

# Migration from Monolithic Applications to Microservices: A Systematic Literature Mapping on Approaches, Challenges, and Anti-patterns

Fernando Muraca, María F. Pollo-Cattaneo

*Universidad Tecnológica Nacional, Buenos Aires, Argentina*

## Abstract

The present work explores the approaches, challenges, and anti-patterns, together with the best practices concerning the transition from a monolith to a microservice architecture. Based on a methodical examination of current scholarly publications across an extensive review of publications in specialized journals, ongoing tendencies are identified, also future research areas in the domain of transitioning from monolithic systems into microservices are mentioned. Additionally, the applied and academic implications of these findings are reviewed. The intention is to present a complete and current summary of progress in this area.

## Keywords

Migration, Monolithic Applications, Microservices, Systematic Mapping, Literature, Approaches, Challenges, Anti-Patterns, Best Practices

## 1. Introduction

Transitioning from monolithic apps to microservices grows as a dynamic research area in Software Engineering, due to the advantages offered by this architecture in terms of scalability, maintainability, and adaptability [1]. Migration involves decomposing a monolithic system into little, autonomous components that can be developed, released and expanded independently [2]. However, that process can be complex and challenging as it involves reorganizing components, managing dependencies, and adapting to new design patterns and architectures [3]. Additionally, the use of microservices in agile environments introduces challenges, benefits, and drawbacks that are considered in [4]. These key considerations motivate a deeper analysis of the migration process in such environments.

Drawing from an exhaustive examination of prior research, multiple publications delve into the advantages as well as challenges related to leveraging Microservices [5, 6]. These works explore various design aspects [7] and highlight the significance of patterns [8]. The industry is achieving considerable progress in this field, however, academic research is still in its early stages [9, 10]. An additional review verifies substantial industry interest in transitioning legacy systems [11].

---


*ICAIW 2023: Workshops at the 6th International Conference on Applied Informatics 2023, October 26–28, 2023, Guayaquil, Ecuador*

✉ [fmuraca@frba.utn.edu.ar](mailto:fmuraca@frba.utn.edu.ar) (F. Muraca); [fpollo@frba.utn.edu.ar](mailto:fpollo@frba.utn.edu.ar) (M. F. Pollo-Cattaneo)

ORCID [0009-0002-1669-4126](https://orcid.org/0009-0002-1669-4126) (F. Muraca); [0000-0003-4197-3880](https://orcid.org/0000-0003-4197-3880) (M. F. Pollo-Cattaneo)



© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

The context and motivations driving the research, intended objectives, and document structure are outlined in the present section.

### **1.1. Context and Motivation**

The software industry gravitates toward microservices adoption as a result of the advantages it offers in terms of efficiency and scalability for application development and maintenance [1]. Despite this trend, many organizations still use monolithic applications, which pose challenges in terms of adaptability and scalability. Shifting from monoliths to microservices is a solution to address these challenges, yet many concerns and obstacles remain to achieve a successful transition [2].

### **1.2. Research Goal**

The present work intends to carry a systematic literature mapping about the migration from monolithic systems to microservices evaluating the present approaches, challenges, antipatterns, and best practices published in recent scholarly literature. Moreover, this study tries to identify tendencies as well as possible areas for further research in this domain of study.

### **1.3. Document Structure**

The structure of the paper is: Section 2 gives the theoretical grounding of the investigation, including important definitions such as monolithic apps, microservices, converting monoliths to microservices, as well as approaches and techniques for service decomposition. In Section 3 the procedures employed to carry out the mapping are discussed. Section 4 presents the discoveries of the systematic mapping, containing approaches, challenges, antipatterns, and best practices reported in the literature. In Section 5 concrete and theoretical effects of these discoveries are reviewed, along with the limits of the examined literature, plus the areas for future research. Section 6 shows the research findings, including the conclusions recap. The references used are presented in the last section.

## **2. Theoretical Framework**

The theories and models that back up the current work are presented in this section. Diverse concepts are explained: monolithic apps, microservices, converting monoliths to microservices, approaches, and techniques for service breakdown, challenges and antipatterns in migration, best practices and suggestions, tools and technologies for microservices migration, success measurement and metrics in microservices migration, also case studies of transitions from monolithic systems to microservices.

Shifting away from monoliths to microservices is a complicated and demanding procedure mandating thoughtful preparation and roll-out. The importance lies in choosing appropriate strategies for breaking down to ensure an easy transition as well as obtain the most of the opportunities that microservice architecture gives. The goal is to find current literature and

viewpoints on migration, guaranteeing that all relevant topics are included, and reducing any research gaps that may occur.

## 2.1. Monolithic Applications

Monolithic applications are computer programs developed as unified, indivisible elements where all components are interconnected and dependent on each other [1]. These computer programs are frequently constructed, verified, and released as a sole artifact, which can make maintenance and scalability challenges. As a monolithic system increases its scope and complication, challenges can emerge in trying to understand, modify, as well as extend them, potentially undermining the developer productivity and code quality [3, 12].

