Guilherme Luciano Donin Villaca

Strategies to mitigate anti-patterns in microservices before migrating from a monolithic system to microservices

Cascavel-PR

2022

Guilherme Luciano Donin Villaca

Strategies to mitigate anti-patterns in microservices before migrating from a monolithic system to microservices

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre pelo Programa de Pós-Graduação em Ciência da Computação (PPGComp) da Universidade Estadual do Oeste do Paraná – Unioeste, campus de Cascavel.

Universidade Estadual do Oeste do Paraná – Unioeste – Cascavel

Centro de Ciências Exatas e Tecnológicas – CCET

Programa de Pós-Graduação em Ciência da Computação – PPGComp

Supervisor: Prof. Dr. Ivonei Freitas da Silva

Cascavel-PR

2022

Guilherme Luciano Donin Villaca

Strategies to mitigate anti-patterns in microservices before migrating from a monolithic system to microservices/ Guilherme Luciano Donin Villaca. – Cascavel-PR, 2022-

140p. : il. (algumas color.) ; 30 cm.

Supervisor: Prof. Dr. Ivonei Freitas da Silva

Dissertação (Mestrado)– Universidade Estadual do Oeste do Paraná – Unioeste – Cascavel
Centro de Ciências Exatas e Tecnológicas – CCET
Programa de Pós-Graduação em Ciência da Computação – PPGComp, 2022.

1. monolithic. 2. migration. 3. microservices. 4. anti-patterns. 5. strategies I. Orientador. II. Universidade xxx. III. Faculdade de xxx. IV. Título

Guilherme Luciano Donin Villaca

Strategies to mitigate anti-patterns in microservices before migrating from a monolithic system to microservices

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre pelo Programa de Pós-Graduação em Ciência da Computação (PPGComp) da Universidade Estadual do Oeste do Paraná – Unioeste, campus de Cascavel.

Trabalho aprovado. Cascavel-PR, 10 de maio de 2022:

[illegible]

Prof. Dr. Ivonei Freitas da Silva
Orientador(a)

Wesley Klewerton

Dr. Wesley Klewerton Guez Assunção
Johannes Kepler Universität Linz

[illegible]

Dr. Sidgley Camargo de Andrade
Universidade Tecnológica Federal do Paraná

Cascavel-PR
2022

[illegible]

Para meus filhos Luís Guilherme e Heloísa, minha inspiraçao todos os dias.

Acknowledgements

Agradeço aos colegas do NTI da Reitoria da Unioeste, onde trabalhei boa parte deste mestrado, pela compreensão devido ao tempo que foi necessário para se dedicar a esta pesquisa. Ao meu orientador pela paciência, ensinamentos e dedicação despendida para que esta pesquisa fosse possível e por fim a minha família, meus pais Sidnei e Maria Inês, pelos conselhos e suporte que foi me dado durante toda a vida, aos meus irmãos, Rafael, Daniele e Maurício pelo apoio e torcida e aos meus amigos pelo apoio e distração nos momentos mais dificeis.

“São quatro os homens
Aquele que não sabe, e não sabe que não sabe. É um tolo: evite-o;
Aquele que não sabe, e sabe que não sabe. É um simples: ensine-o;
Aquele que sabe, e não sabe que sabe. Está dormindo: acorde-o;
Aquele que sabe, e sabe que sabe. É um sábio: siga-o“
Provérbio Árabe

Abstract

DONIN VILLACA, Guilherme Luciano. **Strategies to mitigate anti-patterns in microservices before migrating from a monolithic system to microservices**. Supervisor: Prof. Dr. Ivonei Freitas da Silva. 2022. 140f. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual do Oeste do Paraná, Cascavel – Paraná, 2022.

Microservice architectures are affected by the so-called anti-patterns, i.e., bad implementation habits that affect software quality. Considering that the vast majority of microservices-based systems are migrated from monolithic legacy systems, these anti-patterns can be an undesirable inheritance that must be avoided. Some of these microservice anti-patterns should be mitigated in the early stages of the migration process, namely during pre-migration. To assist practitioners and researchers in mitigating microservice anti-patterns during pre-migration, this dissertation presents an exploratory study that examines existing strategies. This study addresses the anti-patterns already catalogued in the literature and defined which ones can be mitigated through better pre-migration planning when moving from monolithic to microservice-based systems. This study relies on multi-methods composed of a systematic literature mapping, a rapid review and interview with practitioners. Then, results are analysed using thematic analysis. As a result, ten strategies were identified, namely adopt the domain-driven design, use the strangler pattern, identify tight coupling, use of backlog strategy, group entities, classify data in business subsystem, look at data first, focus on clean architecture, adopt the twelve factor app, and adoption of the evolvability assurance.

Keywords: monolithic, migration, microservices, anti-patterns, strategies.

Resumo

DONIN VILLACA, Guilherme Luciano. **Estratégias para mitigar antipadrões em microsserviços antes da migração de um sistema monolítico para microsserviços**. Supervisor: Prof. Dr. Ivonei Freitas da Silva. 2022. 140f. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual do Oeste do Paraná, Cascavel – Paraná, 2022.

As arquiteturas de microsserviços são afetadas pelos chamados antipadrões, que são maus hábitos de implementação que afetam a qualidade do software. Considerando que a grande maioria dos sistemas baseados em microsserviços são migrados de sistemas legados monolíticos, esses antipadrões podem ser uma herança indesejável que deve ser evitada pois isto afeta negativamente a arquitetura de microsserviços, causando problemas de manutenibilidade, gerenciamento e evolução do sistema. Alguns desses antipadrões de microsserviços devem ser mitigados nas fases iniciais do processo de migração, ou seja, durante a pré-migração. Para ajudar profissionais e pesquisadores a mitigar antipadrões de microsserviços durante a pré-migração, esta dissertação apresenta um estudo exploratório que examina as estratégias existentes. Este estudo aborda os antipadrões já catalogados na literatura e define quais podem ser mitigados por meio de um melhor planejamento na fase de pré-migração ao passar de sistemas monolíticos para sistemas baseados em microsserviços. Este estudo se baseia em multi-métodos compostos por um mapeamento sistemático da literatura, uma revisão rápida e entrevistas com profissionais. Em seguida, os resultados são analisados por meio da análise temática. Como resultado, dez estratégias foram identificadas, que são, adotar o *domain-driven design*, uso da *strangler pattern*, identificar o alto acoplamento, usar uma estratégia de *backlog*, agrupamento de entidades, classificar dados como subsistema de negócios, olhar primeiro para os dados, uso de *clean architecture*, adotar *twelve factor app* e adotar uma estratégia de garantia de evolução do sistema.

Palavras-chave: monolitico, migração, microsserviços, anti-padrões, estratégias

List of Figures

List of Tables

List of abbreviations and acronyms

MSA	Microservice Architecture
TD	Techinical Debt
DDD	Domain-Drive Design
API	Application Programming Interface
RQ	Research Question
SRQ	Subresearch Question
SLM	Systematic Literature Mapping
RR	Rapid Review
GQM	Goal, Question and Metric
CD/CI	Continuous integration and continuous deployment

Contents

1 Introduction

1.1 Context

Monolithic software suffers degradation over time and, to maintain quality, the best approach is to reduce software defects (bug fixes) and design problems (refactoring) (Ahmed et al., 2015). These problems can also be called anomalies in the code and are key factors for degradation, if not removed, the software can suffer from erosion, which occurs when architectural violations are introduced and drift, when bad design decisions affect the architecture (Macia et al., 2012). Degradation occurs due to constant changes and implementations of new features, which is the way to meet the requirements that arise as the system grows. Such changes end up in conflict with the way the software was originally built, either with the design or with the initial implementation, and become expensive to revert, in addition, the technology used to build the software may become outdated after some years, becoming incompatible with current demands like the cloud. Cloud computing is a paradigm shift from the practice of on-premise computing and resource utilization to a pool of virtualized computing services. (Ogbole; Ogbole; Olagesin, 2021). To contain degradation, there are several approaches, such as system complete rebuilding using new design techniques and technologies. For Candela et al. (2016) modernization of software is one of the paths to be taken, whether in the form of refactoring, remodeling or migration to another approach such as software product lines or to a new architecture, such as microservices.

The essence of microservices is to decompose the application's components into small functionalities logically grouped as services, each running in its own process and communicating with lightweight mechanisms, usually an HTTP resource API (Jambunathan; Y., 2016a). A constant idea in this universe of applications in the cloud is the reuse of software, which brings efficiency to the application as a whole, making it possible to consume services without the need to create functionality from scratch (Linthicum, 2016). Migrating from monolithic systems to microservices has been an industry trend in recent years, inspired by large companies like Amazon, Linkedin, Netflix and others, and gaining popularity. However, if you are considering a new architecture such as microservices, you should keep in mind that an architecture based on distributed systems also brings more complexity than a monolithic system.

1.2 Motivation

In the literature there are researches that point out problems that occur in the migration from monolithic to microservices and that after the migration is completed these problems negatively affect the quality of the new architecture. According to Taibi and Lenarduzzi (2018), Taibi, Lenarduzzi and Pahl (2019) these problems also called bad smells or antipatterns are caused by bad practices that occur during development and affect some quality attributes of the software such as understandability, testability, extensibility, reusability and maintainability. It is also concluded that practitioners need to carefully make the decision of migrating to MSA based on the return on investment, since this architectural style additionally causes some pains in practice (Li et al., 2021b). Based on this, it raised the question if *the migration of monolithic systems that suffer from degradation can cause these antipatterns, and whether an understanding, analysis, refactoring or reengineering of the monolithic system can mitigate antipatterns*, i.e, whether antipatterns are inherited by the new architecture. Based on this gap, the aim of this research is what strategies can be taken in monolithic system before the migration process began so that antipatterns in microservices can be mitigated.

From the antipatterns cataloged (Taibi; Lenarduzzi, 2018; Taibi; Lenarduzzi; Pahl, 2019; Carrasco; Bladel; Demeyer, 2018), it was identified that some of them are unique for microservices, that is, they are problems related to distributed systems, communication between microservices, use or lack of use of tools or patterns etc. Other antipatterns was considered that can be analyzed even with the monolithic system, and with a better planning or study/analysis can be mitigated when the system with the architecture in microservices is ready.

1.3 Problem - Research Question

This dissertation starts with the following research question: *Before migrating from a monolithic system to microservices, what strategies are used to mitigate the antipatterns present in microservices?*. The aim is to gain a deep understanding of the challenges and solutions in this matter. There are many researches published with a catalog of bad smells and antipatterns in microservices with proposed solutions of this antipatterns in microservices systems (Taibi; Lenarduzzi, 2018; Taibi; Lenarduzzi; Pahl, 2019; Carrasco; Bladel; Demeyer, 2018), several others showed catalogs of bad smells, architectural smells in monolithic system, tools for correction and detection of smells in monolithic systems. However, none of these studies focused on strategies to be taken for the monolithic system before migrating to microservices. However, the aforementioned studies do propose solutions for the microservices context.

1.4 Methodology

To achieve the purpose of this research, the adopted methodology used was initially systematic literature mapping (Kitchenham; Charters, 2007), in the sequence, as it was also an idea from the beginning, look at gray literature (Kamei et al., 2021) with a rapid review (B. Pinto G., 2020) in order to analyze and confirm whether the SLM results would be similar. In parallel, the GQM technique (Basili; Rombach, 1988) (goal, question, metric) was used to define how to apply the interview (Marshall; Brereton; Kitchenham, 2015), and at the end was used a thematic synthesis approach (Cruzes; Dyba, 2011) to assist in the data analysis, which was the last part of this research. More details will be described in the methodology chapter.

1.5 Contribution

With this research, the intention is to contribute so that researchers and professionals can analyze the monolithic system regarding state of degradation, defects and architectural design problems, as well as have a better planning in the pre-migration and migration phase to microservices so that the antipatterns existing microservices are minimized or even avoided.

1.6 Dissertation Structure

The remainder of this dissertation is organized as follows, in chapter 2 I discuss the background that supports this research, chapter 3 has the Methodology to data collect and analysis. In chapters 4 (SLM), 5 (RR) and 6 (Interview), I describe the data collected to provide evidence of what strategies have been taken to avoid antipatterns in microservices. In chapter 7, I describe the thematic analysis process on collected data to answer the dissertation research question. And in chapter 8 the conclusion and future work.

2 Background

This chapter introduces basic information for the rest of the dissertation.

2.1 Monolithic Architecture

In the traditional application, all the business components are packaged together, distributed and deployed as a unit, this development and deployment pattern is called monolithic (Figure 1). When it comes to the monolithic architecture system, it cannot replicate the component or extension module in question, and deploying the entire application to multiple nodes results in wasted resources (Ren et al., 2018). Monolithic are rigid and tightly coupled. Any small changes to be made in any component in this kind of architecture will have challenges and large impact on productivity, time, cost and deployment. As business demands that these applications be distributed, scalable, and portable across different technologies and platforms, it is more complicated for developers to make changes, test, and deploy these applications in a short period of time (Jambunathan; Y., 2016b).

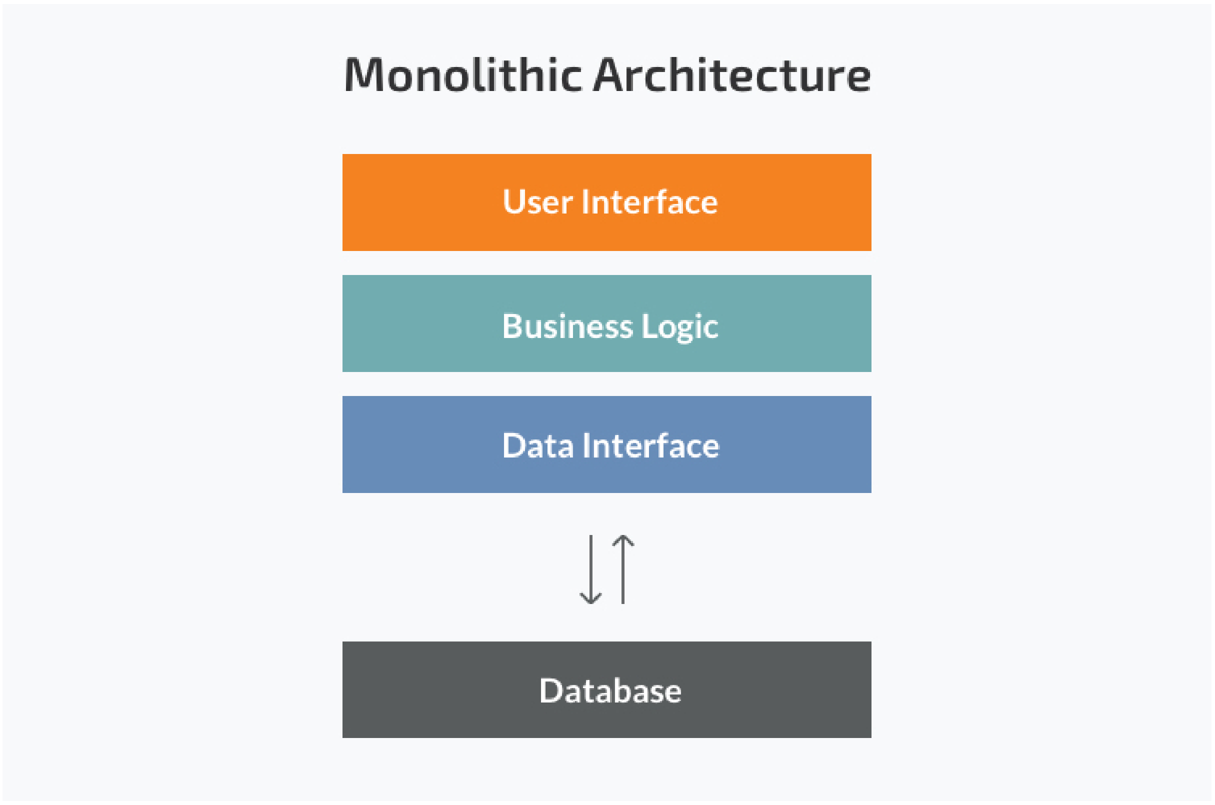


Figure 1 – Monolithic Architecture (source: (Vasylenko, 2018))

The following benefits of monolithic architecture can be mentioned: fewer cross-cutting concerns – it is simpler to hook up components to cross-cutting concerns when everything is running through the same application, less operational overhead – only one application needs to be set up, Less complex to deploy – only one application needs to be deployed. Drawbacks of monolithic architecture are as follows: coupled – is very difficult to make changes when monolith becomes very complex, continuous deployment – the entire application should be deployed on each update, scalability – difficult to scale when different modules have conflicting resource requirements, reliability – bug in any component can potentially bring down the entire application. Usually, legacy applications are always growing in size and complexity, therefore, it led to monstrous monolithic software after a few years of development and disadvantages of monolithic architecture become outweigh its advantages. Fixing bugs and adding new features to such application is a very complex and time-consuming operation. Scalability usually is not possible or require a lot of work. Once this happens, organizations start looking for a new architectural solution (Kazanavičius; Mažeika, 2019).

2.2 Why does the monolithic degrade?

Monoliths suffer from degradation/erosion over time, which happens due to constant modifications in code, requirements changes, growing number of defects (bugs) and design problems. In addition, it can decrease software performance, substantially increase evolutionary costs, and degrade software quality. Moreover, in an eroded architecture, code changes and refactorings may introduce new bugs and aggravate the brittleness of the system (Li et al., 2021a).

In the development of software systems, some bad smells are identified and fixed to maintain the systems. These smells can be classified into three categories according to their granularity: architecture smells, design smells, and code smells. Code smells are design issues that degrade understandability and changeability at the code level, which results in poor maintainability and hinders evolution of a system. Design smells indicate violation of design principles such as cyclic dependencies principle, and have a potential impact on a set of classes in design structure. Architecture smells are a typical type of architectural technical debt and can negatively affect system-wide maintainability of a software system (Tian; Liang; Babar, 2019). The growing number of smells results in degradation, and because of that the software demands a constant adaptation or enhancement within an operational software system. In the software engineering, it is widely accepted that real world software must be continually adapted or enhanced to remain operational (Khadka et al., 2015).

Software evolution has three main activities: maintenance, modernization, and replacement. Modernization is important to increase maintainability, increase flexibility and reduce costs (Comella-Dorda et al., 2000). Yang et al. (2005) mention two options, one is a complete redevelopment and the other is an evolutionary migration or reengineering and emphasizes that the first one is high risk, high cost, with long development and understanding business logic and documentation. While the second may take longer, and have the ability to minimize risks. One of the forms for modernization or reengineering is to migrate monolithic software to microservices due to the benefits that this architecture brings (Levcovitz; Terra; Valente, 2016). A monolithic architecture better suits a simple, lightweight application. The microservices architecture solution is the better choice for complex, evolving applications. Monolith should be modernizing to microservices when: (i) monolith became too big and complex to maintain or extend, (ii) Modularity and decentralization are an important aspect and (iii) the preference for long-term benefits over short-term (Kazanavičius; Mažeika, 2019).

2.3 Microservices

According to Fowler (2014) microservice (see Figure 2) consist of suites of independently deployable services organized around business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data. Microservices are built around business functions and are deployed independently. As the term "microservice"implies, services are the building blocks and the primary means of modularization in microservice architectures. Services run in separate process contexts and can be deployed, replaced, and retired individually. Each microservice focuses on providing a single business function and follows the principle of single responsibility (Knoche; Hasselbring, 2019). Thereupon, each service can be developed in the programming language that best fits the characteristics of the service, uses the most appropriate mechanism for data persistence, runs in an appropriate hardware and software environment, and is developed by different teams (Levcovitz; Terra; Valente, 2016).

Benefits of microservices architecture are as follows (Kazanavičius; Mažeika, 2019):

- Agility (Deployability) – microservices can be deployed independently and there is no need to restart an entire application;
- Reliability – a microservice fault affects that microservice alone and its consumers;
- Scalability – each microservice can be scaled independently using pools, clusters, grids;
- Cloudability – the deployment characteristics make microservices a great match for the elasticity of the cloud;

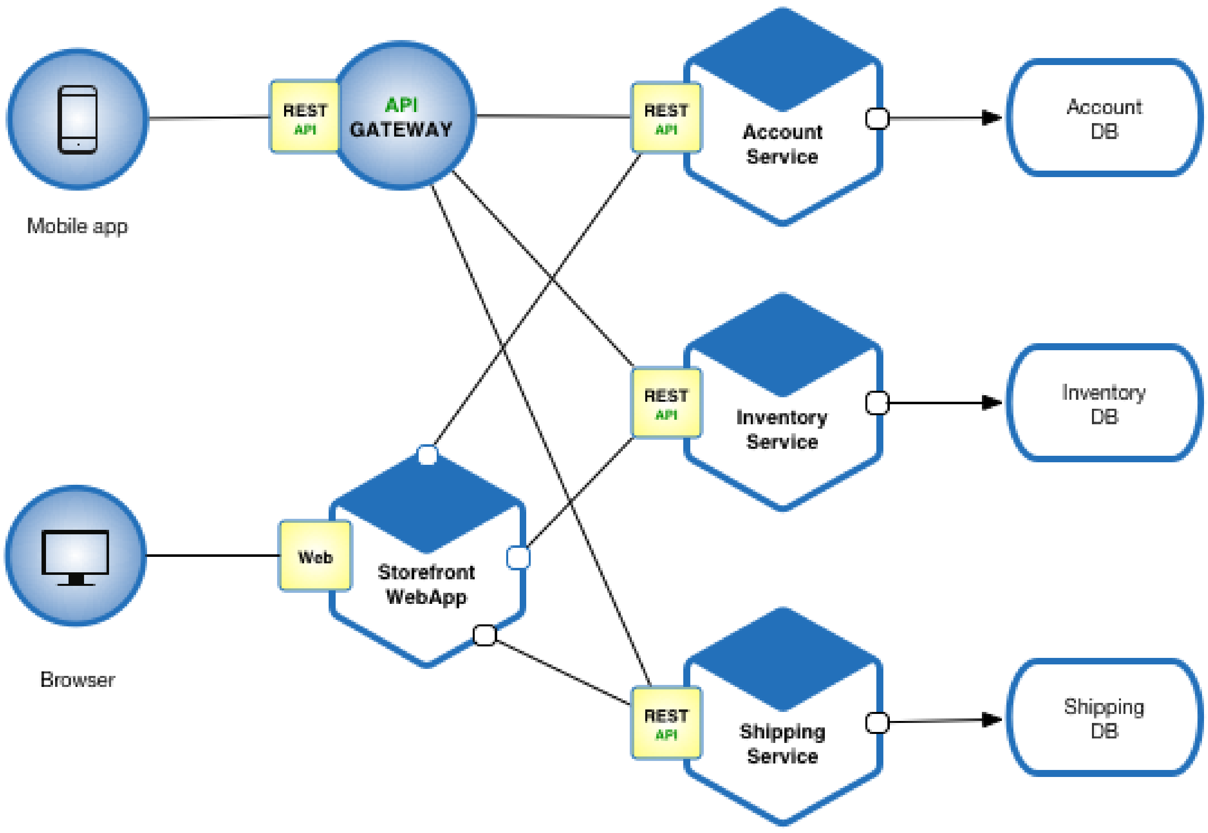


Figure 2 – Microservice Architecture (source: (Richardson, 2018))

- Modifiability – each microservice is incapsulated therefore it is more flexibility to use new frameworks, libraries, data sources, and other resources.

2.3.1 Modernization to Microservices

An approach to modernize monolithics to microservices comprising three major phases (Knoche; Hasselbring, 2019):

1. New features are implemented only as microservices.
2. An interface layer is created through which the newly created microservices can access the functions of the monolith. This interface layer serves as an anti-corruption layer to clearly separate the old and new domain models.
3. Functionality is gradually removed from the monolith and re-implemented as microservices.

A closer look at this approach reveals several important challenges, and highlights that such a modernization is far from trivial. The benefits of adopting a microservices architecture come with the complexities of distributed systems (Carrasco; Bladel; Demeyer, 2018). During the migration process, practitioners often face common problems, which are due mainly to their lack of knowledge regarding bad practices and patterns. The so called bad smells are indicators of situations that negatively affect software quality attributes such as understandability, testability, extensibility, reusability, and maintainability of the system under development (Taibi; Lenarduzzi, 2018).

2.3.2 Bad Smells/Antipatterns in Microservices

To aid practitioners in identifying the potential pitfalls, Taibi and Lenarduzzi (2018) defined 11 bad smells related to the microservices architecture. Later Carrasco, Bladel and Demeyer (2018) present 5 new architecture and 4 new migration bad smells found by digesting 58 different sources from the academia and grey literature. Taibi continues his research and identified a taxonomy of 20 antipatterns, including organizational (team oriented and technology/tool oriented) antipatterns and technical (internal and communication) antipatterns (Taibi; Lenarduzzi; Pahl, 2019).

2.3.3 List Bad Smells/Antipatterns

First the 9 smells proposed by Carrasco, Bladel and Demeyer (2018):

- **Single layer teams** - if a team is divided by layers, simple changes can require time and effort to approve between the teams.
- **Greedy Service Container** - not using containers adds a whole new level of complexity to the management and orchestration of the application.
- **Single DevOps toolchain** - microservices architecture require DevOps toolchains, such as continuous delivery, continuous integration, monitoring, and orchestration.
- **Dismiss documentation** - in microservices documenting the exposed API is particularly important. With the increased number of independent services, the overview of the system can be easily lost.
- **Grinding dusty or coarse services** - a monolith must be broken down into services, for this the proper granularity must be determined. This depends on what the application actually is doing and the wrong granularity can have profound side-effects.
- **Thinking microservices are a silver bullet** - however promising microservices seem to be, their advantages come with many inherent complexities and challenges at possibly a high cost. Therefore, this architectural style might not be worthwhile for many applications.

- **Rewrite all services into microservices at once** - many new faults can be inadvertently introduced, leading to unnecessary risks.
- **Learn as you go** - put inexperienced developers in distributed systems is generally not a good idea. Their inexperience may result in higher cost and development time.
- **Forgetting about the CAP theorem** - a distributed data store cannot simultaneously provide more than two of the following guarantees: consistency, availability, and partition tolerance.

Following the antipatterns proposed by Taibi and Lenarduzzi (2018), Taibi, Lenarduzzi and Pahl (2019):

- **Hardcoded endpoints** - hardcoded IP addresses and ports of the services between connected microservices.
- **Wrong cuts** - microservices should be split based on business capabilities, not on technical layers (presentation, business, data layers).
- **Cyclic dependency** - a cyclic chain of calls between microservices
- **API versioning** - APIs are not semantically versioned (e.g., v1.1, 1.2, etc.) For example, the returning data might be different or might need to be called differently.
- **Shared persistence** - different microservices access the same relational database. In the worst case, different services access the same entities of the same relational database.
- **ESB usage** - the microservices communicate via an Enterprise Service Bus (ESB).
- **Legacy organization** - the company still work without changing their processes and policies. As example, with independent Dev and Ops teams, manual testing and scheduling common releases.
- **Local logging** - logs are stored locally in each microservice, instead of using a distributed logging system
- **Megaservice** - a service that does a lot of things.
- **Inappropriate service intimacy** - the microservice keeps on connecting to private data from other services instead of dealing with its own data.
- **Lack of Monitoring** - Lack of usage of monitoring systems, including systems to monitor if a service is alive or if it responds correctly.

- **No API-Gateway** - microservices communicate directly with each other. In the worst case, the service consumers also communicate directly with each microservice, increasing the complexity of the system and decreasing its ease of maintenance.
- **Shared libraries** - usage of shared libraries between different microservices.
- **Too many technologies** - usage of different technologies, including development languages, protocols, frameworks.
- **Lack of microservice skeleton** - each team develop microservices from scratch, without benefit of a shared skeleton that would speed-up the connection to the shared infrastructure (e.g. connection to the API-Gateway)
- **Microservice greedy** - teams tend to create of new microservices for each feature, even when they are not needed.
- **Focus on latest technologies** - the migration is focused on the adoption of the newest and coolest technologies, instead of based on real. The decomposition is based on the needs of the different technologies aimed to adopt.
- **Common ownership** - one team own all the microservices.
- **No DevOps tools** - the company does not employ CD/CI tools and developers need to manually test and deploy the system.
- **Non-homogeneous adoption** - only few teams migrated to microservices, and the decision if migrate or not is delegated to the teams.

There are antipatterns that can be closely linked to the monolithic system and, then, they should be considered in pre-migration phase: **Wrong Cuts, Cyclic Dependency, Shared Persistence, Megaservice, Shared Libraries, Microservice Greedy, Innappropriate Service Intimacy**. However, there some of the bad smells/antipatterns mentioned above can be exclusively for microservices and, then, for this research, cannot be considered unless they are analyzed in a planning phase before migrating, with no code of the monolithic involved, and that should be focus for more research.

2.4 Phases for Migration to Microservices

In migration process, there are suggested phases there need to be taken for the transformation. In Wolfart et al. (2021), 4 phases were identified - Phase 1: Comprehension the monolithic legacy system; Phase 2: Definition of the microservice architecture; Phase 3: Execution of the transformation; Phase 4: Monitoring after the migration. In this study, based on these 4 identified phases, phases 1 and 2 are considered pre-migration because they are actions taken prior to the migration of the monolithic system, phase 3 as migration and phase 4 as post-migration. The last phase was not considered in this study.

2.5 Summary

This chapter provides the foundation for this dissertation with the current state of the art in bad smells and antipatterns for microservices. Following the dissertation, I discuss how to mitigate the antipatterns in monolithic software with strategies and refactoring approaches before migrating to the microservices approach.

3 Methodology

In this chapter, I will present the methodology. Start with Figure 3 showing the research methodology. The *First Reading* step took place at the beginning of this master thesis and then the gap was found. This was followed by the *Gap Found* step with the development of the research question below (see 3.1.1). A systematic literature mapping was conducted. The implementation and results are presented in the next chapter. Next, I started planning the survey/interviews and then used the GQM technique to elaborate the questions. In parallel, a *Rapid Review* was conducted, searching the grey literature for results that matched the SLM results. This process was important in formulating the questions for the interview and later the analysis.

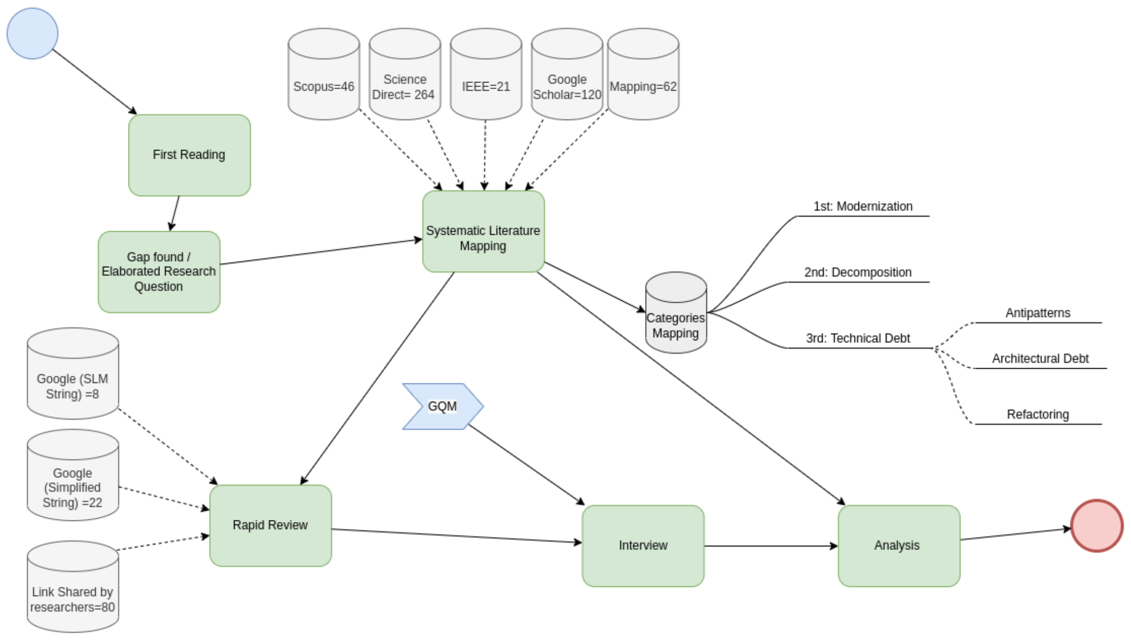


Figure 3 – Methodology

3.1 Systematic Literature Mapping

A systematic literature review is a means of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest (Kitchenham; Charters, 2007). Also accordingly with Kitchenham and Charters (2007), if, during the initial examination of a specific subject in literature, it is discovered that very little evidence is likely to exist or that the topic it is likely that a systematic mapping study may be a more appropriate exercise than a systematic review.

3.1.1 Research question

The main question in this dissertation is **What are the adopted strategies by the researchers or practitioners to refactor the architecture of a monolithic system before adopting a modernization process for Microservice?**
Rationale: Understand whether they are preparing to avoid (or mitigate) future (and possible) antipatterns after migrating. Strategies here can mean recommendations or refactoring techniques to improve the design of an existing code (Fowler, 2018), the use of design patterns (Gamma et al., 1995), design principles (Martin, 2000) or even approaches of reactive manifesto (Bonér et al., 2014). Other meanings for strategy are: criteria, processes, guidelines, tools, patterns, metrics.

Based on this question, the following subquestions emerged to guide this systematic mapping[1].

SQ1. **Are the microservice antipatterns considered in those strategies?**
Rationale: Antipatterns can be related to monolith or microservice architecture. Tools, techniques, and so on, can exist to mitigate these microservice antipatterns. It is important to understand whether microservice antipatterns can be observed in monolithic systems before migrating to microservice architecture. For example, megaservice (a microservice antipatterns) can be observed as a big module in monolithic architecture.

SQ2. **Which strategy is used to determine the extent of monolithic software degradation?**
Rationale: If the software is very degraded with, for example, outdated technology, having many defects (bugs), and design problems, what strategies in this scenario have been adopted for modernization and mitigate that this degradation be inherited in microservices.

SQ3. **Which strategy is used to identify whether upgrading to microservices is possible?**
Rationale: Knowing any strategy that considers a feasibility analysis before migrating to microservices can provide evidence about whether or not the team should migrate, as well as why.

SQ4. **What was the context (scenario) of the monolithic system?**
Rationale: Context characterized by architecture, business goals, organizational, and restrictions.

SQ5. **What were the challenges and recommendations?**
Rationale: Migration to microservices is still on the rise in the industry and is important to identify challenges and good practices to assist new practitioners who wish to migrate to microservices.

[1] The protocol for this SLM can be found in the appendix .1.

The research question and the subquestions are discussed from the following viewpoints of the PICO structure that is according to its guidelines, articulating a question in terms of its four anatomic parts—Problem/Population, Intervention, Comparison, and Outcome (PICO)—facilitates searching for a precise answer (Huang; Lin; Demner-Fushman, 2006):

- **Study Population:** practitioners/projects that refactor monolithic software before migrating to microservices, researchers/projects that study refactoring of monolithic software before migrating to microservices.
- **Study Intervention:** use of some software engineering strategy such as activities, practices, tools, standards, guidelines, patterns, metrics, criteria, evidence during the refactoring of the monolithic before migrating to microservices.
- **Study Comparisons:** -
- **Study Outcomes:** (i) what are the adopted strategies; (ii) how was the adoption of the strategy. (iii) What the problems, lessons learned, or challenges they have found during the refactoring. (iv) What are adopted criteria and evidence when selecting and adopting refactoring strategies.
- **Study design:** all empirical studies designs such as case studies, technical reports on feasibility study, controlled experiments (and quasi-experiments), experience reports, the survey with the practitioners, and action research that show adoption of refactoring of monolithic software before migrating to microservices.

3.1.2 Search process

The primary studies should be searched by following keywords: *monolith, micro-service; smell, antipattern, bad practice, pitfall, refactor, reengineer, violation, defect and degradation.* Search strings construction is based on research questions, PICO structure. They are assembled by using boolean ANDs and ORs to merge keywords. Following the search strings used in this review are listed:

((monolith) AND (smell* OR antipattern* OR badpractice* OR pitfall* OR refactor* OR reengineer* OR violation OR defect OR degradation) AND (microservice* OR micro-service* OR "micro services"))*

Based on Teixeira et al. (2019), the strategy for search engines was defined in a similar way, by analysing their ability to use logic expressions, full text recovery, automated searches of paper content and searches using specific fields (for example, title, abstract, keywords). Primary studies search should be focused on important journals, conferences proceedings, books, thesis and technical reports. The search process is manual and based on web search engines, studies references and digital libraries of relevant publishers and organizations in software engineering, such as:

- Scopus https://www.scopus.com/home.uri
- IEEEXplore (https://ieeexplore.ieee.org/Xplore/home.jsp)
- ScienceDirect (www.sciencedirect.com)
- Google Scholar (https://www.googlescholar.org/)

Specific researchers can also be contacted directly through their WebPages and emails, if it is necessary to gather further information on any relevant research and the paper (or set of papers) related to the work does not provide enough information on such issues.

