

Modernization of Legacy Systems to Microservice Architecture: A Tertiary Study

Nathalia Rodrigues de Almeida, Gabriel Nagassaki Campos,
Fernando Rodrigues de Moraes and Frank José Affonso^a

*Department of Statistics, Applied Mathematics and Computation, São Paulo State University – UNESP,
PO Box 178, Rio Claro, São Paulo, 13506-900, Brazil*

Keywords: Modernization, Monolithic System, Microservice Architecture, Systematic Mapping Study.

Abstract: Modernization of legacy systems to Microservice Architecture (MSA) has been a subject of interest in both academia and industry. MSA facilitates the development of systems by composing them as a collection of small, loosely coupled, and independent services. Despite the importance of this architectural style for system modernization, there is a lack of comprehensive study of how the modernization process must be conducted, regarding the main steps that must be executed, besides architectural patterns that may be used during this process. The goal of this paper is to provide a panorama and analysis, providing a panorama on the modernization of legacy systems to MSA. To do so, we conducted a tertiary study following the guidelines of a Systematic Mapping Study, based on 20 secondary studies. Our overview reveals some evidence related to modernization, such as the main motivations, the main architectural patterns, and the macro steps of a process. Moreover, a discussion of the main findings is addressed in our study. As the study presented in this paper addresses a research theme that is constantly evolving, the presented panorama may be an important guide for researchers and practitioners in the development of future work to advance and consolidate such theme.


1 INTRODUCTION

Nowadays, software systems have played an important role in our society, operating in various segments, such as public and private institutions, airports, and communication systems. In parallel, a notable increase in the complexity of these systems and their computational environments has been observed in recent years (Lewis and Fowler, 2014). In a scenario not far, system integration was a feasible alternative to expand the accessibility of legacy systems to a larger user base, achieved simply by modernizing its interface. According to Pressman and Maxim (2019), legacy can be understood as a system developed with obsolete computing resources (e.g., architectural models, programming languages, databases, among others) for the current computing scenario. Regarding operation, these systems have been continually maintained to meet changing business requirements and computing platforms. In this paper, this concept is restricted to systems designed based on a monolithic architecture built on logical units (e.g., client-side, server-side, and database). Thus, any

changes to the system involve building and deploying a new version of the application (e.g., server-side) (Lewis and Fowler, 2014).

In a short period ago, integration techniques based on API (Application Programming Interface) have been used as a feasible alternative adopted by companies to provide new functionalities to their users from a legacy system (Erl et al., 2012). Over time, this type of system does not meet the most current requirements, besides beginning to present execution adversities (Grubb and Takang, 2003; Newman, 2019; Colanzi et al., 2021). In these systems, monolithic architectures encompass functionalities in large individual components, rising to inherent limitations in terms of scalability, maintenance, and deployment performance (Abgaz et al., 2023).

Based on the exposed context, software modernization has been an alternative adopted by companies to make their systems available to their users, considering recent technologies, architectural models, and computing scenarios (Lewis and Fowler, 2014). The Microservices Architecture (MSA) has been adopted because it enables the achievement of structural, functional, and data decoupling. MSA enables us to meet

^a  <https://orcid.org/0000-0002-5784-6248>

new recent trends of development (e.g. DevOps – Development and Operations) that demand ever-faster release cycles through partitioning systems into separately compilable services (Abgaz et al., 2023).

In this paper, modernization is a process starting from the legacy system (e.g., monolithic) to MSA. Some approaches, techniques, and methods for guiding modernization activities have been proposed in the literature, focusing on some aspects (e.g., the legacy's modules and their dependencies) and neglecting others (e.g., the data model structure). Researchers and practitioners have also focused their efforts on establishing panoramas or state-of-the-art on this research topic in secondary studies. However, these studies lack guidance on aspects that permeate the activities and interests related to the modernization of the system. Therefore, to the best of our knowledge, there is no tertiary study on the modernization of legacy systems to MSA.

According to Kitchenham et al. (2010); Petersen et al. (2015), a tertiary study aims to synthesize knowledge from secondary studies, providing a wide view of a research area. Therefore, a better understanding concerning the modernization of legacy systems to MSA can contribute to the development of new solutions (e.g., approaches, methods, and tools). Aiming to contribute to this research area, we present in this paper a Systematic Mapping Study (SMS) on modernization of legacy systems to MSA. Our analyses were based on 20 secondary studies, where we sought to explore aspects related to the studies themselves, the motivation to modernize, the patterns used in modernization, and the main activities of a modernization process itself.

The remainder of this paper is organized as follows. Section 2 presents background about microservice and software modernization and an overview of related work. Section 3 addresses the main findings of our study. A discussion of these findings is reported in Section 4. Finally, Section 5 presents our conclusions and perspectives for future work.

2 BACKGROUND AND RELATED WORK

In this section we present the background and related work that contributed to the development of our study. Initially concepts of microservice and software modernization are reported. Next, related work (i.e., tertiary studies) on modernization of legacy system to microservices is addressed.

Microservices. MSA has emerged as a dominant architectural style in recent years. Although count-

less studies have been conducted on the subject, there is no precise definition of this style. However, it is defined by shared features, including but not limited to, scalable services, organization around business capabilities, automated deployment, and a decentralized approach to languages and data. According to Lewis and Fowler (2014), “the term “Microservice Architecture” has sprung up over the last few years to describe a particular way of designing software applications as suites of independently deployable services”. This approach enables each component to be updated, modified, or replaced without disrupting the functionality of others, offering increased adaptability in the development process. As explained by Richardson (2018), “Microservices are a way of breaking up large monolithic applications into a set of small, loosely coupled, and independently deployable services that enable organizations to deliver software more quickly”. Within an MSA, services are fine-grained, and communication protocols are lightweight, which enhances modularity, making applications easier to comprehend, develop, test, and more resilient. Additionally, services can be developed and deployed independently, providing flexibility and scalability.

