. The authors highlight the role of service mesh technologies, API gateways, and centralised logging systems in providing visibility into inter-service communication, performance metrics, and system health.

Tokmak et al. (2024) also propose several strategies to improve service visibility, including the adoption of observability frameworks and real-time monitoring tools that aggregate data from multiple microservices. They emphasise the importance of integrating distributed tracing systems such as Open Telemetry, which allows for end-to-end tracking of requests as they pass through various services, providing insights into service interactions and latency issues. They discuss how this visibility can also be

extended to security by identifying potential vulnerabilities through continuous monitoring. The study recommends a combination of architectural patterns and tools to ensure that all microservices within an architecture are properly monitored and visible, enabling better system maintenance, troubleshooting, and scaling.

Although microservices offer significant potential for improving IoT system architectures, there is a growing need to focus more on design, integration, and security. Siddiqui et al. (2023) review the use of microservices-based architectures for the Internet of Things (IoT) systems. Their study examines the strengths, weaknesses, and opportunities of using microservices to address the functional and non-functional challenges associated with large-scale IoT connectivity. They also highlight how microservices can enhance fault tolerance and improve the overall reliability of IoT systems, as failures in one service do not necessarily impact the entire system. The review includes a taxonomy for microservices-based IoT systems that offers a detailed picture of the current landscape. It also discussed the integration of microservices with edge computing and cloud platforms, which are often crucial for processing and analysing data generated by IoT devices. The study identified key issues such as service orchestration, efficient communication between microservices, and ensuring data consistency. The study highlights the need for robust security mechanisms to protect against potential vulnerabilities in microservices-based IoT systems, especially in distributed environments.

Barua and Kaiser (2024) propose a cloud-enabled MSA designed specifically for next-generation online airline reservation systems. They observe that traditional monolithic architectures for airline reservation systems can be divided into smaller, independent services, each focussing on a specific business function such as flight booking, customer management, or payment processing. The study outlines how these services can be hosted in the cloud, leveraging elastic scalability to efficiently handle fluctuating workloads, and ensuring that the system can adapt to peak and off-peak demands. Their work suggests that cloud-enabled microservices can provide the foundation for more agile, scalable, and resilient systems. The benefits of such cloud-native microservices include improved fault tolerance, reduced downtime, and the ability to rapidly deploy new features and updates.

Challenges of Microservices Architecture

Although microservices offer significant benefits in terms of system effectiveness and scalability, careful attention to security is crucial to mitigate the risks associated with their decentralised and interconnected nature. Söylemez et al. (2022) categorise the challenges associated with MSA into several key areas, including service decomposition, inter-service communication, data management, and system security. They note that the primary challenge lies in properly decomposing monolithic applications into microservices while ensuring that the resulting services are cohesive, loosely coupled, and manageable. The complexity of managing communication between numerous

microservices, particularly in terms of latency and consistency, is the next significant hurdle. In addition, there are difficulties related to maintaining data consistency and integrity across distributed services, as well as the security concerns that arise from the decentralised nature of microservices-based systems. While working with location-based services, Minakova et al. (2022) also noted that ensuring data consistency across distributed services, managing service dependencies, and dealing with potential latency issues are the key challenges.

Razzaq and Ghayyur (2023) also identified the key challenges associated with adoption of microservices, and how these challenges are compounded by the cultural and organisational shifts required for successful adoption, such as the need for a DevOps mindset and a shift towards agile methodologies. They noted that there is a lack of comprehensive frameworks for guiding organisations through the migration process.

Velepucha and Flores (2023) highlight the significant challenges organisations face during the migration to microservices. One of the primary challenges is decomposing legacy monolithic systems into manageable independent services without disrupting ongoing operations. Other challenges are the difficulties related to data management, consistency, and inter-service communication, which can become more complex as the number of microservices increases. In addition, organizational and cultural changes are required to implement microservices, such as fostering collaboration between cross-functional teams and adopting new DevOps practices. To overcome the challenges, the study recommends adopting a gradual, iterative migration approach, using containerisation for deployment, and leveraging cloud-native technologies.

Ataei and Staegemann (2023) examine the potential risks and challenges of integrating microservice patterns with big data systems, such as issues with data consistency, latency, and the complexity of managing distributed services. They provide a detailed analysis of several case studies where organisations have successfully implemented microservices in big data contexts, offering valuable insights into the best practices for overcoming these challenges. The study suggests that microservices can enable more efficient handling of large volumes of data by decentralising processing tasks and allowing for independent scaling of services. However, they caution that the adoption of microservices in big data systems requires careful consideration of service orchestration, data synchronisation, and inter-service communication mechanisms to avoid pitfalls like data fragmentation and performance bottlenecks.