In a monolithic application, components such as the front end, backend logic, and data are tightly coupled, making it challenging to implement changes in a specific part of the application without affecting other parts [1]. Additionally, monolithic applications may be less scalable than microservices-based applications, as the scalability of a specific component can be limited by the scalability of the application as a whole [3].

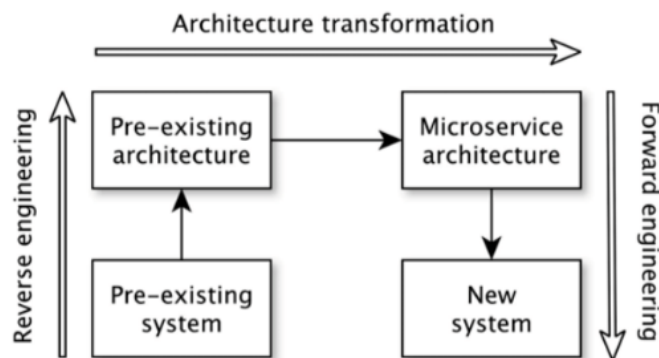
## 2.2. Microservices

Microservice is a software architecture that involves decomposing an app into tiny, autonomous components, each handling particular functionality [1, 13, 14]. Such components are created, validated, and installed separately, which enables scalability as well as maintenance of the system. Additionally, microservices allow development squads to operate more flexibly and also adapt to changes in business requirements [4].

Microservices also promote modularity and separation of responsibilities, which can improve software quality and facilitate collaboration among development teams [1, 13]. The independent nature of microservices enables teams to work simultaneously in addition to reducing risks associated with implementing changes in the application [4].

## 2.3. Migration from Monolithic Applications to Microservices

Over time, companies tend to collect apps that, due to their longevity, commonly have a monolithic architecture and a tendency to accumulate technical debt. Given the necessity to update these legacy systems due to technological advances, and shifting from monoliths to microservices represents a convincing approach [15]. Numerous companies are attracted to decomposing monoliths into a microservice architecture model [16, 17]. Making the switchover from monolithic apps to microservices involves breaking apart a monolithic app into shorter, self-contained components that can be engineered, released, and extended separately [2]. That process may be complicated, as it involves reorganizing components, managing dependencies, and adjusting to new design patterns and architecture [3]. Successfully transitioning from monoliths to microservices can improve extensibility, and maintainability, along with adaptability related to the application, as well as facilitate teamwork and continuous software delivery [4]. The migration process can be represented by the following diagram suggested in Figure 1, whereas 3 stages are identified: reverse engineering, architecture transformation, and forward engineering.



**Figure 1:** Migration process (Di Francesco et al., 2018)

Transitioning from monoliths to microservices can be carried out incrementally, starting by identifying and extracting the components that can benefit the most from decomposition into microservices [3]. This approach allows organizations to minimize risks associated with migration and obtain quick benefits by addressing the most problematic areas of the monolithic application first [2].

In [18], an approach is presented for transforming one monolithic model into a microservices-based paradigm, focusing on critical points such as modularity, scalability, and software reliability during the migration process. Coping with these challenges is crucial to ensure a successful shift to a microservice architecture.

#### 2.4. Approaches and Techniques for Service Decomposition

The process of shifting from monoliths to microservices involves various approaches together with techniques to achieve a successful transition. Some common approaches include functionality-based decomposition, which involves dividing the application into microservices that represent specific business functions; domain-based decomposition, which focuses on dividing the application into microservices that represent specific business areas; and responsibility-based decomposition, which involves dividing the application into microservices that represent specific responsibilities, such as user management or order management [2].

Another proposed approach focuses on identifying and analyzing the business functionalities of the monolith, theoretically assigning them to microservices using statistical techniques. The possibility of automating the process in the future is also mentioned, but the need for testing and demonstrations to validate its effectiveness is emphasized [18].

In [13], additional approaches and techniques for microservice decomposition are presented, offering practical perspectives and strategies for such transitioning efficiently as well as effectively. These approaches encompass methodologies based on functionality, domain, and responsibility, providing tailored options for the specific context of each project.

## 2.5. Tools and technologies for Microservice Migration

The choice of appropriate tools and technologies is crucial to facilitate the switchover from monolithic systems to microservices. Some common tools along with technologies include containers (such as Docker), container orchestrators (such as Kubernetes), API management systems (such as API Gateway), and monitoring and observability platforms (such as Prometheus and Grafana) [1]. These tools and technologies can assist development teams in efficiently and effectively managing, deploying, and scaling microservices, as well as monitoring and analyzing the performance and health of the application during and after the migration [1].

Another study [19] shows tools as Linux Containers over Docker Swarm. Containerization offers many capabilities via Docker. Also, automated continuous integration and deployment workflow on in-house hosting.