Modernization. The monolith was the primary architectural approach used for years in the industry, as it was easy to develop, deploy, and offer inherent benefits, such as simplified codebase management, streamlined debugging processes, and direct scalability within a single software unit. However, over time and with its increasing size, it presents various issues, impacting productivity, deliveries, and including challenges related to codebase complexity, difficulties in accommodating evolving technologies, and the potential for slower development cycles and releases. With these drawbacks and the emerging success of microservices, numerous companies set their sights on this new architecture, sparking widespread interest and prompting various organizations to consider migrating. Migrating or modernizing from a legacy system to a microservice-based one is not a trivial task. Many academic studies have been conducted to comprehend and discover the best techniques for optimizing this process. However, similar to the microservices themselves, the migration/modernization processes lack a consolidated definition and a single and solid step-by-step guide, providing companies with an array of options (Colanzi et al., 2021).

As related work, to the best of our knowledge, there is no tertiary study about the modernization of legacy systems to MSA. To do so, we combined the terms “tertiary study” and “microservices” with their synonyms and executed the search string in the same

publication databases of this study (Almeida et al., 2023). As presented in Section 3, the secondary studies available in the literature address different aspects of the investigation, leaving gaps related to the modernization process itself and its associated aspects.

3 A PANORAMA ON MODERNIZATION OF LEGACY SYSTEMS TO MSA

This section presents the main findings of our tertiary study, which provides a panorama on the modernization of legacy systems to MSA. To do so, we elaborated a research protocol based on guidelines proposed by Kitchenham et al. (2010) and Petersen et al. (2015), which can be found in full in Almeida et al. (2023). We conducted our study from August 31 to November 30, 2023, and involved four people: two software engineering researchers and two microservice researchers/practitioners with strong professional experience in the area.

After strictly following the guidelines of our research protocol, 20 studies were selected to compose the final list of our investigation, as shown in Table 1. The first column shows the ID for each study, and the second one is the reference for each study. Each reference is composed of the author’s name(s), and the publication title with an external link where the secondary study was published. The third column shows the publication year of each study. Next, we report the main findings of our study in Sections 3.1 – 3.4 based on Research Questions (RQ) defined in our protocol (Almeida et al., 2023).

3.1 Overview of Secondary Studies

The results reported in this section are related to RQ1, which describes the relationship between the studies’ publication venues, publication year, studies’ nature, and types of each secondary study. Figure 1 illustrates the distribution of studies relating the interests of RQs 1.1 and 1.2. The columns illustrate the distribution of studies in relation to the publication venues identified in our study (i.e., Conference, Journal, and Symposium) for each year. We limited our analysis to the last six full years and aimed to get the most up-to-date and recent data. Thus, 55% of the studies were published in Journals (S3, S4, S7, S8, S10, S11, S12, S13, S16, S18, and S19), 30% in Conferences (S6, S9, S14, S15, S17, and S20), and only 15% in Symposium (S1, S2, and S5). This information alone provides solid data that reveals a wider concentration

of studies in publication venues with a high rigor of evaluation (i.e., Journal). Moreover, it can be said that studies published in venues with a lower level of maturity (i.e., Symposium and Conference) are moving towards becoming more solid studies for interested scientific and practitioner communities.

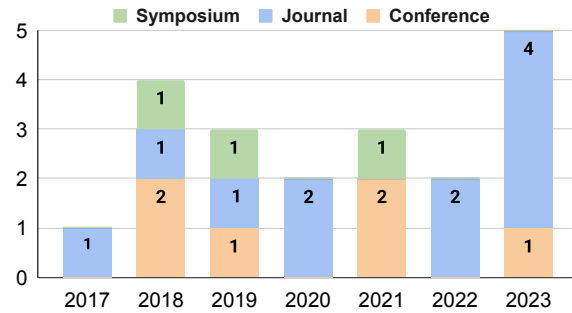


Figure 1: Yearly distribution of publications.

The nature and type of studies reveal the research interests of the scientific and industry communities in understanding aspects related to the modernization of legacy systems to MSA. Concerning the studies’ nature (RQ 1.3), we elaborated three categories, namely: (i) *Academic*, a category that covers studies conducted mainly by academic researchers, focusing only on theoretical and experimental investigations, without direct emphasis on industrial application; (ii) *Industrial*, a category that covers studies led by researchers with a focus on applications aimed at industry and without academic bias; and (iii) *Mixed*, a category that covers studies conducted in a hybrid way, involving researchers and practitioners, with content aimed at industry and/or academia. Regarding the studies’ type (RQ 1.4), we categorized the secondary studies in three categories: Systematic Literature Review (SLR); SMS, and Survey. Figure 2 illustrates the distribution of the publications per nature and type.

With 10 studies (50%), “Survey” was the most recurrent type of study in our investigation. “SMS” with 6 studies (30%) and “SLR” with 4 studies (20%) complete our distribution. Drawing a parallel between this information and the nature of the studies, it is observed a balance between studies considered academic (i.e., Academic category) and studies with industry participation (i.e., Industrial and Mixed categories). This relationship reveals the interest of academia and industry in the modernization of legacy systems to MSA. From the academic side, it is noted an interest in understanding the industry’s pains in modernizing such systems and developing solutions (e.g., processes, methodologies, frameworks) that can minimize the impacts related to this type of activity. On the industry side, evidence shows interest in so-

Table 1: List of studies analyzed in our study.