According to Kitchenham and Charters (2007), it is necessary to define inclusion and exclusion criteria for the papers selection. These criteria should identify the primary studies, i.e. the ones identified during the research, which provide evidence about the research question. These criteria are presented in the following sections.

3.1.3 Inclusion/exclusion criteria

Inclusion criteria establish the inclusion reasons for each study found during the search. Studies on the following topics will be included:

1. The study outlines strategies to refactor monolithic systems before migrating or upgrading to microservice.
2. The study describes challenges or recommendations to refactor the monolithic system before migrating or upgrading to microservices.
3. The study shows degradation conditions, failures or defects before migrating or upgrading from monolithic to microservices.
4. The study presents tools or frameworks that assist or monitor defects in the design of the monolithic system.

Exclusion criteria describe the following reasons to discard studies:

1. Duplicated studies. Whenever a certain study has been published in different publications, venues or search engines (or databases)
2. The study is not a scientific research article nor a conference article.
3. Study is not written in English

3.1.4 Primary study selection process

The selection process for primary studies is performed by a joint effort of two researchers.

As papers were read, exclusion/inclusion criteria must be obey and, if any are achieved, it must be rejected but its details are kept in a reference repository system (spreadsheets) containing information to further analysis and research, when necessary.

As showed in Figure 4, 513 studies were retrieved, 62 of them from a previous study realized in this master course (Wolfart et al., 2020). After the first round of reading, 367 studies were left for the next round, which was the reading of the abstract where 87 studies were left for the final reading, where, finally, 59 studies were left.

3.1.5 Data collection

The data extraction forms must be designed to collect all the information needed to address the review questions and the study quality criteria. The following were extracted from each study:

The study's title and authors. The source (conference, journal, and so on). The year when the study was published. In case of study was published in different sources, the most relevant was used in any analysis although both dates can be recorded. Classification (related work or approach). Scope (Refactoring strategies, Modularization Techniques). The answers for research questions addressed by the publication. Summary (a summary with brief analysis, overview of its weaknesses and strengths).

The main data extraction effort was performed by two researchers.

3.1.6 Data analysis and synthesis

The primary studies were categorized since there are many techniques and methods applicable to different scenarios. After analyzing the primary studies, they were grouped in categories to facilitate the tabulation and further comparative analysis.

According to the comparisons and study analysis, the following criteria were raised:

- The strategies strengths and weaknesses;
- Limitations and trends in refactoring (remodularization) of monolithic systems before migrating to microservice architecture;
- Challenges in the area to be addressed;
- Best practices (recommendations) of refactoring (remodularization) of monolithic systems before migrating to microservice architecture.

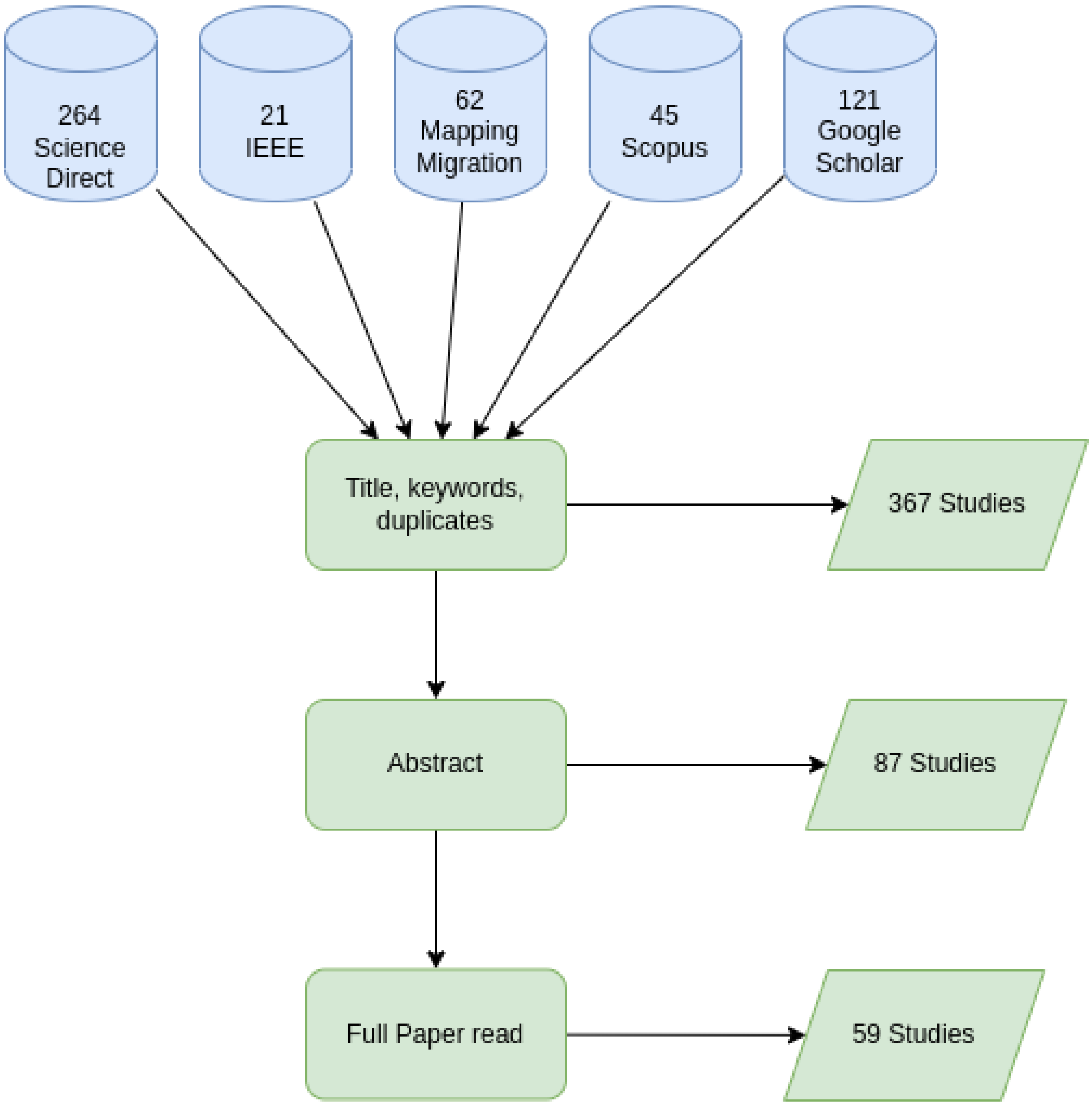


Figure 4 – Papers Selection

3.2 Rapid review

Rapid Reviews are lightweight secondary studies focused on delivering evidence to practitioners in a timely manner. RRs tend to be more connected to practice, when compared to Systematic Reviews (SRs) The kick start of a RR is a practical problem that exist in a software project. This particular problem must motivate researchers to screen the literature looking for potential answers. As a consequence, researchers must work closely to practitioners to guarantee that the RR is close tied to a practical context (B. Pinto G., 2020).

This rapid review[2] is intended to search the Grey Literature what has been published by experts in the field (software engineers/architects, developers, analysts) during the pre-migration and migration phase to understand the results of the mapping are similar or if they bring another perspective. Over recent years, Grey Literature stands out as an essential source of knowledge to be used alone or complementing research findings within the traditional literature (Kamei et al., 2021). It has increased over the last few years, mainly due to the widespread presence of GL media used by SE professionals, including various types of social media and publication channels.

The intention with this rapid-review it is to understand whether antipatterns are considered and mentioned and in what context and understand what techniques, technologies, methods and solutions are adopted to face antipatterns before migrating from monolithic systems to microservices.

I want to discover if are there experiences in Grey Literature of teams, projects or organizations that describe the process/steps taken before migrating from the monolith system to microservices. If Yes, Did they consider microservice antipatterns?, How were the steps or strategies adopted? Did they consider refactoring catalogs or bad smells in monoliths? and What are the challenges and benefits reported?. If not, What are the challenges, benefits, lessons learned reported on the strategies adopted before migrating to microservice architecture?

3.2.1 Rapid Review - Research Question

For this rapid review I develop the following research questions (Table 1) with the purpose to understand and to discover what strategies was taken on monolithic systems before migrating to microservice. This rapid review also differs from the systematic literature mapping, because here what is important is the practitioners perspective.

3.2.2 Rapid Review - Criteria

3.2.2.1 Inclusion Criteria

Contribution type of gray literature:

- Blog posts, video, white paper;
- Practice experience in migration from monolithic system to microservices;
- Tips, suggestions, steps;
- Failures in microservices that are connected with antipatterns (that can be mitigated in monolithic) identified in mapping.

[2] The protocol for this RR can be found in the appendix .2.

Table 1 – GQM for Rapid Review

Goal	Purpose	To Understand	
	Object	Strategies taken on monolithic systems before migrating to microservice systems	
	Issue	With respect to the effectiveness in addressing microservice antipatterns	
	Viewpoint	from the practitioners perspective	
RQ	Q1	What strategies was taken on monolithic systems before migrating to microservice systems from the viewpoint of the practitioners?	
	Q2	How were applied the strategies on monolithic systems before migrating to microservice systems from the viewpoint of the practitioners?	
Metrics	Q1	M1	List of Strategies
	Q2	M1	Steps
		M2	Tools
		M3	Techniques
		M4	Limitations

3.2.2.2 Exclusion Criteria

- Create microservices as a green development, which means not migrate from monolithic.
- SOA, or any other service solution;
- Only theoretical opinion, without practical experience; Point out just differences between microservices and monoliths, such as advantages and drawbacks;
- Product marketing information recommendation content only (if any articles are included that have this product marketing information, mark them as “more likelihood of bias” in the quality review.);
- No snowballing in links of articles, videos or blog posts;
- Tips, suggestions or experience in a brief or shallow manner;
- Video without transcription;
- Link unavailable / signature required;
- Copy of another white paper without credit.

3.2.2.3 Stop Criteria

- First 100(hundred) hits on Google;
- No snowballing in links of articles, videos, blog posts;

3.2.3 Rapid Review - Search Process

The same string of the SLM was used. Google was used in this case, because Google Scholar was already used in SLM. Kamei et al. (2021) says that this are the two search engines used for Grey Literature along with websites, for this reason another bases was not considered as well as others search engines. With the SLM string most of the results were similar to the mapping, but even so 8 grey articles were collected. After run the SLM string I use a more direct approach using the following string: *migration from monolithic to microservice.*

This string brought better results: 21 out of 100 analyzed links. 80 links to a spreadsheet shared by software engineering researchers who collect and fill in / update the spreadsheet regularly also served for this analysis (Figure 6). In total, 109 articles were used for inclusion and exclusion criteria. After the first diagonal reading, 33 articles met the inclusion criteria (Figure 5).

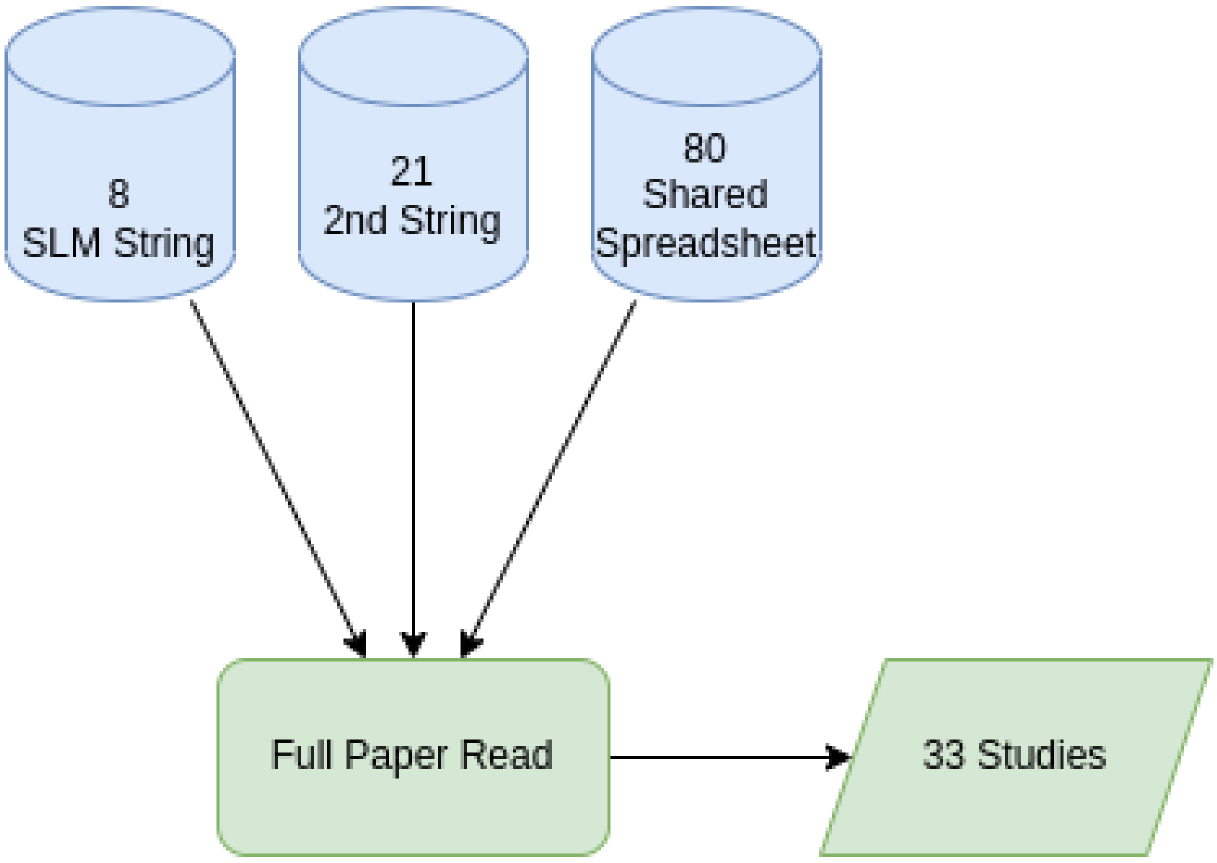
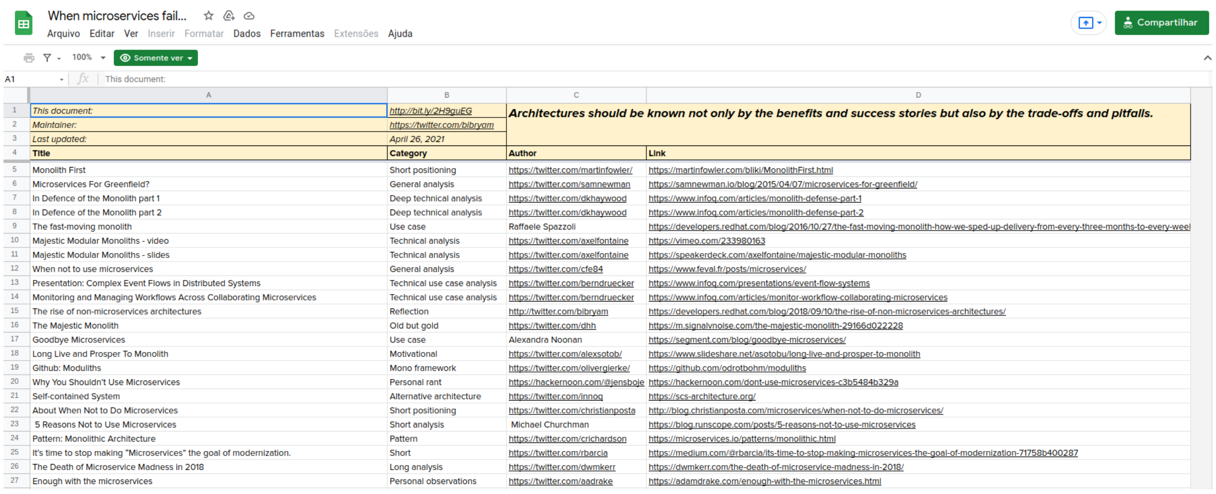


Figure 5 – Rapid Review Papers Selection

3.2.4 Rapid Review - Quality criteria

As stated in Table 4 of Garousi, Felderer and Mäntylä (2017) the responses that was a motivation to do a Grey Literature Review were *Yes* for the following questions:



	A	B	C	D
1	*This document:*	*http://bit.ly/2H9guEG*	***Architectures should be known not only by the benefits and success stories but also by the trade-offs and pitfalls.***	
2	*Maintainer:*	*https://twitter.com/bibryam*		
3	*Last updated:*	*April 26, 2021*		
4	**Title**	**Category**	**Author**	**Link**
5	Monolith First	Short positioning	https://twitter.com/martinfowler/	https://martinfowler.com/bliki/MonolithFirst.html
6	Microservices For Greenfield?	General analysis	https://twitter.com/samnewman	https://samnewman.io/blog/2015/04/07/microservices-for-greenfield/
7	In Defence of the Monolith part 1	Deep technical analysis	https://twitter.com/dkhaywood	https://www.infoq.com/articles/monolith-defense-part-1
8	In Defence of the Monolith part 2	Deep technical analysis	https://twitter.com/dkhaywood	https://www.infoq.com/articles/monolith-defense-part-2
9	The fast-moving monolith	Use case	Raffaele Spazzoli	https://developers.redhat.com/blog/2016/10/27/the-fast-moving-monolith-how-we-sped-up-delivery-from-every-three-months-to-every-weel
10	Majestic Modular Monoliths - video	Technical analysis	https://twitter.com/axelfontaine	https://vimeo.com/233980163
11	Majestic Modular Monoliths - slides	Technical analysis	https://twitter.com/axelfontaine	https://speakerdeck.com/axelfontaine/majestic-modular-monoliths
12	When not to use microservices	General analysis	https://twitter.com/cfe84	https://www.feval.fr/posts/microservices/
13	Presentation: Complex Event Flows in Distributed Systems	Technical use case analysis	https://twitter.com/berndruecker	https://www.infoq.com/presentations/event-flow-systems
14	Monitoring and Managing Workflows Across Collaborating Microservices	Technical use case analysis	https://twitter.com/berndruecker	https://www.infoq.com/articles/monitor-workflow-collaborating-microservices
15	The rise of non-microservices architectures	Reflection	http://twitter.com/bibryam	https://developers.redhat.com/blog/2018/09/10/the-rise-of-non-microservices-architectures/
16	The Majestic Monolith	Old but gold	https://twitter.com/dhh	https://m.signalvnoise.com/the-majestic-monolith-29166d022228
17	Goodbye Microservices	Use case	Alexandra Noonan	https://segment.com/blog/goodbye-microservices/
18	Long Live and Prosper To Monolith	Motivational	https://twitter.com/alexsotob/	https://www.slideshare.net/asotobu/long-live-and-prosper-to-monolith
19	Github: Moduliths	Mono framework	https://twitter.com/olivergierke/	https://github.com/odrotbohm/moduliths
20	Why You Shouldn't Use Microservices	Personal rant	https://hackernoon.com/@jensboje	https://hackernoon.com/dont-use-microservices-c3b5484b329a
21	Self-contained System	Alternative architecture	https://twitter.com/innoq	https://scs-architecture.org/
22	About When Not to Do Microservices	Short positioning	https://twitter.com/christianposta	http://blog.christianposta.com/microservices/when-not-to-do-microservices/
23	5 Reasons Not to Use Microservices	Short analysis	Michael Churchman	https://blog.runscope.com/posts/5-reasons-not-to-use-microservices
24	Pattern: Monolithic Architecture	Pattern	https://twitter.com/crichardson	https://microservices.io/patterns/monolithic.html
25	It's time to stop making "Microservices" the goal of modernization.	Short	https://twitter.com/rbarcia	https://medium.com/@rbarcia/its-time-to-stop-making-microservices-the-goal-of-modernization-71758b400287
26	The Death of Microservice Madness in 2018	Long analysis	https://twitter.com/dwmkerr	https://dwmkerr.com/the-death-of-microservice-madness-in-2018/
27	Enough with the microservices	Personal observations	https://twitter.com/aadrake	https://adamdrake.com/enough-with-the-microservices.html

Figure 6 – When Microservices Fail - Representation of the Spreadsheet (source: (Ibryam, 2021))

- Is the subject "complex" and not solvable by considering only the formal literature?
- Is there a lack of volume or quality of evidence, or a lack of consensus of outcome measurement in the formal literature?
- Is the contextual information important to the subject under study?
- Is it the goal to validate or corroborate scientific outcomes with practical experiences?
- Would a synthesis of insights and evidence from the industrial and academic community be useful to one or even both communities?

3.2.5 Rapid Review - Data Analysis and Synthesis

Following (Kamei et al., 2021) Qualitative Approach for this rapid review analysis with the steps: Familiarizing ourselves with data, Initial coding, From codes to categories and Categories refinement. After this rapid review reading and analysis, codes and categories refinement, the categories raised will be detailed in chapter 5.

3.3 Interview Based on Expert Opinion

The goal of the study is to explore the experiences and opinions of software engineering/developer who migrate from monolithic to microservices, so it can be considered primarily as being qualitative in nature. Semi-structured interviews are particularly suitable for collecting qualitative data because, they provide the opportunity for discussion or exploration of new topics that arise during data collection (Marshall; Brereton; Kitchenham, 2015).

The purpose of this semi-structured interview is to understand the state of practice at the following points: (i) Validity of the research, context of what can be done to mitigate antipatterns, and if there is something to be done, which techniques and strategies. (ii) Organizational context of the pre-migration phase, if the necessary investments are made in training, specialized staff, infrastructure, planning, etc.

3.3.1 Goal, Question and Metrics

To assist and guide this research to conduct a survey/interview with practitioners in industry, the GQM technique was used. The goal/question/metric (GQM) paradigm is intended as a mechanism for formalizing the characterization, planning, construction, analysis, learning and feedback tasks. GQM represents a systematic approach for tailoring and integrating goals to models of the software processes, products and quality perspectives of interest, based upon the specific needs of the project and the organisation (Basili; Rombach, 1988). GQM defines a certain goal, refines this goal into questions, and defines metrics that should provide the information to answer these questions. By answering the questions, the measured data defines the goals operationally, and can be analysed to identify whether or not the goals are attained (Solingen; Berghout, 1999).

3.3.1.1 Goals

The GQM model starts top-down with the definition of an explicit measurement goal. This goal is refined into several questions that break down the issue into its major components. Each question is then refined into metrics that should provide information to answer those questions (Solingen; Berghout, 1999).

The main goal of the interview is to gather initial evidence, through expert opinion, about the effectiveness of practices (actions) of strategies and measures taken in monolithic systems to address microservices antipatterns.

Accordingly to Basili and Rombach (1988) the basic template/guidelines for goal definition is to set a purpose, perspective and environment. For this research the goal was defined as:

(purpose) Understand reengineering practices in monolithic systems before migrating to microservices architecture for the purpose of characterization with respect to their effectiveness for addressing the microservices antipatterns **(perspective)** from the point of view of experts in the area of migration from monolithic system to microservices architecture **(environment)** in the context of the software engineering community.

3.3.1.2 Questions

Q1: Do you agree that the following association (Table 5) can mitigate bad smells in microservices?
This question was intended to present the mapping results to interviewees.

Q2: Point out other possible solutions that you believe could be applied to address known bad smells.
This question was optional and provided to allow the expert to suggest and evaluate other possible solutions that was not collected in the mapping.

3.3.1.3 Metrics

Once goals are refined into a list of questions, metrics should be defined that provide all the quantitative information to answer the questions in a satisfactory way. Therefore, metrics are a refinement of questions into a quantitative process and/or product measurements. After all these metrics have been measured, sufficient information should be available to answer the questions (Solingen; Berghout, 1999). In the metric definitions (Table 2) both quantitative metrics and qualitative metrics was used to take also a subject opinion of the interviewees.

Table 2 – Metrics

#	**Metric**
M1	How many Years of experience
M2	Numbers of Motivations to Migrate from monolithic to microservices
M3	What and how many were the forms of study to learn about microservices complexity
M4	Members and quantity in the migration team and their experiences
M5	How many microservices were planned in pre-migration phase
M6	What and how many were the metrics to define the size of each microservice
M7	What is the time of preparation, planning, study for migration from monolithic to microservices
M8	What are the technologies of monolithic and microservices
M9	What were the domains of the systems
M10	What antipatterns were known to respondents before to the interview
M11	Which antipatterns were considered in pre-migration to mitigate
M12	What approaches to separate the monolith or create the microservice and database

3.3.2 Motivation

The interview on expert opinion can have high external validity, good potential for theory generation, are straightforward to replicate, and are relatively cheap to organize. The semi-structured interview based on expert opinion can be important to understand what strategies are adopted to refactoring the monolithic systems before migrating to microservice architecture, identifying the main challenges and recommendations.

The main goal of this interview is to confirm or refuse the insights and findings from the systematic mapping study and rapid review, as well, as, to explore more evidence from the experts.

3.3.3 Research design

The research approach is based on two other popular techniques to extract data from people, survey (Kitchenham; Pfleeger, 2008) and expert opinion (Li; Smidts, 2003). A survey is a system for collecting valid information from or about people to describe, compare, or explain their knowledge, attitudes, and behavior. The system consists of interrelated activities starting with defining precise survey objectives, choosing respondents, preparing a reliable and valid survey instrument, testing the survey with respondents, and conducting all activities in an ethical manner. The survey researcher is responsible for implementing all survey activities and ensuring their quality (Fink, 2010). The approach for the expert opinion is composed of the following steps:

1. **Problem Statement.** The background and problem need to be clearly systematized and defined.
2. **Selection of Experts.** A number of experts need to be identified based on a set of criteria such as credibility, knowledge ability, and dependability of experts.
3. **Opinion Elicitation.** This step poses the right question and ensures conditions conductive to an elicitation process.
4. **Opinion Aggregation.** The idea is to reach an aggregated opinion or a consensus based on each individual opinion.
5. **Decision Making.** This last step makes the decision based on aggregated opinion.

3.3.3.1 Questions

The questionnaire was composed of 17 questions (Table 3) which was defined around the main research question *What and How are the adopted strategies by the researchers or practitioners to refactor the architecture of a monolithic system before adopting a modernization process for microservice?*. The questions was divided in 2 groups - the first group has open questions about microservices, experience of the experts, knowledge about antipatterns in microservices. The second group was specific of this research, to extract of the respondents if they had knowledge about antipatterns before the interview, if they have strategies to mitigate antipatterns and, if yes, what antipatterns was considered, also what strategies to decompose the monolith, and if was intended do mitigate any antipatterns. I select the insights from the systematic mapping to develop each question. The questions were developed by 2 researchers. The first interview was conducted with the participation of a third researcher who helped with criticism and suggestions for the following interviews.

3.3.3.2 Expert selection

There is not a known standard or mathematical theory involving how the selection of experts could be conducted (Li; Smidts, 2003). The authors from (Brooks et al., 1989) recommend the selection of experts from industry and academia with different backgrounds and affiliations and defined the following guidelines for the expert selection:

Table 3 – Interview Questionnaire

ID	Group	Question
1	1	What was the motivation for you to consider the migration from monolith to microservices in what was the context of the monolith?
2	1	What were the steps to study/understand/research the complexities of microservices? What training were done?
3	1	How many stakeholders participated in the pre-migration and migration process?
4	1	How many microservices were planned in the pre-migration phase?
5	1	Was any metric used to define the size of each microservice? Ex: lines of code, number of features, classes, models, etc.
6	1	What is the estimated time for each phase of the pre-migration and migration process? Planning, Study, Decomposition, Refactoring, Implementation, etc.
7	1	What roles are involved? Software engineer/architect, developer, responsible for infrastructure etc.
8	1	What is the technology of the monolith? What technologies are adopted for microservices? If it's the same, why keep it? if it changed, why did it change? If it changed how much time was considered/ spent for the learning curve of the new technology, was investment made in hiring/training?
9	1	What is the level of knowledge and experience of the team in relation to Object Orientation, design standards, microservices, distributed systems, etc.
10	1	Which projects have already participated? What were the areas/domains? How many?
11	2	Did you know about existing antipatterns in microservices before this interview? If so, which ones did you know?
12	2	In the pre-migration phase, was any action taken to address antipatterns? whether by studying what these antipatterns were in detail or any other approach? If yes, which antipattern was considered?
13	2	What were the steps to separate the monolith into microservices? Was an approach such as DDD used?
14	2	What was the approach to separating the database?
15	2	Has any refactoring technique or software been thought of to eliminate other antipatterns, other than those related to microservices, during pre-migration?
16	2	What are your considerations regarding the pre-migration and migration phase, what are the difficulties/barriers envisioned and/or lessons learned that could be useful for future migrations to microservices?
17	2	To conclude, in a possible new implementation, would you consider developing a direct software in microservices or would you prefer to develop in monolith and then migrate? What are the reasons for your choice?

- Experts should have demonstrated experience by publications, hands-on experience, and consulting or managing research in the areas regarding the issues under investigation;
- Each expert should be versatile enough to be able to address several issues and have extensive experience to consider how these issues would be used;
- Experts should represent a wide variety of experiences as obtained in universities, consulting firms or laboratories;
- Experts should represent a wide perspective of the issue as possible; and
- Experts should be willing to be elicited under the methodology to be used. In this case, some basic orientations about the survey or interview should be provided to the expert.

The primary requirement to select the researchers was the knowledge and experience in migration from monolithic to microservice. So of the above guidelines required mainly experts hands-on experience and wide perspective of the issue of this research was considered.

3.3.3.3 Interview Procedures

The interviews for this research were carried out between November and December 2021. In all, 7 interviews were carried out, the first of which was carried out with 3 respondents simultaneously. Of the total 2 interviews were conducted using google forms, where the questions were adapted in order to provide more dynamism and facilitate answers, with multiple choice for example. This option to respond via google forms was made available to those interested in participating in the survey, but who did not have time on their agendas or who did not feel comfortable talking with this researchers. Of the 10 people who participated in the interview, 1 of them stood out in terms of length of experience, having more than 20 years of experience and having a management position. The others ranged in roles, developer, software engineering, backend developer, and architect software ranging from 2 to 8 years of experience (Table 4).

Table 4 – Roles of interviewees

Interview ID	**Role**	**Years Experience**
1	Developer	+2
1	Software Engineering	+5
1	Manager	+20
2	Software Engineering	+8
3	Software Engineering	No reply
4	Software Architect	+2
5	Software Engineering	+7
6	Backend Developer	+8
7	Software Engineering	+8

3.4 Thematic Analysis

For the Analysis a thematic analysis approach was used. Thematic analysis is an approach that is often used for identifying, analyzing, and reporting patterns (themes) within data in primary qualitative research. It minimally organizes and describes the data set in rich detail and frequently interprets various aspects of the research topic. Thematic analysis can be used within different theoretical frameworks, and it can be an essentialist or realist method that reports experience, meanings, and the reality of participants (Cruzes; Dyba, 2011). The steps for this are Extract Data, Code data, Translate codes into themes, Create a model of higher-order themes and Assess the trustworthiness of the synthesis (Figure 7). Codes are descriptive labels that are applied to segments of text from each study. Coding is the process of examining and organizing the data contained in each study of the systematic review. A theme at a minimum describes and organizes possible observations or at the maximum interprets aspects of the phenomenon.
The themes that emerged will be further explored and interpreted to create a model consisting of higher-order themes and the relationships between them. The aim is to return to the original research questions and the theoretical interests that support them, and to approach them with arguments based on the themes that emerged in the exploration of the texts.

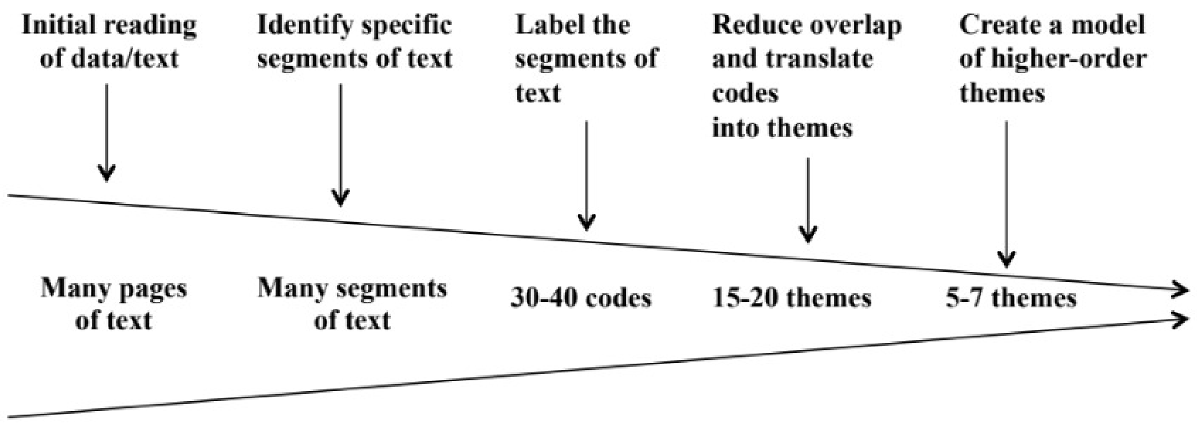

Initial reading of data/text
Identify specific segments of text
Label the segments of text
Reduce overlap and translate codes into themes
Create a model of higher-order themes
Many pages of text
Many segments of text
30-40 codes
15-20 themes
5-7 themes

Figure 7 – Thematic Synteshis process (adapted from (Cruzes; Dyba, 2011))

3.5 Summary

This chapter presented the methodology for this study, beginning with Systematic Literature Mapping, Rapid Review, and Interview. Prior to the interview, the GQM was used as a technique to support and define the questions for the interview. The results of the study may be of interest to the software engineering community as well as researchers interested in migrating from a monolithic to a microservice architecture or, in some cases, some other form of modernization.

4 Systematic Literature Mapping

Based on the research questions[1], I began to search for the strategies in the pre-migration phase. The first step was to categorize the mapping based on the keywords and summary of each paper. Based on the whole study, the following categories were found: technical debt, modernization, and decomposition. The technical debt category was divided into three subcategories: Antipatterns, Architectural Odors, and Refactoring.

The following chart pie (Figure 8) represent the categories.

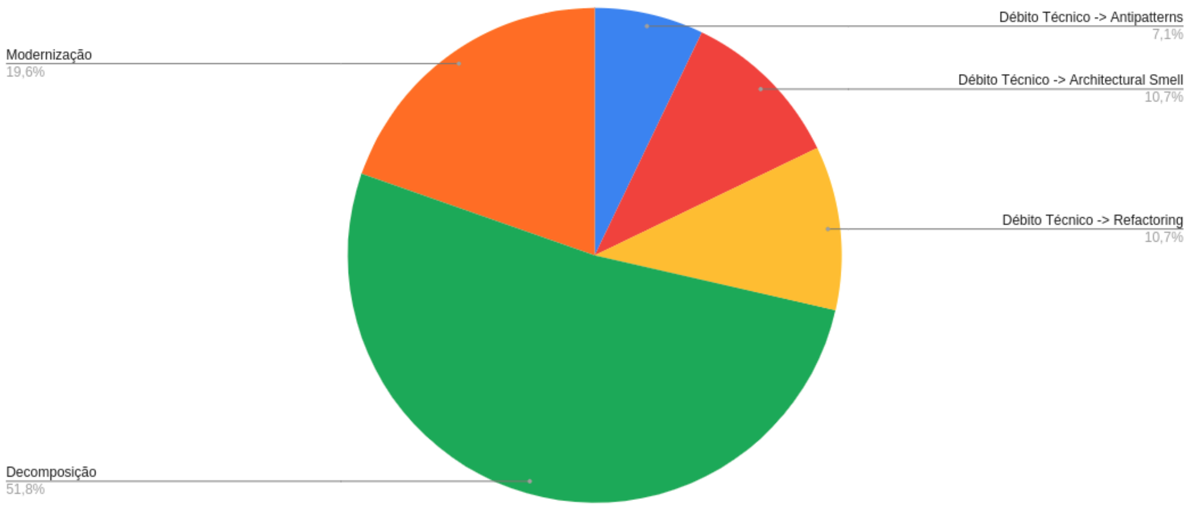


Figure 8 – Categories

4.1 Technical Debt

4.1.1 Antipatterns

The category of antipatterns identified in the SLM refers to studies that have focused on analysing problems in migrating from monolithic systems to microservices, and highlights the lack of understanding of the concepts associated with these antipatterns.

The migration to microservices is not an easy task, mainly because of its novelty and because microservice migration patterns are not clear yet (Pigazzini et al., 2020).

