Strategies for Mitigating Microservice Anti-Patterns in the Pre-Migration of Monolithic Legacy Systems

Guilherme L. D. Villaca[1], Ivonei F. da Silva[1], Wesley K. G. Assunção[2], Rodrigo B. Rocha[1], Gabriel C. Fermino[1]

[1]PPGComp – Western Paraná State University (UNIOESTE)
Cascavel – PR – Brazil

[2]Department of Computer Science, North Carolina State University
Raleigh – North Carolina – USA

guidvillaca@gmail.com, wguezas@ncsu.edu
{ivonei.silva, rodrigo.rocha5}@unioeste.br
gabriel.cf2009@hotmail.com

Abstract. *This article analyzes strategies for mitigating antipatterns during the pre-migration phase of monolithic systems to microservices architectures. Using a multi-method research approach, the study examines the challenges and best practices involved in this transition. The research highlights that Domain-Driven Design (DDD) and the Strangler Fig Pattern are essential strategies for the gradual decomposition and migration of systems. Key success factors also include modularization, effective management of coupling, and the implementation of automated testing. The analysis emphasizes the importance of considering the specific context when selecting migration strategies. Overall, the study identified 25 strategies and concluded that these approaches, when applied in the pre-migration phase, can effectively mitigate antipatterns during the transition to microservices.*

1. Introduction

Monolithic legacy systems suffer degradation over time due to extensive maintenance, architectural violations, and inadequate design decisions [Wolfart et al. 2021, Macia et al. 2012]. To mitigate this degradation, one option is complete reconstruction using new technologies, but software modernization is a more economical approach [Mahanta and Chouta 2020, Wolfart et al. 2021], whether through refactoring, remodeling, or migration to modern paradigms such as microservices [Candela et al. 2016]. The migration to microservices, adopted by companies like Amazon and Netflix, involves decomposing systems into smaller services, each operating independently and communicating through lightweight APIs [Newman and Reisz 2020].

Despite its advantages, the migration from monoliths to microservices is challenging, introducing problems such as boundary conflicts and versioning difficulties. Moreover, microservices architectures may suffer from antipatterns, inadequate practices that arise during development and impact several software quality attributes, such as comprehensibility, testability, extensibility, reusability, and maintainability [Taibi et al. 2020, Li et al. 2021, Cerny et al. 2023]. The most common antipatterns in microservices include Nano-service, Mega-service, Duplicated services, Wrong cuts, Knot service, Content coupling, Tightly coupled services, Shared libraries, Hard-coded endpoint, Mi-

croservice greedy, Data-driven migration, Lack of communication standards among microservices, Inappropriate service intimacy, Shared persistence, and Legacy organization [Cerny et al. 2023]. This study investigates strategies to mitigate these antipatterns during the pre-migration phase of migrating legacy systems to microservices. While there are few studies on microservices antipatterns, none focus on monolith migration [Taibi et al. 2020, Carrasco et al. 2018]. This study aims to fill this gap by assisting professionals in more effectively planning migrations and preventing antipatterns from the beginning.

2. Background

2.1. Monolithic Architecture and Its Modernization

Monolithic architectures are rigid and highly coupled, making even small changes challenging and negatively impacting productivity, cost, and deployment [Jambunathan and Y. 2016]. Over time, these architectures may experience difficulties related to constant modifications, evolving requirements, an increase in defects, and design issues, which might affect performance and increase costs. In software engineering, it is recommended to continuously adapt or improve systems to keep them operational [Khadka et al. 2015]. Modernization, alongside maintenance and replacement, is essential to enhance maintainability, flexibility, and potentially reduce costs [Comella-Dorda et al. 2000]. The decision to modernize a monolithic system to a microservices architecture is typically considered when: (i) they become too large and complex to maintain, (ii) modularity and decentralization are crucial, and (iii) there are clear long-term benefits anticipated from such a transition [Kazanavičius and Mažeika 2019].