ID	Reference	Year
S1	Luz, Welder and Agilar, Everton and de Oliveira, Marcos César and de Melo, Carlos Eduardo R. and Pinto, Gustavo and Bonifácio, Rodrigo, An Experience Report on the Adoption of Microservices in Three Brazilian Government Institutions	2018
S2	Colanzi, Thelma and Amaral, Aline and Assunção, Wesley and Zavadski, Arthur and Tanno, Douglas and Garcia, Alessandro and Lucena, Carlos, Are We Speaking the Industry Language? The Practice and Literature of Modernizing Legacy Systems with Microservices	2021
S3	Velepucha, Victor and Flores, Pamela, A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges	2023
S4	Razzaq, Abdul and Ghayyur, Shahbaz A. K., A systematic mapping study: The new age of software architecture from monolithic to microservice architecture—awareness and challenges	2023
S5	Cojocar, Michel-Daniel and Uta, Alexandru and Oprescu, Ana-Maria, Attributes Assessing the Quality of Microservices Automatically Decomposed from Monolithic Applications	2019
S6	Kalske, Miika and Mäkitalo, Niko and Mikkonen, Tommi, Challenges When Moving from Monolith to Microservice Architecture	2018
S7	Abgaz, Yalemisew and McCarren, Andrew and Elger, Peter and Solan, David and Lapuz, Neil and Bivol, Marin and Jackson, Glenn and Yilmaz, Murat and Buckley, Jim and Clarke, Paul, Decomposition of Monolith Applications Into Microservices Architectures: A Systematic Review	2023
S8	Knoche, Holger and Hasselbring, Wilhelm, Drivers and Barriers for Microservice Adoption – A Survey among Professionals in Germany	2019
S9	Carvalho, Luiz and Garcia, Alessandro and Assunção, Wesley K. G. and Bonifácio, Rodrigo and Tizzei, Leonardo P. and Colanzi, Thelma Elita, Extraction of Configurable and Reusable Microservices from Legacy Systems: An Exploratory Study	2019
S10	Muhammad Hafiz Hasan and Mohd Hafeez Osman and Novia Indriaty Admodisastro and Muhamad Sufri Muhammad, Legacy systems to cloud migration: A review from the architectural perspective	2023
S11	Ghayyur, Shahbaz Ahmed Khan and Razzaq, Abdul and Ullah, Saeed and Ahmed, Salman, Matrix clustering based migration of system application to microservices architecture	2018
S12	Hassan, Sara and Bahsoon, Rami and Kazman, Rick, Microservice transition and its granularity problem: A systematic mapping study	2020
S13	Bamberger, Burkhard and Körber, Bastian, Migrating Monoliths to Microservices Integrating Robotic Process Automation into the Migration Approach	2022
S14	Salii, Sh. and Ajdari, J. and Zenuni, Xh., Migrating to a microservice architecture: benefits and challenges	2023
S15	Di Francesco, Paolo and Lago, Patricia and Malavolta, Ivano, Migrating Towards Microservice Architectures: An Industrial Survey	2018
S16	Velepucha, Victor and Flores, Pamela and Torres, Jenny, Migration of Monolithic Applications Towards Microservices Under the Vision of the Information Hiding Principle: A Systematic Mapping Study	2020
S17	Velepucha, Victor and Flores, Pamela, Monoliths to microservices-Migration Problems and Challenges: A SMS	2021
S18	Taibi, Davide and Lenarduzzi, Valentina and Pahl, Claus, Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation	2017
S19	Weerasinghe, Sidath and Perera, Indika, Taxonomical Classification and Systematic Review on Microservices	2022
S20	Michael Ayas, Hamdy and Leitner, Philipp and Hebig, Regina, The Migration Journey Towards Microservices	2021

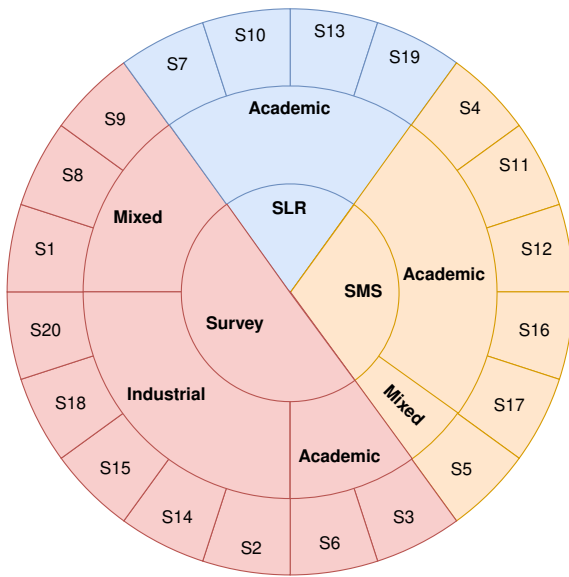


Figure 2: Publications’ distribution per nature and type.

lutions that suit their needs without bringing major technological impacts or high learning curves.

3.2 Motivations

The motivations outlined in each secondary study were analyzed to address RQ2, as shown in Table 2. The first column presents all motivations identified in our investigation, while the other columns correspond to studies that presented some evidence for such motivations. Next, we addressed a description of each motivation concerning each study.

Before presenting our analysis of the motivations extracted from secondary studies, it is worth highlighting that seven of the twenty studies did not present motivational evidence about the modernization activity (S3, S7, S9, S12, S13, S16, and S17). According to our analysis, identified modernization motivations can be classified into two categories, namely: (i) **pre-modernization**, which represents studies that highlighted some adversity in legacy systems and proposed software modernization to address it; and (ii) **post-modernization**, which encompasses the studies that reported interest in achieving the modernization advantages.

Regarding **pre-modernization** motivations, six studies (S1, S4, S5, S8, S14, and S20) reported difficulties in carrying out frequent deployments (“*Deployment frequency*”), impacting their ability to stay updated with market innovations and reducing competitive advantage. “*Adding new functionalities*” was addressed by two studies (S6 and S15). This motivation reveals the complexity of integrating new features because of the legacy nature of systems, imped-

ing agility in modifications. Adverse issues related to integrating new developers were also highlighted by two studies (S1 and S5), which reported extended adaptation time (“*Developer onboarding*”), delaying the effective commencement of development.