[1] Research questions e sub research questions: RQ: What are the adopted strategies by the researchers or practitioners to refactor the architecture of a monolithic system before adopting a modernization process for Microservice?. SRQ1: Are the microservice bad smells considered in those strategies?. SRQ2: Which strategy is used to determine the extent of monolithic software degradation?. SRQ3: What was the context (scenario) of the monolithic system?. SRQ4: What were the challenges and recommendations?. SRQ5: Which strategy is used to identify whether upgrading to microservices is possible?

Companies commonly start the migration without experience with microservices, only in few cases hiring a consultant to support them during the migration. Therefore, companies often face common problems, which are mainly due to their lack of knowledge regarding bad practices and patterns (Taibi; Lenarduzzi; Pahl, 2019). The highly dynamic nature of microservice based systems as well as the continuous integration and continuous delivery of microservices can lead to design and implementation decisions, which might be applied often and introduce poorly designed solutions, called antipatterns (Tighilt et al., 2020). Code and architectural smells, which are patterns commonly considered symptoms of bad design and indicators of situations that negatively affect software quality attributes such as understandability, testability, extensibility, reusability, and maintainability of the system under development and should be removed (Taibi; Lenarduzzi, 2018).

Not a single contribution provided answers to the question of how to refactor monolithic software before migrating to microservices.

4.1.2 Architectural Smells

As mentioned earlier, architectural defects are a typical type of architectural technical debt and can negatively impact the system-wide maintainability of a software system (Tian; Liang; Babar, 2019). In this subsection, we present this subcategory of mapping.

In Toledo, Martini and Sjøberg (2020) the authors investigate the use of shared libraries in microservices, some of the issues they found are: coupling among teams, delays on fixes due to overhead on libraries development teams, and need to maintain many versions of the libraries. Two solutions are presented: creating additional microservices or implementing the code in the microservices themselves. Afterwards Pigazzini, Fontana and Maggioni (2019) introduce an approach involves static analysis of the system architecture, architectural smell detection and topic detection, a text mining method used here to model software domains starting from code analysis. Walker, Das and Černý (2020) introduce a new approach to detect code smells in distributed applications based on microservices. The research mentions that smells are patterns of poor programming practice and deteriorate program quality and they can affect a wide range of quality attributes in a program including reusability, testability, and maintainability. The authors proposes a tool called MSANose to detect up to eleven different microservice specific code smells. As a result the authors show that it is possible to detect code smells within microservice applications using bytecode and/or source code analysis throughout the development process or even before its deployment to production.

Carrasco, Bladel and Demeyer (2018) present 9 common pitfalls (already mentioned in this research) in terms of bad smells with their potential solutions. Using these bad smells, pitfalls can be identified and corrected in the migration process and provides a foundation for actionable information on the successful refactoring of monolithic application towards microservices.

Lenarduzzi et al. (2020) monitored the technical debt of an small and medium sized enterprise while it migrated from a legacy monolithic system to an ecosystem of microservices. their goal was to analyze changes in the code technical debt before and after the migration to microservices. The TD data was obtained by analyzing the system's commits using SonarQube. For the analysis, was used SonarQube's standard quality profile and analyzed each commit two years before the migration and after the start of the migration. As a brief result considering the trend of TD analyzed by SonarQube, the overall TD of the monolithic system before the migration was lower than the overall TD right after the migration. When microservice became stable, the TD decreased significantly and started growing with a lower trend compared to the growth of the monolithic system before the migration.

Another author, de Toledo, Martini and Sjøberg (2021) aim to identify architectural technical debts, theirs costs, and their most common solutions. As a result, the authors found 16 ATD issues, negative impact and solutions to repay each debt together with related costs.

4.1.2.1 Research Question Answers

In this section, a set of responses from the RQ and SRQ found in each study is organized, separately by author.

Answers found in Pigazzini, Fontana and Maggioni (2019): **RQ1**: In addition to being time consuming, this process (migration) requires specialized personnel on software analysis with knowledge about the system to be refactored; Arcan tool in order to identify possible architectural smells before or during the migration process. **SRQ1**: Yes, bad smells are mentioned in the related works and what differs from this research is that the tool is used to detect bad smells still in the monolith. **SRQ2**: Arcan is a software analysis tool for architectural smell detection. **SRQ4**: Moreover the AS detection made him aware of a problem regarding a specific entity named Deadline: the reation of a Deadline requires the information present in Suspension and Proceedings and vice-versa, part of the problem was solved by incorporating entity Deadline with Suspension, while the Cyclic Dependency between Deadline and Suspension should be analyzed and possibly removed during the migration process in order to decouple the services. **SRQ5**: In general the migration process is not easy to carry out, since a deep knowledge of the project subjected to the migration is needed in order to have significant results. Arcan can be very useful: to retrieve knowledge about the project using the architectural smell detection and the vertical functionality view.

Answers found in Carrasco, Bladel and Demeyer (2018): **RQ1**: Look at migration smells described, could help the planning phase before migration.

Answers found in Lenarduzzi et al. (2020): **RQ1**: The TD data was obtained by analyzing the system's commits using SonarQube9 (version 7.0). For the analysis, was used SonarQube's standard quality profile and analyzed each commit two years before the migration and after the start of the migration. **SRQ2**: In this work, as required by the case company, was adopted SonarQube, SonarQube is also open source, while the other well known competitors have a commercial license. SonarQube calculates several metrics such as number of lines of code and code complexity, and verifies the code's compliance against a specific set of "coding rules". If the analyzed source code violates a coding rule or if a metric is outside a predefined threshold (also called "quality gate"), SonarQube generates an "issue". Issues are problems that generate TD and therefore should be solved. **SRQ4**: The company is migrating the bookkeeping document management system for Italian tax accountants. The company needs to frequently update the system, as the annual tax rules usually change every year. The Italian government normally updates the bookkeeping process between December and January, which involves not only changing the tax rate but also modifying the process of storing the invoices. However, tax declarations can be made starting in March/April of each year. Therefore, in the best case, the company has two to four months to adapt their software to the new rules in order to enable tax accountants to work with the updated regulations from March/April.

4.1.3 Refactoring

One of the approaches for migrate to microservices is through refactoring, to pay the technical debt. The authors Shimoda and Sunada (2018) mention that is difficult to apply this architecture to a new business that is not well understood and to cut out an appropriate service. For this reason, so called "Monolithic first", in which a monolithic system is first constructed and the microservices are gradually made, has attracted attention. The author proposes method for clarifying the desirable order for converting the function constituting the monolithic system into the microservices and confirm the feasibility by applying in a fictitious e-commerce.

Brogi et al. (2019) review the white and grey literature on the topic, in order to identify the most recognised architectural smells for microservices and to discuss the architectural refactorings allowing to resolve them.

Furda et al. (2018) show how to identify challenges in legacy code and explain refactoring and architectural pattern based migration techniques relevant to microservice architectures. The authors explain how multitenancy enables microservices to be utilised by different organisations with distinctive requirements, why statefulness affects both availability and reliability of a microservice system and why data consistency challenges are encountered when migrating legacy code that operates on a centralised data repository to microservices operating on decentralised data repositories.

4.1.3.1 Research Question Answers

Answers found in Linthicum (2016): **RQ1**: Pattern one decides quickly how the application is to be broken into components that will be run inside of containers in a distributed environment. This means breaking the application down to its functional primitives, and building it back up as component pieces to minimize the amount of code that needs to be changed.

Answers found in Shimoda and Sunada (2018): **RQ1**: evaluate the database tables and classify them into business subsystems. Next, the business subsystem accesses the source code and database model to mechanically identify candidates for microservices.

4.2 Modernization

In this section you will find studies in which modernization or evolution was categorized as modernization. In Santos and Silva (2020), Bogner et al. (2019), Zirkelbach, Krause and Hasselbring (2019a), Balalaie, Heydarnoori and Jamshidi (2016), Levcovitz, Terra and Valente (2016), Knoche and Hasselbring (2019) the main motivation for seeking migration is the evolution/modernization of the software and techniques are presented. Other studies Zirkelbach, Krause and Hasselbring (2019b) it is analyzed how the sustainable evolution of architecture was guaranteed after the migration while in Mahanta and Chouta (2020), Balalaie, Heydarnoori and Jamshidi (2015), Escobar et al. (2016) modernization is driven by modularization techniques and is used based on the microservice architecture. In Kaplunovich (2019) DevOps practices are adopted and how this knowledge helped in the migration process.

4.2.0.1 Research Question Answers

Answers found in Santos and Silva (2020): **RQ1**: Step 1 Collect information from the monolith system. Most approaches rely on the source code, and usually use static and dynamic code analysis. Other approaches do not use the source code, and rely on interface information, or models of the system. Step 2 Grouping entities of the system into candidate microservices. One common approach is to define similarity measures between domain entities and use a clustering algorithm, which returns clusters of entities. Step 3 A visualization of the decomposition as a graph, where some proposals support a modeling tool where the architect can interact with the graph in order to modify the decomposition.

Answers found in Bogner et al. (2019): **RQ1**: To provide sufficient confidence that such a system can be sustainably evolved, software professionals apply a set of numerous activities that is refer to as evolvability assurance. Activities are usually either of an analytical nature to identify issues or of a constructive nature to remediate issues [4]. This includes for example techniques like code review, refactoring, standardization, guidelines, conscious technical debt management, and the usage of tools (e.g. for static code analysis), metrics, or design patterns. For larger systems, these activities often form a communicated assurance process and are an important part of the development workflow. **SRQ5**: There are no useful tools to split up a monolith. It's always a very difficult manual activity. You can use something like Domain-Driven Design, but that's just a methodology which doesn't give you a concrete solution; "In a monolith with 100.000 FindBugs warnings, you are completely demotivated to even fix a single one of those. In a Microservice with 100 warnings, you just get to work and remove them."

Answers found in Zirkelbach, Krause and Hasselbring (2019b): **RQ1**: The modularization planning phase was started with a requirement analysis for the modernized software system and identified technical and development process related impediments in the project. The focus was to provide a collaborative development process, which encourages developers to participate in the research project; The modularization approach started by dividing the old monolith into separated frontend and backend projects. **SRQ1**: ExplorViz Legacy was also covered by this survey and categorized by the "Single DevOps toolchain" pitfall. This pitfall concerns the usage of a single toolchain for all microservices. Fortunately, this pitfall was addressed since their observation during their survey by employing independent toolchains by means of pipelines within the continuous integration system for the backend and frontend microservices. **SRQ2**: Students of the university know and use supporting software for code quality, e.g., static analysis tools such as Checkstyle [12] or PMD [13]. However, they did not define a common code style supported by these tools in ExplorViz Legacy. **SRQ4**: Initially the Java based Google Web Toolkit (GWT) was used [11], which seemed to be a good fit in 2012, since Java is the most used language in the lectures. GWT provides different wrappers for Hypertext Markup Language (HTML) and compiles a set of Java classes to JavaScript (JS) to enable the execution of applications in web browsers. Employing GWT in the project resulted in a monolithic application (hereinafter referred to as ExplorViz Legacy), which introduced certain problems over the course of time; Every change affects the whole project due to its single code base. New developed features were hard wired into the software system. Thus, a feature could not be maintained, extended, or replaced by another component with reasonable effort. This situation was a leading motivation to look for an up to date framework replacement. With the intention to take advantage of this situation and modularize the software system.

Answers found in Balalaie, Heydarnoori and Jamshidi (2016): **RQ1**: Migrating the system towards the target architecture was not a one step procedure and it was done incrementally without affecting the end users. The migration steps was treated as architectural changes (adding or removing components) that consists of two states: (i) before the migration, and (ii) after the migration. **SRQ4**: Backtory is written in Java using the Spring framework. The underlying RDBMS is an Oracle 11g. Maven is used for fetching dependencies and building the project. All of the services were in a Git repository, and the modules feature of Maven was used to build different services. The deployment of services to development machines was done using the Maven's Jetty plugin. However, the deployment to the production machine was a manual task. The architecture of Backtory before the migration to microservices is illustrated in Figure 2(a). In this Figure, solid arrows and dashed arrows respectively illustrate the direction of service calls and library dependencies. Figure 2(a) also demonstrates that Backtory consisted of five major components. **SRQ5**: Deployment in the development environment is difficult; Service contracts are critical; Distributed system development needs skilled developers; Creating service development templates is important; Microservices is not a silver bullet.

Answers found in Levcovitz, Terra and Valente (2016): **SR4**: applied the proposed technique on a large system from a Brazilian bank. The system handles transactions performed by clients on multiple banking channels (Internet Banking, Call Center, ATMs, POS, etc.). It has 750 KLOC in the C language and runs on Linux multicore servers. The system relies on a DBMS with 198 tables that performs, on average, 2 million transactions a day.

Answers found in Balalaie, Heydarnoori and Jamshidi (2015): **SRQ4**: SSaaS is written in Java using the Spring framework. The underlying RDBMS is an Oracle 11g. Maven is used for fetching dependencies and building the project. All of the services were in a Git repository, and the modules feature of Maven was used to build different services. At the time of writing this paper, there were no test cases for this project. The deployment of services in development machines was done using the Maven's Jetty plugin. However, the deployment to the production machine was a manual task that had many disadvantages.

4.3 Decomposition

I considered as decomposition any research that did not mentioned any other approach other than a generic migration from monolithic software to microservices.

The researches Kazanavičius and Mažeika (2019), Sayara, Towhid and Hossain (2017), Mazlami, Cito and Leitner (2017), Kecskemeti, Marosi and Kertesz (2016), Selmadji et al. (2018), Jin et al. (2018), Carvalho et al. (2019), Christoforou, Odysseos and Andreou (2019), Kuryazov, Jabborov and Khujamuratov (2020), Francesco, Lago and Malavolta (2018), Silva, Carneiro and Monteiro (2019), Amiri (2018), Ren et al. (2018), Abdellatif et al. (2021), Taibi, Lenarduzzi and Pahl (2017), Kazanaviius and Mazeika (2020), Laigner et al. (2019) show techniques, challenges, automation for migration / decomposition of monolithic to microservices; In Janes and Russo (2019) a tool to measure software degradation during migration to microservices is presented. In Cojocaru, Uta and Oprescu

(2019), Fan and Ma (2017), Taibi et al. (2019), Taibi and Systä (2019), Abdullah, Iqbal and Erradi (2019), Henry and Ridene (2020), Desai, Bandyopadhyay and Tamilselvam (2021) a framework is developed to decompose monolithic for microservices.

4.3.0.1 Research Question Answers

Answers found in Kazanavičius and Mažeika (2019): **RQ1**: Monolith should be modernizing to microservices when: Monolith became too big and complex to maintain or extend; Modularity and decentralization are an important aspect; Preference for long term benefits in comparison to those in the short term. **SRQ2**: The following type of legacy applications are not recommended to refactor: Very old applications that are built using very old languages and databases; Applications what has a poor design; A application that are tightly coupled to the database. **SRQ3**: To choose migration methods and technics which best suits for an organization is very a hard task. The first question which always should be answered: refactor or rebuild? In most cases there is not so many resources and time to completely rebuild the solution.

Answers found in Fan and Ma (2017): **RQ1**: Use of DDD to modularize the monolithic before migration. **SRQ5**: Complex environment settings: The configuration is not as simple as in a Monolithic architecture system, and many automation tools must be carefully set up to achieve the desired results. This includes monitoring tools, server environment, automatic deployment tools, and automatic integration. Using more resources: Microservices use multiple tools to achieve architectural flexibility, such as Service Discovery and API Gateway. This consumes more resources and increases the complexity of the system.

Answers found in Jin et al. (2018): **RQ1**: First leverage execution traces collected at runtime to guide microservice extraction. Execution traces are able to expose actual software behavior accurately. A lot of tools can help obtain execution traces. Here Kieker was used to monitor four software projects, in which 204 unique execution traces were extracted. This study shows that an execution trace inherently represent a business function. Another contribution was an execution oriented clustering method towards grouping similar functionalities as a service. FoME (Functionality oriented Microservice Extraction) was used to refer to the proposed method. Concretely, execution traces was clustered to recommend microservice skeletons, and then apply two strategies, namely, “Move class” and “Pull up class”, to deal with crossovers overlapping between services.

Answers found in Taibi et al. (2019): **RQ1**: In order to finalize the decision on whether or not to migrate to Microservices, teams should first analyze their existing monolithic system. The system should be analyzed by considering the metrics reported in Table 16. **SRQ2**: Metrics identified and analysed before migration. **SRQ5**: The vast majority of the interviewees migrated to Microservices in order to improve maintainability.

Answers found in Kuryazov, Jabborov and Khujamuratov (2020): **RQ1**: This step helps to identify how high is the tight coupling among system parts. For instance, the analysis steps identifies if a legacy system already has parts that are not tightly coupled (i.e., loosely coupled) with the rest of the system. A system can be predicted (whether its modules are tightly or loosely coupled) by measuring several metrics [8], e.g., Cohesion and Coupling. This phase is needed for estimating how much effort is required for migration. The result of this phase is analyzed a software system including their analysis reports.

Answers found in Taibi and Systä (2019): **RQ1**: Analysis of the System Structure. All processes start by analyzing dependencies mainly with the support of tools (Structure101, SchemaSpy 2 , or others).

Answers found in Francesco, Lago and Malavolta (2018): **RQ1**: Analyse Source code, Test suits, architectural docs, textual docs; understanding legacy system, find dependencis legacy, analysis legacy, document legacy. **SRQ5**: Almost all participants (15/18) have acknowledged the long time to release new features as the main challenge faced in the pre-existing system. From an architectural perspective, participants find challenging the high degree of coupling among modules (13/18) in the pre-existing system.

Answers found in Silva, Carneiro and Monteiro (2019): **RQ1**: Domain-Driven Design (DDD) concepts to translate functionalities into domain and subdomain and thereby support the migration; related to the restructuring of the legacy system to a modularized version. **SRQ1**: Using DDD and bounded contexts lowers the chances of two microservices needing to share a model and corresponding data space, risking a tight coupling. Avoiding data sharing facilitates treating each microservice as an independent deployment unit. Independent deployment increases speed while still maintaining security within the overall system. DDD and bounded contexts seems to make a good process for designing components. **SRQ2**: Manual identification of candidate features and their respective relationships, by navigating among the directories and files and identifying the purposes of each class. **SRQ4**: EPromo system was selected as the subject of the Pilot study. It comprises a typical example of a corporate/business coupon web system implemented in the PHP programming language for the management of outreach campaigns. The web server is Nginx and its features include: creation of personalized offers and issuance of tickets made by the customer. All functionalities are implemented in a large artifact, connected to a single relational database (MySQL).

Answers found in Ren et al. (2018): **RQ1**: Firstly, static and dynamic characteristics of application was obtained through static analysis of the source code and tracing data during application's execution.

Table 5 – SLM Analysis

Response Collected	**Antipattern to Mitigate**
Use of sonarQube or Arcan Tool to detect smell;	
Evaluate the database tables and classify them into business subsystems in the pre-migrations phase; Grouping entities of the system into candidate microservices;	Shared persistence; Inappropriate service intimacy;
Apply a set of numerous activities that is refer to as evolvability assurance;	Microservice greedy; Megaservice;
Use of DDD to modularize and to translate functionalities into domain and subdomain before migration; Strategy to identify how high is the tight-coupling;	Wrong cuts; Megaservice; Shared Libraries; Cyclic Dependence

4.4 Summary

In this chapter, the systematic literature mapping was discussed with the categories of mapping, a brief overview of each research, and the answers to the research questions found. Not all research provided the expected answers. But as the Table 5 shows, it was possible to extract some answers from a brief analysis of the SLM. In the data analysis 7 chapter, a full analysis of the study is shown.

5 Rapid Review

Similar to SLM, not all studies in this rapid review provided the expected answers. Although some strategies were found, the authors did not show exactly how they were applied.

5.1 General Approaches

5.1.1 Log Aggregation

Part of the articles found mention some techniques, such as Newman and Reisz (2020) mentions that it is necessary to implement some form of log aggregation or backlog to process log files locally, and that these files can be automatically aggregated to a central location where they can be queried and processed. In this case of log aggregation, it is reasonable to assume that a variety of tools [1] are available that can solve this problem. Log aggregation is a strategy to find problems more easily and have real-time monitoring, but it is not really related to antipatterns. According to Sigma (2021), the task in creating a backlog is to identify candidate microservices. The next steps are to select a refactoring strategy, design the microservice and make changes according to CI/CD and testing procedures, set up CI/CD, and implement the microservice.

5.1.2 Data First

To Hubers (2021) start with Data is the first step before migrate to microservices. Relational data needs to be designed conceptually from the start, before you even consider the placement and design of functions to carry out business processes. Roos (2020) Explain about organization matter for cultural shift "Stop feeding the monster, leave monolithic as it is start developing in microservice".

5.1.3 Coupling and Cohesiveness

Richardson (2021) mention a review of the so called AS-IS code. The first step is to split the code and turn delivery management into a separate, loosely coupled module within the monolith. To Lea (2016), when designing applications, avoid using library code that increases coupling between applications, so that developers building clients are free to make different technology choices if that becomes advantageous. To Deshpande and Singh (2020) a good approach is to stop adding things to the monolithic app. Fix what is

[1] Backlog tools: (Humio, ELK stack)

broken and accept only small changes. Also identify components that are more loosely coupled than others and use them as a starting point. A key step in identifying the core components of any monolithic application is analyzing the codebase. It also helps you understand the cohesiveness between various modules and prepare a list of microservices that you want to build. To Gupta (2021) one of the most important and challenging steps is to select the business functionality to be decomposed. One can start with either selecting parts of the code that are frequently modified or functionality that is needed to be scaled on demand. One may end up with a list of such functionalities. The next step would be to select the one amongst the list which is fairly loosely coupled.

5.2 Strangler Pattern

Strangler Pattern (Fowler, 2004) it is a way to handle the release of refactored code in a large web application. Strangler Pattern is a popular design pattern to incrementally transform a monolithic application into microservices by replacing a particular functionality with a new service. Once the new functionality is ready, the old component is “strangled Transform, Co-Exist and Eliminate. The author Sitnikova (2021) applied Strangler pattern, identify all local components and determine which of them are best for migrating. This involves checking for duplicated data, checking all data formats; Identifying Component Dependencies. This is done using a static analysis of the source code to search for calls between various libraries and data types.

To Goel (202) there are tree good strategies, using the Strangler Pattern, using Domain-Driven Design "Requires an understanding of the domain for which the application will be written. The necessary domain knowledge to create the application resides with the people who understand it". And the general migration approach that has three steps: Stop adding functionality to the monolithic application; Split the frontend from the backend; Decompose and decouple the monolith into a series of microservices.

To Kornilov (2020) use strangler pattern with the anti-corruption layer pattern. The main advantage of the strangler pattern is that it offers an incremental migration process without breaking the whole application functionality. After applying the pattern the extracted microservice and the monolith are part of the bigger application and communicate with each other. Most likely, they use different domain models that are converted during the communication process. The process involves some glue code, which resides in the monolith, in the microservice, or on both sides. This code is called the anti-corruption layer. In this case, anti-corruption means that the design solutions of the extracted microservice and the monolith don’t affect each other. In other words, the anti-corruption layer pattern is applicable to each extracted microservice and makes its design independent of the monolith design.

To Richardson (2016) the best way to migrate it is gradually build a new application consisting of microservices, and run it in conjunction with your monolithic application. Over time, the amount of functionality implemented by the monolithic application shrinks until either it disappears entirely or it becomes just another microservice.

5.3 Modularity

Domain-driven design (DDD) has some ways to find service boundaries. When working with organizations that are looking at microservice migration, is common to start with a DDD modeling exercise on the existing monolithic application architecture (Newman; Guimarães, 2021). To Lavann, EdPrice and neilpeterson (2021) applying DDD has a few steps: Applied retroactively to an existing application, as a way to begin decomposing the application:

- Start with a ubiquitous language, a common vocabulary that is shared between all stakeholders.
- Identify the relevant modules in the monolithic application and apply the common vocabulary to them.
- Define the domain models of the monolithic application.
- The domain model is an abstract model of the business domain.
- Define bounded contexts for the models. A bounded context is the boundary within a domain where a particular domain model applies.
- Apply explicit boundaries with clearly defined models and responsibilities.;

Accordingly to Assouline and Grazi (2017) domain-driven design advocates modeling based on the practical use cases of the actual business. In its simplest form, DDD consists of decomposing a business domain into smaller functional chunks, possibly at either the business function or business process level, so that the complexity of both a business and problem domain can be better apprehended and resolved through technology. For Samokhin (2018), first concentrate on user stories forming cohesive business-capability. Then, within each one, focus on objects that possess some identity and behavior. That's how you'd get the higher-level view of your business-processes expressed with DDD aggregates (which are often sagas). Put the rest of the logic in value-objects. Thus you can end up with almost no service classes. The author Janssen (2021) explain how to migrate in a few steps. Each service has to have its own database to keep it independent of all other services. Step 1: Identify independent modules and split your business code; Step 2: Remove queries and associations across module boundaries; Step 3: Each module becomes a service and final step: Refactor your modules into independent services.

To Haywood and Betts (2017) modules it's about ensuring that the code is understandable, encapsulating functionality and constraining how different parts of the system interact. If any object can interact with any other object, then it's just too difficult for a developer to fully anticipate all side-effects when code is changed. Instead the better way it is to break the app into modules small enough that a developer can understand each module's scope and can reason about its function and responsibility. To Mak (2017) Modularity can also be achieved by other means. The key is that it can effectively draw and enforce boundaries during development. However, this can also be achieved by creating a well-structured monolith. Creating good modules requires the same design rigor as creating good microservices. A module should model (part of) a single bounded context of the domain. Module boundaries in a modular application are easier to change. Refactoring across modules is typically supported by the type-system and the compiler. Redrawing microservice boundaries involves a lot of inter-personal communication to not let things blow up at run-time.

5.4 Summary

In this chapter some findings of the gray literature was showed, as it happened in the SLM it was difficult to find clear evidence about the research question, some strategies that are taken before the migration process was found, but they are not directly linked with the antipatterns in microservices, during the thematic analysis this subject will be discuss again with the conclusions.

6 Interview

Interviews were recorded (audio and video) and converted to text. If doubts about an answer arose during the analysis, it was possible to return to the recording to clarify them. The shortest interview lasted 28 minutes and the longest 42 minutes. The transcribed interviews can be found in the appendix .3.

6.1 First Group of Questions

The first group of questions had the intention to learn about the interviewees about this general knowledge about microservices, distributed systems, about the migration process, planning, members teams, members knowledge etc.

6.1.1 Question 1

In the first question (Table 6), the interviewee #1 said that they had a system with more than thirty years of development and outdated technology, and needed modern technology that could be easily adapted to the current market. #2 also wanted a modern technology and mentioned the difficult with scalability and maintenance of the monolithic system, the high number of bugs, the high cohesion and coupling, and the high number of functionalities with many modules with dependencies on each other. #4 Wanted scalability and flexibility to change the system as needed, including changing technology. Old system with high complexity and no money to invest in staff. Develop monolithic first to migrate incrementally. #5 Difficulty to maintain code, upgrade technology which was Java and Hibernate, "this was never considered because they were afraid to break the code". It was also mentioned that maintenance was difficult because of the size of the system and the hours of configuring the system in the customer's environment because they had to have their own infrastructure, their own server, etc. So it was very expensive and took a long time just to develop a new functionality. #6 The motivation for the migration was essentially decoupling. The old system was in Delphi with services that did not work properly and were hard to maintain, plus it was getting harder to find professionals with knowledge of Delphi. #7 Mentioned that they had a monolithic with client/server architecture, front end and back end, they just wanted a system that was more scalable. They do not really migrate, they just use monolithic systems as knowledge repositories. They assure that they are building microservices from scratch.

Table 6 – Responses of 1st question - Motivation to migrate

ID	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Wanted Modern Technology	✓	✓				✓	✓
Scalability		✓		✓			
Flexibility				✓			
Maintainability		✓			✓		
Decoupling						✓	

Table 7 – Responses of 2nd question

ID	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Book	✓	✓	✓	✓			✓
Lectures		✓	✓				✓
Case Studies		✓	✓				✓
Blog Posts			✓	✓			
Consultants					✓		
Previous knowledge				✓		✓	

6.1.2 Question 2

From the 2nd question (Table 7) #1 buys a book and each member of the team reads it and each uses his own form of study. #2 and #7 also read a book, lectures and case studies from big companies, #7 starts studying with three architects. #3 mentions that he read everything he found and stresses the importance of reading the pros and cons, especially the cons of microservices. #4 studied how to migrate for four months. #5 studied what technologies are used in the market today, hired consultants to help with development and understand the complexity of microservices. #6 Had no training, used initial time to understand monolithic systems, how to decouple them, started conversations with customers to identify business rules.

6.1.3 Question 3

To the 3rd question (Table 8) they all give simple answers, but #7 mentions that they form so-called squads with three to five members each, and #5 mentions that today they have more than forty people.

6.1.4 Question 4

For the fourth question (Table 9), respondents #4, #5, and #6 could not measure the number of microservices because it varied during the pre-migration process. #1, #2, and #3 start with 5. #1 has 13 today and #2 has 10. #7 plan 25 microservices from the beginning.

Table 8 – Responses of 3rd question

ID	How many stakeholders participated in the pre-migration and migration process?
1	1 to 5
2	5 to10
3	More than 50
4	1 to 5
5	1 to 5
6	1 to 5
7	1 to 5

Table 9 – Responses of 4th question

ID	How many microservices were planned in the pre-migration phase?
1	Start with 5, today 13
2	5 to10
3	5
4	Did not planned a number of MS
5	Did not planned a number of MS
6	Did not planned a number of MS
7	25 to 30

6.1.5 Question 5

#1 Had no metric, cited DDD as a form of defining microservices based on domain, but size was not considered. #2 used business domains such as people, courses, shopping carts, products, etc. #4 and #5 had no metrics, while #6 mentioned common sense to evaluate whether each microservice is needed. #7 Had no metrics, just the motivation of the business, context boundaries. "It was easy to understand communication between objects than to look for other ways to have that metric, like lines of code or functionality. Sometimes a lot more code was required to communicate between two microservices because they were separate, when they were kept together the communication was not required.

6.1.6 Question 6

#1 Says the process started by one year and six months and was done in parallel, planning, study and implementation. #2 Built the system from scratch, with a year of development to have the first stable version of the main services. #4 Said that everything was happening at the same time, but they had orchestrated everything very well. #5 Said that the time to deliver the functionality was well planned. #6 Four months of planning and six months of study, and they are still in the development phase. #7 Six months pre-migration phase and 2 years migration.

Table 10 – Responses of 7th question

ID	Members who participated in the migration to microservice
1	2 Senior Developers, 1 Senior Engineering, 1 Manager, 1 Junior Developer
2	1 Engineering 1 Architect 2 Developers (Back/Front-end) 1 Dev/Ops
3	No reply
4	1 Front-end Developer 1 Dev/Ops 1 Back-end Developer
5	5 Developers/Architects 1 UX/UI Designer
6	No reply
7	1 Product Owner 1 Architect 1 Test Engineer 5 Developers

6.1.7 Question 7

For the seventh question (Table 10), #1 had two senior developers and one senior engineer, one manager and one junior developer. #2 One engineer/architect, two developers (back-end and front-end), one focused on infrastructure "Was a very small team, but the idea with growth was to use a metric to control a number of microservices per team."#4 Started with one front-end, one infrastructure and one back-end. #5 Had 5 developers/architects and one person responsible for UX. #7 Had one PO, one architect, one test and five developers. They have this pattern for each team.

6.1.8 Question 8

In the eighth question (Table 11), #1 says that the technology of monolithic was Delphi and they chose GoLang for microservice, "this choice was made due to practicality, easy learning curve, easy scalability and a set of tools and frameworks compatible with cloud infrastructure."#2 Monolithic was PHP and MySQL. With microservice, it was PHP, MySQL, MongoDB, Redis, Docker, OAuth2, Kubernetes. "Using the right tool for the proposed problem, in terms of programming language, was maintained because the team was very small. The databases were used for different purposes to better solve each area."#4 Python is used in the microservice, React in the frontend, Terraform in the infrastructure, Postgresql in the database along with Kubernetes and AWS. #5 Monolithic

Table 11 – Responses of 8th question

ID	Monolithic Technology	Microservice Technology
1	Delphi	GoLang
2	PHP, Mysql	PHP, MySql, MongoDB, Redis, Docker, OAuth2, Kubernetes
3	No reply	No reply
4	No reply	Python, React, TerraForm, PostgreSql, Kubernetes, AWS
5	Java 4, Hibernate	Java8, Kotlin, PostgreSql, React, React Native, Sql Lite
6	Delphi, Firebird	GoLang, VueJS, PostgreSql
7	Java, Jboss, PostgreSql, Oracle, Sql Server Native Android app	Skala, Kotlin, Node, MySql, Postgres, Cassandra Neo4J, S2, Elasticsearch

was Java 4 with Hibernate, in front-end javascript, html css and jQuery. Microservice continued with java 8 and kotlin and postgresql. Reactjs in the front-end along with mobile react native and sql lite. "The people who were already on the team helped the new guys, there was no training, they just started coding."#6 The technology in Monolith was Delphi with Firebird, for microservice C#, VueJS, GoLang and the database Postgresql. #7 Monolith was a Java application running on a Jboss client server along with Postgresql, Oracle and Sql server, the front end was a native Android app. Microservices started from the idea that there would not be just one language. Today, besides Java, there are also Skala and Kotlin, services also with Node front with Javascript / Mysql, postgresql, cassandra, neo4j, s3, elasticsearch and REST JSON for communication.

6.1.9 Question 9

In the ninth question #1 team with focus and experience in design patterns and microservices. #2 "Lots of knowledge in object-oriented programming, lots of knowledge in design patterns, intermediate knowledge in microservices and distributed systems."#3 Expert in object-oriented programming and design patterns, intermediate knowledge related to microservices, and much knowledge related to distributed systems."#4 Basic knowledge from studies. #5 "Initially it was higher seniority members, then new lower seniority members. Knowledge of microservices and distributed systems was low at first, they learned over time."#6 "The leader had a greater knowledge of design patterns and distributed systems. The rest of the team had basic to intermediate knowledge. The team started to learn about clean architecture in depth during the project."

6.1.10 Question 10

About domain systems of the interviewees (Table 12) it varied from finances, e-commerce's, trade merchandising (industries to agencies) to biggest Brazilian ERP.

Table 12 – Responses of 10th question

ID	Domain of the system
1	commerce, sales, client management
2	E-commerce and Online Teaching Trade Merchandising (industries and agencies)
3	ERP
4	commerce, sales, client management
5	commerce, sales, client management
6	Finances, Bank
7	commerce, sales, client management

Table 13 – Responses of 11th question

ID	1	2	3	4	5	6	7
Megaservice	✓	✓	✓		✓	✓	✓
Wrong Cuts	✓	✓	✓				
Cyclic Dependency		✓	✓				
Shared Persistency		✓	✓	✓			
Inappropriate Service Intimacy			✓				✓
Shared Libraries		✓	✓				
Microservice Greedy		✓	✓				✓
Legacy organization			✓				

6.2 Second Group of Questions

The second group of questions was about finding out whether the respondents already knew about microservices antipatterns, lessons learned, and other things.

6.2.1 Question 11

For the eleventh question the table was refined (Table 13) after the data analysis, although some respondents did not confirm that they knew about antipatterns in this question, they mentioned them later in the interview. #1 Already knew megaservice and wrong cuts, other antipatterns they did not know. #2 "I already knew megaservice, wrong cuts, cyclic dependency, shared persistence, shared libraries and microservice greedy".#3 "Yes i knew, megaservice, wrong cuts, cyclic dependency, shared persistence, Inappropriate Service Intimacy, shared libraries, microservice greedy e Legacy organization". #4 and #6 did not know academically, only from concept or with another names from books. #5 Not knew exactly antipatterns, but the problems reported were familiar. Megaservice was a concern, not building a very large microservice, had microservices split wrongly (wrong cuts), caused a big impact on the system. There was also a call cycle between microservices. Shared libraries were used, with a plan to avoid rework. #7 "Yes, megaservice, inappropriate service intimacy, microservice greedy".