2.2. Microservices and Antipatterns

Microservices are sets of independent services, organized around business functions, with automated deployment and decentralized control of languages and data [Fowler 2014]. Each service should follow the single responsibility principle, focusing on a specific function [Knoche and Hasselbring 2019]. Although they bring benefits, microservices increase the complexity of distributed systems [Carrasco et al. 2018]. Common problems, known as antipatterns, arise during migration due to a lack of knowledge about best practices, impacting attributes such as comprehensibility and maintainability [Taibi et al. 2020].

Antipatterns, such as wrong cuts, cyclic dependency, and shared persistence, have been identified in microservices [Taibi et al. 2020], with solutions focused on the microservices context, without considering the migration process from monolithic systems. Studies suggest that some pitfalls, such as mega-service and shared libraries, need to be addressed in the pre-migration phase [Carrasco et al. 2018].

2.3. Phases of Microservices Migration

The migration from monolithic systems to microservices follows four phases: (i) Understanding the monolithic system, (ii) Defining the microservices architecture, (iii) Executing the transformation, and (iv) Post-migration monitoring [Wolfart et al. 2021]. This study focuses on the first two phases, considered pre-migration, as the execution and monitoring phases are beyond the scope of this research.

3. Methodology

The research was conducted in three Steps, as illustrated in Figure 1. Step 1, which took place between 2020 and 2022, aimed to identify strategies for mitigating antipatterns in microservices before migrating monolithic systems. A Systematic Literature Mapping (SLM) was carried out with the central research question: *What strategies adopted during the pre-migration phase of legacy systems migration to microservices can mitigate antipatterns?* This question was analyzed using the PICO framework [Huang et al. 2006]. Alongside the SLM, a rapid review of grey literature was conducted to validate the data [Kamei et al. 2021]. Subsequently, interviews were conducted with industry professionals to gather insights into their experiences and perspectives on the migration of monolithic systems to microservices. These interviews were systematically planned and guided by the Goal-Question-Metric (GQM) approach [Basili and Rombach 1988, Solingen and Berghout 1999], ensuring a focused exploration of the challenges and best practices in the migration process. Nine experts from industry were selected based on their practical experience with microservices migration.

Steps 2 and 3, completed in 2024, involved a Systematic Literature Review (SLR) focusing on both scientific and grey literature. In Step 2, the goal was to investigate how existing refactoring strategies contribute to mitigating antipatterns during the pre-migration phase, with data sourced from grey literature, including blogs, publications, and videos. For Step 3, the focus shifted to exploring how refactoring strategies applied before migration aid in the transition to microservices, with data collected from a review of scientific literature, including conference papers and research. The scientific review utilized databases such as ACM Digital Library, IEEE Xplore, Scopus, and SpringerLink, where quality was assessed based on context clarity and how well the research questions were addressed. For the grey literature, quality was measured by the practical relevance and reputation of the sources, using tools like Google and arXiv for data collection.

The search string used was: *(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")*. A total of 847 studies were identified. Inclusion criteria were defined to ensure that the selected works addressed strategies implemented prior to the migration from monolithic systems to microservices. These included detailed descriptions of strategies, practical migration experiences, and the provision of guidelines or step-by-step procedures for the process. Exclusion criteria were applied to eliminate studies that did not align with the research scope, such as systems originally developed in microservices architecture, works outside the migration context, and those that did not address strategies prior to the migration process. After applying the inclusion and exclusion criteria, 135 studies were selected for the final analysis.

3.1. Thematic Analysis

All studies employed thematic analysis as a research method, a widely used approach for identifying, analyzing, and reporting patterns in qualitative data. This method allows for organizing and describing data in detail, as well as interpreting various aspects of the investigated topic. Thematic analysis can be applied in different theoretical frameworks, whether essentialist or realist, capturing participants' experiences and mean-