## 2.6. Success Measurement and Metrics in Microservice Migration

To assess the success in transitioning from monoliths to microservices, it is crucial to establish clear metrics and evaluation criteria. Typical measurements incorporate response duration, latency, error frequency, resource consumption, and user satisfaction [4]. These metrics can help development teams identify areas for improvement and adjust their migration strategies as needed to achieve the desired goals [4].

## 2.7. Case Studies and Successful Examples of Monolithic to Microservices Migration

Several case studies demonstrate successful migration from monolithic applications to microservices [3, 20]. One notable example is the work conducted in [21], which investigates the feasibility and effects of transforming an advanced driver assistance function into a microservice architecture. The results indicate that microservices reduced system complexity and streamlined the development process, thereby preparing the systems to tackle future challenges. By analyzing such cases, there is the potential to achieve enhanced comprehension of successful practices along with approaches applied to their migration projects [3]. In [19] another example is presented to display how scale is beneficially influenced by reconfiguring a monolithic design to microservices model.

## 3. Methodology

The present section delineates the approach taken for this research, including the systematic literature mapping, document selection system, criteria of inclusion and exclusion, data extraction, and synthesis.

Specific filters are applied to select relevant articles within the domain of microservices design. An approach followed in [22] serves as a reference, and a general overview of the approach is provided along with an explanation of the filters used.

### 3.1. Systematic Literature Mapping

Systematic literature mapping refers to a research approach that enables the spotting, assessment, along fusion of available evidence in a specific area of study [23]. This approach involves conducting comprehensive and systematic searches in academic databases and other sources of information to identify relevant studies and gain a panorama of cutting-edge progress on the study topic [24].

### 3.2. Research Queries

The chosen research queries for this investigation are on approaches, challenges, and antipatterns in shifting from monoliths to microservices. These queries are displayed in the following exhibit:

**Table 1**  
Research Queries

RQ	Research Query	Justification
Q1	Which are the approaches utilized for transitioning from monoliths into microservices?	Understanding those approaches applied in shifting from monoliths to microservices offers a panorama of the strategies and implements utilized in the present process.
RQ2	What are the common challenges in the process of converting monolithic apps to microservices?	Identifying the common challenges in the transition of monoliths into microservices will provide an understanding of every difficulty that arises during this shift together with developing strategies to recover from them.
RQ3	What antipatterns have been singled out when moving from monolithic systems to microservices?	Identifying antipatterns in converting from monoliths to microservices will help avoid common mistakes and adopt recommended practices for achieving successful migration.
RQ4	Which are the ideal approaches and recommendations when it comes to transitioning from monolithic apps to microservices?	The ideal approaches and recommendations can be helpful as guidelines to manage migration more effectively.
RQ5	What are the advantages and disadvantages associated with transitioning from monolithic apps to microservices?	Assessing the positives and negatives of shifting away from monoliths to microservices gives an understanding into the upsides and possible constraints of this transition.

These research queries intend to cope with different viewpoints presented in the process of switching monoliths into microservices.

### 3.3. Query String

A carefully designed search string is created to address the research focus, ensuring the inclusion of relevant studies on microservices and migration. The Boolean operators AND and OR are used to ensure the inclusion of studies that contemplate both terms: microservices and migration. The use of AND ensures the presence of both topics, while OR allows for variations in terminology, such as "Microservices" or "Microservice".

Initially, the search string is broad and iteratively refined to narrow down the results. Different combinations of terms and Boolean operators are tested until an optimal string is obtained that returns a manageable number of relevant articles. The final search string used is: ("Microservices" OR "Microservice") AND ("Migration" OR "Challenges" OR "Approaches" OR "Antipatterns").

### 3.4. Eligibility Criteria and Search Filters

Eligibility criteria are rules applied during the present study selection process to determine if a study is relevant and should be included in the mapping [24]. These criteria can be based on aspects such as study type, methodology used, study population, and relevance to the research questions [23].

Table 2 summarizes the eligibility criteria established in the present systematic literature mapping, based on [22]. These criteria were essential to delimit the scope of the mapping and ensure the inclusion of relevant studies that provide valuable information.

**Table 2**  
Eligibility Criteria

Criteria	Inclusion	Exclusion
Study category	Any type of research	Non-academic sources, opinion articles, editorials
Methodology	Any methodology	Non-relevant methodologies for the research topic
Study Population	Studies on the migration of monolithic applications to microservices	Studies on other topics or unrelated to the research questions
Publication Language	English or Spanish	Other languages
Publication Date	Recent publications (last 5 years)	Outdated publications
Availability of Full Text	Full-text available	Abstract-only or unavailable studies

These criteria enable the selection of studies that align with the research topic and contribute valuable insights to the investigation.