Table 14 – Responses of 12th question

ID	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Megaservice	✓	✓	✓		✓		✓
Wrong Cuts	✓	✓	✓				
Cyclic Dependency			✓				
Shared Persistence		✓	✓	✓	✓		
Inappropriate Service Intimacy			✓				
Shared Libraries		✓	✓				
Microservice Greedy		✓	✓				
Legacy Organization			✓				

6.2.2 Question 12

For the twelfth question (Table 14) #1 Mention that in the planning and study phase, attempts were made to avoid two antipatterns megaservice and wrong cuts, no other antipatterns were mentioned. #2 Considered megaservice, wrong cuts, shared persistence, shared libraries and microservice greedy. "Services were divided into business domains (DDD - Bounded Context). In case of non-action: cyclic dependency, lack of knowledge in asynchronous communication (queues) and project development time."#3 "Yes, megaservice, wrong cuts, cyclic dependency, shared persistence, inappropriate service intimacy, shared libraries, microservice greed and legacy organization. #4 "Today we have already figured out how to avoid antipatterns, e.g., the antipattern of shared persistence. The plan is to have one messenger, meaning no microservice communicates with another, so if one fails, there is no problem, only the connection between the messenger and the microservice is changed."#5 The plan was to avoid a megaservice based solely on not creating too large a service that would be difficult to maintain, and it was assumed that shared persistence would prevent high coupling. #6 "Common sense and the team's experience from previous work on bad practices was used. #7 "Yes, during the migration, a mega service was discovered that was disconnected and required a lot of code to integrate."

6.2.3 Question 13

#1 "Yes, the organization's principle is to use DDD as a basis for development, and DDD was used to separate the monolith". #2 Yes, some concepts were used and so was The Twelve-Factor App (Wiggins, 2017). #3 "DDD causes as much trouble as microservices, and actually one place where DDD doesn't make sense is the microservice, like, almost always, OOP. OOP is to manage complexity, DDD is to manage extreme complexity, microservice is to make everything small and avoid (individual) complexity, it makes no sense to mix things up. People are doing random things with no real study. Anyone using DDD is making a monolith broken into parts and not MS, and has no idea what they are doing."#4 Separate service based on demand, if a service has more access, or is using more resources. #5 "No specific approach was used, it was separated by business". #6 "It was thought about architecture, technologies, ways of communication between microservices, event-drive (message), DDD and Clean Architecture was applied". #7 "Breaking down by business context, then, during the migration, different microservices were detected that worked on the same data domain and synchronization between them was necessary. After understanding this problem, a study on DDD was started to better structure it in the context of domains."

6.2.4 Question 14

#1 "Database is already separated, the same technique was uMatheus de Lara Dias da Silvased for database separation". #2 "(DDD – Bounded Context) of the services involved". #3 "If you are going to adopt microservices, each one must have their own, otherwise it's not microservice. Doing this means that the application will have to take care of what the DB used to take care of for free for you. There are strategies in the monolith when the DB can't take it. ".#6 "The old database was still used for a while, database normalization practices". #7 "see what were the existing standards for database separation in the context of microservices from the beginning, he already understood that he should separate. isolation between microservices 99% of communications between miscroservice is used rest api or messaging each service has its information domain have different instances to increase resilience horizontal distribution of data each tenant has its specific maintenance shema to have insulation"

6.2.5 Question 15

#1 and #2 Recreate the system from scratch, so they eliminate any antipattern from monolithic system, like code smells #1 says "Yes, the system was rebuilt practically from 0, so it was planned to eliminate any kind of antipattern existing in the monolithic system". #2 "Was recreated from scratch."#3 "Well done refactoring solves problems without adopting microservice. Microservice is an excuse to refactor. Solve application problems without adopting microservice, which creates a new problem that you didn't have before". #4 Part of the system is in the cloud and part of the system's infrastructure, when the system was designed, part of the data that would be in the cloud was changed to be on the client side, which led to maintenance problems. #5 Has used sonarQube and code review to eliminate code smells and bugs. #6 "It was tried to apply TDD, but due to deadlines and the team's lack of experience, it was not possible to continue with this practice".

6.2.6 Question 16

The sixteenth question was related to lessons learned. #2 mentioned problems with provisioning, communication between services, authentication and authorization. "I think few projects really need this approach (microservice), especially if it is created from scratch, with a team of up to 15 people I think it is completely unfeasible. #3 No, hardly anyone needs it. There may be some point microservices where it makes sense, but not a microservice architecture, not something until you have a good monolith. Do not use crutches. #4 "So far there was no big problem, problems were easy to solve Data access problem requires a lot of rework, few people add new features or do refactoring, sometimes time is wasted on refactoring Lack of system documentation, difficulty in updating."#5 "Levelling of knowledge about microservices, distributed systems, training about technologies to be used more people involved in technology decisions, more testing. After implementing and starting to use the system, a poor choice of technology was identified and there was no time to change the situation."#6 "Difficult because I was not involved in the planning, had no knowledge of architecture and software engineering. The lesson would be to study more, consume more materials."#7 "Do the best you can with the information you have today, you will make mistakes and learn and correct from them. Problems were found that brought good experiences to help other teams and serve as a foundation for future implementations, do not wait until you have all the knowledge in the world to get started"

6.2.7 Question 17

The last question (Table 15) #1 "I would develop microservices directly, remembering that if I develop in a monolith and then have to migrate, I will not have duplicate work. The lessons they learned during the migration were useful to develop in a way that also avoids the antipatterns."#2 "It depends on the team. If it is a big team that already knows the project and really needs it, I would use it without a doubt, otherwise not. In my case, I have always worked in small and medium-sized companies, I would prefer to start with a monolith without a doubt and migrate gradually."#3 "You do not need a microservice and if you do, it has to be PROVEN. You must have something simple that has proven incapable of meeting the requirements when done right, and EVERY has been tried to solve the problem without it. Almost always, accepting MS means a testimony of incompetence.". #4 "develop monolithically at first, now you get a feel for using shared libraries, for example, and a macro understanding of the system due to the complexity of microservices, at the beginning it would be harder, an initial project sometimes has no idea of basic things like business rules that mature over time. The initial investment was also low, it would not have been possible to develop in MS in the beginning. #5 "it depends on the maturity of the team a mature team already has the baggage to work directly

Table 15 – Responses of 17th question

ID	Monolithic	Microservice
1		✓
2	✓	
3	✓	
4	✓	
5	✓	
6		✓
7	✓	

with microservices, with separate teams if it is not clear how the system will work, not enough knowledge in microservices, it is best to start in monolithic when the knowledge related to the business is very mature and advanced."#6 "Developing in microservices is what allows you to deliver new products because of the scalability and the development practices are not always good. Maybe the best way would be to think about strategies that make the developer's life easier."#7 "develop in monolithic first with monolithic it would be much easier to involve the customer to use the information to teach the best configuration to solve his needs they started in microservice without fully understanding monolithic business contexts, it is easier".

6.3 Data Analysis

6.3.1 1st Group of questions

In the first group of answers, not many motivations for migration were discovered. Looking at all the benefits of microservices known from the literature, only five motivations were mentioned, most of which related to modern technology, followed by scalability and maintainability with two mentions. One of the so-called advantages of microservices is the ability to use different technologies. One of the respondents mentioned that he used this feature initially and encountered so many problems that he preferred to give up and use the same technology for all microservices today.

The motivation for modern technologies was confirmed when respondents were asked what technologies exist in monolith. Most of them were using outdated technologies or outdated versions of a particular technology, such as Java 4, so it is a trend to be up to date in terms of versions of frameworks and technologies.

The most common method to learn about microservices was learning from books, then lectures and case studies. Academic articles/papers were not mentioned. Regarding the members of the initial migration teams, it became clear that most of them start with a small number of people that gradually grow over time. The teams also change over time. One of them said that after some time they had only one architect per team.

Almost half of the respondents did not think about planning a set of microservices from the beginning. They feel that this was not as important initially, although it was critical given the need for at least an estimated number of functionalities or modules. The other responses mention a low number of microservices, with the exception of one that initially stands out with a high number of microservices. This suggests that it is not easy to have a metric for creating a microservice. Most of the respondents use business logic to determine the right size of each microservice. Migration time is also not easy to measure, with most respondents taking at least six months between study and development.

6.3.2 2nd Group of Questions

Nearly all respondents indicated that they knew about the megaservice antipattern prior to this interview. This was to be expected, as it became clear during the RR that a distributed monolith was a concern and a warning. Although other antipatterns such as misuse of shared libraries were also a concern, respondents did not cite this as a major problem. In fact, they said they intentionally use cyclic dependencies, and when the problem of one microservice crashing and the others crashing as well due to this cycle was explained, they said that none of this was a big problem to solve, but just organizing messaging between microservices.

The most commonly cited approach to mitigating antipatterns was the use of DDD. Considering that this approach is also used to separate monoliths, it makes sense when considering the most common antipatterns such as megaservice, the correct use of the context boundary, and the contexts business and logical data separation.

In general, everyone who participated in this interview is eager to avoid antipatterns, especially to avoid future problems, such as those related to maintainability. For example, one of the interviewees mentioned a wrong choice of technology. This has nothing to do with an antipattern, but he regretted it because there is no easy way to get rid of this technology anymore, and today it is necessary to rewrite a good part of the system. That's why it's important to make your decisions carefully when planning. That does not mean that bad decisions will never happen, but this investigation should make it clear that antipatterns exist and that there are ways to avoid them. At the beginning of this research, I had the idea of looking for refactoring techniques that could avoid antipatterns. However, during the interviews I realized that refactoring is not a suitable approach due to changing technologies and the need to rewrite the whole system into microservices. Studies and frameworks are being developed in the literature that analyze source code and separate it based on various factors, such as logs. None of the interviewees mentioned any kind of reuse of the monolithic system, except as a source of information or in some form of extracting functionality, not as copy and paste, but as a way to look at the system and replicate it with the new technology and architecture.

Respondents demonstrated a good understanding of the complexity of microservices and reported issues they faced that can be summarized as a lack of preparation for working with microservices due to inadequate training and study. I have found that it is easier to learn while working than to study and then apply the knowledge. This is due to the time it takes to study and the cost that companies sometimes do not want to pay. One of the interviewees mentioned that it is interesting to execute with the knowledge you have right now, and when you make mistakes, you learn from them.

Developing a new application in microservices does not seem to be the best solution. Martin Fowler mentioned this (Fowler, 2015), and the interviewees confirmed this due to the complexity of microservices. In a monolith, it would be easier to involve the customer, find the best way to solve the problem, the best configuration, etc. This problem may also depend on the size and maturity of the team. As some interviewees mentioned and can also be found in the literature, a small team is not a reason to struggle with microservices.

6.4 Summary

This chapter presents the results of the interviews, how the migration process to microservices and the process of studying and understanding the complexity of microservices went, what experience and knowledge the interviewees had. And also about the knowledge of the interviewees about antipatterns in microservices and their strategies to mitigate them in the future. The next chapter focuses on a more direct discussion and final conclusions from this research.

7 Analysis and Discussions

7.0.1 Analysis procedure

In this chapter I discuss the results based on previous chapters. A thematic synthesis map will be drawn to represent graphically codes/themes and its relationships. The main research question will be answered through the analysis and discussion of the data compared, codified and themed.

7.1 Systematic Literature Mapping Analysis

The initial readings and surveys of the main points of each study, using a spreadsheet with rows, columns, where the first column was the article ID and each following column represented a research question and subresearch questions. In analyzing the responses, they were divided into 3 categories: Modularization, Decomposition, and Technical Debt. Within technical debt, 3 subcategories were found: Antipatterns, Architectural Debt, and Refactoring..

- Modularization: In this category, was grouped studies focused on modularizing the monolithic system with strategies to achieve this goal and assist in the definition of each microservice. Was noticed that there is a tendency to seek the modularized system, or even develop a modularized monolithic system in order to migrate to microservices more easily in the future.
- Decomposition: In this category was grouped studies that aimed to extract microservices from the monolithic system, using strategies or frameworks for this.
- Technical Debt:

  Antipatterns: Studies that focus on analyzing problems that occurred in the phase of migration from monoliths to microservices, which show a lack of understanding about the complexity of microservices.

  Architectural debts: Studies that showed types of defects related to the difficulty of maintaining a system.

  Refactoring: Studies that show refactoring techniques or strategies as a way to migrate the monolithic to microservices.

After organize and separate the findings into categories and with further reading and analysis the following strategies emerged: DDD, Evolvability Assurance, Strategy to

Table 16 – SLM Codes

Code	**Description**
DDD for Modularization	Using DDD for a better understanding of the domain, this approach is useful to help find the right boundaries for each microservice
Evolvability Assurance	Use of methods, techniques, and tools to evaluate or improve evolvability and to facilitate sustainable long-term development
Strategy to Identify Tight Coupling	Ways to prevent high coupling in the system
Group Entities	Ways to ensure group entities based on Business Domain
Classify DB in Business Subsystem	Ways to separate and classify tables of database based on business Domain

Identify Tight Coupling, Group Entities and Classify DB in Business Subsystem. Each strategy was transformed in code (Table 16).

For more details on the responses collected in the SLM, see the Villaca (2022b).

7.2 Rapid Review Analysis

For this rapid review the same strategy was adopted, I created a spreadsheet to collect and group the data, where the first column was the study ID and the following were the research questions for the rapid review and metrics to collect. Analyzing the texts, it was difficult to extract the strategies the way I wanted, this happens due to the novelty of this research, antipatterns were recently cataloged and not all practitioners know them, and for this reason mitigate them it is not always a concern. But I was able to group the information into the following categories with a thorough analysis of the responses collected: Log Aggregation, Data, Coupling and Cohesion, Strangler Pattern, and Modularity.

- Log Aggregation: Use of a log tool to assist developers to identify with bugs, functionalities that could be decomposed in microservices;
- Data: Look at data first, to analyse entities and group them in subsystems;
- Coupling and Cohesion: Look at coupling and cohesiveness to identify bottlenecks in the system, this category also could be related to refactoring in SLM categories;
- Strangler Pattern: Strategy to decompose the monolithic in migration phase, develop each functionality until the monolithic is extinct.

Table 17 – Rapid Review Codes

Code	Description
Use of Backlog strategy	Use of a log tool to assist developers
Look at data first	Analyse entities and group them in subsystems before looking at code
Measure coupling and cohesion	Strategy to Identify Tight Coupling
Use of strangler pattern	Extract each functionality until the monolithic is extinct
DDD for Modularization	Using DDD for a better understanding of the domain, this approach is useful to help find the right boundaries for each microservice

- Modularization: Use of strategies, like DDD, for modularization. Category also founded in SLM.

When analysing each category, the following strategies emerged: Use of Backlog strategy, Look at data first, measure coupling and cohesion, use of strangler pattern and DDD for modularization. Each strategy was transformed in code (Table 17).

For more details on the responses collected in the Rapid Review, see the Villaca (2022a).

7.3 Interview Analysis

The interview analysis process was different because I did another type of data collection, seeking initial information that could help to understand the migration context of each interviewee, so different categories were defined, but not all of them will be transformed into code. The categories defined were experience and members, motivation for migration, technologies used, planning, antipatterns knowledge, antipatterns mitigate strategies, antipatterns decomposition strategies and lessons learned.

List of Categories:

- Experience and members: collected experience of each interviewee and members of the migration team;
- Motivation for migration: what was the motivation for migration, if was related with the know benefits from microservices;
- Technologies used: what was the technologies used;
- Planning: how was the planning process from pre-migration and migration;

Table 18 – Interview Codes

Code	**Description**
DDD for Modularization	Using DDD for a better understanding of the domain, this approach is useful to help find the right boundaries for each microservice
Clean Architecture	Create layered architectures that are simple, extendable and easy to maintain.
Twelve Factor App	12 principles of making software that can be scaled quickly and maintained in a consistent manner

- Antipatterns knowledge: what was the knowledge of antipatterns of interviewee and members;
- Antipatterns mitigate strategies: what was the strategies to mitigate antipatterns;
- Antipatterns decomposition strategies: what was the strategies to decompose the microservices;
- Lessons learned: what was the lessons learned of the pre-migration and migration process.

Of the categories analysed the following strategies appear: DDD, Clean Architecture and Twelve Factor App (Table 18).

Table 19 – Themes & Codes

Theme	Codes	SLM	RR	Interview
Modularity	Domain-Driven Design	✓	✓	✓
	Strangler Pattern		✓	
	Identify Tigh Coupling	✓	✓	✓
	Backlog		✓	
Data	Group Entities	✓		
	Classify DB in Business SubSystem	✓	✓	
	Data First		✓	
Organisational Culture	Clean Architecture			✓
	Twelve Factor App			✓
	Evolvability Assurance	✓		

7.4 Themes

Accordingly to Cruzes and Dyba (2011) the codes founded in the 3 phases of this research (Tables 16, 17 and 18), were grouped and transformed in the themes, which are as follows: modularity, data and organization culture (Table 19).

Modularity includes DDD which was mentioned in SLM, RR and Interview, strangler pattern, identify tight coupling and cohesion and use of backlog. Data includes group entities, classify DB in business subsystem and data first. Organizational Culture includes use of clean architecture, twelve factor app and evolvability assurance.

Accordingly with Cruzes and Dyba (2011) the process for thematic analysis is to reduce overlap and transform codes into themes and then create models or high order themes. For this research I didn't fell the need for do this process of create a theme and then another high order themes, because did not have enough data. So from codes I could extract themes and this make sense at the time.

As showed in Figure 9 for a better visualization, I grouped the codes into themes. Modularity was linked with Data because each module or later each microservice will have his own data, so make sense that this strategies could be related, and on the other hand, this Databases strategies could not be inside Modularity theme because you look at data before you look at the code, or at the system as a whole. Clean Architecture was linked with DDD because this was mentioned in the interviews, as an strategy to take in parallel and more than a strategy do create a microservice, clean architecture it is to implement in an intrinsic way inside the organization. Same way I linked twelve factor app to data first, because make sense to use a strategy that can help the organization to maintain the system in a consistent manner, which means to start to use an approach and stick with it at the end.

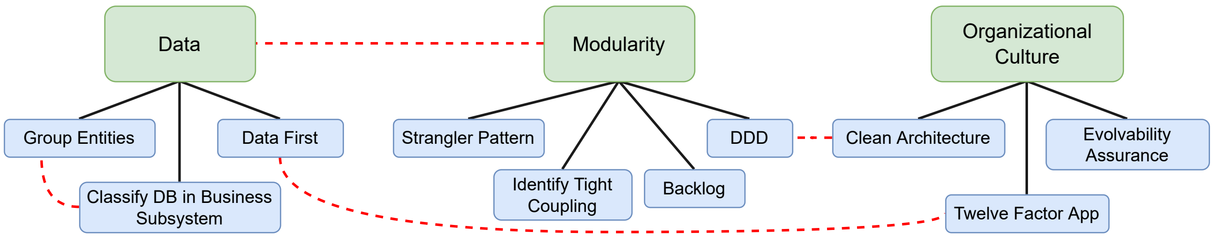


Figure 9 – Thematic map of strategies to mitigate anti-patterns in microservices

7.5 Discussion

So of the main research question - *What are the adopted strategies before the migration from a monolithic system to microservices in order to mitigate existing antipatterns in microservices* - came clear that the use of some strategies can help to mitigate antipatterns in microservices. I highlight the use of DDD (Fan; Ma, 2017; Silva; Carneiro; Monteiro, 2019; Newman; Reisz, 2020; Lavann; EdPrice; neilpeterson, 2021; Assouline; Grazi, 2017; Samokhin, 2018) or Strangler Pattern (Kornilov, 2020; Sitnikova, 2021; Goel, 202; Richardson, 2016) to mitigate megaservice and wrong cuts. As mentioned by Silva, Carneiro and Monteiro (2019) DDD lowers the chances of two microservices needing to share a model and corresponding data space, risking a tight coupling, which means that the practitioner would have better control over the right size of the microservice and its boundaries.

Any of Data theme to mitigate shared persistence. Relational data needs to be designed conceptually from the start, before you even consider the placement and design of functions to carry out business processes (Hubers, 2021). Group entities strategy is to define similarity measures between domain entities and use a clustering algorithm that returns clusters of entities (Santos; Silva, 2020).

Each of the organizational culture themes can mitigate shared libraries and cyclic dependencies as mentioned by the interviewees, as well as concerns about maintaining and developing clean and scalable code in the long term.

Use of backlog strategy (Sigma, 2021; Newman; Reisz, 2020) can also mitigate megaservice, microservice greedy because you can measure if a functionality uses few resources or if it can be aggregated with another. Also identify tight coupling can help to mitigate shared libraries. When designing applications, avoid using library code that increases coupling between applications, so that developers building clients are free to make different technology choices if that becomes advantageous (Deshpande; Singh, 2020).

These strategies found in this study are not a guarantee to avoid antipatterns, they are only ways to mitigate them. It is important to keep in mind that these strategies must be taken along with good planning, team training, investment in consulting when possible, and other measures that can ensure that the migration to microservices can be done gradually and with as few errors as possible.

Some of the antipatterns cataloged by Carrasco, Bladel and Demeyer (2018) like thinking microservices are a silver bullet; rewrite all services into microservices at once; learn as you go - although it has not been considered in most of this study, can be mentioned at this moment because it can also be mitigated, if you think that a thorough study of microservices and better planning is enough to invalidate the idea that microservices are a silver bullet. Rewrite all services into microservices at once can be mitigates using strangler pattern (Kornilov, 2020; Sitnikova, 2021; Goel, 202; Richardson, 2016). One of the interviewees cited "learning as you go"as one of the reasons for the problems encountered during the migration phase to microservices. As mentioned earlier, a thorough study can help to mitigate this antipattern.

7.6 Limitations of the study

The limitations of the proposed strategies for mitigate antipatterns in microservices are listed as follows:

- Strategies was defined in high level, more information could be extracted in each step from the methodology, especially from the interviewees;
- Each strategy could be more explored, e.g. more studies from DDD could be analysed and detailed strategies could be described;
- Strategies comparable to those that appeared in this study could be analyzed, e.g., for clean architecture there would be other strategies that could have the same effect. The same could be done for DDD and other strategies.
- Although they produced good responses, the interviews conducted for this study could be done differently, with a longer implementation time that would bring more respondents into the study, and the questionnaire could also be designed differently to obtain more information from respondents.

7.7 Summary

In this chapter, I have discussed data analysis using a thematic synthesis approach. It can be stated that the main research question has been answered, although these conclusions have only been drawn from the reading and analysis and further studies can be conducted to prove whether the answers are true or not.

8 Conclusion & Future Work

8.1 Conclusion

This dissertation advances in finding forms to mitigate the common antipatterns in microservices before migrating from monolithic to microservices.

At the beginning of the study, I set out the background to familiarize the reader with the context of the study. I wrote about monolithic architectures, why monolithic architectures degrade, I talked about one of the different ways to modernize monolithic systems, namely migrating to microservices, and finally I presented the catalog of antipatterns in microservices that exist in the literature so far. Some of the antipatterns are exclusive to microservices as they relate to the architecture of distributed systems, while others can also be mitigated in monolithic systems, which are the focus of this research.

In Chapter 3, I talked about the methodology, starting with the systematic literature mapping, the definition of the research question and sub-questions, and the search string, after I deemed it necessary to do a rapid review of the grey literature to confirm or not the SLM findings. Then, I prepared the interviews using the GQM technique to identify the goals, questions, and metrics. This process was helpful to develop the interview questions. Finally, I discuss the thematic synthesis that I used in the data analysis.

Chapters 4 and 5 presented the SLM and RR, respectively, where I found the categories that would later be transformed into codes and themes.

Chapter 6 addressed the interview process, in which I elaborate the questions based on the information previously obtained in the methodology. I conducted seven interviews, two via Google Forms and five via Google Meet, all of which were conducted remotely due to time, distance, and COVID-19. During the interviews, I found that antipatterns were indeed an issue. All interviewees demonstrated knowledge of Antipatterns, some even under different names. The use of DDD proved to be the most practical way to avoid the most important antipattern, megaservice. However, in the analysis and reading, I realized that having a megaservice in microservices may be a symptom of another problem, a poor choice, because perhaps the microservice was not the best solution, but the monolith itself. Some of the interviewees brought this to my attention in the last question.

Chapter 7 focused on the data analysis and discussions where I presented the thematic analysis by extracting codes from SLM, Rapid Review, and interview and then converting them into codes and then into themes. This process was helpful to define the answers to my research question. I believe that antipatterns such as megaservice, which was most frequently mentioned by the respondents, can be mitigated with strategies such as DDD. The other antipatterns can be mitigated through better planning and use of the other strategies found in this study.

8.2 Future Work

Microservices are still a new topic in software engineering and there is still a lot of work to be done on this topic. Perhaps new antipatterns will emerge in the future and this research will need to be revised and updated. It might also be necessary to start from the direct development of microservices and apply the strategies mentioned in this research to see if antipatterns can be mitigated.

In a new interview, it could be done similarly, but with more emphasis on each antipattern, i.e., ask the respondent to answer to each antipattern, which would be the best strategy to mitigate, because I did it in an open way where the respondent could freely answer and quote as they saw fit.

The strategies founded in this research can be validated in other studies, like a case study, to see if the strategies can be applied, what is the scenario that would be applied and what would be the results.

This study focused on researching the literature and experts' opinions on what strategies exist to mitigate anti-patterns in microservices. Future research can focus on how to apply these strategies and what practical impact they have on microservices architecture.

References

Abdellatif, M. et al. A taxonomy of service identification approaches for legacy software systems modernization. *Journal of Systems and Software*, v. 173, p. 110868, 2021. ISSN 0164-1212. Available on: <https://www.sciencedirect.com/science/article/pii/S0164121220302582>.

Abdullah, M.; Iqbal, W.; Erradi, A. Unsupervised learning approach for web application auto-decomposition into microservices. *Journal of Systems and Software*, v. 151, 02 2019.

Ahmed, I.; Mannan, U. A.; Gopinath, R.; Jensen, C. An empirical study of design degradation: How software projects get worse over time. In: *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 2015. p. 1–10.

Amiri, M. J. Object-aware identification of microservices. In: . 2018.

Assouline, P.; Grazi, V. *Perspective on Architectural Fitness of Microservices*. 2017. <https://www.infoq.com/articles/Microservices-Architectural-Fitness/>.

B. Pinto G., S. S. C. Rapid reviews in software engineering. 2020.

Balalaie, A.; Heydarnoori, A.; Jamshidi, P. Migrating to cloud-native architectures using microservices: An experience report. In: . 2015. ISBN 978-3-319-33312-0.

Balalaie, A.; Heydarnoori, A.; Jamshidi, P. Microservices architecture enables devops: an experience report on migration to a cloud-native architecture. *IEEE Software*, v. 33, p. 1–1, 05 2016.

Basili, V.; Rombach, H. The tame project: towards improvement-oriented software environments. *IEEE Transactions on Software Engineering*, v. 14, n. 6, p. 758–773, 1988.

Bogner, J.; Fritzsch, J.; Wagner, S.; Zimmermann, A. Assuring the evolvability of microservices: Insights into industry practices and challenges. In: . 2019.

Bonér, J.; Farley, D.; Kuhn, R.; Thompson, M. *The Reactive Manifesto*. 2014.

Brogi, A.; Neri, D.; Soldani, J.; Zimmermann, O. Design principles, architectural smells and refactorings for microservices: A multivocal review. 06 2019.

Brooks, A.; Daly, J.; Miller, J.; Roper, M.; Wood, M. *Reactor risk reference document - Technical Report NUREG-1150*. Washington D.C, 1989.

Candela, I.; Bavota, G.; Russo, B.; Oliveto, R. Using cohesion and coupling for software remodularization: Is it enough? *ACM Trans. Softw. Eng. Methodol.*, Association for Computing Machinery, New York, NY, USA, v. 25, n. 3, jun. 2016. ISSN 1049-331X. Available on: <https://doi.org/10.1145/2928268>.

Carrasco, A.; Bladel, B. van; Demeyer, S. Migrating towards microservices: migration and architecture smells. In: . 2018. p. 1–6.

Carvalho, L. et al. Extraction of configurable and reusable microservices from legacy systems: An exploratory study. In: *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A*. New York, NY, USA: Association for Computing Machinery, 2019. (SPLC '19), p. 26–31. ISBN 9781450371384. Available on: <https://doi.org/10.1145/3336294.3336319>.

Christoforou, A.; Odysseos, L.; Andreou, A. Migration of software components to microservices: Matching and synthesis. In: *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering*. Setubal, PRT: SCITEPRESS - Science and Technology Publications, Lda, 2019. (ENASE 2019), p. 134–146. ISBN 9789897583759. Available on: <https://doi.org/10.5220/0007732101340146>.

Cojocaru, M.; Uta, A.; Oprescu, A.-M. Microvalid: A validation framework for automatically decomposed microservices. In: *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. 2019. p. 78–86.

Comella-Dorda; Wallnau; Seacord; Robert. A survey of black-box modernization approaches for information systems. In: *Proceedings 2000 International Conference on Software Maintenance*. 2000. p. 173–183.

Cruzes, D. S.; Dyba, T. Recommended steps for thematic synthesis in software engineering. In: *2011 International Symposium on Empirical Software Engineering and Measurement*. 2011. p. 275–284.

de Toledo, S. S.; Martini, A.; Sjøberg, D. I. Identifying architectural technical debt, principal, and interest in microservices: A multiple-case study. *Journal of Systems and Software*, v. 177, p. 110968, 2021. ISSN 0164-1212. Available on: <https://www.sciencedirect.com/science/article/pii/S0164121221000650>.

Desai, U.; Bandyopadhyay, S.; Tamilselvam, S. Graph neural network to dilute outliers for refactoring monolith application. 02 2021.

Deshpande, A.; Singh, N. P. *Challenges and patterns for modernizing a monolithic application into microservices*. 2020. <https://developer.ibm.com/articles/challenges-and-patterns-for-modernizing-a-monolithic-application-into-microservices/>.

Escobar, D. et al. Towards the understanding and evolution of monolithic applications as microservices. In: *2016 XLII Latin American Computing Conference (CLEI)*. 2016. p. 1–11.

Fan, C.-Y.; Ma, S.-P. Migrating monolithic mobile application to microservice architecture: An experiment report. In: *2017 IEEE International Conference on AI Mobile Services (AIMS)*. 2017. p. 109–112.

Fink, A. Survey research methods. In: _____. 2010. p. 152–160. ISBN 9780080448947.

Fowler, M. *StranglerFigApplication*. 2004. <https://martinfowler.com/bliki/StranglerFigApplication.html>.

Fowler, M. *Microservices a definition of this new architectural term*. 2014. <http://martinfowler.com/articles/microservices.html>.

Fowler, M. *Monolith First*. 2015. <https://martinfowler.com/bliki/MonolithFirst.html>.

Fowler, M. *Refactoring Improving the Design of Existing Code*. 2018. <https://martinfowler.com/books/refactoring.htm>.

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

Furda, A.; Fidge, C.; Zimmermann, O.; Kelly, W.; Barros, A. Migrating enterprise legacy source code to microservices: On multitenancy, statefulness, and data consistency. *IEEE Software*, v. 35, n. 3, p. 63–72, 2018.

Gamma, E.; Helm, R.; Johnson, R. E.; Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995. (Addison-Wesley Professional Computing Series). ISBN 978-0-201-63361-0. Available on: <https://www.safaribooksonline.com/library/view/design-patterns-elements/0201633612/>.

Garousi, V.; Felderer, M.; Mäntylä, M. Guidelines for including the grey literature and conducting multivocal literature reviews in software engineering. 07 2017.

Goel, A. *CHOOSING THE RIGHT STRATEGY TO MIGRATE YOUR MONOLITHIC APPLICATION TO A MICROSERVICES-BASED ARCHITECTURE*. 202. <https://capgemini-engineering.com/us/en/insight/part-3-choosing-the-right-strategy-to-migrate-your-monolithic-application-to-a-microservices-based-a
>.

Gupta, A. *From Monolith to Microservices: An epic Migration Journey*. 2021. <https://dzone.com/articles/from-monolith-to-microservices-an-epic-migration-j>.

Haywood, D.; Betts, T. *In Defence of the Monolith, Part 1*. 2017. <https://www.infoq.com/articles/monolith-defense-part-1/>.

Henry, A.; Ridene, Y. Migrating to microservices. In: _____. *Microservices: Science and Engineering*. Cham: Springer International Publishing, 2020. p. 45–72. ISBN 978-3-030-31646-4. Available on: <https://doi.org/10.1007/978-3-030-31646-4_3>.

Huang, X.; Lin, J.; Demner-Fushman, D. Evaluation of pico as a knowledge representation for clinical questions. *AMIA Annu Symp Proc*, p. 359–363, 02 2006.

Hubers, T. *How big is a microservice?* 2021. <https://medium.com/geekculture/the-size-of-a-microservice-b9e6bc90475>.

Ibryam, B. *When Microservice Fails*. 2021. <https://docs.google.com/spreadsheets/d/1vjnjAII_8TZBv2XhFHra7kEQzQpOHSZpFIWDjynYYf0/edit#gid=0>.

Jambunathan, B.; Y., K. Microservice design for container based multi-cloud deployment. *International Journal of Engineering and Technology (IJET)*, v. 8, 02 2016.

Jambunathan, B.; Y., K. Multi cloud deployment with containers. v. 8, p. 421–428, 02 2016.

Janes, A.; Russo, B. Automatic performance monitoring and regression testing during the transition from monolith to microservices. In: *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. 2019. p. 163–168.

Janssen, T. *From Monolith to Microservices – Migrating a Persistence Layer*. 2021. <https://thorben-janssen.com/monolith-to-microservices-persistence-layer/>.

Jin, W.; Liu, T.; Zheng, Q.; Cui, D.; Cai, Y. Functionality-oriented microservice extraction based on execution trace clustering. In: . 2018. p. 211–218.

Kamei, F. et al. Grey literature in software engineering: A critical review. *CoRR*, abs/2104.13435, 2021. Available on: <https://arxiv.org/abs/2104.13435>.

Kaplunovich, A. Tolambda–automatic path to serverless architectures. In: *2019 IEEE/ACM 3rd International Workshop on Refactoring (IWoR)*. 2019. p. 1–8.

Kazanaviius, J.; Mazeika, D. Analysis of legacy monolithic software decomposition into microservices. In: *Doctoral Consortium/Forum*. 2020.

Kazanavičius, J.; Mažeika, D. Migrating legacy software to microservices architecture. In: *2019 Open Conference of Electrical, Electronic and Information Sciences (eStream)*. 2019. p. 1–5.

Kecskemeti, G.; Marosi, A.; Kertesz, A. The entice approach to decompose monolithic services into microservices. *2016 International Conference on High Performance Computing & Simulation (HPCS)*, p. 591–596, 2016.

Khadka, R. et al. Does software modernization deliver what it aimed for? a post modernization analysis of five software modernization case studies. In: *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2015. p. 477–486.

Kitchenham, B. A.; Charters, S. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. 2007. Available on: <https://www.elsevier.com/___data/promis_misc/525444systematicreviewsguide.pdf>.

Kitchenham, B. A.; Pfleeger, S. L. Personal opinion surveys. In: Shull, F.; Singer, J.; Sjøberg, D. I. K. (Ed.). *Guide to Advanced Empirical Software Engineering*. : Springer London, 2008. p. 63–92.

Knoche, H.; Hasselbring, W. Drivers and barriers for microservice adoption - a survey among professionals in germany. v. 14, p. 1–35, 01 2019.

Kornilov, D. *Monolithic to microservices: Design patterns to ensure migration success*. 2020. <https://blogs.oracle.com/cloud-infrastructure/post/monolithic-to-microservices-how-design-patterns-help-ensure-migration-success>.

Kuryazov, D.; Jabborov, D.; Khujamuratov, B. Towards decomposing monolithic applications into microservices. In: *2020 IEEE 14th International Conference on Application of Information and Communication Technologies (AICT)*. 2020. p. 1–4.

Laigner, R.; Lifschitz, S.; Kalinowski, M.; Poggi, M.; Salles, M. A. V. Towards a technique for extracting relational actors from monolithic applications. In: . 2019.

Lavann; EdPrice; neilpeterson. *Migrate a monolith application to microservices using domain-driven design*. 2021. <https://docs.microsoft.com/en-us/azure/architecture/microservices/migrate-monolith>.

Lea, G. *Why "Don't Use Shared Libraries in Microservices" is Bad Advice*. 2016. <https://www.grahamlea.com/2016/04/shared-libraries-in-microservices-bad-advice/>.

Lenarduzzi, V.; Lomio, F.; Saarimäki, N.; Taibi, D. Does migrating a monolithic system to microservices decrease the technical debt? *Journal of Systems and Software*, v. 169, 07 2020.

Levcovitz, A.; Terra, R.; Valente, M. Towards a technique for extracting microservices from monolithic enterprise systems. 05 2016.

Li, M.; Smidts, C. A ranking of software engineering measures based on expert opinion. *IEEE Transactions on Software Engineering*, v. 29, n. 9, p. 811 – 824, sept. 2003. ISSN 0098-5589.