Concerning **post-modernization** motivations, “*Maintainability*” and “*Scalability*” with seven studies (S2, S8, S10, S14, S18, S19, and S20) in common were the main motivational factors related to post-modernization. Individually, these motivational factors were identified in S4 (Maintainability) and S11 (Scalability). Therefore, the need for scalable, adaptable, and maintainable systems emerged as a significant priority in the aforementioned studies. “*Flexibility*” was identified in three studies (S1, S4, and S10). This feature enables independent updates of system components, promoting agility, and facilitating incremental innovations during the transition to MSA. “*Cost and time savings*” were factors detected in two studies (S4 and S14). This is attributed to the modular nature of MSA, enabling precise updates on specific components and reducing overall maintenance costs. Additionally, shorter development cycles and agile deployments lead to cost savings and financial benefits. “*Polyglot*”, motivational factor identified in two studies (S4 and S8), refers to microservices’ flexibility in supporting multiple programming languages, expanding development options, and preserving developers’ knowledge. “*Team responsibilities*”, feature identified in one study (S18), represents the effective task and responsibility distribution among teams during the system transition to MSA. This feature enables greater autonomy and agility in development decisions, driving independent improvements in each microservice. We identified three core motivations in study S14, namely: “*Performance*”, “*Fault tolerance*”, and “*Agility*”. The first relates to operational efficiency, the second addresses the system’s ability to handle issues without significant interruptions, and the third refers to the capability of quick adaptation-essential elements in pursuing MSA. S19 underscores that modernization offers opportunities for more granular and flexible security policies, safeguarding individual services and reducing breach impacts, strengthening overall system resilience (“*Security*”).

Drawing a counterpoint concerning modernization, two studies (S6, S18) highlighted the use of “*DevOps practices*” as an essential factor for the continuous delivery of software. Considering the scenario of constant innovations, companies are avoiding distancing themselves from modernization (or technological transformation). In this direction, S18 mentioned an important jargon “*Because everybody does it*” as a

Table 2: An overview of the motivations found in our study.

Motivation	S1	S2	S4	S5	S6	S8	S10	S11	S14	S15	S18	S19	S20
Adding new functionalities					✓					✓			
Agility									✓				
Because everybody does it											✓		
Cost and time savings			✓						✓				
Deployment frequency	✓		✓	✓		✓			✓				✓
Developer onboarding	✓			✓									
DevOps practices					✓						✓		
Fault tolerance									✓				
Flexibility	✓		✓				✓						
Maintainability		✓	✓			✓	✓		✓		✓	✓	✓
Performance									✓				
Polyglot			✓			✓							
Scalability		✓				✓	✓	✓	✓		✓	✓	✓
Security												✓	
Team responsibilities											✓		

motivational factor, highlighting the desire to keep up to date, even though these companies fail because of the lack of minimum knowledge to conduct a modernization activity.

3.3 Patterns Addressed in Modernization

As found in our investigation, we must follow a step set to guarantee the success of the modernization of legacy systems to MSA. This recommendation is essential to avoid the feature perpetuation of legacy systems in the new systems. Moreover, it is also important to apply specific patterns in each modernization phase, as these patterns significantly contribute to transitioning from a legacy system to MSA.

To answer RQ3, we analyzed the patterns reported by each secondary study. Of the 20 studies, 14 studies (70%) reported some evidence of the use of patterns during the modernization process. S4, S7, S9, S10, S11, and S16 were the studies that did not present concrete evidence on the investigation subject of this RQ. To organize such patterns and their applicability, we adopted and extended the structure presented in the study S19, as illustrated in Figure 3. Next, a description of the main patterns of each category is addressed for reasons of space and scope.

Service decomposition is a category that encompasses a pattern set related to the system's ability to operate independently across multiple connected services. These patterns play important roles in the breakdown of a legacy system into microservices, creating smaller services, and optimizing application

performance. Effective use of these patterns requires architects to have solid business domain knowledge, combined with a deep and specific technical understanding of it (S19). We found evidence of the *Strangler* pattern in five studies (S1, S2, S14, S15, and S18) and we elect it as the main pattern in this category. During the transition phase, the Strangler pattern facilitates the gradual replacement of the old system with the new architecture (S1).

Data management encompasses different patterns for organizing and structuring system data. The pattern selection in this category must be guided by specific business demands, enabling architects to choose those that best meet the identified requirements (i.e., MSA). According to our analyses, the *Database per Service* pattern can be considered one of the most important in this category, since it provides that each service maintains its exclusive database, which is only accessed through its API. The strategy behind this pattern enables independence between services, giving each one full control over its persistent data, without direct sharing with other services. This pattern also allows the use of different databases, according to the specific needs of each service (Richardson, 2018).

Microservices deployment is a category that deals with deploying services independently, enabling seamless updates and scalability for microservices. By deploying these microservices across various servers, it becomes feasible to scale the system and optimize performance as required. To do so, it is recommended to maintain a dependable deployment environment and implement ongoing monitoring. Of

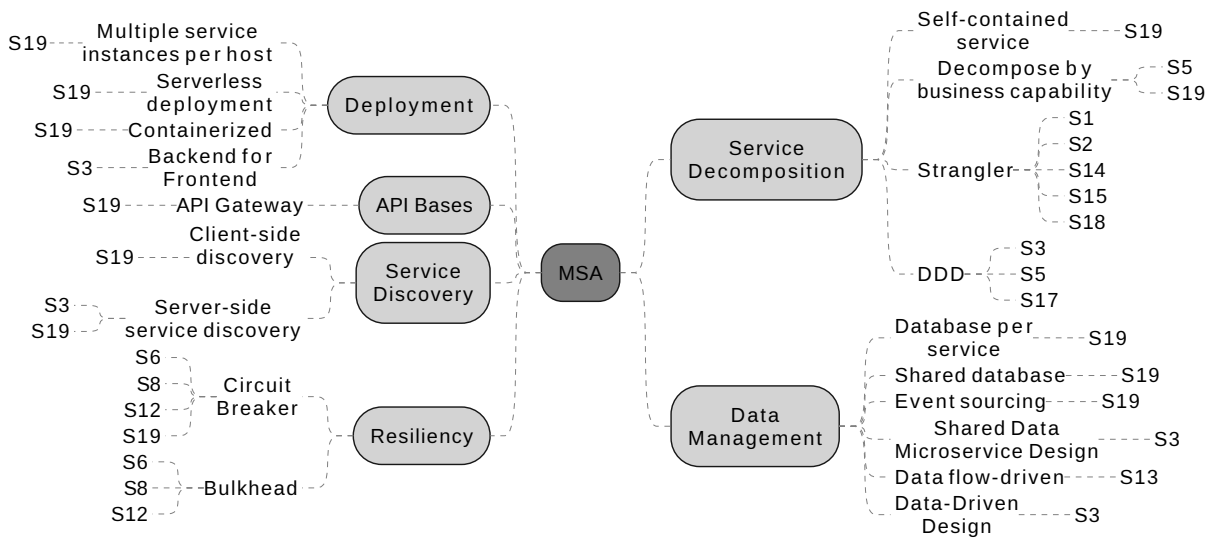


Figure 3: Taxonomy of architectural patterns for modernization of legacy systems to MSA.

the four patterns of this category, *Serverless deployment* (S19) represents a solution that hides any server concept (i.e. reserved or allocated resources), hosts, or containers. In short, the server cloud receives a service code and runs it transparently. The customers are charged by the requests for the resources consumed.