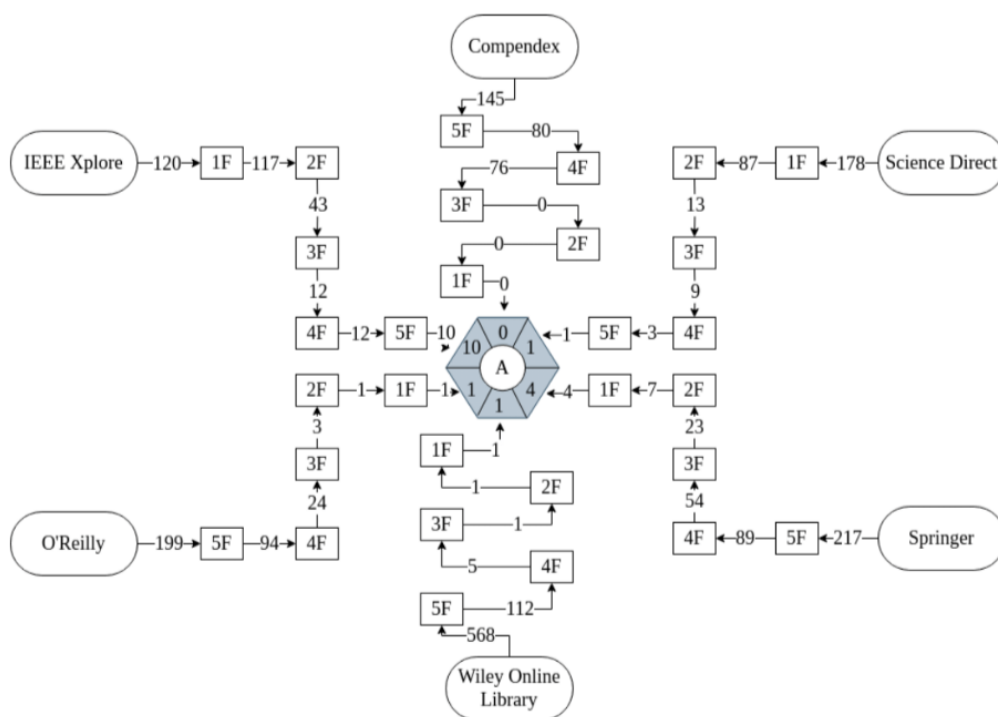
### 3.5. Filters

Five filters (1F to 5F) are applied to refine the search results:

- **1F**: Review of articles by submitting the query through research databases. Search specifics include publication date span, publication category, and vocabulary.
- **2F**: Reading the headline, tags, and synopsis of the research to assess its relevance.
- **3F**: Elimination of duplicate studies to avoid data repetition.
- **4F**: Checking the opening, discoveries, and deductions of the paper for enhanced comprehension of the study.
- **5F**: Reading each full article to have a complete understanding of the study along with its relevance to this research.

### 3.6. Information seeking Execution

The information seeking takes place sequentially across the listed databases in the following order: IEEE Xplore, Springer, Science Direct, O’Reilly, Wiley Online Library, and Compendex. The numbers shown represent the cumulative filtered studies after searching each database. In addition to the search string, the first filter applied was the publication date (2018–2023), content type or publication, and language (English). At this stage, Figure 2 was generated, illustrating the study filtering process.



**Figure 2:** Systematic Mapping Results

In the above figure, the filtered studies are later revised to identify the relevant research articles for the systematic mapping.



### 3.7. Study Selection Process

In this process, which involves the revision of the filtered articles in the information-seeking phase, is determined their significance related to the research queries [24]. This process is carried out in several stages, starting with the removal of duplicates and the review of study titles and abstracts to identify those that address the research topic [23].

Next, a full-text review is conducted on the selected studies from the previous stage, aiming to assess their quality and relevance in detail [24]. That involves the review of various factors including the methods employed, the legitimacy and dependability of the findings, plus the enrichment of understanding in the research area [23].

The conceptual categories used for document classification were defined based on the key topics covered in the research questions, including approaches for migration, challenges, antipatterns, and best practices.

Finally, the studies that meet the established criteria for quality and relevance are selected and included in the data synthesis and results analysis [24].

### 3.8. Information selection

At this point, relevant data coming from selected studies is collected and analyzed to dig the research queries together with providing an overview of the current state of this topic under study, in this case, transitioning from monolithic systems to microservices [24]. This process includes distinguishing tendencies within the content, associating outcomes, along finding areas of agreement and distinction through the literature [23].

## 4. Findings

Within this area, every finding obtained from this structured literature mapping around transitioning from monoliths to microservices is presented. Each following subsection corresponds to each research query. In essence, the final intention is to connect the following findings with the research queries previously stated.

### 4.1. Most Common approaches utilized for a transition from monoliths into microservices (RQ1)

In this section, the most common approaches spotted in the literature are revised. One spotted tactic is Service Cutter [2], a structured procedure for module breakdown based on two concepts: coupling and cohesion. This method applies many rules to identify component boundaries, as a result, it provides support in the process of changing to a microservices model.