Li, R.; Liang, P.; Soliman, M.; Avgeriou, P. Understanding architecture erosion: The practitioners' perceptive. *CoRR*, abs/2103.11392, 2021. Available on: <https://arxiv.org/abs/2103.11392>.

Li, S. et al. Understanding and addressing quality attributes of microservices architecture: A systematic literature review. *Information and Software Technology*, v. 131, p. 106449, 2021. ISSN 0950-5849. Available on: <https://www.sciencedirect.com/science/article/pii/S0950584920301993>.

Linthicum, D. S. Practical use of microservices in moving workloads to the cloud. *IEEE Cloud Computing*, v. 3, n. 5, p. 6–9, 2016.

Macia, I. et al. Are automatically-detected code anomalies relevant to architectural modularity? an exploratory analysis of evolving systems. In: *Proceedings of the 11th Annual International Conference on Aspect-Oriented Software Development*. New York, NY, USA: Association for Computing Machinery, 2012. (AOSD '12), p. 167–178. ISBN 9781450310925. Available on: <https://doi.org/10.1145/2162049.2162069>.

Mahanta, P.; Chouta, S. Translating a legacy stack to microservices using a modernization facade with performance optimization for container deployments. In: _____. 2020. p. 143–154. ISBN 978-3-030-40906-7.

Mak, S. *Modules vs. microservices: Apply modular system design principles while avoiding the operational complexity of microservices*. 2017. <https://www.oreilly.com/radar/modules-vs-microservices/>.

Marshall, C.; Brereton, P.; Kitchenham, B. Tools to support systematic reviews in software engineering: A cross-domain survey using semi-structured interviews. In: *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2015. (EASE '15). ISBN 9781450333504. Available on: <https://doi.org/10.1145/2745802.2745827>.

Martin, R. C. *Design Principles and Design Patterns*. 2000.

Mazlami, G.; Cito, J.; Leitner, P. Extraction of microservices from monolithic software architectures. In: *2017 IEEE International Conference on Web Services (ICWS)*. 2017. p. 524–531.

Newman, S.; Guimarães, L. *Migrating Monoliths to Microservices with Decomposition and Incremental Changes*. 2021. <https://www.infoq.com/articles/migrating-monoliths-to-microservices-with-decomposition/>.

Newman, S.; Reisz, W. *Sam Newman: Monolith to Microservices*. 2020. <https://www.infoq.com/podcasts/monolith-microservices/>.

Ogbole, M.; Ogbole, E.; Olagesin, A. Cloud systems and applications : A review. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, p. 142–149, 02 2021.

Pigazzini, I.; Fontana, F. A.; Lenarduzzi, V.; Taibi, D. Towards microservice smells detection. In: *Proceedings of the 3rd International Conference on Technical Debt*. New York, NY, USA: Association for Computing Machinery, 2020. (TechDebt '20), p. 92–97. ISBN 9781450379601. Available on: <https://doi.org/10.1145/3387906.3388625>.

Pigazzini, I.; Fontana, F. A.; Maggioni, A. Tool support for the migration to microservice architecture: An industrial case study. In: Bures, T.; Duchien, L.; Inverardi, P. (Ed.). *Software Architecture*. Cham: Springer International Publishing, 2019. p. 247–263. ISBN 978-3-030-29983-5.

Ren, Z. et al. Migrating web applications from monolithic structure to microservices architecture. In: *Proceedings of the Tenth Asia-Pacific Symposium on Internetware*. New York, NY, USA: Association for Computing Machinery, 2018. (Internetware '18). ISBN 9781450365901. Available on: <https://doi.org/10.1145/3275219.3275230>.

Richardson, C. *Refactoring a Monolith into Microservices*. 2016. <https://www.nginx.com/blog/refactoring-a-monolith-into-microservices/>.

Richardson, C. *Microservice Architecture*. 2018. <https://microservices.io/patterns/microservices.html>.

Richardson, C. *Refactoring a monolith to microservices*. 2021. <https://microservices.io/refactoring/>.

Roos, W. *Getting ready for microservices – strangling the monolith*. 2020. <https://amazicworld.com/getting-ready-for-microservices-breaking-down-the-monolith/>.

Samokhin, V. *Why Microservices Fail*. 2018. <https://hackernoon.com/why-microservices-fail-6cdc006f9540>.

Santos, N.; Silva, A. R. A complexity metric for microservices architecture migration. In: *2020 IEEE International Conference on Software Architecture (ICSA)*. 2020. p. 169–178.

Sayara, A.; Towhid, M. S.; Hossain, M. S. A probabilistic approach for obtaining an optimized number of services using weighted matrix and multidimensional scaling. In: *2017 20th International Conference of Computer and Information Technology (ICCIT)*. 2017. p. 1–6.

Selmadji, A.; Seriai, A.; Bouziane, H.; Dony, C.; Mahamane, R. Re-architecting oo software into microservices: 7th ifip wg 2.14 european conference, esocc 2018, como, italy, september 12-14, 2018, proceedings. In: _____. 2018. p. 65–73. ISBN 978-3-319-99818-3.

Shimoda, A.; Sunada, T. Priority order determination method for extracting services stepwise from monolithic system. In: *2018 7th International Congress on Advanced Applied Informatics (IIAI-AAI)*. 2018. p. 805–810.

Sigma. *MIGRATING MONOLITH TO MICROSERVICES: STEP-BY-STEP GUIDE*. 2021. <https://sigma.software/about/media/migrating-monolith-microservices-step-step-guide>.

Silva, H.; Carneiro, G.; Monteiro, M. Towards a roadmap for the migration of legacy software systems to a microservice based architecture. In: . 2019. p. 37–47.

Sitnikova, A. *Monolith vs Microservices: Everything You Need To Know*. 2021. <https://bambooagile.eu/insights/monolith-vs-microservices/>.

Solingen, R.; Berghout, E. The goal/question/metric method: A practical guide for quality improvement of software development. 01 1999.

Taibi, D.; Auer, F.; Lenarduzzi, V.; Felderer, M. From monolithic systems to microservices: An assessment framework. 09 2019.

Taibi, D.; Lenarduzzi, V. On the definition of microservice bad smells. *IEEE Software*, v. 35, n. 3, p. 56–62, 2018.

Taibi, D.; Lenarduzzi, V.; Pahl, C. Processes, motivations and issues for migrating to microservices architectures: An empirical investigation. *IEEE Cloud Computing*, v. 4, 10 2017.

Taibi, D.; Lenarduzzi, V.; Pahl, C. Microservices anti patterns: A taxonomy. In: _____. 2019.

Taibi, D.; Systä, K. From monolithic systems to microservices: A decomposition framework based on process mining. In: . 2019.

Teixeira, E. et al. Software process line as an approach to support software process reuse: a systematic literature review. *Information and Software Technology*, v. 116, 08 2019.

Tian, F.; Liang, P.; Babar, M. A. How developers discuss architecture smells? an exploratory study on stack overflow. In: *2019 IEEE International Conference on Software Architecture (ICSA)*. 2019. p. 91–100.

Tighilt, R. et al. On the study of microservices antipatterns: a catalog proposal. In: . 2020. p. 1–13.

Toledo, S.; Martini, A.; Sjøberg, D. Improving agility by managing shared libraries in microservices. In: _____. 2020. p. 195–202. ISBN 978-3-030-58857-1.

Vasylenko, A. *Monolithic Architecture*. 2018. <https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/>.

Villaca, G. L. D. rapidreview2.pdf. 4 2022. Available on: <https://figshare.com/articles/figure/rapidreview2_pdf/19527781>.

Villaca, G. L. D. systematic-literature-mapping.pdf. 4 2022. Available on: <https://figshare.com/articles/figure/systematic-literature-mapping_pdf/19526230>.

Walker, A.; Das, D.; Černý, T. Automated code-smell detection in microservices through static analysis: A case study. *Applied Sciences*, v. 10, 11 2020.

Wiggins, A. *Twelve Factor App*. 2017. <https://12factor.net/pt_br/>.

Wolfart, D. et al. Modernizing legacy systems with microservices: A roadmap. In: *Evaluation and Assessment in Software Engineering*. New York, NY, USA: Association for Computing Machinery, 2021. (EASE 2021), p. 149–159. ISBN 9781450390538. Available on: <https://doi.org/10.1145/3463274.3463334>.

Wolfart, D. et al. Towards a process for migrating legacy systems into microservice architectural style. In: . 2020. p. 255–264.

Yang, X.; Chen, L.; Wang, X.; Cristoforo, J. A dual-spiral reengineering model for legacy system. In: . 2005. v. 2007, p. 1 – 5.

Zirkelbach, C.; Krause, A.; Hasselbring, W. Modularization of research software for collaborative open source development. 07 2019.

Zirkelbach, C.; Krause, A.; Hasselbring, W. On the modularization of explorviz towards collaborative open source development. In: . 2019.

Appendix

.1 Systematic Literature Mapping

.1.1 Protocol Mapping

PROTOCOL - MAPPING

Research Topic: Refactoring Monolithic System Towards Migrations to Microservices

Authors: Guilherme Luciano Donin Villaca / Ivonei Freitas da Silva

Review History

Date	Review	Author	Role	Description
30/09/2020	1	Guilherme / Ivonei	Main researchers	Initial draft
26/10/2020	2	Guilherme	Main researcher	Reescrita background
16/11/2020	3	Ivonei	Main researcher	Revisão background
21/03/2021	4	Guilherme	Main researcher	Inclusion/Exclusion e QC
08/04/2021	5	Guilherme	Main researcher	String Review
28/04/2021	6	Guilherme	Main researcher	Revisão Final

1 Team

MEMBER	ACADEMIC DEGREE	RESPONSIBILITY	ROLE
Guilherme Luciano Donin Villaca	M.Sc. Candidate, University of the State of Paraná.	Develop protocol. Select studies. Collect, analyze and synthesize data. Assess studies quality. Write report.	Main Researcher
Ivonei Freitas da Silva	Ph.D., Federal University of Pernambuco.	Review report.	Advisor

2 Background

It is known that over time the software suffers degradation and, in order to maintain quality, the best approach is to reduce software defects (bug fixes) and design problems (refactoring) [1]. These problems can also be called anomalies in the code and are key factors for degradation, if not removed, the software can suffer from erosion, which occurs when architectural violations are introduced and drift, when bad design decisions affect the architecture [8]. Degradation occurs due to constant changes and implementations of new features, which is the way to meet the requirements that arise as the system grows. Such changes end up in conflict with the way the software was originally built, either with the design or with the initial implementation, and become expensive to revert, in addition, the technology used to build the software may become out of date after some years, becoming incompatible with current demands like the cloud. To contain the degradation, it is common to think of several approaches, whether to completely rebuild the system from scratch, using new design techniques and technologies.The modernization of software is one of the paths to be taken, whether in the form of refactoring, remodeling [5] or migration to another approach such as software product lines or to a new architecture, such as microservices. However, when thinking about a new architecture, such as microservices, it should be taken into account that an architecture based on distributed systems also brings greater complexity when compared to a monolithic system, where there was a single system to manage (deploy) , sometimes on a single server, and even if it were necessary to replicate that system on different servers, it is not comparable to dozens or hundreds of parts of a system being replicated on different servers. But when looking for scalability, resilience, heterogeneity of technologies and other benefits that the cloud provides, there must be a preparation for this scenario.

The essence of microservices is to decompose the application's components into small functionalities logically grouped as services, each running in its own process and communicating with lightweight mechanisms, usually an HTTP resource API [6]. In addition, a constant idea in this universe of applications in the cloud is the reuse of software, which brings efficiency to the application as a whole, making it possible to consume services without the need to create functionality from scratch [7]. The migration from monolithic systems to microservices has been an industry trend in recent years, gaining popularity and hype, inspired by big players such as Amazon, Linkedin, Netflix, among others.

However, in the literature (state of the art) there are researches that point out problems that occur in the migration from monolithic to microservices and that after the migration is completed these problems negatively affect the quality of the new architecture. According to taibi [3] [4] these problems also called bad smells and anti patterns are caused by bad practices that occur during development and affect some quality attributes of the software such as understandability, testability, extensibility, reusability and maintainability of the system under development. It is also concluded that practitioners need to carefully make the decision of migrating to MSA based on the return on investment, since this architectural style additionally causes some pains in practice [9]. In other words, the inside chaos of monoliths were transformed into some other external and visible complications after the separation and componentization.

This study aims to review and find what activities, and important practical and research issues concerning research directions. This review is systematically performed, following Kitchenham's guideline [10], which aids in assuring consistent results from a set of individual studies, called primary studies, since a single primary study does not answer a research question from every possible perspective at the same time and does not produce practical recommendations. Secondary studies, such as our study, are expected to give practical recommendations.

This document's rationale is stated at delineating the research objectives and clearly explaining the review carrying on process, through defining research questions and planning on how the sources and studies selection will be accomplished. The research questions will guide the design of the review process.

3 Research Questions

The research questions that will guide this study are:

Our main question is **What and How are the adopted strategies by the researchers or practitioners to refactor the architecture of a monolithic system before adopting a modernization process for Microservice?**
Rationale: *We want to understand whether they are preparing to avoid (or mitigate) future (and possible) bad smells after migrating.*
Strategies here can mean recommendations or refactoring techniques to improve the design of an existing code [13], the use of design patterns [14], design principles [15] or even approaches of reactive manifesto [16]. Other meanings for strategy are: **criteria, processes, guidelines, tools, patterns, metrics.**

Based on this question, we refine it into specific subquestions .

SQ. Are the microservice bad smells considered in those strategies?
Rationale: *Bad smells can be related to monolith or microservice architecture. Tools, techniques, etc maybe exist to mitigate these microservice bad smells.* On one hand, we want to understand whether microservice bad smells can be observed in monolithic systems before migrating to microservice architecture. For example, megaservice (a microservice bad smell) can be observed as a big module in monolithic architecture. On the other hand, we *also want to know whether those tools or techniques that deal with microservice bad smells could be customized or not when dealing with monolithic bad smells, and vice-versa.*

SQ. Which strategy is used to determine the extent of monolithic software degradation?
Rationale: *If the software is very degraded outdated technology, many defects (bugs) and design problems, what strategies in this scenario have been adopted for modernization.*

SQ. Which strategy is used to identify whether upgrading to microservices is possible?
Rationale: *If the software is very degraded or outdated technology, many defects (bugs) and design problems can be found, what strategies have been adopted for modernization.*

SQ. What was the context (scenario) of the monolithic system?
Rationale: *Context characterized by architecture, business goals, organizational, restrictions.*

SQ. What were the challenges and recommendations?
Rationale: *Migration to microservices is still on the rise in the industry and we still need to*

identify many challenges and good practices to assist new practitioners who wish to migrate to microservices

The research question and the sub questions are discussed from the following viewpoints of the PICO structure that is according to its guidelines, articulating a question in terms of its four anatomic parts—Problem/Population, Intervention, Comparison, and Outcome (PICO)—facilitates searching for a precise answer [11]:

Study Population: practitioners/projects that refactor monolithic software before migrating to microservices, researchers/projects that study refactoring of monolithic software before migrating to microservices.
Study Intervention: use of some software engineering strategy such as activities, practices, tools, standards, guidelines, patterns, metrics, criteria, evidence during the refactoring of the monolithic before migrating to microservices.
Study Comparisons: -
Study Outcomes:(i) what are the adopted strategies; (ii) how was the adoption of the strategy. (iii) What the problems, lessons learned, or challenges they have found during the refactoring. (iv) What are adopted criteria and evidence when selecting and adopting refactoring strategies.
Study design: all empirical studies designs such as case studies, technical reports on feasibility study, controlled experiments (and quasi-experiments), experience reports, the survey with the practitioners, and action research that show adoption of refactoring of monolithic software before migrating to microservices.

4 Search Process

The primary studies should be searched by following keywords: *monolith, microservice; smell, antipattern, bad practice, pitfall, refactor, reengineer, violation, defect and degradation.*

Search strings construction is based on research questions, PICO structure. They are assembled by using boolean ANDs and ORs to merge keywords. Following the search strings used in this review are listed:

- ((monolith*) AND (smell* OR antipattern* OR badpractice* OR pitfall* OR refactor* OR reengineer* OR violation OR defect OR degradation) AND (microservice* OR micro-service* OR "micro services"))

Based on [12] our strategy for search engines was defined in a similar way, by analysing their ability to use logic expressions, full text recovery, automated searches of paper content and searches using specific fields (for example, title, abstract, keywords). Primary studies search should be focused on important journals, conferences proceedings, books, thesis and technical reports. The search process is manual and based on web search engines, studies references and digital libraries of relevant publishers and organizations in software engineering, such as:

- Scopus (https://www.scopus.com/home.uri)
- IEEEXplore (https://ieeexplore.ieee.org/Xplore/home.jsp)
- ScienceDirect (www.sciencedirect.com)
- Google Scholar (https://www.googlescholar.org/)

Specific researchers can also be contacted directly through their WebPages and emails, if it is necessary to gather further information on any relevant research and the paper (or set of papers) related to the work does not provide enough information on such issues.

According to [10], it is necessary to define inclusion and exclusion criteria for the papers selection. These criteria should identify the primary studies, i.e. the ones identified during the research, which provide evidence about the research question. These criteria are presented in the following 5 and 6 sections.

5 Critério de Inclusão e Exclusão

Inclusion Criteria establish the inclusion reasons for each study found during the search. Studies on the following topics will be included:

1. Estudo descreve estratégia para refatorar sistemas monolíticos antes de migrar ou modernizar para microsserviço.
2. Estudo descreve algum desafio ou recomendação para refatorar o sistema monolítico antes de migrar ou modernizar para microsserviços.
3. Estudo mostra condições de degradação, falhas ou defeitos antes de migrar ou modernizar de monolítico para microsserviços.
4. Estudo apresenta ferramentas ou frameworks que auxiliam ou monitoram defeitos no projeto do sistema monolíticos.

Exclusion Criteria describe the following reasons to discard studies:

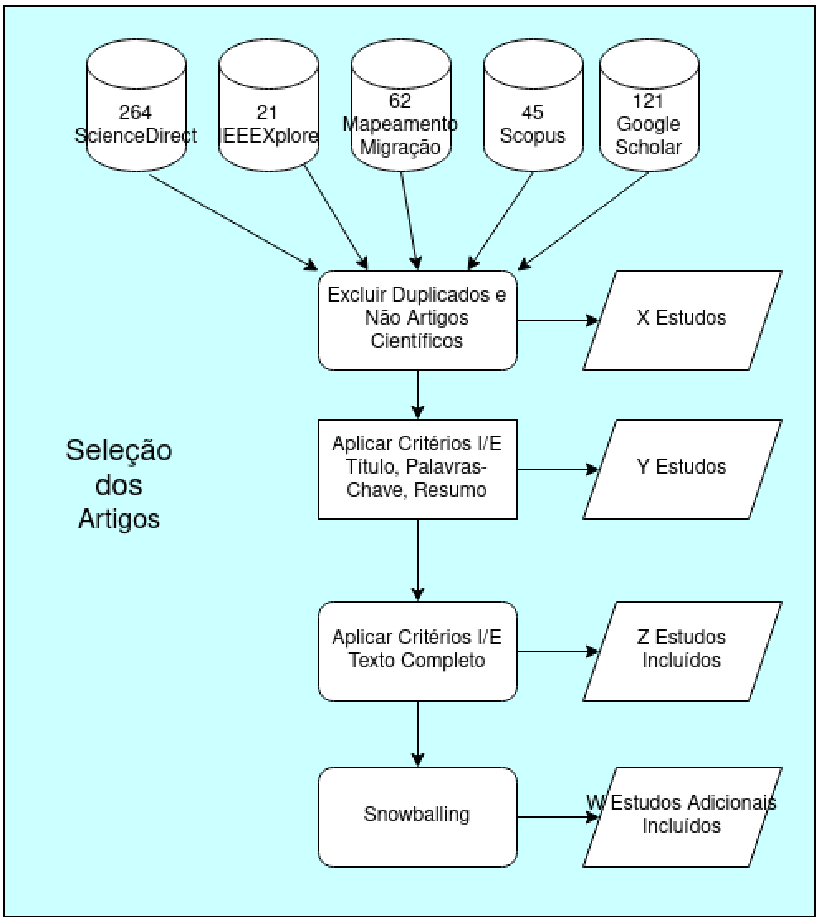
1. Duplicated studies. *Whenever a certain study has been published in different publications, the most recent and/or complete version will be used, unless they are complementary information.*
2. *O estudo não é um artigo de pesquisa científica ou é um artigo de conferência.*
3. *Estudo não está escrito em inglês.*
4. *Versões anteriores de trabalhos selecionados*

6 Primary Study selection process

The selection process for primary studies is performed by a joint effort of two researchers; one of them is M.Sc. Student and the other is a Doctor and advisor.

It is based on automated search in a digital database, then the reading of titles, abstracts, keywords. After, the included studies are read the introduction and conclusion. Finally, the complete study is read.

As papers are read, exclusion/inclusion criteria must be obey and, if any are achieved, it must be rejected but its details are kept in a reference repository system (e.g. JabRef) containing information to further analysis and research, when necessary.


264 ScienceDirect
21 IEEXplore
62 Mapeamento Migração
45 Scopus
121 Google Scholar
Excluir Duplicados e Não Artigos Científicos
X Estudos
Seleção dos Artigos
Aplicar Critérios I/E Título, Palavras-Chave, Resumo
Y Estudos
Aplicar Critérios I/E Texto Completo
Z Estudos Incluídos
Snowballing
W Estudos Adicionais Incluídos

7 Reliability of inclusion decisions

Whenever a paper is analyzed by more than one researcher, it is necessary to evaluate their viewpoints, in order to have a consensus on the topic. A faithfully agreement level must exist. If the researchers do not find a common point of view after discussion, the advisor will give appropriate directions. Finally, the authors can be also contacted.

8 Study Quality Assessment

The purpose of quality criteria is to evaluate the primary studies, as a means of weighting the importance of individual studies and guiding their understanding.

The following primary studies can be viewed in searches as approaches and surveys. Hence, we define common and specific quality criteria for all of the studies types. Table 4 shows the quality criteria for approaches associated to Testing in Software Product Line, and Table 5 presents a quality criteria for surveys found in that area. These surveys correspond to related work with this review.

The scope presented on Table 4 and Table 5 represent how the quality criteria should be applied to each study.

Table 4. Quality Criteria of Software Refactoring for migration to microservices.

ID	QUALITY CRITERIA	OPTIONS	SCORE
QC1	**Are there any refactoring strategies described?**	**Yes.** They are explicitly defined in the study.	1
		Partly. They are implicit.	0,5
		No. They are not defined and cannot be readily inferred.	0
QC2	**Are there any references to refactoring for migration to microservice described?**	**Yes.** They are explicitly defined in the study.	1
		Partly. They are implicit	0,5
		No. They cannot be found in study activities.	0
QC3	**Are there any references to bad smells in microservices described?**	**Yes.** They are explicitly defined in the study.	1
		Partly. They are implicit.	0,5
		No. They are not defined and cannot be readily inferred.	0
QC4	**Are there any references to design problems ?**	**Yes.** They are explicitly defined in the study.	1
		Partly. They are implicit.	0,5
		No. They are not defined and cannot be readily inferred.	0
QC5	**Are there any references to defects of software (code smells in monolithic) ?**	**Yes.** They are explicitly defined in the study	1
		Partly. They are implicit.	0,5
		No. They are not defined and cannot be readily inferred.	0

9 Data Collection

The data extraction forms must be designed to collect all the information needed to address the review questions and the study quality criteria. The following will be extracted from each study:

- The study's title and authors.
- The source (conference, journal, and so on.).
- The year when the study was published. In case of study was published in different sources, the most relevant will be used in any analysis although both dates will be recorded.
- Classification (related work or approach).
- Scope (Refactoring strategies, Modularization Techniques).
- The answers for research questions addressed by the publication.
- Summary (a summary with brief analysis, overview of its weaknesses and strengths).
- Quality criteria score.

The main data extraction effort will be performed by two researchers. Reviewers will check the work, as pointed in the previous section (Section 7).

Supplementary publications to a study should be grouped. Each study will be documented in a form pattern. It will also be documented the reviewer's name and date of the review.

10 Data Analysis and Synthesis

The primary studies will be categorized since there are many techniques and methods applicable to different scenarios. After analyzing the primary studies, they will be grouped in categories to facilitate the tabulation and further comparative analysis. The studies containing any exclusion criteria will also be tabulated, where it will be described the exclusion reason.

According to the comparisons and study analysis, the following criteria will be raised:

- The strategies strengths and weaknesses;
- Limitations and trends in refactoring (remodularization) of monolithic systems before migrating to microservice architecture;
- Challenges in the area to be addressed;
- Best practices (recommendations) of refactoring (remodularization) of monolithic systems before migrating to microservice architecture.

11 Dissemination

The results of the study can be of interest for the software engineering community as well as researchers interested in migrating from monolithic to a microservice architecture or, in some cases, any other form of modernization. It will also be documented that the study accomplished results in a technical report format in order to ensure that readers are able to properly evaluate the rigor and validity of the review.

12 References

[1] I. Ahmed, U. A. Mannan, R. Gopinath and C. Jensen, "An Empirical Study of Design Degradation: How Software Projects Get Worse over Time," *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Beijing, China, 2015, pp. 1-10, doi: 10.1109/ESEM.2015.7321186.

[2] Ganesh, Samarthyam & Suryanarayana, Girish & Sharma, Tushar. (2016). Refactoring for software architecture smells. 1-4. 10.1145/2975945.2975946.

[3] D. Taibi and V. Lenarduzzi, "On the Definition of Microservice Bad Smells," in IEEE Software, vol. 35, no. 3, pp. 56-62, May/June 2018, doi: 10.1109/MS.2018.2141031.

[4] Taibi, Davide & Lenarduzzi, Valentina & Pahl, Claus. (2019). Microservices Anti-Patterns: A Taxonomy.

[5] Ivan Candela, Gabriele Bavota, Barbara Russo, and Rocco Oliveto. 2016. Using Cohesion and Coupling for Software Remodularization: Is It Enough? ACM Trans. Softw. Eng. Methodol. 25, 3, Article 24 (August 2016), 28 pages. DOI:https://doi.org/10.1145/2928268

[6] Jambunathan, Baskaran & Y., Kalpana. (2016). Microservice Design for Container based Multi-cloud Deployment. International Journal of Engineering and Technology (IJET). 8.

[7] D. S. Linthicum, "Practical Use of Microservices in Moving Workloads to the Cloud," in IEEE Cloud Computing, vol. 3, no. 5, pp. 6-9, Sept.-Oct. 2016, doi: 10.1109/MCC.2016.114.

[8] Isela Macia, Joshua Garcia, Daniel Popescu, Alessandro Garcia, Nenad Medvidovic, and Arndt von Staa. 2012. Are automatically-detected code anomalies relevant to

architectural modularity? an exploratory analysis of evolving systems. In <i>Proceedings of the 11th annual international conference on Aspect-oriented Software Development</i> (<i>AOSD '12</i>). Association for Computing Machinery, New York, NY, USA, 167–178. DOI:https://doi.org/10.1145/2162049.2162069

[9] Shanshan Li, He Zhang, Zijia Jia, Chenxing Zhong, Cheng Zhang, Zhihao Shan, Jinfeng Shen, Muhammad Ali Babar, Understanding and addressing quality attributes of microservices architecture: A Systematic literature review, Information and Software Technology, Volume 131, 2021, 106449, ISSN 0950-5849, https://doi.org/10.1016/j.infsof.2020.106449

[10] Kitchenham, B. (2007). "Guidelines for performing Systematic Literature Reviews in Software Engineering ," V 2.3 EBSE Technical Report, EBSE-2007-01.

[11] Huang, X., Lin, J., & Demner-Fushman, D. (2006). Evaluation of PICO as a knowledge representation for clinical questions. *AMIA ... Annual Symposium proceedings. AMIA Symposium*, *2006*, 359–363.

[12] Teixeira, E. N., Aleixo, F. A., de Sousa Amâncio, F. D., OliveiraJr, E., Kulesza, U., & Werner, C. (2019). Software Process Line as an Approach to Support Software Process Reuse: a Systematic Literature Review. Information and Software Technology. doi:10.1016/j.infsof.2019.08.007

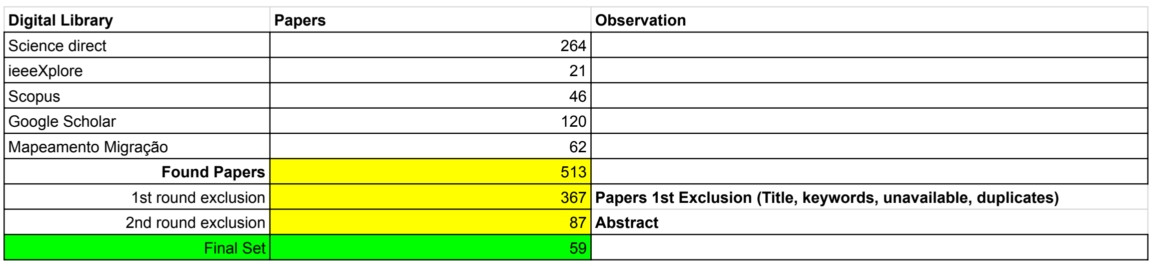
[13] Refactoring Improving the Design of Existing Code. Disponível em: <https://martinfowler.com/books/refactoring.html >. Acesso em 20 mar.2021

[14] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1995. <i>Design patterns: elements of reusable object-oriented software</i>. Addison-Wesley Longman Publishing Co., Inc., USA.

[15] Martin, Robert C. "Design principles and design patterns." *Object Mentor* 1.34 (2000): 597.

[16] The Reactive Manifesto. Disponível em: <https://www.reactivemanifesto.org/pt-BR>. Acesso em 20 mar.2021

.1.2 SLM Paper Count



Digital Library	Papers	Observation
Science direct	264	
ieeeXplore	21	
Scopus	46	
Google Scholar	120	
Mapeamento Migração	62	
Found Papers	513	
1st round exclusion	367	**Papers 1st Exclusion (Title, keywords, unavailable, duplicates)**
2nd round exclusion	87	**Abstract**
Final Set	59	

.2 Rapid Review

PROTOCOL - RAPID REVIEW

Research Topic: Refactoring Monolithic System Towards Migrations to Microservices

Authors: Guilherme Luciano Donin Villaca / Ivonei Freitas da Silva

Review History

Date	Review	Author	Role	Description
30/10/2021	1	Guilherme / Ivonei	Main researchers	Initial draft
08/11/2021	2	Guilherme	Main researchers	Revisão GQM da RR
08/11/2021	3	Guilherme	Main researchers	Revisão Final

1 Team

MEMBER	ACADEMIC DEGREE	RESPONSIBILITY	ROLE
Guilherme Luciano Donin Villaca	M.Sc. Candidate, University of the State of Paraná.	Develop protocol. Select studies. Collect, analyze and synthesize data. Assess studies quality. Write report.	Main Researcher
Ivonei Freitas da Silva	Ph.D., Federal University of Pernambuco.	Review report.	Advisor

Resumo do tópico para revisão rápida

… por aqui o mesmo resumo do mapping, só que tratando de GL. Talvez o parágrafo abaixo sirva para indicar que será uma rapid review também. Mas quando passar isso para a dissertação, decidimos se colocamos ou não.

Exemplo: This Rapid Review (RR) [1] on GL aims to analyze <<facet>> to characterize it in the IoT field, regarding what, how, where, when and why is used in the context of IoT projects, verifying the existence of published studies supporting the previous results. The 5W1H aims to give the observational perspective on which information is required to the understanding and management of the facet in a system (what); to the software technologies (techniques, technologies, methods, and solutions) defining their operationalization (how); the activities location being geographically distributed or something external to the software system (where); the roles involved to deal with the facet development (who); the effects of time over the facet, describing its transformations and states (when); and to translate the motivation, goals, and strategies going to what is implemented in the facet (why), in respect of IoT projects.

1. Necessidade para focar na revisão da literatura cinza

aplicar a tabela 2 da [referencia do artigo de GL] , original está no trabalho do garousi. Acho que os Yes/No são iguais ao do trabalho sobre DevSecOps.

JUSTIFICATIVA PARA USAR GLR

ver introdução do artigo “Grey Literature in Software Engineering: A critical review” do Fernando Kamei et al. publicado no IST, 2021

2 - GQM

Ver/Colocar refs de GQM

Goal	Purpose	To understand	
	Object	Approaches of modularizing monolithic systems before migrating to microservice systems	
	Issue	with respect to the effectiveness in addressing microservice bad smells	
	Viewpoint	from the practitioners’ perspective	
Research Questions	Q1	What are the approaches made in monolithic systems before migrating to microservice systems from the viewpoint of the practitioners?	
	Q2	How approaches made in monolithic systems before migrating to microservice systems work from the viewpoint of the practitioners?	
Metrics	Q1	M1	List of approaches
		M2	List of categories of approaches
		M3	Structure of categories and approaches
	Q2	M1	Steps
		M2	tools
		M3	techniques
		M4	monolith bad smells adopted catalog
		M5	monolith refactoring adopted catalog
		M6	Limitações

*** PARA AS MÉTRICAS DE Q2, marcar/pontuar se estão ligadas à microservice bad smells????**

3. Especificar as Rqs (research questions)

Há experiências na GL de times, projetos ou organizações que descreve o processo de correção/reengenharia/refatoração do sistema monolito antes de migrar para microsserviços?

Se sim:
I) No relato consideraram microservice bad smells? Se sim, como foi conduzido?
II) No relato consideraram catálogos de refactoring ou bad smells de monolito? Se sim, como foi?
III) Quais os desafios, benefícios reportados?

Se não:
III) Quais os desafios, benefícios, lições aprendidas reportados sobre correção/reengenharia/refatoração do monolito antes de migrar para arquitetura de microsserviço?