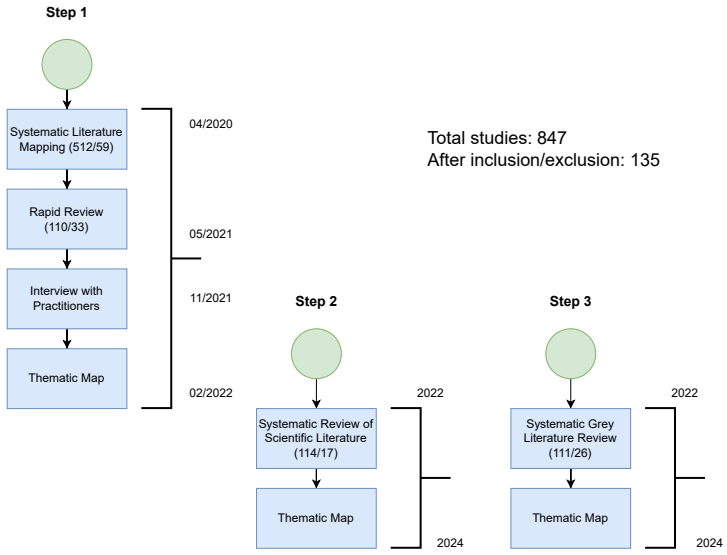

Figure 1. Research Timeline

ings [Cruzes and Dyba 2011]. In this study, the information was coded and grouped into broader themes.

The methodological differences between the steps reflect the complexity of system migration, highlighting the need for multiple perspectives. The analysis of the adopted methodologies emphasizes the importance of using diverse sources and approaches to build a more comprehensive understanding of the migration of monolithic systems to microservices.

4. Results and Discussions

The thematic analysis followed a similar process across all studies, with stages of coding, grouping into themes, and analyzing the relationships between them. However, the data sources and identified themes varied. Below, we describe the results of each step.

In Step 1, the main themes identified were modularity, data, and organizational culture. As a result, ten strategies to mitigate antipatterns were found, with a focus on DDD and the Strangler Fig Pattern. In Step 2, the main themes were decomposition, modularity, and data. The analysis identified 13 refactoring strategies and techniques, which were mapped based on the identified themes. In Step 3, nine strategies related to the discussed themes were identified.

The identified strategies are detailed in the spreadsheet [Villaca 2024]. Modularity emerged as a central theme across all steps, reinforcing the importance of breaking down the monolithic system into smaller components before migration. Additionally, DDD and the Strangler Fig Pattern were consistently mentioned across all three steps as effective refactoring strategies. The three steps of the study resulted in some duplicated strategies (see Table 1), and after removing these duplications, the total number of strategies is 25.

Table 1. Themes & Codes

Theme	Code	Step 1	Step 2	Step 3
Modularity	Domain-Driven Design	✓	✓	✓
	Strangler Fig Pattern	✓	✓	✓
	Measure Coupling and Cohesion	✓		
	Backlog	✓		
	Bounded Context		✓	✓
	Model-Driven Design		✓	✓
	Anti Corruption Layer			✓
	Branch by Abstraction		✓	
	Glue Code		✓	✓
	Aggregates		✓	
	Modular Monolith		✓	
	UI Composition		✓	
Data	Group Entities	✓		
	Classify DB in Business SubSystem	✓		
	Data First	✓		
	CQRS		✓	
	Data Access Object		✓	
Organizational Culture	Clean Architecture	✓		
	Twelve Factor App	✓		
	Evolvability Assurance	✓		
Testability	Test-Driven Development			✓
Intermediate Architecture	Service Oriented Architecture			✓
	Hexagonal Architecture			✓
Decomposition	Sidecar		✓	
	Functional Decomposition		✓	

4.1. Themes

The codes identified in the three steps of this research were grouped and transformed into themes [Cruzes and Dyba 2011], which are: modularity, data, organizational culture, testability, intermediate architecture, and decomposition, as presented in Table 1 and the thematic map in Figure 2. In the table, each strategy is also related to a column that indicates the step in which it was identified.