Another approach is domain-driven decomposition, which involves dividing the application into services that represent specific business domains [3]. Domain-driven design is the main concept that motivates this type of decomposition, as a consequence it permits better alignment between software architecture and business needs.

Responsibility-based decomposition is another technique found in the literature, where services are divided based on functional and non-functional responsibilities [4]. It allows better

separation of concerns, modularity, scalability with maintainability.

Another proposal is a cooperative clustering-oriented solution for automated microservices detection by extracting them from business processes [25]. Here, a separation between models to capture structural dependencies, data dependencies, together with semantic dependencies of the business processes is made. Collaborative clustering means preventing loss of details and summing the extracted information, and as a consequence, to have better precision in microservice detection. In [26] it is recommended to use continuous integration and continuous delivery for end-to-end automation of release administration and installation, individual databases per service, service discovery, monitoring, and the use of virtual machines or containers.

Additionally, hybrid approaches that combine different decomposition techniques, such as functionality-based decomposition and domain-based decomposition are identified [3]. Multiple perspectives and criteria are considered in this hybrid approach, which leads to greater flexibility in the whole migration process.

In [27] most systems are rebuilt using modern technologies, whereas legacy code is restructured in only two examples. This validates a trend also spotted in [3].

## 4.2. Most Common Challenges Encountered During the Migration (RQ2)

Every long-standing monolithic system is singular, and shifting to microservices presents obstacles [16]. One of the top issues is managing dependencies between services, as the decomposition of a monolithic application can lead to a complex set of interdependencies [3]. The negative aspects are, on one hand, scalability and maintainability of the solution can be affected, and on the other hand complexity of the architecture can increase.

Another common difficulty throughout the transition is adapting to new design together with architectural patterns, such as communication between services and managing distributed data [2]. These patterns require important changes in the system's architecture, including additional skills along with knowledge from the development team.

The article [28] remarks three significant challenges in microservice migration: timeshare, persistence, and information integrity. The concept of timeshare allows multiple services to be utilized by diverse organizations possessing unique needs while accommodating data protection priorities. Persistence or statefulness affects non-functional criteria around flexibility, robustness, and accessibility within the upgraded platform. Information integrity issues arise upon transitioning legacy scripts that tap into centralized data storage into microservices accessing decentralized data servers [28].

Troubles around undertaking microservices decoupling, underappreciating the impacts of service isolation, and having a single data storage per microservice along with problems in managing microservice paradigm are challenges described in [26].

The reorganization of components and code refactoring can also pose challenges during the migration, as they can introduce regression risks and affect the software's quality [4]. Also, the migration may require the adoption of a set of new technologies, including containers and orchestration platforms, this may result in compatibility problems and a slow learning curve [3]. In a study conducted in [27] technological obstacles are informed across entities of all scales, though large institutions primarily face challenges to sync big teams and build bridges between them. Experienced corporations need to begin a mentality shift while instituting agile

workflows.

### **4.3. Antipatterns singled out when moving from monolithic systems to microservices (RQ3)**

Antipatterns are bad habits that can be counterproductive when moving monolithic systems to microservices. The first identified antipattern: is excessive service decomposition, which can result in many microservices that are difficult to manage and maintain [3]. To address this issue, a more balanced and pragmatic approach to service decomposition is proposed, considering factors such as cohesion, coupling, and application complexity [2].

Another antipattern is the lack of consideration for the implications of migration on non-functional aspects such as security, privacy, and performance [4]. To address this issue, it is recommended to perform an extensive analysis of non-functional needs including adopting design and architectural practices that ensure the fulfillment of these requirements during the migration process [3]. Over-prioritizing technology over design along with the transition process [26] can be problematic too.

Regarding databases, [29] points out that several companies implemented microservices wired into legacy data stores or existing repository clusters, despite this potentially shortening the benefits of implementing microservice architectures because they were not able to divide the databases. This situation brings unwanted coupling. In this case, the professionals propose splitting the information in present data sources so each microservice taps its exclusive repository.

Another thing that could happen is to grow large components, as described in [19] as numerous entities find, feature upon feature, modules can inflate over time. Troubled by monolithic components with overstuffed functionality. This introduces unneeded complications, uncertainty on where to put the new features, and lastly poor software quality.

### **4.4. Best Practices and Recommendations (RQ4)**

Many best practices and recommendations were found in the literature. One best practice is to adopt an incremental and evolutionary approach to migration, starting with the decomposition of the most critical or problematic components of the application and gradually progressing towards a complete microservice architecture [3, 30]. The Strangler pattern [27] can be used to incrementally shift the current legacy solution into a microservice architecture.

Another recommendation is to establish effective communication and close collaboration between development, operations, and business teams to ensure proper alignment between the software architecture and business needs [2]. This can facilitate adaptation to changing requirements and team skills, as well as refine quality and effectiveness over the transition plan.