3. Definir o protocolo

Base de busca Piloto: google e google scholar [referencia do artigo de GL]

String de busca: Mesma do mapping trouxe respostas similares ao mapeamento, portanto foi utilizada uma string mais direta para o google:

“migration from monolithic to microservice”

Critério de Parada:
- somente as 5 primeiras páginas do google
- sem snowballing nos links dos artigos, videos, blog posts

Inclusão

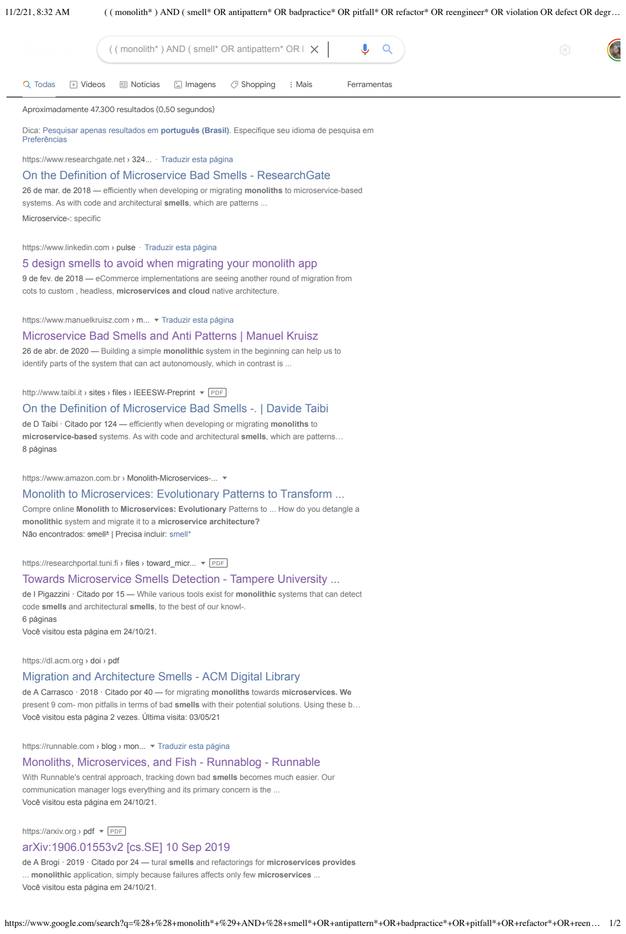
1	blog posts, video, white paper
2	(experiência prática) migração de monolito para microsserviço
3	Dicas, sugestões, passos (steps)
4	falhas em MSA que estejam interligadas com smells (que podem ser mitigadas ainda no monolito) identificadas no mapping

Exclusão

Tópico:

1	criar microsserviços do zero, ou seja, não migrou do monolito.
2	SOA, outras tecnologias de serviços.
3	apenas opinião teórica, sem experiência prática; aponte somente diferenças entre monolito e microsserviços com comparações, vantagens/desvantagens etc
4	Somente conteúdo para recomendação de product marketing infomation (se algum artigo for incluído tendo esse product marketing information, marcá-lo como “more likelihood of bias” na avaliação de qualidade.)
5	sem snowballing nos links dos artigos, videos, blog posts
6	Dicas, sugestões, experiência de forma breve/rasa;
7	Vídeo que não tenha transcrição
8	Link Indisponível / requer assinatura
9	Cópia de outro white paper sem dar créditos

.2.1 Rapid Review Search Results



11/2/21, 8:32 AM ((monolith*) AND (smell* OR antipattern* OR badpractice* OR pitfall* OR refactor* OR reengineer* OR violation OR defect OR degr…

Aproximadamente 47.300 resultados (0,50 segundos)

Dica: Pesquisar apenas resultados em **português (Brasil)**. Especifique seu idioma de pesquisa em Preferências

https://www.researchgate.net › 324... · Traduzir esta página
On the Definition of Microservice Bad Smells - ResearchGate
26 de mar. de 2018 — efficiently when developing or migrating **monoliths** to microservice-based systems. As with code and architectural **smells**, which are patterns ...
Microservice-: specific

https://www.linkedin.com › pulse · Traduzir esta página
5 design smells to avoid when migrating your monolith app
9 de fev. de 2018 — eCommerce implementations are seeing another round of migration from cots to custom , headless, **microservices and cloud** native architecture.

https://www.manuelkruisz.com › m... Traduzir esta página
Microservice Bad Smells and Anti Patterns | Manuel Kruisz
26 de abr. de 2020 — Building a simple **monolithic** system in the beginning can help us to identify parts of the system that can act autonomously, which in contrast is ...

http://www.taibi.it › sites › files › IEEESW-Preprint PDF
On the Definition of Microservice Bad Smells -. | Davide Taibi
de D Taibi · Citado por 124 — efficiently when developing or migrating **monoliths** to **microservice-based** systems. As with code and architectural **smells**, which are patterns…
8 páginas

https://www.amazon.com.br › Monolith-Microservices-...
Monolith to Microservices: Evolutionary Patterns to Transform ...
Compre online **Monolith** to **Microservices: Evolutionary** Patterns to ... How do you detangle a **monolithic** system and migrate it to a **microservice architecture?**
Não encontrados: ~~smell*~~ | Precisa incluir: smell*

https://researchportal.tuni.fi › files › toward_micr... PDF
Towards Microservice Smells Detection - Tampere University ...
de I Pigazzini · Citado por 15 — While various tools exist for **monolithic** systems that can detect code **smells** and architectural **smells**, to the best of our knowl-.
6 páginas
Você visitou esta página em 24/10/21.

https://dl.acm.org › doi › pdf
Migration and Architecture Smells - ACM Digital Library
de A Carrasco · 2018 · Citado por 40 — for migrating **monoliths** towards **microservices. We** present 9 com- mon pitfalls in terms of bad **smells** with their potential solutions. Using these b…
Você visitou esta página 2 vezes. Última visita: 03/05/21

https://runnable.com › blog › mon... Traduzir esta página
Monoliths, Microservices, and Fish - Runnablog - Runnable
With Runnable's central approach, tracking down bad **smells** becomes much easier. Our communication manager logs everything and its primary concern is the ...
Você visitou esta página em 24/10/21.

https://arxiv.org › pdf PDF
arXiv:1906.01553v2 [cs.SE] 10 Sep 2019
de A Brogi · 2019 · Citado por 24 — tural **smells** and refactorings for **microservices provides** ... **monolithic** application, simply because failures affects only few **microservices** ...
Você visitou esta página em 24/10/21.

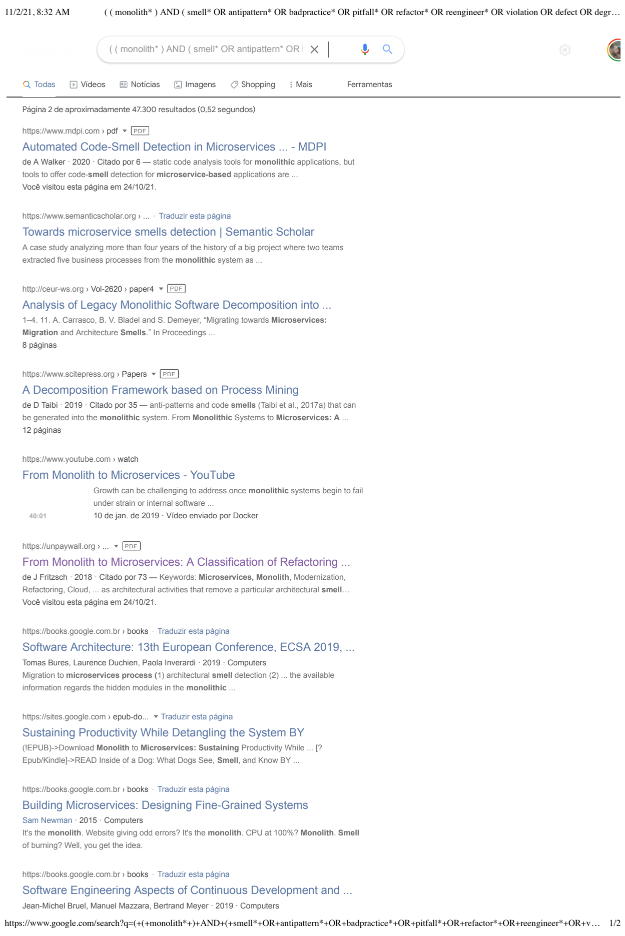
https://www.google.com/search?q=%28+%28+monolith*+%29+AND+%28+smell*+OR+antipattern*+OR+badpractice*+OR+pitfall*+OR+refactor*+OR+reen… 1/2

11/2/21, 8:32 AM ((monolith*) AND (smell* OR antipattern* OR badpractice* OR pitfall* OR refactor* OR reengineer* OR violation OR defect OR degr…

https://www.infoq.com › podcasts Traduzir esta página
Sam Newman: Monolith to Microservices - InfoQ
25 de mai. de 2020 — Problems That Require a **Microservices Approach**. Reisz: Decompose that a bit. What are the **smells**? You're running a **monolith** and you start to ...
Você visitou esta página em 24/10/21.

Brasil | **85806-264 - Esmeralda, Cascavel - PR** - Com base nos seus lugares (Casa) - Atualizar local

https://www.google.com/search?q=%28+%28+monolith*+%29+AND+%28+smell*+OR+antipattern*+OR+badpractice*+OR+pitfall*+OR+refactor*+OR+reen… 2/2



11/2/21, 8:32 AM ((monolith*) AND (smell* OR antipattern* OR badpractice* OR pitfall* OR refactor* OR reengineer* OR violation OR defect OR degr…

((monolith*) AND (smell* OR antipattern* OR I

Todas · Vídeos · Notícias · Imagens · Shopping · Mais · Ferramentas

Página 2 de aproximadamente 47.300 resultados (0,52 segundos)

https://www.mdpi.com › pdf PDF
Automated Code-Smell Detection in Microservices ... - MDPI
de A Walker · 2020 · Citado por 6 — static code analysis tools for **monolithic** applications, but tools to offer code-**smell** detection for **microservice-based** applications are ...
Você visitou esta página em 24/10/21.

https://www.semanticscholar.org › ... · Traduzir esta página
Towards microservice smells detection | Semantic Scholar
A case study analyzing more than four years of the history of a big project where two teams extracted five business processes from the **monolithic** system as ...

http://ceur-ws.org › Vol-2620 › paper4 PDF
Analysis of Legacy Monolithic Software Decomposition into ...
1–4. 11. A. Carrasco, B. V. Bladel and S. Demeyer, "Migrating towards **Microservices: Migration** and Architecture **Smells**." In Proceedings ...
8 páginas

https://www.scitepress.org › Papers PDF
A Decomposition Framework based on Process Mining
de D Taibi · 2019 · Citado por 35 — anti-patterns and code **smells** (Taibi et al., 2017a) that can be generated into the **monolithic** system. From **Monolithic** Systems to **Microservices: A** ...
12 páginas

https://www.youtube.com › watch
From Monolith to Microservices - YouTube
40:01
Growth can be challenging to address once **monolithic** systems begin to fail under strain or internal software ...
10 de jan. de 2019 · Vídeo enviado por Docker

https://unpaywall.org › ... PDF
From Monolith to Microservices: A Classification of Refactoring ...
de J Fritzsch · 2018 · Citado por 73 — Keywords: **Microservices, Monolith**, Modernization, Refactoring, Cloud, ... as architectural activities that remove a particular architectural **smell**...
Você visitou esta página em 24/10/21.

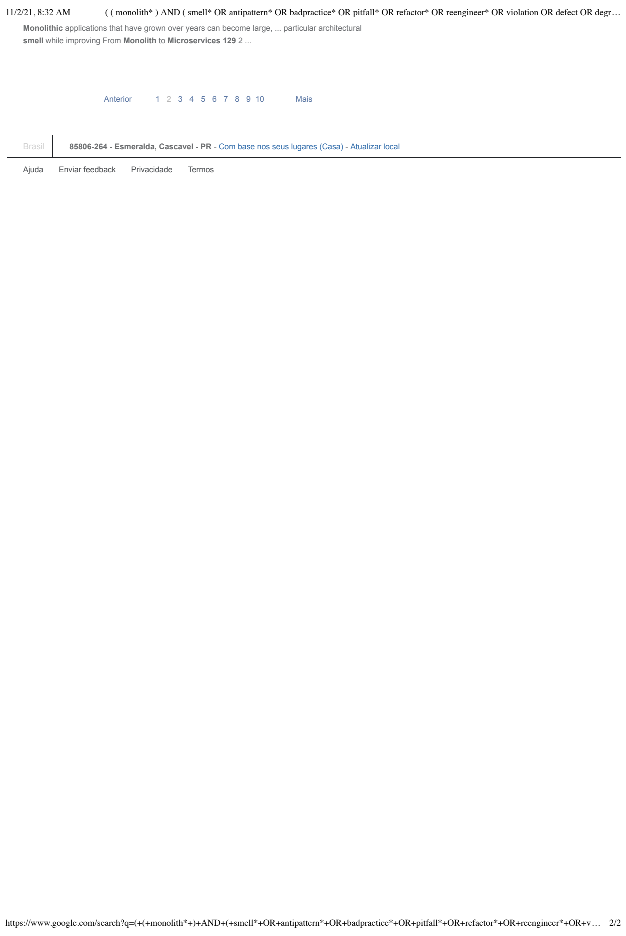
https://books.google.com.br › books · Traduzir esta página
Software Architecture: 13th European Conference, ECSA 2019, ...
Tomas Bures, Laurence Duchien, Paola Inverardi · 2019 · Computers
Migration to **microservices process (**1) architectural **smell** detection (2) ... the available information regards the hidden modules in the **monolithic** ...

https://sites.google.com › epub-do... Traduzir esta página
Sustaining Productivity While Detangling the System BY
(!EPUB)->Download **Monolith** to **Microservices: Sustaining** Productivity While ... [?Epub/Kindle]->READ Inside of a Dog: What Dogs See, **Smell**, and Know BY ...

https://books.google.com.br › books · Traduzir esta página
Building Microservices: Designing Fine-Grained Systems
Sam Newman · 2015 · Computers
It's the **monolith**. Website giving odd errors? It's the **monolith**. CPU at 100%? **Monolith**. **Smell** of burning? Well, you get the idea.

https://books.google.com.br › books · Traduzir esta página
Software Engineering Aspects of Continuous Development and ...
Jean-Michel Bruel, Manuel Mazzara, Bertrand Meyer · 2019 · Computers

https://www.google.com/search?q=(+(+monolith*+)+AND+(+smell*+OR+antipattern*+OR+badpractice*+OR+pitfall*+OR+refactor*+OR+reengineer*+OR+v… 1/2



11/2/21, 8:32 AM ((monolith*) AND (smell* OR antipattern* OR badpractice* OR pitfall* OR refactor* OR reengineer* OR violation OR defect OR degr…

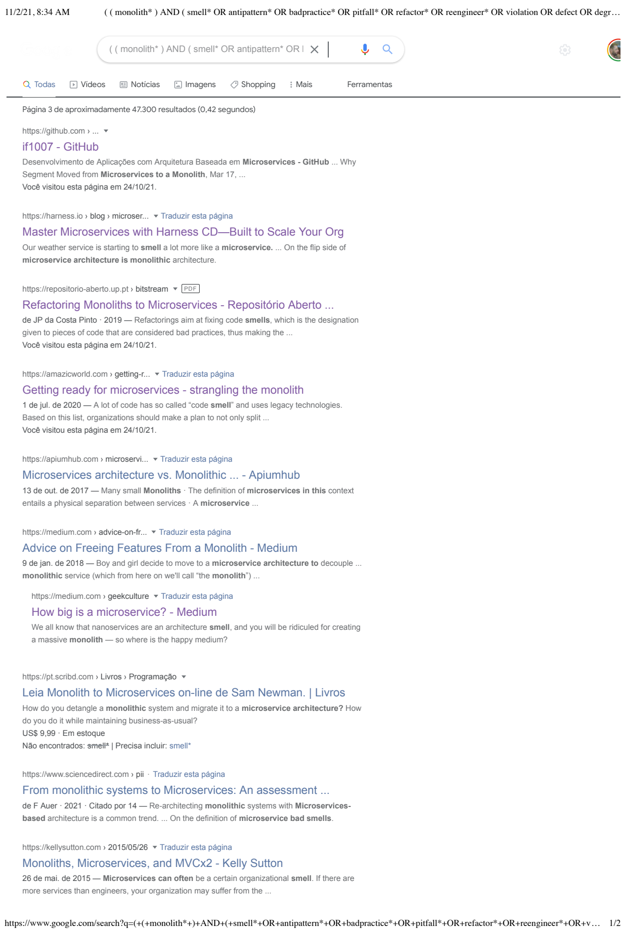
Monolithic applications that have grown over years can become large, ... particular architectural **smell** while improving From **Monolith** to **Microservices 129** 2 ...

Anterior 1 2 3 4 5 6 7 8 9 10 Mais

Brasil | **85806-264 - Esmeralda, Cascavel - PR** - Com base nos seus lugares (Casa) - Atualizar local

Ajuda Enviar feedback Privacidade Termos

https://www.google.com/search?q=(+(+monolith*+)+AND+(+smell*+OR+antipattern*+OR+badpractice*+OR+pitfall*+OR+refactor*+OR+reengineer*+OR+v… 2/2



11/2/21, 8:34 AM ((monolith*) AND (smell* OR antipattern* OR badpractice* OR pitfall* OR refactor* OR reengineer* OR violation OR defect OR degr…

((monolith*) AND (smell* OR antipattern* OR I

Todas Vídeos Notícias Imagens Shopping Mais Ferramentas

Página 3 de aproximadamente 47.300 resultados (0,42 segundos)

https://github.com › ...
if1007 - GitHub
Desenvolvimento de Aplicações com Arquitetura Baseada em **Microservices - GitHub** ... Why Segment Moved from **Microservices to a Monolith**, Mar 17, ...
Você visitou esta página em 24/10/21.

https://harness.io › blog › microser... Traduzir esta página
Master Microservices with Harness CD—Built to Scale Your Org
Our weather service is starting to **smell** a lot more like a **microservice.** ... On the flip side of **microservice architecture is monolithic** architecture.

https://repositorio-aberto.up.pt › bitstream PDF
Refactoring Monoliths to Microservices - Repositório Aberto ...
de JP da Costa Pinto · 2019 — Refactorings aim at fixing code **smells**, which is the designation given to pieces of code that are considered bad practices, thus making the ...
Você visitou esta página em 24/10/21.

https://amazicworld.com › getting-r... Traduzir esta página
Getting ready for microservices - strangling the monolith
1 de jul. de 2020 — A lot of code has so called "code **smell**" and uses legacy technologies. Based on this list, organizations should make a plan to not only split ...
Você visitou esta página em 24/10/21.

https://apiumhub.com › microservi... Traduzir esta página
Microservices architecture vs. Monolithic ... - Apiumhub
13 de out. de 2017 — Many small **Monoliths** · The definition of **microservices in this** context entails a physical separation between services · A **microservice** ...

https://medium.com › advice-on-fr... Traduzir esta página
Advice on Freeing Features From a Monolith - Medium
9 de jan. de 2018 — Boy and girl decide to move to a **microservice architecture to** decouple ... **monolithic** service (which from here on we'll call "the **monolith**") ...

https://medium.com › geekculture Traduzir esta página
How big is a microservice? - Medium
We all know that nanoservices are an architecture **smell**, and you will be ridiculed for creating a massive **monolith** — so where is the happy medium?

https://pt.scribd.com › Livros › Programação
Leia Monolith to Microservices on-line de Sam Newman. | Livros
How do you detangle a **monolithic** system and migrate it to a **microservice architecture?** How do you do it while maintaining business-as-usual?
US$ 9,99 · Em estoque
Não encontrados: ~~smell*~~ | Precisa incluir: smell*

https://www.sciencedirect.com › pii · Traduzir esta página
From monolithic systems to Microservices: An assessment ...
de F Auer · 2021 · Citado por 14 — Re-architecting **monolithic** systems with **Microservices-based** architecture is a common trend. ... On the definition of **microservice bad smells**.

https://kellysutton.com › 2015/05/26 Traduzir esta página
Monoliths, Microservices, and MVCx2 - Kelly Sutton
26 de mai. de 2015 — **Microservices can often** be a certain organizational **smell**. If there are more services than engineers, your organization may suffer from the ...

https://www.google.com/search?q=(+(+monolith*+)+AND+(+smell*+OR+antipattern*+OR+badpractice*+OR+pitfall*+OR+refactor*+OR+reengineer*+OR+v… 1/2

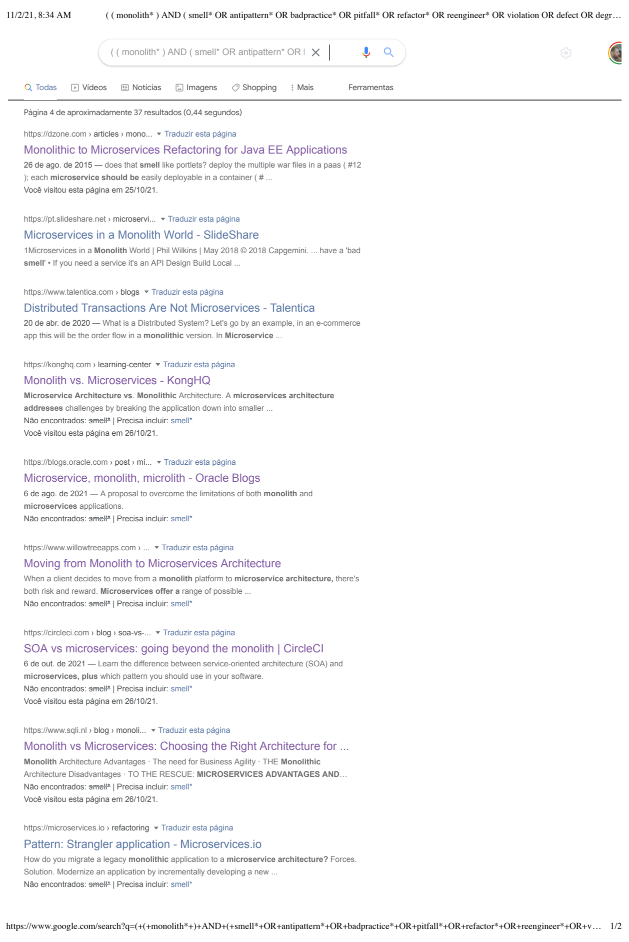
11/2/21, 8:34 AM ((monolith*) AND (smell* OR antipattern* OR badpractice* OR pitfall* OR refactor* OR reengineer* OR violation OR defect OR degr…

Anterior 1 2 3 4 5 6 7 8 9 10 Mais

Brasil | **85806-264 - Esmeralda, Cascavel - PR** - Com base nos seus lugares (Casa) - Atualizar local

Ajuda Enviar feedback Privacidade Termos

https://www.google.com/search?q=(+(+monolith*+)+AND+(+smell*+OR+antipattern*+OR+badpractice*+OR+pitfall*+OR+refactor*+OR+reengineer*+OR+v… 2/2



11/2/21, 8:34 AM ((monolith*) AND (smell* OR antipattern* OR badpractice* OR pitfall* OR refactor* OR reengineer* OR violation OR defect OR degr…

((monolith*) AND (smell* OR antipattern* OR I

Todas Vídeos Notícias Imagens Shopping Mais Ferramentas

Página 4 de aproximadamente 37 resultados (0,44 segundos)

https://dzone.com › articles › mono... Traduzir esta página
Monolithic to Microservices Refactoring for Java EE Applications
26 de ago. de 2015 — does that **smell** like portlets? deploy the multiple war files in a paas (#12); each **microservice should be** easily deployable in a container (# ...
Você visitou esta página em 25/10/21.

https://pt.slideshare.net › microservi... Traduzir esta página
Microservices in a Monolith World - SlideShare
1Microservices in a **Monolith** World | Phil Wilkins | May 2018 © 2018 Capgemini. ... have a 'bad **smell**' • If you need a service it's an API Design Build Local ...

https://www.talentica.com › blogs Traduzir esta página
Distributed Transactions Are Not Microservices - Talentica
20 de abr. de 2020 — What is a Distributed System? Let's go by an example, in an e-commerce app this will be the order flow in a **monolithic** version. In **Microservice** ...

https://konghq.com › learning-center Traduzir esta página
Monolith vs. Microservices - KongHQ
Microservice Architecture vs. **Monolithic** Architecture. A **microservices architecture addresses** challenges by breaking the application down into smaller ...
Não encontrados: ~~smell*~~ | Precisa incluir: smell*
Você visitou esta página em 26/10/21.

https://blogs.oracle.com › post › mi... Traduzir esta página
Microservice, monolith, microlith - Oracle Blogs
6 de ago. de 2021 — A proposal to overcome the limitations of both **monolith** and **microservices** applications.
Não encontrados: ~~smell*~~ | Precisa incluir: smell*

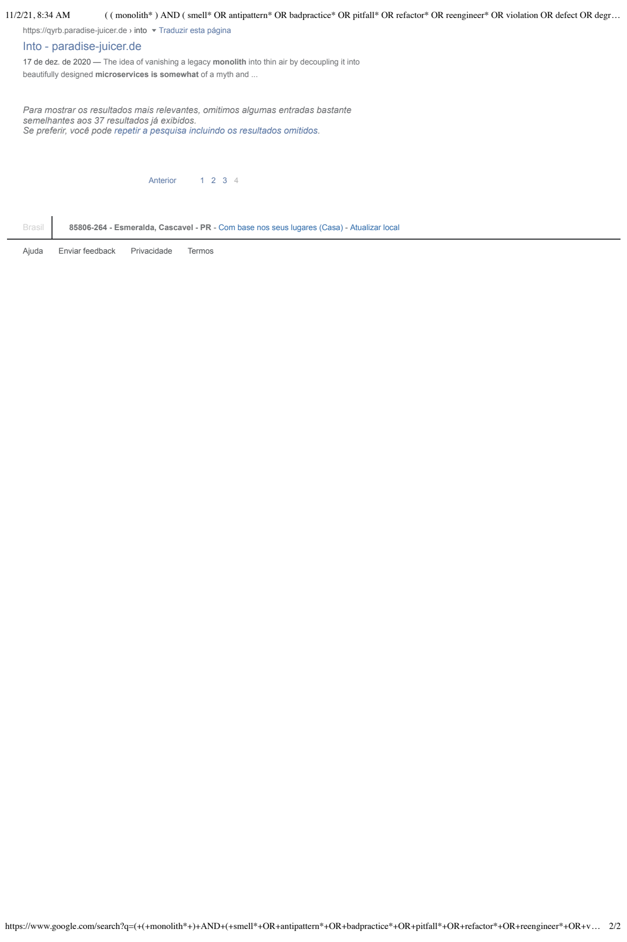
https://www.willowtreeapps.com › ... Traduzir esta página
Moving from Monolith to Microservices Architecture
When a client decides to move from a **monolith** platform to **microservice architecture,** there's both risk and reward. **Microservices offer a** range of possible ...
Não encontrados: ~~smell*~~ | Precisa incluir: smell*

https://circleci.com › blog › soa-vs-... Traduzir esta página
SOA vs microservices: going beyond the monolith | CircleCI
6 de out. de 2021 — Learn the difference between service-oriented architecture (SOA) and **microservices, plus** which pattern you should use in your software.
Não encontrados: ~~smell*~~ | Precisa incluir: smell*
Você visitou esta página em 26/10/21.

https://www.sqli.nl › blog › monoli... Traduzir esta página
Monolith vs Microservices: Choosing the Right Architecture for ...
Monolith Architecture Advantages · The need for Business Agility · THE **Monolithic** Architecture Disadvantages · TO THE RESCUE: **MICROSERVICES ADVANTAGES AND**…
Não encontrados: ~~smell*~~ | Precisa incluir: smell*
Você visitou esta página em 26/10/21.

https://microservices.io › refactoring Traduzir esta página
Pattern: Strangler application - Microservices.io
How do you migrate a legacy **monolithic** application to a **microservice architecture?** Forces. Solution. Modernize an application by incrementally developing a new ...
Não encontrados: ~~smell*~~ | Precisa incluir: smell*

https://www.google.com/search?q=(+(+monolith*+)+AND+(+smell*+OR+antipattern*+OR+badpractice*+OR+pitfall*+OR+refactor*+OR+reengineer*+OR+v… 1/2



11/2/21, 8:34 AM ((monolith*) AND (smell* OR antipattern* OR badpractice* OR pitfall* OR refactor* OR reengineer* OR violation OR defect OR degr…

https://qyrb.paradise-juicer.de › into Traduzir esta página
Into - paradise-juicer.de
17 de dez. de 2020 — The idea of vanishing a legacy **monolith** into thin air by decoupling it into beautifully designed **microservices is somewhat** of a myth and ...

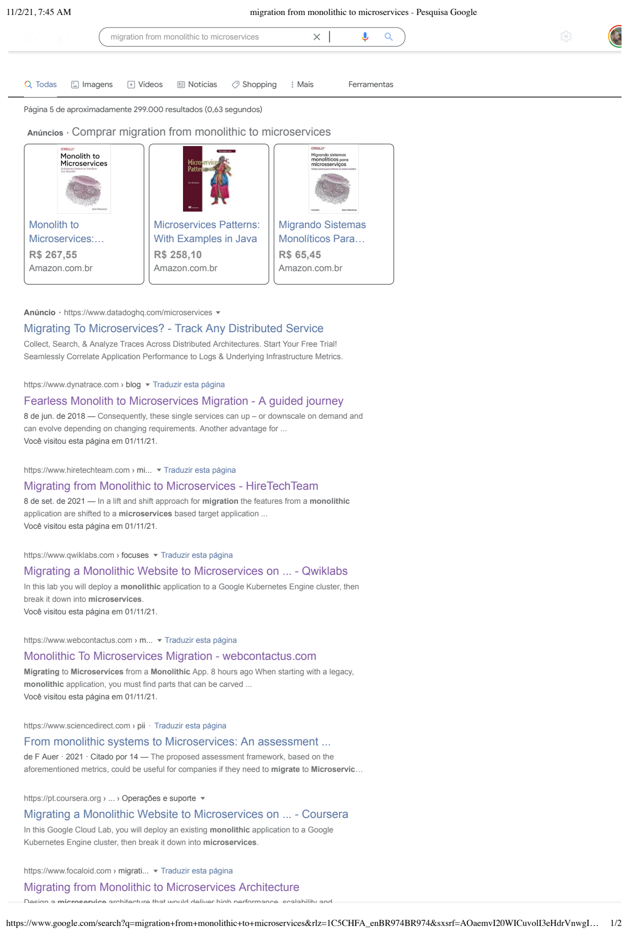
Para mostrar os resultados mais relevantes, omitimos algumas entradas bastante semelhantes aos 37 resultados já exibidos.
Se preferir, você pode repetir a pesquisa incluindo os resultados omitidos.

Anterior 1 2 3 4

Brasil | **85806-264 - Esmeralda, Cascavel - PR** - Com base nos seus lugares (Casa) - Atualizar local

Ajuda Enviar feedback Privacidade Termos

https://www.google.com/search?q=(+(+monolith*+)+AND+(+smell*+OR+antipattern*+OR+badpractice*+OR+pitfall*+OR+refactor*+OR+reengineer*+OR+v… 2/2



11/2/21, 7:45 AM migration from monolithic to microservices - Pesquisa Google

migration from monolithic to microservices

Todas · Imagens · Vídeos · Notícias · Shopping · Mais · Ferramentas

Página 5 de aproximadamente 299.000 resultados (0,63 segundos)

Anúncios · Comprar migration from monolithic to microservices

Monolith to Microservices:…	Microservices Patterns: With Examples in Java	Migrando Sistemas Monolíticos Para…
R$ 267,55	**R$ 258,10**	**R$ 65,45**
Amazon.com.br	Amazon.com.br	Amazon.com.br

Anúncio · https://www.datadoghq.com/microservices
Migrating To Microservices? - Track Any Distributed Service
Collect, Search, & Analyze Traces Across Distributed Architectures. Start Your Free Trial! Seamlessly Correlate Application Performance to Logs & Underlying Infrastructure Metrics.

https://www.dynatrace.com › blog · Traduzir esta página
Fearless Monolith to Microservices Migration - A guided journey
8 de jun. de 2018 — Consequently, these single services can up – or downscale on demand and can evolve depending on changing requirements. Another advantage for ...
Você visitou esta página em 01/11/21.

https://www.hiretechteam.com › mi... · Traduzir esta página
Migrating from Monolithic to Microservices - HireTechTeam
8 de set. de 2021 — In a lift and shift approach for **migration** the features from a **monolithic** application are shifted to a **microservices** based target application ...
Você visitou esta página em 01/11/21.

https://www.qwiklabs.com › focuses · Traduzir esta página
Migrating a Monolithic Website to Microservices on ... - Qwiklabs
In this lab you will deploy a **monolithic** application to a Google Kubernetes Engine cluster, then break it down into **microservices**.
Você visitou esta página em 01/11/21.

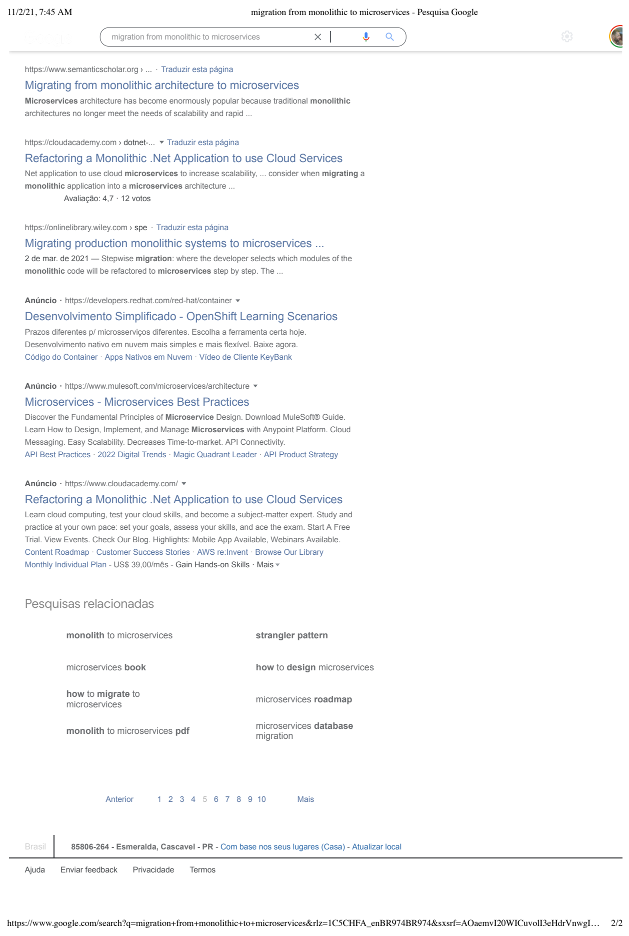
https://www.webcontactus.com › m... · Traduzir esta página
Monolithic To Microservices Migration - webcontactus.com
Migrating to **Microservices** from a **Monolithic** App. 8 hours ago When starting with a legacy, **monolithic** application, you must find parts that can be carved ...
Você visitou esta página em 01/11/21.

https://www.sciencedirect.com › pii · Traduzir esta página
From monolithic systems to Microservices: An assessment ...
de F Auer · 2021 · Citado por 14 — The proposed assessment framework, based on the aforementioned metrics, could be useful for companies if they need to **migrate** to **Microservic**…

https://pt.coursera.org › ... › Operações e suporte
Migrating a Monolithic Website to Microservices on ... - Coursera
In this Google Cloud Lab, you will deploy an existing **monolithic** application to a Google Kubernetes Engine cluster, then break it down into **microservices**.

https://www.focaloid.com › migrati... · Traduzir esta página
Migrating from Monolithic to Microservices Architecture
Design a **microservice** architecture that would deliver high performance, scalability and

https://www.google.com/search?q=migration+from+monolithic+to+microservices&rlz=1C5CHFA_enBR974BR974&sxsrf=AOaemvI20WICuvolI3eHdrVnwgI… 1/2



11/2/21, 7:45 AM migration from monolithic to microservices - Pesquisa Google

migration from monolithic to microservices

https://www.semanticscholar.org › ... · Traduzir esta página
Migrating from monolithic architecture to microservices
Microservices architecture has become enormously popular because traditional **monolithic** architectures no longer meet the needs of scalability and rapid ...

https://cloudacademy.com › dotnet-... · Traduzir esta página
Refactoring a Monolithic .Net Application to use Cloud Services
Net application to use cloud **microservices** to increase scalability, ... consider when **migrating** a **monolithic** application into a **microservices** architecture ...
Avaliação: 4,7 · 12 votos

https://onlinelibrary.wiley.com › spe · Traduzir esta página
Migrating production monolithic systems to microservices ...
2 de mar. de 2021 — Stepwise **migration**: where the developer selects which modules of the **monolithic** code will be refactored to **microservices** step by step. The ...

Anúncio · https://developers.redhat.com/red-hat/container
Desenvolvimento Simplificado - OpenShift Learning Scenarios
Prazos diferentes p/ microsserviços diferentes. Escolha a ferramenta certa hoje. Desenvolvimento nativo em nuvem mais simples e mais flexível. Baixe agora.
Código do Container · Apps Nativos em Nuvem · Vídeo de Cliente KeyBank

Anúncio · https://www.mulesoft.com/microservices/architecture
Microservices - Microservices Best Practices
Discover the Fundamental Principles of **Microservice** Design. Download MuleSoft® Guide. Learn How to Design, Implement, and Manage **Microservices** with Anypoint Platform. Cloud Messaging. Easy Scalability. Decreases Time-to-market. API Connectivity.
API Best Practices · 2022 Digital Trends · Magic Quadrant Leader · API Product Strategy

Anúncio · https://www.cloudacademy.com/
Refactoring a Monolithic .Net Application to use Cloud Services
Learn cloud computing, test your cloud skills, and become a subject-matter expert. Study and practice at your own pace: set your goals, assess your skills, and ace the exam. Start A Free Trial. View Events. Check Our Blog. Highlights: Mobile App Available, Webinars Available.
Content Roadmap · Customer Success Stories · AWS re:Invent · Browse Our Library
Monthly Individual Plan - US$ 39,00/mês - Gain Hands-on Skills · Mais

Pesquisas relacionadas

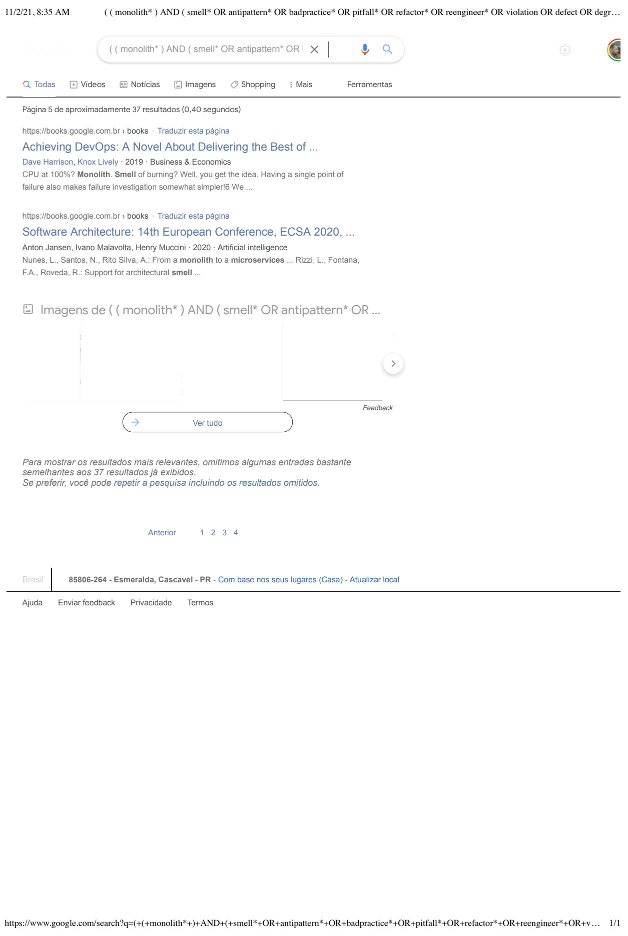
monolith to microservices	**strangler pattern**
microservices **book**	**how** to **design** microservices
how to **migrate** to microservices	microservices **roadmap**
monolith to microservices **pdf**	microservices **database** migration

Anterior 1 2 3 4 5 6 7 8 9 10 Mais

Brasil | **85806-264 - Esmeralda, Cascavel - PR** - Com base nos seus lugares (Casa) - Atualizar local

Ajuda · Enviar feedback · Privacidade · Termos

https://www.google.com/search?q=migration+from+monolithic+to+microservices&rlz=1C5CHFA_enBR974BR974&sxsrf=AOaemvI20WICuvolI3eHdrVnwgI… 2/2



11/2/21, 8:35 AM ((monolith*) AND (smell* OR antipattern* OR badpractice* OR pitfall* OR refactor* OR reengineer* OR violation OR defect OR degr…

Página 5 de aproximadamente 37 resultados (0,40 segundos)

https://books.google.com.br › books · Traduzir esta página
Achieving DevOps: A Novel About Delivering the Best of ...
Dave Harrison, Knox Lively · 2019 · Business & Economics
CPU at 100%? **Monolith**. **Smell** of burning? Well, you get the idea. Having a single point of failure also makes failure investigation somewhat simpler!6 We ...