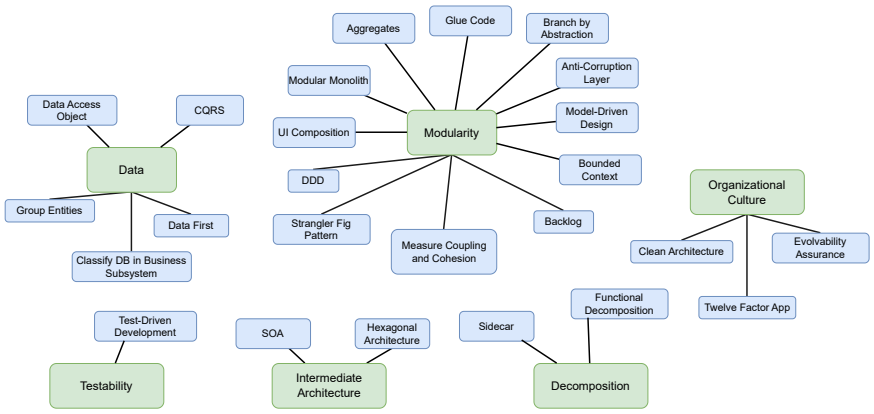

Figure 2. Thematic map of strategies to mitigate antipatterns in microservices

4.2. Discussion

This study addresses a central challenge in modern software engineering: the migration from monolithic systems to microservices architectures. Focusing on the pre-migration phase, it highlights the importance of applying specific strategies to mitigate antipatterns,

facilitating a more effective transition. In addition to acknowledging the complexity involved in the process, the study directly contributes to practical problems faced by developers and companies, identifying practical solutions to prevent common difficulties and ensure a successful migration.

4.2.1. Strategies to Mitigate Antipatterns

The association between strategies and antipatterns was made by the authors of this study during Steps 1, 2, and 3 and are listed in Table 2. In this analysis, we identified a set of strategies and techniques that, if applied in the pre-migration phase can minimize the occurrence of antipatterns. Among the key findings, the importance of modularizing the monolithic system before migration stands out. Breaking down the system into smaller, independent components facilitates the subsequent decomposition into microservices. The research identifies DDD as a powerful tool for achieving modularity. By focusing on business domains and dividing the system into Bounded Contexts, DDD helps avoid antipatterns such as Mega Service and Wrong Cuts [Fan and Ma 2017, Silva et al. 2019, Newman and Reisz 2020, Lavann et al. 2021, Assouline and Grazi 2017, Samokhin 2018].

Another highlighted technique is the Strangler Fig Pattern, which proposes a gradual and incremental migration, proving effective in reducing risks and migration complexity [Kornilov 2020, Sitnikova 2021, Goel 202, Richardson 2016]. Data management is also a crucial point in the process. The research emphasizes the importance of analyzing, classifying, and organizing data before migration, ensuring that each microservice has a well-defined and independent data structure [Hubers 2021, Santos and Rito Silva 2020, Baeldung 2018]. The CQRS (Command Query Responsibility Segregation) pattern, especially recommended for complex domains, allows for the selective application of Bounded Contexts in DDD. When combined with event sourcing, CQRS significantly improves application organization [Gonçalves 2023].

Additionally, the research emphasizes the implementation of automated tests during the pre-migration phase. Comprehensive testing ensures code quality and helps identify issues early, facilitating migration and preventing failures [Martin 2023]. Another aspect addressed is the use of intermediate architectures, such as Service-Oriented Architecture (SOA), which can serve as a gradual step in the transition to microservices [AWS 2024].

The research demonstrates the importance of pre-migration planning as a fundamental step for successful adoption of microservices. The identified strategies offer a practical guide for developers and architects aiming to execute this transition efficiently and securely, minimizing risks and costs associated with migration.

4.3. Study Limitations

This study contributes to software migration but has limitations. Some strategies are only superficially explored, with limited practical examples, and validation relies heavily on literature rather than real-world case studies. The focus on general migration to microservices, without considering specific application domains, may overlook domain-specific challenges. Organizational factors like company culture and resistance to change are not