As [18] conveys, the importance of maintaining high levels of communication between multidisciplinary teams included in the construction of the latest model is also mentioned. Additionally, thorough testing is recommended before, during, and after the migration process to ensure smooth transitions and maintain software quality.

The following migration patterns are recommended in [31]: enabling CI, retrieving the existing structure, dismantling the legacy system, shifting code interdependencies to service

invocations, presenting service catalog, placing an internal load balancer, implementing circuit breaker, containerizing modules, clustering implementation with container orchestration, among others.

Lastly, adopting agile development practices and continuous delivery in the migration process is recommended to facilitate adaptation to changing requirements and enable rapid incorporation of enhancements and fixes in the application [4]. This can improve responsiveness, and flexibility, and reduce the risks associated with the migration process.

#### **4.5. Advantages and disadvantages associated with transitioning from monoliths to microservices regarding scalability, maintainability, along performance (RQ5)**

An important advantage is that microservices can improve the scalability of the application by allowing individual services to scale independently [3, 29, 19]. Like related studies [29, 32], maintainability, as well as scalability, were spotlighted as prime motivators for such transition. The fact of scaling independently provides an enormous advantage in application performance and responsiveness, particularly on peaks of consumption. Capabilities such as facilitating differentiated accessibility and scale tuning for system segments, leveraging diverse frameworks avoiding vendor encasement, accelerated service launch, along with boosted codebase legibility [31].

Another advantage is that microservices can improve the maintainability of the application by enabling individual services to be engineered, validated, and released separately [2]. This may reduce the time as well as the effort required for modifying the system and enhancing the all-round software quality. In terms of technical debt [33] shows that past a comparatively small timescale, the technical debt is inclined to expand less aggressively versus the monolithic approach.

Regarding fault tolerance [29] describes that microservices crashes tend not to broadly disrupt the entire application. Whereas in a monolithic system, a module failure may halt the complete program.

However, there are also disadvantages associated with transitioning away from monolithic systems to microservices. One negative aspect is that the migration process can be complex and challenging, requiring careful planning and execution [3]. Challenges including system partitioning into discrete services alongside tracking and supervising those services figure among other considerations turn microservices adoption non-trivial [31]. Additionally, managing dependencies between services can be difficult and increase the complexity of the architecture [2].

Regarding effort estimation: sizing up the coding cycle of microservices is viewed as more imprecise compared to estimates for monolithic solutions [29].

Another disadvantage is that microservices can increase the cost and complexity of operations, as each service requires individualized tracking, administration, and scaling [4]. Additional expertise and capabilities may be required from the operations team and increase the total cost of ownership of the application. The study made in [34] concludes that despite migration presenting natural complications, said complications seem simpler to resolve over monolith system difficulties.

## 5. Discussion

Based on the results, approaches, and tactics for transitioning from monoliths to microservices are compared, highlighting advantages and disadvantages. Consequently, and given their importance, research implications are presented. In the latter subsections, feedback is given about limitations in the reviewed studies along with future research areas.

### 5.1. Approaches and Techniques

The review of literature displays many approaches for shifting from monoliths into microservices, each one of them having its pros and cons [3]. Responsibility-driven decomposition offers greater modularity, although it can generate complex interdependencies [4]. On the other hand, domain-driven decomposition aligns the architecture with business needs but can be complex in unclear or changing domains [3]. Hybrid approaches, such as mixing functionality and domain-based breakdown, hand over greater flexibility, however, they need more coordination [2]. The selection of several migration patterns is described in [31].

### 5.2. Research Implications

From the theoretical point of view, this research tries to give a structured perspective about the state-of-the-art phase of investigation along with practices in this field [23]. Is the expectation of this work that some of the findings discussed can open future lines of investigation. Identifying areas of consensus as well as discrepancy among the studies and suggesting opportunities for improvement in the microservices migration phase [24].

Conversely, from the practical side, identified approaches can serve as a guide for engineers and software architects in the process of handling a successful migration, by helping them to tackle the challenges and maximizing the advantages associated with the microservice model [3].

### 5.3. Reviewed Studies Limitations

Limitations around the reviewed studies must be taken into consideration while observing the results. Primarily, they are based on case studies or practical experiences, which limits their generalizability to other contexts [3]. Moreover, biases from the authors are possible, and that can influence the neutrality of the results [2]. The absence of strict empirical evidence of the approaches is another limitation, which makes it challenging to identify which are the truly best practices and validate the effectiveness of the proposed solutions in other contexts [1]. Worth highlighting, most of the current works emphasize technical and architectural aspects, neglecting organizational and cultural concerns, which are also important for the success of the migration [3, 18].

Future lines of investigation have to consider wide-ranging contexts, to collect rigorous empirical evidence, as well as to take into account every technical and non-technical requirement to improve the reliability of the results. All of the above limitations emphasize the efforts that have to be made to address these gaps.

## 5.4. Future Research

Future research areas within transitioning from monoliths to microservices are identified. Upcoming research should consider developing more automated approaches for service decomposition, taking into consideration different criteria [2].