The **API Bases** category deals with the exposure of microservices through API managers, which supervise and control the interfaces provided by such services. From an operational viewpoint, it facilitates the scaling of microservices based on demand, dynamically adjusting to changes in traffic. *API Gateway*, a pattern identified in the S19 study, enables customers to access services centrally, directing requests to specific services, simplifying access to microservices, and adapting APIs according to the individual needs of each customer. Thus, this pattern enables the improvement of the user experience, isolating the customer from the complexity of the microservices structure (Richardson, 2018).

Service Discovery is a category that covers the discovery and communication of microservices. Richardson (2018) emphasizes the importance of enabling dynamic and efficient communication among microservices, facilitating automatic discovery and interaction without dependence on static or manual setups. We identified *Server-side Service Discovery* pattern in two studies (S3 and S19) of our investigation, which is a method of dynamic service discovery in an MSA. To do so, it uses a router that consults a service registry, directing requests to available instances. Although it simplifies client code and some cloud platforms offer this feature, this pattern requires the configuration of an additional component (i.e., router), which may introduce more networking

steps compared to *Client-side Discovery*.

Resiliency encompasses different patterns for recovering and maintaining the operational stability of a system in the face of internal or external failures. *Circuit Breaker* was the most recurrent pattern in our investigation, being found in 4 studies (S6, S8, S12, and S19). In short, the goal of this pattern is to prevent failures in services from affecting the stability of the system. For instance, when a service is unavailable or has high latency, *Circuit Breaker* pauses requests to that service, avoiding overloading the calling service and, after a while, enabling the service to be tested again. If the problematic service is working, requests are resumed, otherwise, the interruption period remains (Richardson, 2018).

3.4 Modernization: Challenges, Benefits, and Processes

The results reported in this section are related to RQ4, which describes the relationship between the challenges faced during modernization, the main expected benefits after the system’s modernization, and the main steps recommended for a modernization process. Next, we address each element of this relationship in Sections 3.4.1 – 3.4.3.

3.4.1 Faced Challenges

To evaluate the RQ4.1, we compiled the extracted data in Figure 4, which shows the challenges faced during the modernization process of our study according to the occurrence number. Of the 20 studies selected in our investigation, four (S5, S9, S16, and S20) did not present concrete evidence regarding

the faced challenges during the modernization process. Next, a description of the main challenges is addressed for space and scope reasons.

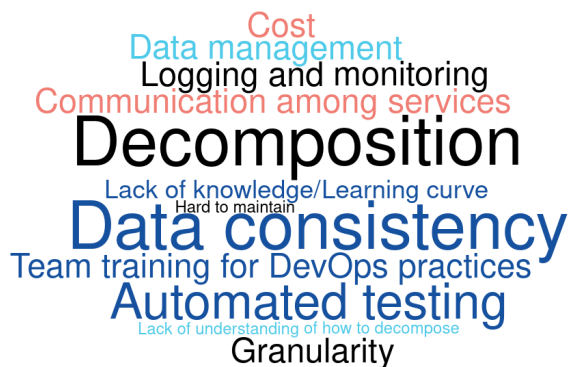


Figure 4: Challenges faced during the modernization process of legacy systems to MSA.

According to our analysis, **Decomposition**, found in 10 studies (S3, S4, S6, S7, S10, S11, S15, S17, S18, and S19), was one of the biggest obstacles faced by developers during the modernization process. This activity consists of fragmenting legacy systems, complex systems, or services into smaller, more manageable parts (i.e., microservices), facilitating the system's development, maintenance, and scalability. In this direction, it is equally important to consider granularity to avoid excessively detailed components, which can harm system performance (S6).

Lack of knowledge/Learning curve was another challenge identified in our analysis with seven studies (S1, S3, S4, S8, S15, S17, and S18). The lack of knowledge can generate severe impacts during system modernization, resulting in severe performance problems if the migration is not correctly conducted. In parallel, the learning curve (S17) can further prolong the duration of the process, making it take longer to complete and generating other discomforts for companies (e.g., continuous software delivery).

Data consistency and **Data management** were the challenges mentioned in four (S2, S4, S8, and S19) and two (S6 and S18) studies, respectively. These issues can be considered first-class elements in the microservice design, since the absence of ACID¹ transactions between services is a limitation that can result in severe problems, including the loss of essential information. Therefore, maintaining data integrity when migrating to a microservice environment is essential to ensuring that critical information is not compromised during the modernization process (S8).

With three studies (S4, S6, S8), **Automated test-**

ing also emerges as a challenge during the modernization of legacy systems to MSA. To do so, requires more comprehensive testing, considering complex interactions between services. Adapting existing tests to this structure involves time and effort, especially because of the database separation, which demands more careful data management to execute the tests. These adversities highlight the importance and complexity of automated testing during the modernization process (S6).

Team training for DevOps practices has been a challenge faced by companies, as identified in two studies (S3 and S18). This activity includes performing tasks such as Continuous Integration and Continuous Deployment (CI/CD), considerably reducing development, testing, and distribution times in different environments, such as pre-production and production. Therefore, investing in DevOps training programs becomes necessary to equip teams with the competencies to use DevOps tools, foster collaboration between development and operations, and get a mindset of continuous improvement.