Table 2. Strategies & Antipattern

Strategies	Antipatterns
Domain-Driven Design Bounded Context Model-Driven Design Aggregates	Mega Service, Duplicated services, Wrong cuts, Lack of communication standards among microservices
Strangler Fig Pattern Anti-Corruption Layer	Mega Service and Big Bang
Measure Coupling and Cohesion	Shared Libraries
Backlog	Mega Service and Microservice Greedy
Branch By Abstraction	Big Bang
Glue Code	No API-Gateway, Too many standards, Inadequate techniques support, Golden Hammer
UI Composition	Ambiguous Service
Classify DB in Business Subsystem Functional Decomposition Group Entities Data first Sidecar	Shared Persistence
CQRS Data Access Object	Data-Driven Migration
Clean Architecture Twelve Factor App and Evolvability Assurance	Shared Libraries and Cyclic Dependencies
Test-Driven Development	Insufficient Monitoring
Service Oriented Architecture Modular Monolith	Mega Service e Wrong cuts
Hexagonal Architecture	Mega Service, Wrong cuts, No API-gateway, Big Bang

deeply examined, despite their importance to migration success. The research was conducted individually, which could introduce bias, and the nine professional interviews, while useful, offer limited conclusions. The subjectivity in constructing the thematic map may have also affected strategy categorization. These limitations suggest the need for further research to deepen the understanding of monolithic to microservices migration.

5. Future Work

Several areas still require further exploration to improve knowledge on migrating from monolithic systems to microservices. Key challenges include managing inter-service communication, ensuring data consistency in distributed environments, and handling failures. Post-migration challenges, such as monitoring, scalability, security, and updates, also need further investigation. Detailed methodological steps, including case studies and real-world examples, would aid in the application of strategies. Comparative studies of different strategies and refactoring techniques, considering factors like system size and team experience, would be valuable.

In addition to technical challenges, it is essential to study how organizational factors—such as company culture, team structure, and resistance to change—affect migration success. The impact on development teams, including the learning curve, training, and changes in work processes, should also be explored. Addressing these gaps will provide a more comprehensive guide for companies and developers, supporting more efficient and sustainable migrations to microservices.

6. Conclusion

This research explored strategies to mitigate anti-patterns in microservices before migrating from monolithic systems to microservice architectures, with an emphasis on modularization as a central point. Dividing the monolithic system into smaller, independent components, using techniques such as DDD and the Strangler Fig Pattern, facilitates the subsequent decomposition into microservices. This approach minimizes the risk of failures and accelerates the migration process, making it more manageable. In addition to modularization, data management also emerges as a crucial aspect. Analyzing, classifying, and organizing data before migration is essential to ensure that each microservice operates with a well-defined and independent data structure, avoiding coupling issues and inconsistencies.

Another key point is testability, underscoring the need to implement automated tests throughout the migration process to ensure code quality and prevent failures. This research demonstrates that the pre-migration phase should not be neglected, as applying appropriate strategies during this phase can mitigate antipatterns in microservice systems.

7. Acknowledgment

We would like to thank the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES)- Programa de Excelência Acadêmica (PROEX) - Brasil for Financial Support.

References

Assouline, P. and Grazi, V. (2017). Perspective on architectural fitness of microservices. https://www.infoq.com/articles/Microservices-Architectural-Fitness/.

AWS (2024). Padrão de figo strangler - aws orientação prescritiva. https://docs.aws.amazon.com/pt_br/prescriptive-guidance/latest/cloud-design-patterns/strangler-fig.html.

Baeldung (2018). The dao pattern in java. https://docs.aws.amazon.com/pt_br/prescriptive-guidance/latest/cloud-design-patterns/strangler-fig.html.

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

Candela, I., Bavota, G., Russo, B., and Oliveto, R. (2016). Using cohesion and coupling for software remodularization. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 25:1 – 28.

Carrasco, A., van Bladel, B., and Demeyer, S. (2018). Migrating towards microservices: migration and architecture smells. pages 1–6.

Cerny, T., Abdelfattah, A. S., Maruf, A. A., Janes, A., and Taibi, D. (2023). Catalog and detection techniques of microservice anti-patterns and bad smells: A tertiary study. *Journal of Systems and Software*, 206:111829.

Comella-Dorda, Wallnau, Seacord, and Robert (2000). A survey of black-box modernization approaches for information systems. In *Proceedings 2000 International Conference on Software Maintenance*, pages 173–183.

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

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

Fowler, M. (2014). Microservices a definition of this new architectural term. `http://martinfowler.com/articles/microservices.html`.