Further research could, for instance, investigate the organizational and cultural aspects of migration, such as communication between teams, and collaboration among the IT department, operations, and business teams, as a consequence of the new model, design, and architectural patterns [3]. Handling these non-technical aspects can enhance the quality and efficiency of the system migration process.

Future lines can dig into the advantages and disadvantages of scaling microservices in the process of assessing challenges related to orchestration and deployment [1]. Another idea for future lines of investigation, based on the works reviewed, is the effects of applying microservices in terms of performance, reliability, and security. The impact of cloud computing represents an emerging trend that warrants exploration in future work. Upcoming works might conduct a more intensive assessment of exposed approaches, motivators, and impediments to ease shifting legacy monolithic apps to microservices [27].

## 6. Conclusions

The approaches identified in the academic literature regarding performing the decomposition of legacy monolithic systems into microservices are domain-driven, responsibility-driven, and hybrid approaches. Some examples are found where a detailed exploration of the existing system can help establish clear boundaries between the resulting microservices after the migration process.

Common troubles and issues in the migration process include dependency management, adaptation to new models and architectural patterns, and reorganization of components as well as code refactoring.

Antipatterns are also discussed, such as excessive service decomposition and lack of consideration for non-functional requirements of the migration. Best practices and suggestions are introduced to address challenges and consolidate the benefits associated with this transition, including the adoption of an incremental approach, the establishment of effective communication among teams, the embrace of agile development practices along the use of CI/CD techniques.

The domain and responsibility-driven approaches align with established decomposition strategies discussed in migration frameworks such as Strangler and microservice refactoring patterns. The identified challenges also mirror issues reported in empirical studies on technical debt and dependency management. However, the literature points to several emerging directions that could complement these findings, including expanded research on organizational change management during migration, and the need for standardized metrics to evaluate migration success across context.

The process of migration into microservices seems promising for further review, with the necessity of finding better approaches to address its challenges, all of this to increase the benefits associated with the adoption of microservice architecture. Yet, it remains essential to acknowledge the nonexistence of silver bullet solutions, as each system and context may require

specific strategies together with techniques to manage its challenges. In any case, industry and academia should continue investigating as well as sharing experiences to improve knowledge and practice in this domain.

## References

- [1] S. Newman, *Building microservices*, " O'Reilly Media, Inc.", 2021.
- [2] M. Gysel, L. Kölbener, W. Giersche, O. Zimmermann, Service cutter: A systematic approach to service decomposition, in: *Service-Oriented and Cloud Computing: 5th IFIP WG 2.14 European Conference, ESOC 2016, Vienna, Austria, September 5-7, 2016, Proceedings 5*, Springer, 2016, pp. 185–200.
- [3] P. Di Francesco, P. Lago, I. Malavolta, Migrating towards microservice architectures: an industrial survey, in: *2018 IEEE international conference on software architecture (ICSA)*, IEEE, 2018, pp. 29–2909.
- [4] D. Taibi, V. Lenarduzzi, C. Pahl, A. Janes, Microservices in agile software development: a workshop-based study into issues, advantages, and disadvantages, in: *Proceedings of the XP2017 Scientific Workshops*, 2017, pp. 1–5.
- [5] W. Luz, E. Agilar, M. C. de Oliveira, C. E. R. de Melo, G. Pinto, R. Bonifácio, An experience report on the adoption of microservices in three brazilian government institutions, in: *Proceedings of the XXXII Brazilian Symposium on Software Engineering*, 2018, pp. 32–41.
- [6] F. Rademacher, J. Sorgalla, S. Sachweh, Challenges of domain-driven microservice design: A model-driven perspective, *IEEE Software* 35 (2018) 36–43.
- [7] D. Taibi, V. Lenarduzzi, C. Pahl, Architectural patterns for microservices: a systematic mapping study, in: *CLOSER 2018: Proceedings of the 8th International Conference on Cloud Computing and Services Science; Funchal, Madeira, Portugal, 19-21 March 2018*, SciTePress, 2018, pp. 1–12.
- [8] J. Soldani, D. A. Tamburri, W.-J. Van Den Heuvel, The pains and gains of microservices: A systematic grey literature review, *Journal of Systems and Software* 146 (2018) 215–232.
- [9] C. Pahl, P. Jamshidi, Microservices: A systematic mapping study., *CLOSER* (1) (2016) 137–146.
- [10] P. Di Francesco, P. Lago, I. Malavolta, Architecting with microservices: A systematic mapping study, *Journal of Systems and Software* 150 (2019) 77–97.
- [11] V. Velepucha, P. Flores, Monoliths to microservices-migration problems and challenges: A sms, in: *2021 Second International Conference on Information Systems and Software Technologies (ICI2ST)*, IEEE, 2021, pp. 135–142.
- [12] D. Sanchez, O. Mendez, H. Florez, An approach of a framework to create web applications, in: *Computational Science and Its Applications–ICCSA 2018: 18th International Conference, Melbourne, VIC, Australia, July 2–5, 2018, Proceedings, Part IV 18*, Springer, 2018, pp. 341–352.
- [13] S. Newman, *Monolith to microservices: evolutionary patterns to transform your monolith*, O'Reilly Media, 2019.
- [14] D. Sanchez, A. E. Rojas, H. Florez, Towards a clean architecture for android apps using