Service **Granularity** was the challenge found in two studies (S6 and S12). The granularity level determines the size and functional scope of each service in an MSA, impacting the efficiency and overall functionality of the system. Therefore, early deciding on this issue can make it difficult to analyze and adapt microservices over time. Accurate and thoughtful choice of granularity is essential to the modernization success and the microservice's effectiveness throughout the software life cycle (S12).

Another significant challenge is **Communication among services**, which was found in two studies (S13 and S18). Efficient communication between services is vital for their integration and synchronization. Lack of clear and consistent communication can result in interoperability problems, leading to failures in the system's functioning. This challenge is especially critical early in development, where the communication infrastructure must be established from the beginning to ensure a smooth and effective transition to the microservices' environment (S18).

Finally, **Logging and monitoring** were critical challenges cited by studies S6 and S8 during the modernization process. Continuous monitoring and event logs are essential elements used to identify and solve problems, ensuring system performance and security. Without a comprehensive logging and monitoring strategy, it becomes difficult to detect faults and diagnose problems early, which compromises the system's stability and reliability (S6).

¹An acronym that refers to four properties, namely: Atomicity, Consistency, Isolation, and Durability.

3.4.2 Expected Benefits

Regarding the expected benefits after modernization (RQ4.2), we analyzed the collected data to delineate and identify the achievements resulting from this transition. Our analyses centered on studies providing explicit and measurable data regarding the benefits realized post-modernization completion (S1, S14, S17, and S18). Three of the 20 studies reviewed did not specify particular benefits (S5, S7, and S11), and the remaining studies did not report an adequate explanation of the real expected benefits. Next, a consideration of our findings is addressed.

Scalability was a benefit identified in three studies (S14, S17, and S18). It embodies the system's capacity for dynamic resource expansion or reduction, ensuring the application can handle increased users or workload without compromising performance (Lewis and Fowler, 2014). For instance, microservices can be flexible through horizontal scalability, enabling business-driven expansion of the application using seamless addition or removal of service instances as necessary. In parallel, these studies also highlighted an improvement in maintenance, suggesting that migrating to microservices not only enhances scalability but also streamlines system maintenance. This benefit is tied to the modular nature of microservices, enabling the development, updating, and correction of specific system parts without affecting others. Thus, maintenance becomes more agile and less prone to disruptions across the entire application (S18).

Performance enhancement was underscored by two studies (S14 and S18) as a significant benefit of migrating to microservices. This improvement stems from the ability of microservices to distribute workloads more efficiently, resulting in faster response times and an overall enhanced user experience. The ability to optimize services individually enables the swift resolution of specific bottlenecks (S14).

Continuous delivery was identified as a benefit in two studies (S1 and S17). This emphasizes the importance of continuous and automated software deliveries. The migration to microservices facilitates the efficient implementation of CI/CD practices, a basis of DevOps, enabling incremental updates, quick adaptation, and more reliable software delivery (S1).

Delving into specific gains, a **reduction in time-to-market** emerged prominently in one particular study (S1). Modernization enables quick product and service launches to the market, aided by the microservice's modularity that facilitates independent work on specific system parts. While highlighted in a specific study, the reduction in the time to make functionalities available to end-users significantly helps companies respond agilely to market demands, giving them

a competitive edge (S1).

Availability was underscored as a benefit in one of the analyzed studies (S14). The modernization of legacy systems to MSA establishes a more resilient architecture, ensuring greater service availability. Microservices enable independent deployment and scaling, reducing the likelihood of widespread failures and enhancing resilience against interruptions (S14).

The S14 study revealed that **team improvement** was a benefit highlighted after modernization because the main purpose was increased productivity. Teams can work independently on specific services using the MSA. The modular organization of this architectural style enables faster and more independent updates and maintenance, enhancing the efficiency of the development process (S14).

Modernization of legacy systems to MSA was linked to better **cost efficiency** in one study (S14). MSA enables more efficient resource allocation, scaling services according to their specific demand. Despite potential initial high migration costs (S4), the long-term flexibility and scalability can lead to significant cost optimization (S14).

Finally, the **use of new technologies** emerged as a post-modernization benefit in one study (S17). With independent services using diverse technologies, organizations can employ advanced frameworks, programming languages, and development tools, ensuring they remain updated and innovative within their technological environment (S17).

3.4.3 Modernization Process

Based on the evidence presented for RQs 4.1 and 4.2, understanding the phases of a modernization process to minimize the impacts of this activity type and aligning future expectations is still an issue that must be investigated (RQ4.3). Of the 20 studies, only four investigated this issue (S2, S15, S18, and S20). Thus, a process for modernization was organized in three macro steps, as illustrated in Figure 5. This process should not be considered a silver bullet solution, but an essential guide for any stakeholder to understand the main steps for modernizing legacy systems to MSA. By proposing this process, we aim to provide these stakeholders with a clear and solid vision of the tasks and knowledge that permeate this activity type so that the challenges presented in RQ4.1 are met and the expectations reported in RQ4.2 are fulfilled. Next, we address a description of each step.

Understanding legacy systems (Step 1) from both a logical (i.e., structural and behavioral) and organizational (i.e., architectural) viewpoint were key aspects investigated and reported by studies S2, S15, and S20. In short, this understanding can be gained through