Goel, A. (202). Choosing the right strategy to migrate your monolithic application to a microservices-based architecture. `shorturl.at/deJV7`.

Gonçalves, M. M. (2023). Cqrs (command query responsibility segregation) em uma arquitetura de micro serviços. `https://medium.com/@marcelomg21/cqrs-command-query-responsibility-segregation-em-uma-arquitetura-de-micro-servi%C3%A7os-71dcb687a8a9`.

Huang, X., Lin, J., and Demner-Fushman, D. (2006). Evaluation of pico as a knowledge representation for clinical questions. *AMIA ... Annual Symposium proceedings. AMIA Symposium*, pages 359–363.

Hubers, T. (2021). How big is a microservice? `https://medium.com/geekculture/the-size-of-a-microservice-b9e6bc90475`.

Jambunathan, B. and Y., K. (2016). Multi cloud deployment with containers. 8:421–428.

Kamei, F., Wiese, I., Lima, C., Polato, I., Nepomuceno, V., Ferreira, W., Ribeiro, M., Pena, C., Cartaxo, B., Pinto, G., and Soares, S. (2021). Grey literature in software engineering: A critical review. *Information and Software Technology*, 138:106609.

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

Khadka, R., Shrestha, P., Klein, B., Saeidi, A., Hage, J., Jansen, S., van Dis, E., and Bruntink, M. (2015). 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)*, pages 477–486.

Knoche, H. and Hasselbring, W. (2019). Drivers and barriers for microservice adoption - a survey among professionals in germany. 14:1–35.

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

Lavann, EdPrice, and neilpeterson (2021). Migrate a monolith application to microservices using domain-driven design. `https://docs.microsoft.com/en-us/azure/architecture/microservices/migrate-monolith`.

Li, S., Zhang, H., Jia, Z., Zhong, C., Zhang, C., Shan, Z., Shen, J., and Babar, M. A. (2021). Understanding and addressing quality attributes of microservices architecture: A systematic literature review. *Information and Software Technology*, 131:106449.

Macia, I., Garcia, J., Popescu, D., Garcia, A., Medvidovic, N., and von Staa, A. (2012). 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*, AOSD '12, page 167–178, New York, NY, USA. Association for Computing Machinery.

Mahanta, P. and Chouta, S. (2020). *Translating a Legacy Stack to Microservices Using a Modernization Facade with Performance Optimization for Container Deployments*, pages 143–154.

Martin, M. (2023). Test driven development. `https://martinfowler.com/bliki/TestDrivenDevelopment.html`.

Newman, S. and Reisz, W. (2020). Sam newman: Monolith to microservices. `https://www.infoq.com/podcasts/monolith-microservices/`.

Richardson, C. (2016). Refactoring a monolith into microservices. `https://www.nginx.com/blog/refactoring-a-monolith-into-microservices/`.

Samokhin, V. (2018). Why microservices fail. `https://hackernoon.com/why-microservices-fail-6cdc006f9540`.

Santos, N. and Rito Silva, A. (2020). A complexity metric for microservices architecture migration. In *2020 IEEE International Conference on Software Architecture (ICSA)*, pages 169–178.

Silva, H., Carneiro, G., and Monteiro, M. (2019). Towards a roadmap for the migration of legacy software systems to a microservice based architecture. pages 37–47.

Sitnikova, A. (2021). Monolith vs microservices: Everything you need to know. `https://bambooagile.eu/insights/monolith-vs-microservices/`.

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

Taibi, D., Lenarduzzi, V., and Pahl, C. (2020). Microservices anti-patterns: A taxonomy. *Microservices: Science and Engineering*, pages 111–128.

Villaca, G. (2024). Strategies to mitigate antipatterns in microservices.

Wolfart, D., Assunção, W. K. G., da Silva, I. F., Domingos, D. C. P., Schmeing, E., Villaca, G. L. D., and Paza, D. d. N. (2021). Modernizing legacy systems with microservices: A roadmap. In *Evaluation and Assessment in Software Engineering*, EASE 2021, page 149–159, New York, NY, USA. Association for Computing Machinery.