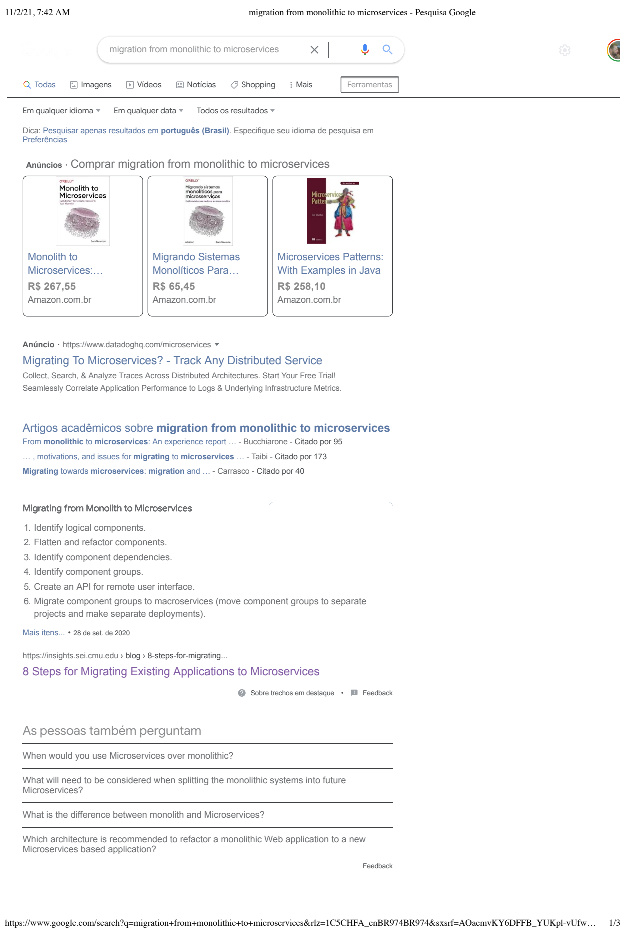
https://books.google.com.br › books · Traduzir esta página
Software Architecture: 14th European Conference, ECSA 2020, ...
Anton Jansen, Ivano Malavolta, Henry Muccini · 2020 · Artificial intelligence
Nunes, L., Santos, N., Rito Silva, A.: From a **monolith** to a **microservices** ... Rizzi, L., Fontana, F.A., Roveda, R.: Support for architectural **smell** ...

Imagens de ((monolith*) AND (smell* OR antipattern* OR ...

Para mostrar os resultados mais relevantes, omitimos algumas entradas bastante semelhantes aos 37 resultados já exibidos.
Se preferir, você pode repetir a pesquisa incluindo os resultados omitidos.

85806-264 - Esmeralda, Cascavel - PR - Com base nos seus lugares (Casa) - Atualizar local

https://www.google.com/search?q=(+(+monolith*+)+AND+(+smell*+OR+antipattern*+OR+badpractice*+OR+pitfall*+OR+refactor*+OR+reengineer*+OR+v… 1/1



11/2/21, 7:42 AM migration from monolithic to microservices - Pesquisa Google

Dica: Pesquisar apenas resultados em **português (Brasil)**. Especifique seu idioma de pesquisa em Preferências

Anúncios · Comprar migration from monolithic to microservices

Monolith to Microservices:…	Migrando Sistemas Monolíticos Para…	Microservices Patterns: With Examples in Java
R$ 267,55	**R$ 65,45**	**R$ 258,10**
Amazon.com.br	Amazon.com.br	Amazon.com.br

Anúncio · https://www.datadoghq.com/microservices
Migrating To Microservices? - Track Any Distributed Service
Collect, Search, & Analyze Traces Across Distributed Architectures. Start Your Free Trial! Seamlessly Correlate Application Performance to Logs & Underlying Infrastructure Metrics.

Artigos acadêmicos sobre **migration from monolithic to microservices**
From **monolithic** to **microservices**: An experience report … - Bucchiarone - Citado por 95
… , motivations, and issues for **migrating** to **microservices** … - Taibi - Citado por 173
Migrating towards **microservices**: **migration** and … - Carrasco - Citado por 40

Migrating from Monolith to Microservices

1. Identify logical components.
2. Flatten and refactor components.
3. Identify component dependencies.
4. Identify component groups.
5. Create an API for remote user interface.
6. Migrate component groups to macroservices (move component groups to separate projects and make separate deployments).

Mais itens... • 28 de set. de 2020

https://insights.sei.cmu.edu › blog › 8-steps-for-migrating...
8 Steps for Migrating Existing Applications to Microservices

As pessoas também perguntam

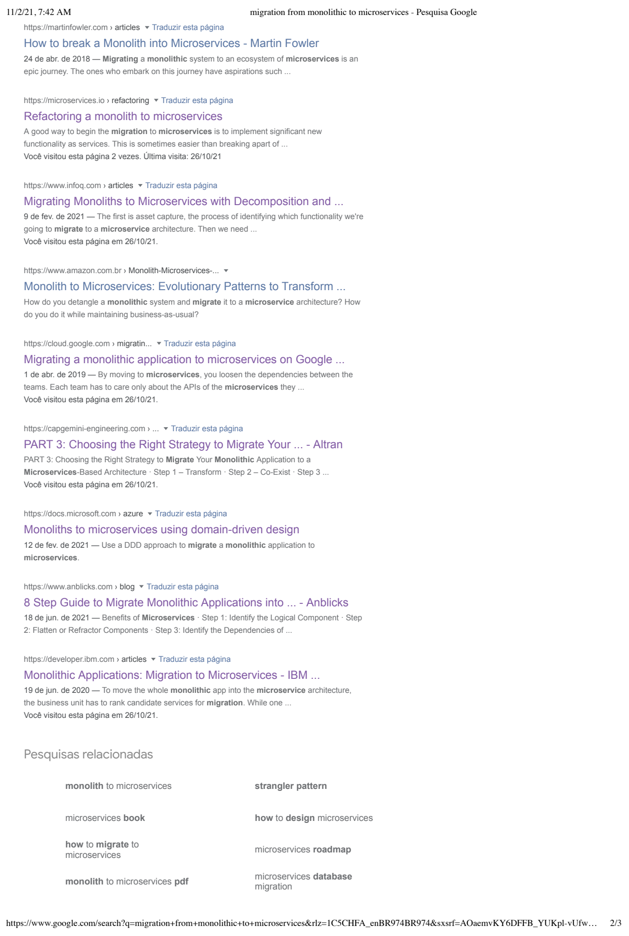
When would you use Microservices over monolithic?

What will need to be considered when splitting the monolithic systems into future Microservices?

What is the difference between monolith and Microservices?

Which architecture is recommended to refactor a monolithic Web application to a new Microservices based application?

https://www.google.com/search?q=migration+from+monolithic+to+microservices&rlz=1C5CHFA_enBR974BR974&sxsrf=AOaemvKY6DFFB_YUKpl-vUfw… 1/3


11/2/21, 7:42 AM migration from monolithic to microservices - Pesquisa Google

https://martinfowler.com › articles ▾ Traduzir esta página
How to break a Monolith into Microservices - Martin Fowler
24 de abr. de 2018 — **Migrating** a **monolithic** system to an ecosystem of **microservices** is an epic journey. The ones who embark on this journey have aspirations such ...

https://microservices.io › refactoring ▾ Traduzir esta página
Refactoring a monolith to microservices
A good way to begin the **migration** to **microservices** is to implement significant new functionality as services. This is sometimes easier than breaking apart of ...
Você visitou esta página 2 vezes. Última visita: 26/10/21

https://www.infoq.com › articles ▾ Traduzir esta página
Migrating Monoliths to Microservices with Decomposition and ...
9 de fev. de 2021 — The first is asset capture, the process of identifying which functionality we're going to **migrate** to a **microservice** architecture. Then we need ...
Você visitou esta página em 26/10/21.

https://www.amazon.com.br › Monolith-Microservices-... ▾
Monolith to Microservices: Evolutionary Patterns to Transform ...
How do you detangle a **monolithic** system and **migrate** it to a **microservice** architecture? How do you do it while maintaining business-as-usual?

https://cloud.google.com › migratin... ▾ Traduzir esta página
Migrating a monolithic application to microservices on Google ...
1 de abr. de 2019 — By moving to **microservices**, you loosen the dependencies between the teams. Each team has to care only about the APIs of the **microservices** they ...
Você visitou esta página em 26/10/21.

https://capgemini-engineering.com › ... ▾ Traduzir esta página
PART 3: Choosing the Right Strategy to Migrate Your ... - Altran
PART 3: Choosing the Right Strategy to **Migrate** Your **Monolithic** Application to a **Microservices**-Based Architecture · Step 1 – Transform · Step 2 – Co-Exist · Step 3 ...
Você visitou esta página em 26/10/21.

https://docs.microsoft.com › azure ▾ Traduzir esta página
Monoliths to microservices using domain-driven design
12 de fev. de 2021 — Use a DDD approach to **migrate** a **monolithic** application to **microservices**.

https://www.anblicks.com › blog ▾ Traduzir esta página
8 Step Guide to Migrate Monolithic Applications into ... - Anblicks
18 de jun. de 2021 — Benefits of **Microservices** · Step 1: Identify the Logical Component · Step 2: Flatten or Refractor Components · Step 3: Identify the Dependencies of ...

https://developer.ibm.com › articles ▾ Traduzir esta página
Monolithic Applications: Migration to Microservices - IBM ...
19 de jun. de 2020 — To move the whole **monolithic** app into the **microservice** architecture, the business unit has to rank candidate services for **migration**. While one ...
Você visitou esta página em 26/10/21.

Pesquisas relacionadas

monolith to microservices	**strangler pattern**
microservices **book**	**how** to **design** microservices
how to **migrate** to microservices	microservices **roadmap**
monolith to microservices **pdf**	microservices **database** migration

https://www.google.com/search?q=migration+from+monolithic+to+microservices&rlz=1C5CHFA_enBR974BR974&sxsrf=AOaemvKY6DFFB_YUKpl-vUfw… 2/3

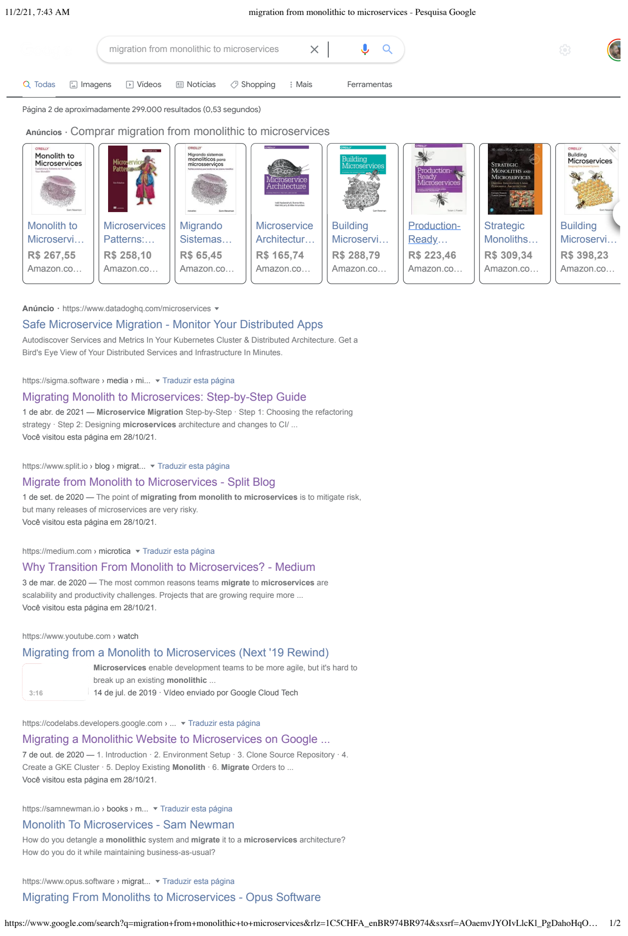
11/2/21, 7:42 AM migration from monolithic to microservices - Pesquisa Google

1 2 3 4 5 6 7 8 9 10 Mais

Brasil 85806-264 - Esmeralda, Cascavel - PR - Com base nos seus lugares (Casa) - Atualizar local

Ajuda Enviar feedback Privacidade Termos

https://www.google.com/search?q=migration+from+monolithic+to+microservices&rlz=1C5CHFA_enBR974BR974&sxsrf=AOaemvKY6DFFB_YUKpl-vUfw… 3/3



11/2/21, 7:43 AM migration from monolithic to microservices - Pesquisa Google

migration from monolithic to microservices

Todas · Imagens · Vídeos · Notícias · Shopping · Mais · Ferramentas

Página 2 de aproximadamente 299.000 resultados (0,53 segundos)

Anúncios · Comprar migration from monolithic to microservices

Monolith to Microservi…	Microservices Patterns:…	Migrando Sistemas…	Microservice Architectur…	Building Microservi…	Production-Ready…	Strategic Monoliths…	Building Microservi…
R$ 267,55	**R$ 258,10**	**R$ 65,45**	**R$ 165,74**	**R$ 288,79**	**R$ 223,46**	**R$ 309,34**	**R$ 398,23**
Amazon.co…	Amazon.co…	Amazon.co…	Amazon.co…	Amazon.co…	Amazon.co…	Amazon.co…	Amazon.co…

Anúncio · https://www.datadoghq.com/microservices

Safe Microservice Migration - Monitor Your Distributed Apps

Autodiscover Services and Metrics In Your Kubernetes Cluster & Distributed Architecture. Get a Bird's Eye View of Your Distributed Services and Infrastructure In Minutes.

https://sigma.software › media › mi... Traduzir esta página

Migrating Monolith to Microservices: Step-by-Step Guide

1 de abr. de 2021 — **Microservice Migration** Step-by-Step · Step 1: Choosing the refactoring strategy · Step 2: Designing **microservices** architecture and changes to CI/ ...
Você visitou esta página em 28/10/21.

https://www.split.io › blog › migrat... Traduzir esta página

Migrate from Monolith to Microservices - Split Blog

1 de set. de 2020 — The point of **migrating from monolith to microservices** is to mitigate risk, but many releases of microservices are very risky.
Você visitou esta página em 28/10/21.

https://medium.com › microtica Traduzir esta página

Why Transition From Monolith to Microservices? - Medium

3 de mar. de 2020 — The most common reasons teams **migrate** to **microservices** are scalability and productivity challenges. Projects that are growing require more ...
Você visitou esta página em 28/10/21.

https://www.youtube.com › watch

Migrating from a Monolith to Microservices (Next '19 Rewind)

3:16

Microservices enable development teams to be more agile, but it's hard to break up an existing **monolithic** ...
14 de jul. de 2019 · Vídeo enviado por Google Cloud Tech

https://codelabs.developers.google.com › ... Traduzir esta página

Migrating a Monolithic Website to Microservices on Google ...

7 de out. de 2020 — 1. Introduction · 2. Environment Setup · 3. Clone Source Repository · 4. Create a GKE Cluster · 5. Deploy Existing **Monolith** · 6. **Migrate** Orders to ...
Você visitou esta página em 28/10/21.

https://samnewman.io › books › m... Traduzir esta página

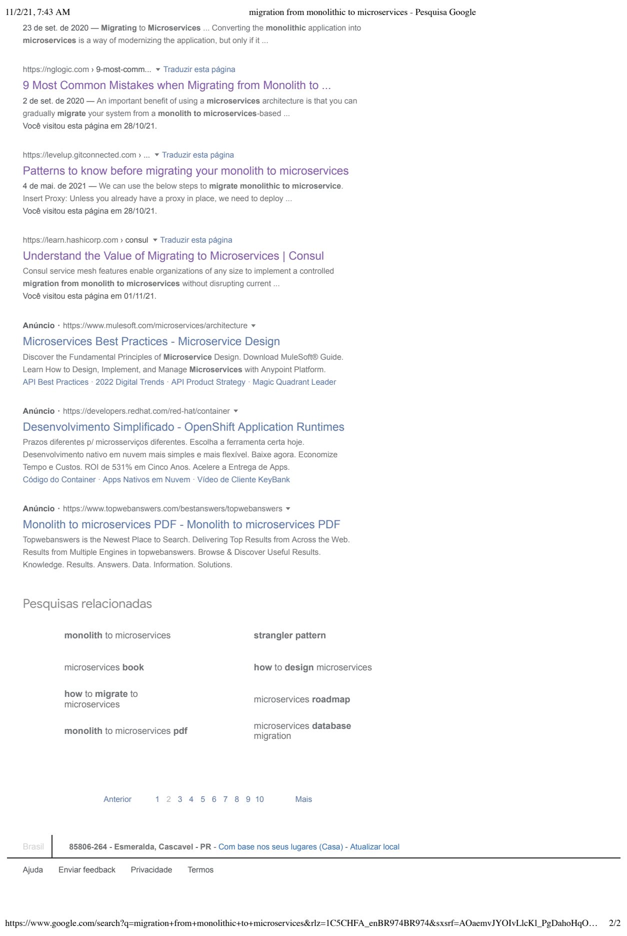
Monolith To Microservices - Sam Newman

How do you detangle a **monolithic** system and **migrate** it to a **microservices** architecture? How do you do it while maintaining business-as-usual?

https://www.opus.software › migrat... Traduzir esta página

Migrating From Monoliths to Microservices - Opus Software

https://www.google.com/search?q=migration+from+monolithic+to+microservices&rlz=1C5CHFA_enBR974BR974&sxsrf=AOaemvJYOIvLlcKl_PgDahoHqO… 1/2



11/2/21, 7:43 AM migration from monolithic to microservices - Pesquisa Google

23 de set. de 2020 — **Migrating** to **Microservices** ... Converting the **monolithic** application into **microservices** is a way of modernizing the application, but only if it ...

https://nglogic.com › 9-most-comm... Traduzir esta página

9 Most Common Mistakes when Migrating from Monolith to ...

2 de set. de 2020 — An important benefit of using a **microservices** architecture is that you can gradually **migrate** your system from a **monolith to microservices**-based ...
Você visitou esta página em 28/10/21.

https://levelup.gitconnected.com › ... Traduzir esta página

Patterns to know before migrating your monolith to microservices

4 de mai. de 2021 — We can use the below steps to **migrate monolithic to microservice**. Insert Proxy: Unless you already have a proxy in place, we need to deploy ...
Você visitou esta página em 28/10/21.

https://learn.hashicorp.com › consul Traduzir esta página

Understand the Value of Migrating to Microservices | Consul

Consul service mesh features enable organizations of any size to implement a controlled **migration from monolith to microservices** without disrupting current ...
Você visitou esta página em 01/11/21.

Anúncio · https://www.mulesoft.com/microservices/architecture

Microservices Best Practices - Microservice Design

Discover the Fundamental Principles of **Microservice** Design. Download MuleSoft® Guide. Learn How to Design, Implement, and Manage **Microservices** with Anypoint Platform.
API Best Practices · 2022 Digital Trends · API Product Strategy · Magic Quadrant Leader

Anúncio · https://developers.redhat.com/red-hat/container

Desenvolvimento Simplificado - OpenShift Application Runtimes

Prazos diferentes p/ microsserviços diferentes. Escolha a ferramenta certa hoje. Desenvolvimento nativo em nuvem mais simples e mais flexível. Baixe agora. Economize Tempo e Custos. ROI de 531% em Cinco Anos. Acelere a Entrega de Apps.
Código do Container · Apps Nativos em Nuvem · Vídeo de Cliente KeyBank

Anúncio · https://www.topwebanswers.com/bestanswers/topwebanswers

Monolith to microservices PDF - Monolith to microservices PDF

Topwebanswers is the Newest Place to Search. Delivering Top Results from Across the Web. Results from Multiple Engines in topwebanswers. Browse & Discover Useful Results. Knowledge. Results. Answers. Data. Information. Solutions.

Pesquisas relacionadas

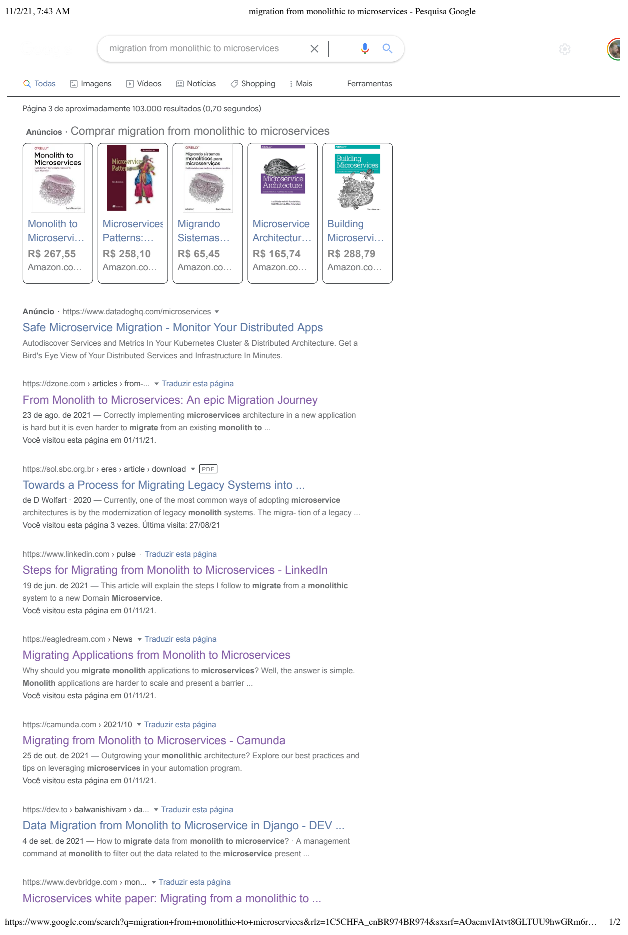
monolith to microservices	**strangler pattern**
microservices **book**	**how** to **design** microservices
how to **migrate** to microservices	microservices **roadmap**
monolith to microservices **pdf**	microservices **database** migration

Anterior 1 2 3 4 5 6 7 8 9 10 Mais

Brasil | **85806-264 - Esmeralda, Cascavel - PR** - Com base nos seus lugares (Casa) - Atualizar local

Ajuda Enviar feedback Privacidade Termos

https://www.google.com/search?q=migration+from+monolithic+to+microservices&rlz=1C5CHFA_enBR974BR974&sxsrf=AOaemvJYOIvLlcKl_PgDahoHqO… 2/2



11/2/21, 7:43 AM migration from monolithic to microservices - Pesquisa Google

migration from monolithic to microservices

Todas · Imagens · Vídeos · Notícias · Shopping · Mais · Ferramentas

Página 3 de aproximadamente 103.000 resultados (0,70 segundos)

Anúncios · Comprar migration from monolithic to microservices

Monolith to Microservi…	Microservices Patterns:…	Migrando Sistemas…	Microservice Architectur…	Building Microservi…
R$ 267,55	R$ 258,10	R$ 65,45	R$ 165,74	R$ 288,79
Amazon.co…	Amazon.co…	Amazon.co…	Amazon.co…	Amazon.co…

Anúncio · https://www.datadoghq.com/microservices

Safe Microservice Migration - Monitor Your Distributed Apps

Autodiscover Services and Metrics In Your Kubernetes Cluster & Distributed Architecture. Get a Bird's Eye View of Your Distributed Services and Infrastructure In Minutes.

https://dzone.com › articles › from-... Traduzir esta página

From Monolith to Microservices: An epic Migration Journey

23 de ago. de 2021 — Correctly implementing **microservices** architecture in a new application is hard but it is even harder to **migrate** from an existing **monolith to** ...
Você visitou esta página em 01/11/21.

https://sol.sbc.org.br › eres › article › download PDF

Towards a Process for Migrating Legacy Systems into ...

de D Wolfart · 2020 — Currently, one of the most common ways of adopting **microservice** architectures is by the modernization of legacy **monolith** systems. The migra- tion of a legacy ...
Você visitou esta página 3 vezes. Última visita: 27/08/21

https://www.linkedin.com › pulse · Traduzir esta página

Steps for Migrating from Monolith to Microservices - LinkedIn

19 de jun. de 2021 — This article will explain the steps I follow to **migrate** from a **monolithic** system to a new Domain **Microservice**.
Você visitou esta página em 01/11/21.

https://eagledream.com › News Traduzir esta página

Migrating Applications from Monolith to Microservices

Why should you **migrate monolith** applications to **microservices**? Well, the answer is simple. **Monolith** applications are harder to scale and present a barrier ...
Você visitou esta página em 01/11/21.

https://camunda.com › 2021/10 Traduzir esta página

Migrating from Monolith to Microservices - Camunda

25 de out. de 2021 — Outgrowing your **monolithic** architecture? Explore our best practices and tips on leveraging **microservices** in your automation program.
Você visitou esta página em 01/11/21.

https://dev.to › balwanishivam › da... Traduzir esta página

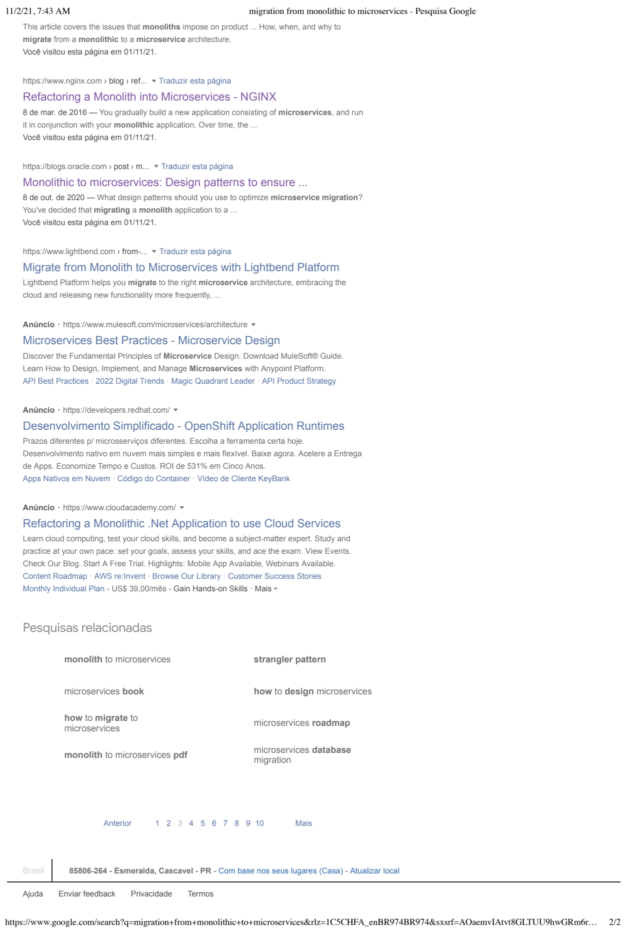
Data Migration from Monolith to Microservice in Django - DEV ...

4 de set. de 2021 — How to **migrate** data from **monolith to microservice**? · A management command at **monolith** to filter out the data related to the **microservice** present ...

https://www.devbridge.com › mon... Traduzir esta página

Microservices white paper: Migrating from a monolithic to ...

https://www.google.com/search?q=migration+from+monolithic+to+microservices&rlz=1C5CHFA_enBR974BR974&sxsrf=AOaemvIAtvt8GLTUU9hwGRm6r… 1/2



11/2/21, 7:43 AM migration from monolithic to microservices - Pesquisa Google

This article covers the issues that **monoliths** impose on product ... How, when, and why to **migrate** from a **monolithic** to a **microservice** architecture.
Você visitou esta página em 01/11/21.

https://www.nginx.com › blog › ref... Traduzir esta página

Refactoring a Monolith into Microservices - NGINX

8 de mar. de 2016 — You gradually build a new application consisting of **microservices**, and run it in conjunction with your **monolithic** application. Over time, the ...
Você visitou esta página em 01/11/21.

https://blogs.oracle.com › post › m... Traduzir esta página

Monolithic to microservices: Design patterns to ensure ...

8 de out. de 2020 — What design patterns should you use to optimize **microservice migration**? You've decided that **migrating** a **monolith** application to a ...
Você visitou esta página em 01/11/21.

https://www.lightbend.com › from-... Traduzir esta página

Migrate from Monolith to Microservices with Lightbend Platform

Lightbend Platform helps you **migrate** to the right **microservice** architecture, embracing the cloud and releasing new functionality more frequently, ...

Anúncio · https://www.mulesoft.com/microservices/architecture

Microservices Best Practices - Microservice Design

Discover the Fundamental Principles of **Microservice** Design. Download MuleSoft® Guide. Learn How to Design, Implement, and Manage **Microservices** with Anypoint Platform.
API Best Practices · 2022 Digital Trends · Magic Quadrant Leader · API Product Strategy

Anúncio · https://developers.redhat.com/

Desenvolvimento Simplificado - OpenShift Application Runtimes

Prazos diferentes p/ microsserviços diferentes. Escolha a ferramenta certa hoje. Desenvolvimento nativo em nuvem mais simples e mais flexível. Baixe agora. Acelere a Entrega de Apps. Economize Tempo e Custos. ROI de 531% em Cinco Anos.
Apps Nativos em Nuvem · Código do Container · Vídeo de Cliente KeyBank

Anúncio · https://www.cloudacademy.com/

Refactoring a Monolithic .Net Application to use Cloud Services

Learn cloud computing, test your cloud skills, and become a subject-matter expert. Study and practice at your own pace: set your goals, assess your skills, and ace the exam. View Events. Check Our Blog. Start A Free Trial. Highlights: Mobile App Available, Webinars Available.
Content Roadmap · AWS re:Invent · Browse Our Library · Customer Success Stories
Monthly Individual Plan - US$ 39,00/mês - Gain Hands-on Skills · Mais

Pesquisas relacionadas

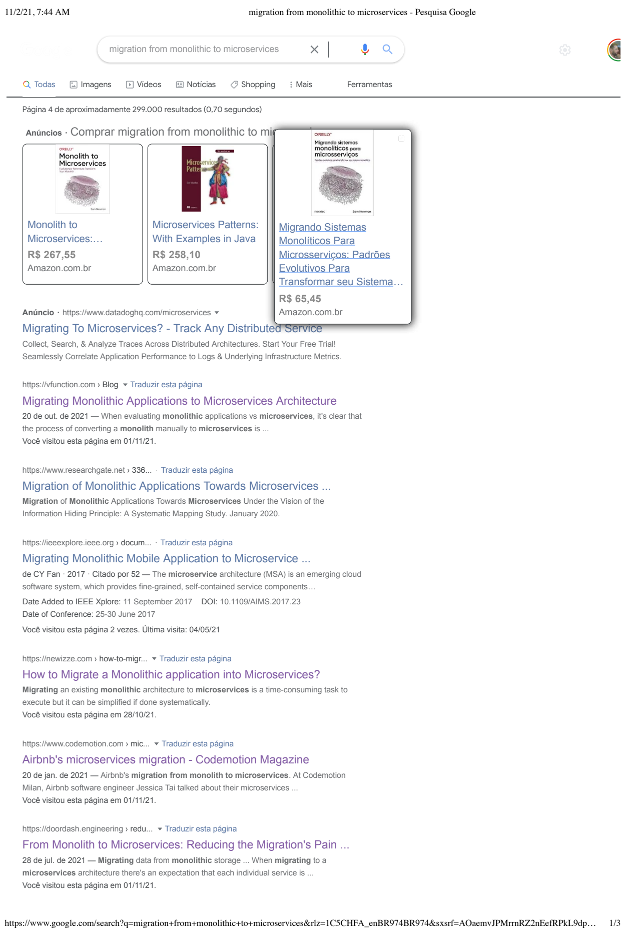
monolith to microservices	**strangler pattern**
microservices **book**	**how** to **design** microservices
how to **migrate** to microservices	microservices **roadmap**
monolith to microservices **pdf**	microservices **database** migration

Anterior 1 2 3 4 5 6 7 8 9 10 Mais

Brasil | **85806-264 - Esmeralda, Cascavel - PR** - Com base nos seus lugares (Casa) - Atualizar local

Ajuda Enviar feedback Privacidade Termos

https://www.google.com/search?q=migration+from+monolithic+to+microservices&rlz=1C5CHFA_enBR974BR974&sxsrf=AOaemvIAtvt8GLTUU9hwGRm6r… 2/2



11/2/21, 7:44 AM migration from monolithic to microservices - Pesquisa Google

migration from monolithic to microservices

Todas · Imagens · Vídeos · Notícias · Shopping · Mais · Ferramentas

Página 4 de aproximadamente 299.000 resultados (0,70 segundos)

Anúncios · Comprar migration from monolithic to mi

Monolith to Microservices:… R$ 267,55 Amazon.com.br

Microservices Patterns: With Examples in Java R$ 258,10 Amazon.com.br

Migrando Sistemas Monolíticos Para Microsserviços: Padrões Evolutivos Para Transformar seu Sistema… R$ 65,45 Amazon.com.br

Anúncio · https://www.datadoghq.com/microservices
Migrating To Microservices? - Track Any Distributed Service
Collect, Search, & Analyze Traces Across Distributed Architectures. Start Your Free Trial! Seamlessly Correlate Application Performance to Logs & Underlying Infrastructure Metrics.

https://vfunction.com › Blog · Traduzir esta página
Migrating Monolithic Applications to Microservices Architecture
20 de out. de 2021 — When evaluating **monolithic** applications vs **microservices**, it's clear that the process of converting a **monolith** manually to **microservices** is ...
Você visitou esta página em 01/11/21.

https://www.researchgate.net › 336... · Traduzir esta página
Migration of Monolithic Applications Towards Microservices ...
Migration of **Monolithic** Applications Towards **Microservices** Under the Vision of the Information Hiding Principle: A Systematic Mapping Study. January 2020.