Matias et al. (2024) explore the effectiveness and security of MSAs, focussing on the trade-offs between scalability, agility, and security concerns. The study highlights how the adoption of microservices has transformed the architecture of extensive systems, that focus on prioritising distinct responsibilities and enabling the division into smaller, more manageable components. However, this transition also introduces new security vulnerabilities due to the increased presence of multiple smaller elements on the Internet. The authors examine the security challenges inherent in MSAs, particularly

regarding service isolation, authentication, and authorisation. They also identify several key security risks, such as increased attack surface due to multiple services communicating over networks and the difficulty in managing consistent security policies across distributed services. To address these issues, they recommend a set of security best practices, including service mesh architectures, API Gateway pattern, and strong encryption techniques for data in transit. They also explore the use of Hyper Text Transfer Protocol Secure (HTTPS) and diverse data payloads to mitigate security vulnerabilities.

Esparza-Peidro et al. (2024) address the challenges of modelling microservices, particularly when it comes to maintaining consistency and managing dependencies in a distributed system. They identify key aspects such as service coordination, transaction management, and API design as critical components to consider during the modelling process. The authors suggest the use of architectural decision records (ADRs) to document and justify design choices, which can help maintain consistency across a microservice ecosystem over time.

Transition Strategies and Best Practices

The use of microservices is a strategy that has been identified for modernizing legacy software systems. According to Razzaq and Ghayyur (2023) many organisations struggle with the complexity of adopting them, particularly in terms of system decomposition, service management, and ensuring efficient inter-service communication. Knoche and Hasselbring (2018) provide detailed overview techniques such as service decomposition, data partitioning, and the adoption of cloud-native infrastructure. They also highlight several key challenges associated with using microservices for modernising legacy software, including managing inter-service communication, ensuring data consistency across distributed services, and maintaining backward compatibility with legacy systems. The authors present a set of guidelines and best practices to mitigate these challenges, such as adopting a gradual migration strategy, leveraging API gateways, and using event-driven architectures to manage service interactions. They also caution that, in the absence of careful planning, the transition to microservices can lead to disruptions in business operations. Although this endeavour is complex and resource intensive, it provides significant long-term benefits in terms of system agility, scalability, and maintainability, making it a viable strategy for legacy software modernisation in a rapidly evolving technology landscape.

Wolfart et al. (2021) present a detailed roadmap for modernising legacy systems by transitioning to MSAs. The proposed roadmap is structured into the initial, planning, execution, and monitoring phases that comprise eight activities. It begins with an assessment of the architecture of the legacy system, business requirements, and technical constraints. This creates an understanding of the limitations of the existing system and identifies areas where microservices can provide the most value, such as improving modularity and enabling independent scaling. The study also stresses the role of

stakeholders, including IT teams and business leaders, in ensuring that the migration plan aligns with organisational goals and capabilities. Findings from Hayretci and Aydemir (2021) reveal that successful migration of legacy banking systems requires not only technological expertise, but also effective change management practices to mitigate risks and ensure smooth transitions. Some of the best practices identified in this study include the phased migration approach, the use of hybrid systems during transition periods, and the importance of maintaining high system availability.

Wolfart et al. (2021) explore the technical and operational challenges of migrating to microservice-based architecture. They address key issues such as service decomposition, data management, and inter-service communication. They discuss transition strategies to ensure that legacy systems can continue to operate during the transition. The authors also examine the importance of adopting modern tools and practices, such as containerisation and continuous integration. The study advocates for a phased migration strategy, which allows for gradual implementation, minimises disruption to business operations, and provides time for the organisation to adapt to new technologies and methodologies. Similar suggestions are shared by Tuusjärvi (2021) in a systematic mapping of research literature on the migration of legacy software systems to MSA.