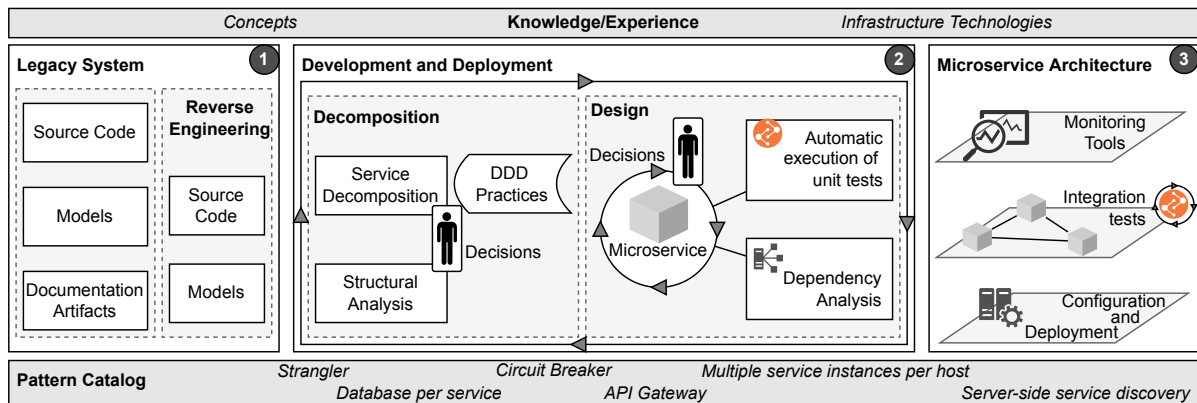


Figure 5: Modernization process of legacy systems to MSA.

pre-existing documentation of the legacy system (e.g., source code, test artifacts, documentation, and architectural artifacts) or reverse engineering techniques. At this stage, software engineers must investigate the motivating forces for system modernization (S2), identifying operational gaps and limitations in the legacy system (i.e., technical, operational, and organizational aspects). In deeper investigation, these engineers must understand, at a minimum, the following aspects of the legacy system: structures, behaviors, organization, and interactions between modules. Therefore, it can be said that the conduction of this activity emphasizes that understanding legacy systems is not just informative, but also provides the basis for microservice design in the new architecture.

As reported by studies S2, S15, S18, and S20, conducting a modernization process involves the transformation of existing implementation artifacts with the system or recovered through reverse engineering techniques, followed by the gradual integration of microservices with the legacy system until the modernization is completed in full (S2). Thus, we suggest that the Step 2 activities be conducted cyclically based on the prior knowledge of the development teams (Soldani et al., 2018) and supported by the use of architectural patterns (Richardson, 2018). According to a study conducted by Soldani et al. (2018), developers’ lack of knowledge has been the main cause of unsuccessful in the adoption or any attempt to transition an existing system to MSA. For space and scope reasons, details about the framework of MSA concepts will not be reported in this paper. However, an overview of this research theme was presented in Section 2. Similarly, details concerning architectural patterns are also not presented in this section, with only the main ones from our investigation being highlighted for each development phase. More details about such patterns can be seen in Section 3.3.

As illustrated in Figure 5, Step 2 represents the de-

velopment and deployment of microservices, a macro activity in the referred process that involves fundamental design decisions for modernization sprints in an incremental/cyclical approach. Furthermore, it is worth highlighting that this step should not be confused with the final architecture (i.e., MSA), focusing on both tasks: the decomposition of the legacy system and the design of microservices. Next, we addressed the details of each task.

The main purpose of decomposition is to prepare the legacy system for the design of microservices. To do so, software engineers must identify services for the new architecture, which can be supported by the application of DDD (Domain-Driven Design) practices (S15) or by the Stranger architectural pattern, as suggested by S15 and S18. According to Di Francesco et al. (2018), decomposing the legacy system into smaller components and immediate experimentation through gradual integration between the legacy and microservices has proven to be a feasible strategy for monitoring the system’s evolution (S15 and S20). It is worth highlighting that the structural analysis of the system, the definition of the architecture, and the prioritization of service development appear as important steps that can be achieved in parallel to the decomposition. Finally, the possibility of inserting new functionalities into the system is a way of modernizing the system to a new computational model, besides meeting new market demands.

Microservices design is based on the definition of the MSA, including design decisions to meet functional and non-functional requirements. In summary, this task entails the creation or modification of software and other necessary components, including microservices. During this stage, the use of practices such as CI/CD, API definition, and DevOps practices, mentioned in Sections 3.2 and 3.3 have proven essential to manage complexity and ensure transition efficiency, offering the knowledge basis necessary for a

dynamic and adaptable architecture. Concerning the development of microservices, the use of an incremental/evolutionary approach with continuous software delivery must be used so that tests and dependencies between microservices can be conducted gradually, considering redeployment of the system from scratch or implementation gradually to eliminate the existing legacy system. At this stage, it is recommended to use automated processes (i.e., tools, and frameworks) to better manage tasks. Finally, it is worth highlighting that documentation of the new architecture, especially through architectural, textual documents and diagrams, has proven to be a fundamental element in the transition from the legacy system to MSA, since it is an important source of information to meet the real motivations for modernization (see RQ2 – Section 3.2) and guarantee future expectations with the new system (see RQ4.2 – Section 3.4).

The MSA is finally introduced in Step 3, showcasing three layers: (i) microservice; (ii) configuration; and (iii) monitoring. Microservices are deployed in the execution environment (Item i), and verification and validation tasks of microservices ensure adequate system integration, whether parts of the legacy system are used as a reference for testing. At this stage, microservices' integration tests also occur as a way of ensuring that the functionalities delivered are reliable and comply with the requirements established in the previous stages. Configuring supporting artifacts is crucial to preparing the DevOps environment (Item ii), facilitating the transition from the legacy system to the new MSA. Finally, continuous monitoring of microservices and infrastructure enables to evaluation of the success of modernization (Item iii), analyzing factors such as service availability, performance, and resource utilization (S2 and S20).

4 DISCUSSION OF RESULTS

This section provides a discussion of the results of our study. Our investigation suggests that the modernization of legacy systems to MSA is a complex activity that must be carefully conducted by experienced teams and supported by development infrastructure. The main findings and results are listed as follows.

Our in-depth analysis revealed critical aspects concerning DDD in modernizing legacy systems to MSA. The ambiguity surrounding DDD, whether considered as a potential pattern (S3, S5, S17) or a comprehensive set of concepts, guidelines, and practices, adds complexity to the modernization of these legacy systems. Varied interpretations of DDD can directly influence the strategies adopted by develop-

ment teams (i.e., practitioners), impacting the effectiveness and scope of the modernization activities.