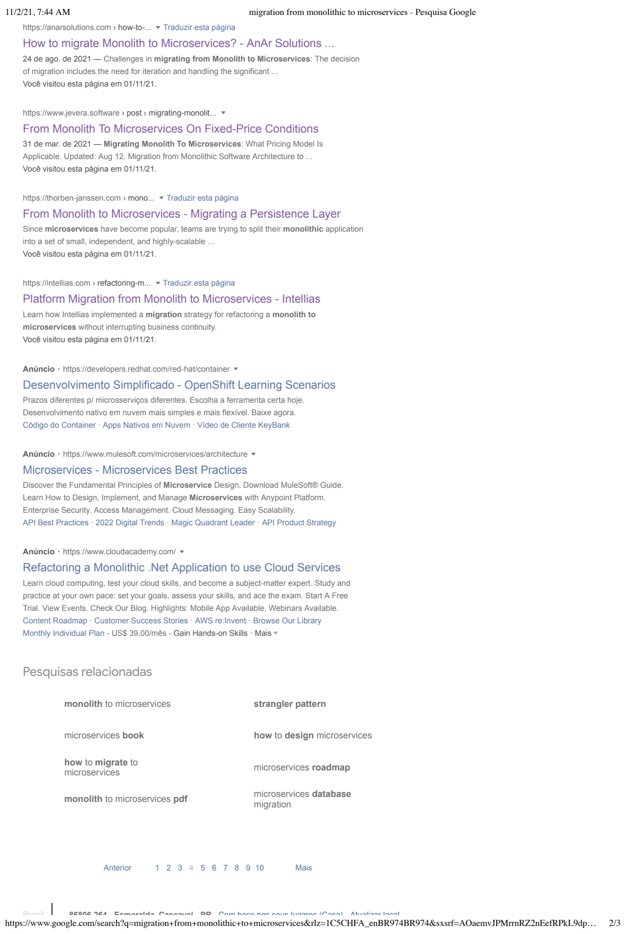
https://ieeexplore.ieee.org › docum... · Traduzir esta página
Migrating Monolithic Mobile Application to Microservice ...
de CY Fan · 2017 · Citado por 52 — The **microservice** architecture (MSA) is an emerging cloud software system, which provides fine-grained, self-contained service components…
Date Added to IEEE Xplore: 11 September 2017 DOI: 10.1109/AIMS.2017.23
Date of Conference: 25-30 June 2017
Você visitou esta página 2 vezes. Última visita: 04/05/21

https://newizze.com › how-to-migr... · Traduzir esta página
How to Migrate a Monolithic application into Microservices?
Migrating an existing **monolithic** architecture to **microservices** is a time-consuming task to execute but it can be simplified if done systematically.
Você visitou esta página em 28/10/21.

https://www.codemotion.com › mic... · Traduzir esta página
Airbnb's microservices migration - Codemotion Magazine
20 de jan. de 2021 — Airbnb's **migration from monolith to microservices**. At Codemotion Milan, Airbnb software engineer Jessica Tai talked about their microservices ...
Você visitou esta página em 01/11/21.

https://doordash.engineering › redu... · Traduzir esta página
From Monolith to Microservices: Reducing the Migration's Pain ...
28 de jul. de 2021 — **Migrating** data from **monolithic** storage ... When **migrating** to a **microservices** architecture there's an expectation that each individual service is ...
Você visitou esta página em 01/11/21.

https://www.google.com/search?q=migration+from+monolithic+to+microservices&rlz=1C5CHFA_enBR974BR974&sxsrf=AOaemvJPMrrnRZ2nEefRPkL9dp… 1/3



11/2/21, 7:44 AM migration from monolithic to microservices - Pesquisa Google

https://anarsolutions.com › how-to-... · Traduzir esta página
How to migrate Monolith to Microservices? - AnAr Solutions ...
24 de ago. de 2021 — Challenges in **migrating from Monolith to Microservices**: The decision of migration includes the need for iteration and handling the significant ...
Você visitou esta página em 01/11/21.

https://www.jevera.software › post › migrating-monolit...
From Monolith To Microservices On Fixed-Price Conditions
31 de mar. de 2021 — **Migrating Monolith To Microservices**: What Pricing Model Is Applicable. Updated: Aug 12. Migration from Monolithic Software Architecture to ...
Você visitou esta página em 01/11/21.

https://thorben-janssen.com › mono... · Traduzir esta página
From Monolith to Microservices - Migrating a Persistence Layer
Since **microservices** have become popular, teams are trying to split their **monolithic** application into a set of small, independent, and highly-scalable ...
Você visitou esta página em 01/11/21.

https://intellias.com › refactoring-m... · Traduzir esta página
Platform Migration from Monolith to Microservices - Intellias
Learn how Intellias implemented a **migration** strategy for refactoring a **monolith to microservices** without interrupting business continuity.
Você visitou esta página em 01/11/21.

Anúncio · https://developers.redhat.com/red-hat/container
Desenvolvimento Simplificado - OpenShift Learning Scenarios
Prazos diferentes p/ microsserviços diferentes. Escolha a ferramenta certa hoje. Desenvolvimento nativo em nuvem mais simples e mais flexível. Baixe agora.
Código do Container · Apps Nativos em Nuvem · Vídeo de Cliente KeyBank

Anúncio · https://www.mulesoft.com/microservices/architecture
Microservices - Microservices Best Practices
Discover the Fundamental Principles of **Microservice** Design. Download MuleSoft® Guide. Learn How to Design, Implement, and Manage **Microservices** with Anypoint Platform. Enterprise Security. Access Management. Cloud Messaging. Easy Scalability.
API Best Practices · 2022 Digital Trends · Magic Quadrant Leader · API Product Strategy

Anúncio · https://www.cloudacademy.com/
Refactoring a Monolithic .Net Application to use Cloud Services
Learn cloud computing, test your cloud skills, and become a subject-matter expert. Study and practice at your own pace: set your goals, assess your skills, and ace the exam. Start A Free Trial. View Events. Check Our Blog. Highlights: Mobile App Available, Webinars Available.
Content Roadmap · Customer Success Stories · AWS re:Invent · Browse Our Library
Monthly Individual Plan - US$ 39,00/mês - Gain Hands-on Skills · Mais

Pesquisas relacionadas

monolith to microservices	**strangler pattern**
microservices **book**	**how** to **design** microservices
how to **migrate** to microservices	microservices **roadmap**
monolith to microservices **pdf**	microservices **database** migration

Anterior 1 2 3 4 5 6 7 8 9 10 Mais

Brasil | 85806-264 - Esmeralda, Cascavel - PR - Com base nos seus lugares (Casa) - Atualizar local

https://www.google.com/search?q=migration+from+monolithic+to+microservices&rlz=1C5CHFA_enBR974BR974&sxsrf=AOaemvJPMrrnRZ2nEefRPkL9dp… 2/3

11/2/21, 7:44 AM migration from monolithic to microservices - Pesquisa Google

Brasil | 85806-264 - Esmeralda, Cascavel - PR - Com base nos seus lugares (Casa) - Atualizar local

Ajuda Enviar feedback Privacidade Termos

https://www.google.com/search?q=migration+from+monolithic+to+microservices&rlz=1C5CHFA_enBR974BR974&sxsrf=AOaemvJPMrrnRZ2nEefRPkL9dp… 3/3

.2.2 RR Paper Count

ID	White Papers aprovados	Inclusão/Exclusão	
1	https://www.infoq.com/podcasts/monolith-microservices/	Inclusão-motivo 3	ok
2	https://medium.com/geekculture/the-size-of-a-microservice-b9e6bc90475	Inclusão-motivo 3	ok
3	https://amazicworld.com/getting-ready-for-microservices-breaking-down-the-monolith/	Inclusão-motivo 3	ok
4	https://bambooagile.eu/insights/monolith-vs-microservices/	Inclusão-motivo 3	ok
5	https://microservices.io/refactoring/	Inclusão-motivo 3	ok
6	https://www.infoq.com/articles/migrating-monoliths-to-microservices-with-decomposition/	Inclusão-motivo 3	ok
7	https://developer.ibm.com/articles/challenges-and-patterns-for-modernizing-a-monolithic-application-into-microservices/	Inclusão-motivo 3	ok
8	https://capgemini-engineering.com/us/en/insight/part-3-choosing-the-right-strategy-to-migrate-your-monolithic-application-to-a-microservices-based-architectur	Inclusão-motivo 3	ok
9	https://docs.microsoft.com/en-us/azure/architecture/microservices/migrate-monolith	Inclusão-motivo 3	ok
10	https://sigma.software/about/media/migrating-monolith-microservices-step-step-guide	Inclusão-motivo 3	
11	https://newizze.com/how-to-migrate-a-monolithic-application-into-microservices/	Inclusão-motivo 3	excluir, argumentos rasos
12	https://dzone.com/articles/from-monolith-to-microservices-an-epic-migration-j	Inclusão-motivo 3	reler, pode ter alguns passos
13	https://www.nginx.com/blog/refactoring-a-monolith-into-microservices/	Inclusão-motivo 3	
14	https://blogs.oracle.com/cloud-infrastructure/post/monolithic-to-microservices-how-design-patterns-help-ensure-migration-success	Inclusão-motivo 3	
15	https://doordash.engineering/2021/07/28/reducing-the-migrations-pain-points/	Inclusão-motivo 3	
16	https://thorben-janssen.com/monolith-to-microservices-persistence-layer/	Inclusão-motivo 3	
17	https://samnewman.io/blog/2015/04/07/microservices-for-greenfield/	Inclusão-motivo 3	
18	https://www.infoq.com/articles/monolith-defense-part-1	Inclusão-motivo 3	
19	https://www.infoq.com/articles/monolith-defense-part-2	Inclusão-motivo 3	
20	https://www.infoq.com/presentations/event-flow-systems	Inclusão-motivo 4	
21	https://segment.com/blog/goodbye-microservices/	Inclusão-motivo 4	
22	https://codeboje.de/developers-problem-not-monoliths/	Inclusão-motivo 4	
23	https://www.oreilly.com/ideas/modules-vs-microservices	Inclusão-motivo 3	
24	http://www.grahamlea.com/2016/04/shared-libraries-in-microservices-bad-advice/	Inclusão-motivo 4	
25	https://www.infoq.com/articles/seven-uservices-antipatterns/	Inclusão-motivo 4	
26	https://itnext.io/anti-patterns-of-microservices-6e802553bd46	Inclusão-motivo 4	
27	https://natalian.org/2019/05/16/Microservices_pitfalls/	Inclusão-motivo 4	
28	https://techbeacon.com/app-dev-testing/forget-monoliths-vs-microservices-cognitive-load-what-matters	Inclusão-motivo 4	
29	https://rclayton.silvrback.com/failing-at-microservices	Inclusão-motivo 3-4	
30	http://chi.pl/2017/01/30/Micro-monolith-Anti-pattern.html	Inclusão-motivo 4	
31	https://www.infoq.com/articles/Microservices-Architectural-Fitness/	Inclusão-motivo 3	
32	https://hackernoon.com/why-microservices-fail-6cdc006f9540	Inclusão-motivo 3	
34	https://www.infoq.com/presentations/microservices-monolith-antipatterns/	Inclusão-motivo 4	

.3 Interviews

UNIVERSIDADE ESTADUAL DO OESTE DO PARANÁ / CAMPUS DE CASCAVEL
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
PROGRAMA DE PÓS-GRADUAÇÃO STRICTO SENSU EM CIÊNCIA DA COMPUTAÇÃO - MESTRADO

PPGComp

Entrevista para dissertação Mestrado

Aluno: Guilherme Villaca
Prof.: Dr. Ivonei Freitas da Silva

Objetivo: Avaliar as estratégias adotadas antes da migração de um sistema monolítico para microsserviços a fim de mitigar anti-padrões existentes em microsserviços

Contexto
A migração de sistemas monolíticos para microsserviços vem sendo uma tendência nos últimos anos na engenharia de software. Entretanto muitas organizações migram sem experiência em microsserviços, principalmente por aprender como migrar através de livros e praticantes [1]. Devido a novidade do tópico, esse processo de aprender durante a migração acarreta em vários problemas após a migração, conhecidos como anti-padrões em microsserviços.
Vários autores vêm, nos últimos anos, catalogando e definindo estes anti-padrões. Dentre os vários anti-padrões em microsserviços conhecidos, alguns são exclusivos de microsserviços (como por ex.: não uso de API gateway ou uso indevido de ESB) e outros são possíveis de mitigar antes da migração, com um melhor planejamento, pesquisa, uso de técnicas e estratégias.
Os anti-padrões que nós consideramos possíveis de mitigar antes da migração são os seguintes:

- **Megaservice**: Um serviço que faz muitas coisas. Um monólito.

- **Wrong Cuts**: Monolítico dividido de forma errada;

- **Cyclic Dependency:** Ciclo de chamadas entre microsserviços;

- **Shared Persistency**; Diferentes microsserviços acessando o mesmo banco de dados;

- **Inappropriate Service Intimacy:** O microsserviço se conecta a dados privados de outros serviços em vez de lidar com seus próprios dados.

- **Shared Libraries**: Bibliotecas compartilhadas entre diferentes microsserviços;

- **Microservice Greedy**: Microsserviços muito pequenos.

- **Legacy organization:** A empresa ainda trabalha sem alterar seus processos e políticas. Como exemplo, com equipes de Dev e Ops independentes, teste manual e agendamento de lançamentos comuns.

Durante nossas pesquisas consideramos que o uso de algumas técnicas ou estratégias podem auxiliar na migração e mitigar alguns anti-padrões:

Anti-padrão:
Shared persistence e Inappropriate service intimacy
Estratégia para mitigar:
Avaliar as tabelas do banco de dados e classificá-las em subsistemas de acordo com o domínio na fase de pré-migração; Agrupar entidades do sistema em candidatos a microsserviços;
Anti-padrão:
Wrong cuts; Megaservice; Shared Libraries; Cyclic Dependence; Microservice greedy;
Estratégia para mitigar:
Uso de DDD para modularizar e traduzir funcionalidades em domínio e subdomínio antes da migração; Estratégia para identificar quão alto está o acoplamento; Aplicar um conjunto de atividades chamadas de evolucionabilidade garantida;

Com base nestas informações, gostaríamos de questionar a comunidade de engenharia de software que já fez ou faz parte de um processo de migração de um sistema monolítico para microsserviços com as questões abaixo.

Para melhor entendimento, considerar nesta entrevista a migração como sendo dividida em três etapas: Pré-migração - toda a fase de planejamento, estudo, análise e decomposição do sistema monolítico; Migração - a implementação do sistema com a arquitetura de microsserviços. E Pós-migração - com a arquitetura em microsserviços já em produção.

[1] Taibi, Davide & Lenarduzzi, Valentina & Pahl, Claus. (2019). Microservices Anti Patterns: A Taxonomy. 10.1007/978-3-030-31646-4_5.

QUESTÕES

1 - Qual foi a motivação para você considerar a migração de monolito para microsserviços e qual era o contexto do monolito?

#1 - Monolito com 30 anos de desenvolvimento em uma tecnologia obsoleta que exigia uma nova versão, com uma tecnologia mais atual e fácil de adaptar ao que o mercado exige

#2 - Flexibilidade nas tecnologias adotadas, fácil manutenção e velocidade no desenvolvimento de novas features, possibilitando ser criadas integrações, sistemas, apps de forma rápida. Problemas monolito eram Coesão e acoplamento, número alto de funcionalidades/módulos, manutenibilidade, muitos bugs e dificuldade para escalar o sistema.
Muitos módulos com dependência entre si, manutenção em um módulo acaba gerando bugs em outros, tudo em um mesmo servidor, banco de dados, cache, arquivos ...
***existe a possibilidade sim de escalar um monólito sendo bem projetado.**

#3 - Não considerei, seriamente, apenas foi avaliado se teria ganhos maiores que os custos, não tinha. Quase sempre a adoção de microsserviços existe para mascarar incompetência da equipe, é quase uma definição automática. Existem exceções mas é raro, quase sempre a pessoa acha que o caso dela é exceção e não é. Todos os problemas acima podem e devem ser resolvidos sem MS.

#4 - Pq gostaria de ter MS - Escalabilidade dentro de MS é muito grande
flexibilidade pra mudar o sistema quando necessário. Sistema era muito complexo, não tinha tempo nem dinheiro pra investir em pessoal. Usam tecnologias de MS kubernetes e desenvolveu monolito primeiro para migrar aos poucos.

#5 - Difícil manutenção do código, dificuldade de atualizar tecnologia que era java, atualizar hibernate. Nunca se atualizava tecnologia por que acarreta quebra de código. Difícil manutenção por ser um sistema muito grande, horas gastas em configuração, pra rodar o sistema na máquina, pra dar build no sistema também. O cliente tinha que ter uma infra, servidor próprio, era necessário ir até o cliente, configurar, instalar etc. Implementar um novo módulo/funcionalidade era mais custoso, poderia afetar todo o sistema, muita amarração de código.

#6 - motivação principal de monolito pra microsserviços foi principalmente desacoplamento, havia um sistema em delphi, serviços que não funcionavam direito
exemplo: uma parte era encarregada pra encaminhar emails pro cliente e esse serviço não funcionava bem era difícil manutenção e delphi era cada vez mais difícil de encontrar pessoas pra trabalhar por isso a demanda de atualizar a tecnologia.

#7 - tinha um sistema cliente servidor, com processos de aplicação de API
entrou num projeto de construir uma aplicação CRM que se tornou microsserviços
produto já foi concebido para ser microsserviços. O monolítico já era um cliente servidor, front e back e quando foi aplicado MSA era usar tecnologia atual que pudesse ser escalável. Foi usado as expertises do monolítico para criar direto em

microsserviços e não foi exatamente uma migração e sim foi recriado do 0, usando apenas como base o monolítico.

2 - Quais foram as etapas para estudar/entender/pesquisar as complexidades de microsserviços? Quais treinamentos foram feitos?

#1 - A organização comprou um livro sobre migração para microsserviços - sam newman - e todos os membros da equipe passaram a estudá-lo. Cada membro da equipe usou uma forma própria de estudar e pesquisar

#2 - Palestras, casos de uso de grandes empresas, cursos livres e livros.

#3 - Ler tudo o que é publicado sobre o assunto, não só os que defendem, ver as entrelinhas, consultar pessoas realmente experientes e que não possuem viés. Fundamentos de computação, o que falta para as pessoas.

#4 - 2 ou 3 meses estudando como fazer

#5 - quando foi criado a equipe o entrevistado ainda não fazia parte. Algumas pessoas da equipe inicial estudaram quais tecnologias e ferramentas estavam sendo utilizadas pelo mercado. Foi contratado consultorias pra auxiliar a desenvolver e entender a complexidade de microsserviços e não teve nenhum treinamento específico.

#6 - Entender as regras do sistema monolítico, identificar o que fazia, como desacoplar, o que a parte deveria fazer, conversar com as pessoas pra identificar as regras de negócio. Não houveram treinamentos específicos, apenas o background da equipe.

#7 - A experiência começou envolvendo 3 arquitetos pra pesquisar e estudar uma forma de desenvolver em microsserviços sem uma ideia de reaproveitar o monolítico. Olhando como big players faziam, netflix etc pra não cometer os mesmos erros enfrentados.

3 - Quantos envolvidos participaram do processo de pré-migração e migração?

#1 - 5 pessoas no total. Hoje são 3 pessoas na equipe.

#2 - Menos que 10.

#3 - Mais que 50.

#4 - 4 pessoas.

#5 - 5 desenvolvedores/arquitetos e o gestor e foi crescendo até ter 40 pessoas.

#6 - uma equipe de 3 pessoas (1 back, 1 front 1 líder) e 1 infra.

#7 - inicialmente 3 e foram criados squads. os primeiros investiram tempo em pesquisa, escolha de tecnologia/frameworks . Cada squad PO, 3 a 5 desenvolvedores 1 testados e depois 1 arquiteto entrou em cada equipe.

4 - Quantos microsserviços foram planejados na fase de pré-migração?

#1 - Foi dividido em 5 módulos e 13 microsserviços até o momento.

Obs: Equipe reduzida utilizando microsserviços, portanto não há separação de microsserviços por times. Nem tecnologias demais envolvidas.

#2 - Entre 5 e 10.

#3 - Menos que 5.

#4 - não tinha número de MS.

#5 - foi planejado de forma macro, pensado em módulos, funcionalidades.

#6 - não soube, variava muito conforme o andamento do projeto, havia muitas discussões e ocorria de 1 microsserviço virar em 2 ou o contrário.

#7 - entre 25 a 30 microsserviços.

5 - Foi utilizado alguma métrica para definir o tamanho de cada microsserviço? Ex: linhas de código, número de funcionalidades, classes, modelos etc.

#1 - Não foi pensado em nenhuma métrica, foi citado o DDD como forma de definir cada microsserviço de acordo com seu domínio, mas tamanho não foi considerado.

#2 - Áreas de domínio do negócio ex: pessoas, cursos, carrinho de compras, produtos.

#3 - Isso não faz sentido, quem faz isso mostra que não ter a menor ideia do que está fazendo.

#4 - não tinha métrica.

#5 - não soube dizer.

#6 - Era usado uma espécie de bom senso avaliar que o microsserviço seria dividido se necessário.

#7 - não existia nenhuma técnica, apenas motivação de negócio context boundaries. Era muito mais fácil entender a comunicação entre objetos do que olhar pra outras formas de ter essa métrica, como linhas de código, funcionalidades etc. Então às vezes era necessário muito mais código entre dois microsserviços pois eles foram separados, se fossem mantidos juntos essa comunicação a mais entre eles não seria necessário.

6 - Qual o tempo estimado para cada fase do processo de pré-migração e migração? Planejamento, Estudo, Decomposição, Refatoração, Implementação etc.

#1 - o processo iniciou há 1 ano e 6 meses, sendo que foi feito em paralelo, planejamento, estudo e implementação.

#2 - Recriamos do zero, cerca de um ano de desenvolvimento para ter a primeira versão estável dos principais serviços.

#3 - Isso não faz sentido, quem faz isso mostra que não tem a menor ideia do que está fazendo.

#4 - Tudo aconteceu junto mas era bem orquestrado como fazer.

#5 - O tempo foi planejado em relação a entregas de funcionalidades, não detalhou.

#6 - 4 mêses planejamento e 6 meses de estudo, ainda está sendo implementado.

#7 - 6 meses pré migração e 2 anos a migração.

7 - Quais os papéis envolvidos? Engenheiro/arquiteto de software, desenvolvedor, responsável por infra-estrutura etc.

#1 - 2 desenvolvedores sênior, 1 engenheiro/desenvolvedor sênior, 1 gestor, 1 desenvolvedor junior.

#2 - 1 engenheiro/arquiteto, 2 desenvolvedores (1 back-end e 1 front-end), um focado em infraestrutura.

#3 - Não tem como não envolver todos.

#4 - 1 frontend, 1 infra e 1 backend.

#5 - era 5 desenvolvedores/arquitetos e 1 destinado a UX.

#6 -

#7 - 1 PO, 1 arquiteto, 1 teste, 5 desenvolvedores.

8 - Qual a tecnologia do monolito? Quais as tecnologias adotadas para microsserviços? Se for a mesma, por que manteve? se alterou por que alterou? Se alterou quanto tempo considerou/despendeu-se para a curva de aprendizado da nova tecnologia, foi feito investimento em contratação/treinamento?

#1 - Tecnologia do monólito era delphi e Tecnologia do microsserviços GOLANG. Foi escolhida pela praticidade, ser uma linguagem com curva de aprendizado simples,

ser facilmente escalável e ter ferramentas e frameworks voltados e compatíveis com a infraestrutura da nuvem

#2 - PHP, MySQL em ambiente AWS. PHP, MySQL, MongoDB, Redis, Docker, OAuth2 em ambiente AWS (Kubernetes e outros serviços). Utilizar a ferramenta correta para o problema proposto, na questão de linguagem de programação mantemos a mesma pois a equipe era bem reduzida. Na questão de banco de dados utilizamos bancos com propósitos diferentes para resolver melhor cada área do negócio envolvida. Em treinamentos online e eventos quando possível.

#3 - Poderiam ser várias, ao contrário da crença popular isso é possível, as pessoas nem conseguem definir o que é monólito ou microsserviço, como podem adotar algo assim? Arquitetura monolítica não é o mesmo que executável monolítico. A motivação de trocar deveria ser sempre pela escolha errada antes. Mas quase sempre é o gosto, as pessoas só não gostam de confessar. Contrate e treine bons profissionais para não precisar de muletas.

#4 - Python é utilizado, no front é react. Na infra terraform e banco postgres usando também kubernetes amazon.

#5 - Monólito era java 4 com hibernate, no frontend era javascript, html, css, jquery microsserviços continuou java 8 / koblin bancos relacionais postgres. Reactjs no front e mobile react native sql lite. As pessoas que estavam na equipe auxiliavam os novos, não havia treinamento, simplesmente já começavam a programar.

#6 - monolito Delphi, firebird. Microsserviços C#, VueJS, GoLang e banco de dados postgresql.

#7 - monolito uma aplicação em java rodando em jboss cliente servidor tradicional com Postgres oracle e sql server,. No front era app android código nativo. Microsserviços partiu da ideia que não teria 1 linguagem apenas, hoje além de java tem skala e kotlin, serviços também com node, front com javascript / frameworks, mysql, postgres, cassandra, neo4j, s3, elasticsearch e comunicação REST JSON.

9 - Qual o nível de conhecimento e experiência da equipe em relação a Orientação a Objetos, padrões de projeto, microsserviços, sistemas distribuídos etc.

#1 - Equipe com foco e experiência em padrões de projetos, microsserviços.

#2 - Muito conhecimento em OO, muito conhecimento em padrões de projeto, conhecimento intermediário em relação a microsserviços e sistemas distribuídos.

#3 - Especialista no assunto em OO e em padrões de projeto, conhecimento intermediário em relação a microsserviços e muito conhecimento em sistemas distribuídos.

#4 - Mais conhecimento que veio da graduação.

#5 - Inicialmente eram membros com senioridade maior, depois novos integrantes eram de níveis menores. Conhecimento de microsserviços e sistemas distribuídos era escasso inicialmente, foram aprendendo com o tempo.

#6 - o líder tinha um conhecimento maior em padrões, sistemas distribuídos. O restante da equipe era entre básico e intermediário e passaram a estudar bastante clean architecture durante o projeto.

#7 -

10 - Quais projetos já participaram? Quais eram as áreas/domínios? Qual a quantidade?

#1 - Domínio - comércio em geral.

#2 - E-commerce e EAD; Trade Merchandising (indústrias e agências); Participou de menos deu 10 projetos

**#3 - O maior ERP do mercado brasileiro (em todos critérios), entre outros.
menos que 10 projetos participou. Principalmente ERPs desktop. Eu trabalho em projetos durante muito tempo, não pulo de projeto em projeto sem comprometimento com o que estou fazendo, são projetos para vida toda. Não considero como projetos as pequenas tarefas de programação.**

#4 -

#5 -

#6 -

#7 -

Questões Específicas

11 - Vocês conheciam os anti-padrões existentes em microsserviços antes dessa entrevista? Se sim, quais vocês conheciam?

#1 - Já conheciam megaservice o wrong cuts. Os demais anti-padrões não conheciam.

#2 - Sim, megaservice, wrong cuts, cyclic dependency, shared persistence, shared libraries, microservice greedy.

#3 - Sim, megaservice, wrong cuts, cyclic dependency, shared persistency, Inappropriate Service Intimacy, shared libraries, microservice greedy e Legacy organization.

#4 - Não conheciam, alguns antipadrões já eram conhecidos através de livros.

#5 - Conhecia, não exatamente antipadrões, mas os problemas relatados eram familiares. Sim, megaservice era uma preocupação, não construir um microsserviço muito grande. Tiveram microsserviços divididos de forma errada (wrong cuts), causando um grande impacto no sistema. Também teve ciclo de chamada entre microsserviços. Foi utilizado bibliotecas compartilhadas, com planejamento de evitar retrabalho.

#6 - Nunca tinha ouvido falar de antipadrões, alguns conceitos sim de forma acadêmica não.

#7 - Sim, megaservice, inappropriate service intimacy, microservice greedy.

12 - Na fase de pré-migração foi tomada alguma medida para enfrentar os anti-padrões? seja por estudo do que eram esses anti-padrões em detalhes ou qualquer outra abordagem? Se sim, qual anti-padrão foi considerado?

#1 - No planejamento e fase de estudos faziam esforços para evitar estes anti-padrões, megaservice e wrong cuts. Outros anti-padrões não foram mencionados.

#2 - Sim, foi considerado megaservice, wrong cuts, shared persistence, shared libraries e microservice greedy. Dividimos os serviços em áreas do negócio (DDD – Bounded Context), e já pensamos em bancos de dados separados. No caso de não tomar medidas: Dependência cíclica, falta de conhecimento em comunicação assíncrona (filas) e tempo de desenvolvimento do projeto.

#3 - Sim, megaservice, wrong cuts, cyclic dependency, shared persistence, Inappropriate Service Intimacy, shared libraries, microservice greedy e Legacy organization. Não fazer microsserviço, que é o problema mais difícil de resolver da computação, todos os outros são mais simples, muitas vezes por usar a tecnologia adequada, montar um time bom, organizar o trabalho corretamente, preocupação com eficiência. Se não considerar todos AP não passe perto de MS. Se considerar de verdade, decidirá não passar perto. MS é anti pattern, engenheiros pensam assim, marketeiros não. APs só devem ser adotados com último recurso.

#4 - Hoje já tem identificado como evitar os antipadrões, como o antipadrão shared persistence. O planejamento é ter um mensageiro, ou seja nenhum microsserviço irá se comunicar com outro, para que se um estiver fora não ter problemas, só muda conexão entre o mensageiro e o microsserviço.

#5 - Sim, houve planejamento pra evitar megaservice, shared persistence.

#6 - Era utilizado de bom senso e experiência da equipe, de trabalhos anteriores sobre más práticas.

#7 - Sim, havia um megaservice detectado durante a migração, ele foi separado e foi necessário uma grande quantidade de código para a integração entre os dois.

13 - Quais foram as etapas para separar o monolito em microsserviços? Foi utilizado uma abordagem como DDD ?

#1 - Sim, a organização tem como princípio utilizar DDD como base para o desenvolvimento, e foi utilizado o DDD para fazer a separação do monolito.

#2 - Sim, alguns conceitos foram utilizados e também The Twelve-Factor App.

#3 - DDD causa tanto problema quanto microsserviço, e na verdade se tem um local onde o DDD não faz sentido é o microsserviço, como, quase sempre, OOP. OOP serve para gerenciar complexidade, DDD para gerenciar extrema complexidade, MS serve para tornar tudo pequeno e evitar complexidade (individual), não faz sentido misturar as coisas. As pessoas estão fazendo coisas aleatórias sem estudo real. Quem usa DDD está fazendo um monólito quebrado em partes e não MS, e não tem a menor ideia do que está fazendo. https://en.wikipedia.org/wiki/Dunning%E2%80%93Kruger_effect.

#4 - separar o serviço baseado em demanda, se um serviço tiver mais acesso, ou estiver usando mais recursos.

#5 - Não foi usado nenhuma abordagem específica, foi separar por negócios.

#6 - Foi pensado na arquitetura, tecnologias, formas de comunicação entre microsserviços foi aplicado event-drive (mensageria) foi aplicado DDD + Clean Architecture.

#7 - Quebrar por contexto de negócio, depois durante a migração foi detectado microsserviços distintos que trabalhavam sobre o mesmo domínio de dados e era necessário sincronização entre eles. Após entender esse problema foi iniciado um estudo sobre DDD para estruturar melhor em contexto de domínios.

14 - Qual foi a abordagem para separar o banco de dados?

#1 - Banco de dados já está separado, foi utilizado a mesma técnica para a separação DDD.

#2 - (DDD – Bounded Context) dos serviços envolvidos.

#3 - Se for adotar MS cada um deve ter seu próprio, caso contrário não é MS. Fazer isso significa que a aplicação terá que cuidar o que antes o DB já cuidava de graça pra você. Existem estratégias no monólito quando o DB não aguenta.

#4 -

#5 -

#6 - Foi usado ainda o banco antigo por um tempo, práticas de normalização do banco.

#7 - ver quais eram os padrões existentes pra separar banco no contexto de microsserviços. Desde o início já entendeu que deveria separar. isolamento entre microsserviços 99% das comunicações entre msa é usado rest api ou mensageria. Cada serviço tem seu domínio de informação e tem instâncias diferentes pra aumentar resiliência com a distribuição horizontal de dados e cada tenant tem seu schema específico de manutenção para ter isolamento.

15 - Foi pensado em alguma técnica de refatoração ou software para eliminar outros anti-padrões, que não sejam estes relacionados a microsserviços, durante a pré-migração?

#1 - Sim, o sistema foi reconstruído praticamente do 0, portanto foi previsto eliminar qualquer tipo de anti-padrão existente no sistema monolítico.

#2 - Recriamos do zero.

#3 - A refatoração bem feita resolve os problemas sem adotar MS. O MS é desculpa para refatorar. Resolva os problemas da aplicação sem adotar MS, que cria um novo problema que não tinha antes, SEMPRE (100%).

#4 - parte do sistema está em nuvem e parte na infra do sistema, quando o sistema foi projetado parte de dados que ficaria em nuvem foi alterado para estar em lado do cliente, o que acarretou em problemas de manutenção.

#5 - foi usado sonarQube e code review.

#6 - Foi tentado aplicar TDD porém devido aos prazos e sem experiência da equipe não foi possível seguir com essa prática.

#7 -

16 - Quais suas considerações a respeito da fase de pré-migração e migração, quais as dificuldades/barreiras vislumbradas e/ou lições aprendidas que poderiam ser úteis para futuras migrações para microsserviços?

#1 -

#2 - Problemas com deploy, comunicação entre serviços, autenticação e autorização.

***penso que poucos projetos realmente precisam dessa abordagem, principalmente se está sendo criado do zero, equipe de até 15 pessoas acho totalmente inviável.**

#3 - Não faça, quase ninguém precisa disso. Pode existir algum microsserviço pontual onde fizer sentido, mas não uma arquitetura de microsserviços, não algo antes de ter um bom monólito. Não use muletas.

#4 - estamos num caminho certo, até o momento não teve um grande problema, os problemas foram fáceis de resolver. problema de acesso a dados demanda muito retrabalho, poucos funcionários. Adicionar uma nova funcionalidade ou refatorar, as vezes perde-se tempo refatorando falta de documentação do sistema, dificuldade em atualizar.

#5 - Fazer um nivelamento de conhecimentos sobre microsserviços, sistemas distribuídos. Treinamento sobre as tecnologias que serão utilizadas. Mais pessoas envolvidas nas decisões sobre as tecnologias, mais testes. Após a implementação e início de utilização do sistema foi percebido uma má escolha de uma tecnologia e não havia mais tempo para reverter a situação.

#6 - Dificuldade pois não havia participado de um planejamento, conhecimento de arquitetura, engenharia de software. Lição seria estudar mais, consumir mais materiais

#7 - Faça da melhor maneira com a informação que você tem hoje. Você vai errar, e vai aprender e vai corrigir dessa forma foi encontrado problemas que trouxeram experiências boas, pra auxiliar outras equipes e servir como base para futuras implementações. Não esperar ter todo conhecimento do mundo pra começar

17 - Para concluir, em uma possível nova implementação, você consideraria desenvolver um software direto em microsserviços ou preferiria desenvolver em monolito e depois migrar? Quais as razões da sua escolha?

#1 - Desenvolveria em microsserviços direto, pensando em não ter um trabalho redobrado, ao desenvolver em monolito e depois ter que migrar.
As lições aprendidas durante a migração foram úteis para que eles possam desenvolver de forma a também evitar os anti-padrões.

#2 - Depende da equipe, se for uma equipe grande e com conhecimento prévio do projeto e realmente necessitar, utilizaria sem dúvidas, caso contrário não.

No meu caso, que sempre trabalhei em empresas de pequeno e médio porte, preferiria começar em um monólito sem dúvidas e migrar aos poucos.

#3 - Você não precisa de microsserviço e se precisar deve ser PROVADO, deve ter algo simples que se mostrou incapaz de atender a demanda quando feito certo, e TODOS os esforços foram feitos para resolver sem ele. Quase sempre a adoção de MS é assinar atestado de incompetência. Arquitetura de microsserviços é uma fad.

A pesquisa já tem viés. Se a pessoa estudar o assunto sem viés, ela não adota essa estratégia. Um exemplo é considerar que uma equipe de dezenas de pessoas (pode ser muitas) precisa de MS. O Linux (a maior base de código com milhares de contribuidoras)

não precisa, boa parte dos 50 sites mais acessados do mundo não precisam (nem considerei os que usam sem precisar de fato).

#4 - desenvolver primeiro em monolítico, agora tem-se noção do uso de shared libraries, por exemplo, e um entendimento macro do sistema. Devido a complexidade de microsserviços, no início seria mais difícil, um projeto inicial às vezes não tem-se uma ideia de coisas básicas como regras de negócio, que vão amadurecendo com o tempo. O investimento inicial também foi pouco, não seria possível no inicio desenvolver em MS.

#5 - depende da maturidade do time e uma equipe madura já tem bagagem pra trabalhar direto com microsserviços com equipes separadas. Se não está claro como o sistema vai funcionar, nem conhecimento suficiente em microsserviços o melhor é iniciar em monolítico. Se o conhecimento em relação ao negócio é muito maduro e avançado.

#6 - Desenvolver em microsserviço, por questão da escalabilidade e nem sempre desenvolver com boas práticas é o que possibilita você entregar produtos novos, talvez pensar em estratégias pra facilitar a vida do desenvolvedor seria o caminho mais assertivo.

#7 - desenvolver em monolítico primeiro. Com monolito seria muito mais fácil envolver o cliente pra usar a informação pra ensinar qual a melhor configuração pra resolver a necessidade dele. Eles começaram em MSA sem entender bem os contextos de negócio. Monolito é mais tranquilo.

Obrigado pela participação!