Hayretci and Aydemir (2021) present a multicase study on the strategies involved in migrating legacy systems in the banking industry to modern IT infrastructures. The study focusses on several banking institutions that have undergone the complex process of migrating their legacy systems, which are often monolithic and rigid, into more flexible and scalable architectures. The authors identify key challenges in the migration process, such as data integration, system downtime, and resistance to change from employees accustomed to legacy systems. They also discuss the technical difficulties in aligning legacy applications with new technologies such as cloud computing, microservices, and agile development methodologies. The study illustrates the various approaches banks have taken to address these challenges, highlighting the importance of a well-planned migration strategy that includes a thorough risk assessment and stakeholder participation. The study also discusses the financial and operational benefits of migration of legacy systems, such as improved efficiency, enhanced customer service capabilities, and the ability to take advantage of new digital technologies for innovation.

Zhou et al. (2023) present an industrial investigation of the real-world practices and challenges of implementing microservice architecture in large-scale organisations. The authors examine how companies have adopted microservices and the various benefits they have realised, including improved scalability, flexibility, and the ability to quickly adapt to business requirements. The study identifies several successful practices, such as the use of continuous integration/continuous delivery (CI/CD) pipelines, the implementation of service mesh technologies, and the adoption of DevOps methodologies, which have helped streamline development and deployment. However, the authors also highlight the complexities involved such as service decomposition, inter-service communication, and the management of distributed data. In addition, industry

professionals cite issues related to monitoring, debugging, and maintaining microservices in production. The authors emphasise that these challenges are often exacerbated by a lack of mature tooling and insufficient experience with microservices within development teams. Based on their findings, Zhou et al. (2023) suggest that organisations should approach microservices adoption incrementally, focussing on building the necessary skills, infrastructure, and operational processes to support the architecture.

Daniel et al. (2023) provide a detailed evaluation of best practices for addressing challenges, such as the use of event-driven architectures to handle asynchronous communication and maintain data integrity. They emphasise the role of continuous integration and continuous deployment (CI/CD) pipelines in enabling seamless updates and reducing the risks associated with system changes.

Abgaz et al. (2023) discuss the various strategies and frameworks that organisations use to facilitate the decomposition process, such as domain-driven design and event-driven architecture. They identify critical success factors, including effective change management, the role of automated testing and continuous integration, and the importance of team collaboration in the adoption of microservices. Furthermore, the authors explore the impact of this architectural shift on software development practices and organisational structure, noting that microservices demand a cultural shift towards DevOps and agile methodologies.

The effective transition from monolithic to microservice architecture demands better tools, frameworks, and training. Razzaq and Ghayyur (2023) evaluated the transition from monolithic to microservice architectures, focussing on awareness of the shift and the challenges faced by organisations during this transition. Esparza-Peidro et al. (2024) discusses the importance of scalability and fault tolerance in the design of microservice models, stressing the need for flexibility in the face of changing business needs. The study proposes modularization, clear separation of concerns, and proactive consideration of system performance and reliability as some of the best practices for microservice modelling.

4. Conclusion

Microservices offer a solution by decoupling monolithic systems into smaller and more manageable components that can evolve independently. The need to modularize legacy systems using MSA is driven by the demands for scalability, flexibility, and integration in modern business environments. The advantages of MSA include enhanced scalability, faster time to market, easier maintenance, and improved integration, which makes them as a viable alternative to inflexible legacy systems. However, the transition to microservices introduces challenges, particularly in terms of data management, complexity, and skill requirements. Despite these challenges, the benefits of modularization through microservices make it a compelling strategy for businesses

seeking to modernise their existing systems. The choice of MSA design and transition strategy is dictated by the specific needs of an organisation after careful planning and consideration. Although MSA presents an approach to modernizing legacy systems, future work should focus on empirical studies comparing the performance, scalability, and maintenance costs of legacy monolithic systems versus modularised microservices architectures to provide valuable insights into the practical benefits of MSA adoption.

References

Abgaz, Y., McCarren, A., Elger, P., Solan, D., Lapuz, N., Bivol, M., Jackson, G., Yilmaz, M., Buckley, J., & Clarke, P. (2023). Decomposition of monolith applications into microservices architectures: A systematic review. *IEEE Transactions on Software Engineering*, *49*(8), 4213–4242.

Ahmed Shaikh, K., & Agaskar, S. S. (2022). Microservices Design Patterns. In *Azure Kubernetes Services with Microservices : Understanding Its Patterns and Architecture* (pp. 61–101). Apress. https://doi.org/10.1007/978-1-4842-7809-3_3

Aljawawdeh, H., Sabri, M., & Maghrabi, L. (2023). Toward Serverless and Microservices Architecture: Literature, Methods, and Best Practices. In *Artificial Intelligence, Internet of Things, and Society 5.0* (pp. 573–584). Springer.