- model transformations, *IAENG International Journal of Computer Science* 49 (2022) 270–278.
- [15] V. Velepucha, P. Flores, J. Torres, Migration of monolithic applications towards microservices under the vision of the information hiding principle: a systematic mapping study, *Advances in Emerging Trends and Technologies: Volume 1* (2020) 90–100.
- [16] J. Kazanavičius, D. Mažeika, Migrating legacy software to microservices architecture, in: *2019 Open Conference of Electrical, Electronic and Information Sciences (eStream)*, IEEE, 2019, pp. 1–5.
- [17] P. Gómez, M. E. Sánchez, H. Florez, J. Villalobos, An approach to the co-creation of models and metamodels in enterprise architecture projects., *Journal of Object Technology* 13 (2014) 2–1.
- [18] L. De Lauretis, From monolithic architecture to microservices architecture, in: *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, IEEE, 2019, pp. 93–96.
- [19] M. Mazzara, N. Dragoni, A. Bucchiarone, A. Giaretta, S. T. Larsen, S. Dustdar, Microservices: Migration of a mission critical system, *IEEE Transactions on Services Computing* 14 (2018) 1464–1477.
- [20] H. Ricaurte, H. Florez, Architectural approach for google services integration, in: *Applied Informatics: Fourth International Conference, ICAI 2021, Buenos Aires, Argentina, October 28–30, 2021, Proceedings 4*, Springer, 2021, pp. 483–496.
- [21] O. B. Benderius, J. L. Lotz, C. B. Berger, A. V. Vogelsang, *Microservice architectures for advanced driver assistance systems: A case-study*, 2019.
- [22] M. Guerrero-Calvache, G. Hernández, Team productivity in agile software development: A systematic mapping study, in: *International Conference on Applied Informatics*, Springer, 2022, pp. 455–471.
- [23] K. Petersen, R. Feldt, S. Mujtaba, M. Mattsson, Systematic mapping studies in software engineering, in: *12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12*, 2008, pp. 1–10.
- [24] B. Kitchenham, S. Charters, et al., *Guidelines for performing systematic literature reviews in software engineering*, 2007.
- [25] M. Daoud, A. El Mezouari, N. Faci, D. Benslimane, Z. Maamar, A. El Fazziki, Towards an automatic identification of microservices from business processes, in: *2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, IEEE, 2020, pp. 42–47.
- [26] A. Henry, Y. Ridene, Migrating to microservices, *Microservices: Science and Engineering* (2020) 45–72.
- [27] J. Fritzsich, J. Bogner, S. Wagner, A. Zimmermann, Microservices migration in industry: intentions, strategies, and challenges, in: *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, 2019, pp. 481–490.
- [28] A. Furda, C. Fidge, O. Zimmermann, W. Kelly, A. Barros, Migrating enterprise legacy source code to microservices: on multitenancy, statefulness, and data consistency, *Ieee Software* 35 (2017) 63–72.
- [29] D. Taibi, V. Lenarduzzi, C. Pahl, Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation, *IEEE Cloud Computing* 4 (2017)



- 22–32.
- [30] H. Florez, M. Sánchez, J. Villalobos, G. Vega, Coevolution assistance for enterprise architecture models, in: *Proceedings of the 6th International Workshop on Models and Evolution*, 2012, pp. 27–32.
  - [31] A. Balalaie, A. Heydarnoori, P. Jamshidi, D. A. Tamburri, T. Lynn, Microservices migration patterns, *Software: Practice and Experience* 48 (2018) 2019–2042.
  - [32] H. Knoche, W. Hasselbring, Drivers and barriers for microservice adoption—a survey among professionals in germany, *Enterprise Modelling and Information Systems Architectures (EMISAJ)–International Journal of Conceptual Modeling: Vol. 14, Nr. 1* (2019).
  - [33] V. Lenarduzzi, F. Lomio, N. Saarimäki, D. Taibi, Does migrating a monolithic system to microservices decrease the technical debt?, *Journal of Systems and Software* 169 (2020) 110710.
  - [34] M. Kalske, N. Mäkitalo, T. Mikkonen, Challenges when moving from monolith to microservice architecture, in: *Current Trends in Web Engineering: ICWE 2017 International Workshops, Liquid Multi-Device Software and EnWoT, practi-O-web, NLPIT, SoWeMine*, Rome, Italy, June 5-8, 2017, Revised Selected Papers 17, Springer, 2018, pp. 32–47.