Alongside the conceptual complexity, the studies selected in our investigation highlighted a significant motivation for migrating to MSA: market pressure based on the “everyone is doing it” phenomenon mentioned by the study S18. While this drive for upgrading is legitimate, it often leads to rushed approaches and implementations without the required expertise per this architectural style (see Section 3.4.2). Consequently, this might inadvertently introduce inappropriate practices, creating anti-patterns that preserve legacy features in the modernized systems.

Other important evidence found in our investigation reveals the close relationship between deployment frequency and time-to-market. Legacy systems (or monoliths) unable to adapt quickly lose the ability to meet market changes, resulting in a disconnect from user needs and competition. This adversity in keeping pace with changes can partly be attributed to the lack of experience and technical knowledge within the involved teams, as evidenced in six studies of our investigation (S1, S4, S5, S8, S14, and S20).

Although deep details of the modernization process proposed in this paper have not been reported in the Section 3.4.3 for space and scope reasons, our view is that the macro activities of this referred process can be considered a useful roadmap, offering a structured guide for any stakeholder (i.e., researchers and practitioners) interested in a process to modernize a legacy system to MSA.

5 CONCLUSIONS AND FUTURE WORK

Our study delved into a wide spectrum of existing research, revealing that this research topic is still emergent, and of common interest in both academia and industry. By synthesizing insights from various secondary studies, we offered a comprehensive overview of motivations, main architectural patterns organized in a taxonomy, faced challenges, expected benefits, and macro steps to conduct the modernization.

By analyzing motivations (see Section 3.2), we uncovered a diversity of driving factors. Pre-motivations highlighted challenges in legacy systems, such as the need to incorporate new functionalities and the difficulty in performing frequent updates because of the monolithic nature of these systems. In contrast, post-motivations emphasized the pursuit of maintainability, scalability, cost efficiency, and benefits, such as enhanced operational performance and flexibility in updating specific components.

The detailed analysis revealed the extensive application of specific patterns at each stage of the modernization process. One example is the “Strangler” pattern, which provides a step-by-step for the identification of candidates for (micro)service, supporting a step of a modernization process of legacy systems to MSA. Similarly, the “Database per Service” pattern, emerges as an effective solution for independence among services and decoupled persistence data. These and other patterns (see Section 3.3), when consciously and systematically applied, play a critical role in the success of modernization toward MSA.

Regarding the modernization process itself, a challenging path involving several steps must be overcome. For instance, “Service decomposition” demands a careful approach to fragmenting legacy systems into smaller, manageable components (i.e., microservices). “Data management” plays a pivotal role in this process, requiring specific strategies to maintain consistency and integrity of data when migrating to a distributed architecture. By systematically and comprehensively addressing the steps of the proposed process in this paper (see Section 3.4.3) combined with architectural patterns reported in Section 3.3, it is possible to establish a cohesive and successful modernization guide, facilitating a smooth and efficient transition from legacy systems to MSA.

As future work, we intend to conduct a more specific investigation of the research topic explored in this paper. Despite various approaches being proposed, a solid guide that details software modernization comprehensively is still a subject to be investigated in-depth. Therefore, our goal is to fill this gap by creating a process that provides clear and practical steps to assist organizations in the modernization of legacy systems to MSA, considering a more dynamic and efficient framework based on microservices. In this sense, it is worth highlighting that the process presented in Section 3.4.3 can be used as a basis for elaboration of this process, as it brings together the most recent evidence related to this research topic.

ACKNOWLEDGEMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES).

REFERENCES

Abgaz, Y., McCarren, A., Elger, P., Solan, D., Lapuz, N., Bivol, M., Jackson, G., Yilmaz, M., Buckley, J., and

Clarke, P. (2023). Decomposition of monolith applications into microservices architectures: A systematic review. *IEEE Transactions on Software Engineering*, 49(8):4213–4242.

Almeida, N. R., Campos, G. N., Moraes, F. R., and Affonso, F. J. (2023). Systematic mapping protocol – mapping study on modernization of legacy systems to microservice architecture. on-line. https://drive.google.com/file/d/1jwx27Noo8lr_zoBSA4xz8pbU8bjCKkza/view?usp=sharing, accessed on March 19, 2024.

Colanzi, T., Amaral, A., Assunção, W., Zavadski, A., Tanno, D., Garcia, A., and Lucena, C. (2021). Are we speaking the industry language? the practice and literature of modernizing legacy systems with microservices. In *The 15th Brazilian Symposium on Software Components, Architectures, and Reuse*, page 61–70, New York, NY, USA. Association for Computing Machinery.

Di Francesco, P., Lago, P., and Malavolta, I. (2018). Migrating towards microservice architectures: An industrial survey. In *IEEE International Conference on Software Architecture (ICSA)*, pages 29–2909.

Erl, T., Balasubramanian, R., Pautasso, C., Wilhelmsen, H., Carlyle, B., and Booth, D. R. (2012). *SOA with REST: principles, patterns & constraints for building enterprise solutions with REST*. Prentice Hall, Philadelphia, PA.

Grubb, P. and Takang, A. A. (2003). *Software Maintenance: Concepts And Practice*. World Scientific Publishing, Singapore, Singapore, 2 edition.

Kitchenham, B., Pretorius, R., Budgen, D., Brereton, O. P., Turner, M., Niazi, M., and Linkman, S. (2010). Systematic literature reviews in software engineering – a tertiary study. *Information and Software Technology*, 52(8):792–805.

Lewis, J. and Fowler, M. (2014). Microservices. on-line. <https://martinfowler.com/articles/microservices.html>, accessed on March 19, 2024.

Newman, S. (2019). *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*. O’Reilly Media, Sebastopol, CA.

Petersen, K., Vakkalanka, S., and Kuzniarz, L. (2015). Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18.

Pressman, R. and Maxim, B. (2019). *Software Engineering: A Practitioner’s Approach*. McGraw-Hill Education. 9th Edition.

Richardson, C. (2018). *Microservices patterns*. Manning Publications Company.

Soldani, J., Tamburri, D. A., and Van Den Heuvel, W.-J. (2018). The pains and gains of microservices: A systematic grey literature review. *Journal of Systems and Software*, 146:215–232.