Ataei, P., & Staegemann, D. (2023). Application of microservices patterns to big data systems. *Journal of Big Data*, *10*(1), 56.

Barua, B., & Kaiser, M. S. (2024). *Cloud-Enabled Microservices Architecture for Next-Generation Online Airlines Reservation Systems*.

Daniel, J., Wang, X., & Guerra, E. (2023). How to design Future-Ready Microservices? Analyzing microservice patterns for Adaptability. *Proceedings of the 28th European Conference on Pattern Languages of Programs*, 1–7.

Esparza-Peidro, J., Muñoz-Esco\'\i, F. D., & Bernabéu-Aubán, J. M. (2024). Modeling microservice architectures. *Journal of Systems and Software*, *213*, 112041.

Fowler, M. (2003). *Patterns of enterprise application architecture*. Addison-Wesley.

Hasan, M. H., Osman, M. H., Novia, I. A., & Muhammad, M. S. (2023). From Monolith to Microservice: Measuring Architecture Maintainability. *International Journal of Advanced Computer Science and Applications*, *14*(5).

Hayretci, H. E., & Aydemir, F. B. (2021). A Multi Case Study on Legacy System Migration in the Banking Industry. In S. and T. E. La Rosa Marcello and Sadiq (Ed.), *Advanced Information Systems Engineering* (pp. 536–550). Springer International Publishing.

Hippchen, B., Giessler, P., Steinegger, R., Schneider, M., & Abeck, S. (2017). Designing microservice-based applications by using a domain-driven design approach. *International Journal on Advances in Software*, *10*(3 & 4), 432–445.

Hohpe, G., & Woolf, B. (2004). *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley Professional.

Knoche, H., & Hasselbring, W. (2018). Using microservices for legacy software modernization. *IEEE Software*, *35*(3), 44–49.

Matias, M., Ferreira, E., Mateus-Coelho, N., Ribeiro, O., & Ferreira, L. (2024). Evaluating Effectiveness and Security in Microservices Architecture. *Procedia Computer Science*, *237*, 626–636.

Minakova, O. V., Akamsina, N. V., & Deniskina, A. (2022). Designing Location-Based Microservice for University Activities. In G. L. and V. T. Solovev Denis B. and Kyriakopoulos (Ed.), *SMART Automatics and Energy* (pp. 651–660). Springer Nature Singapore.

Oyeniran, C. O., Adewusi, A. O., Adeleke, A. G., Akwawa, L. A., & Azubuko, C. F. (2024). Microservices architecture in cloud-native applications: Design patterns and scalability. *Computer Science & IT Research Journal*, *5*(9), 2107–2124.

Razzaq, A., & Ghayyur, S. A. K. (2023). A systematic mapping study: The new age of software architecture from monolithic to microservice architecture—awareness and challenges. *Computer Applications in Engineering Education*, *31*(2), 421–451.

Siddiqui, H., Khendek, F., & Toeroe, M. (2023). Microservices based architectures for IoT systems-State-of-the-art review. *Internet of Things*, 100854.

Söylemez, M., Tekinerdogan, B., & Kolukısa Tarhan, A. (2022). Challenges and Solution Directions of Microservice Architectures: A Systematic Literature Review. *Applied Sciences*, *12*(11). https://doi.org/10.3390/app12115507

Tokmak, A. V., Akbulut, A., & Catal, C. (2024). Boosting the visibility of services in microservice architecture. *Cluster Computing*, *27*(3), 3099–3111.

Tuusjärvi, K. (2021). *Modernization of legacy systems and migrations to microservice architecture*.

Velepucha, V., & Flores, P. (2023). A survey on microservices architecture: Principles, patterns and migration challenges. *IEEE Access*.

Wolfart, D., Assunção, W. K. G., da Silva, I. F., Domingos, D. C. P., Schmeing, E., Villaca, G. L. D., & Paza, D. do N. (2021). Modernizing Legacy Systems with Microservices: A Roadmap. *Proceedings of the 25th International Conference on Evaluation and*

Assessment in Software Engineering, 149–159. https://doi.org/10.1145/3463274.3463334

Zhou, X., Li, S., Cao, L., Zhang, H., Jia, Z., Zhong, C., Shan, Z., & Babar, M. A. (2023). Revisiting the practices and pains of microservice architecture in reality: An industrial inquiry. *Journal of Systems and Software*, *195*, 111521.