Are we speaking the industry language? The practice and literature of modernizing legacy systems with microservices

Thelma E. Colanzi
Aline M. M. M. Amaral
State University of Maringá
Maringá, PR, Brazil

Wesley K. G. Assunção
Pontifical Catholic
University of Rio de Janeiro
Rio de Janeiro, RJ, Brazil

Arthur C. Zavadski
Douglas Tanno
State University of Maringá
Maringá, PR, Brazil

Alessandro Garcia
Carlos Lucena
Pontifical Catholic
University of Rio de Janeiro
Rio de Janeiro, RJ, Brazil

ABSTRACT

Microservice architecture has gained much attention in the last few years in both industry and academia. Microservice is an architectural style that enables developing systems as a suite of small loosely coupled, and autonomous (micro)services that encapsulate business capabilities and communicate with each other using language-agnostic APIs. Despite the microservice adoption for modernizing legacy systems, few studies investigate how microservices are used in practice. Furthermore, the literature still scarce on presenting studies on why and how the modernization is conducted in practice in comparison to existing literature on the subject. Thus, our goal is to investigate if industry and academy are speaking the same language concerning the modernization of legacy systems with microservices, by means of a rigorous study on the use of microservices in the industry. For doing so, we design a survey to understand the state of practice from the perspective of a modernization process roadmap derived from the literature. In this paper, we report the results of a survey with 56 software companies, from which 35 (63.6%) adopt the microservice architecture in their legacy systems. Results pointed out the most expected benefits that drive the migration to microservices are easier software maintenance, better scalability, ease of deployment, and technology flexibility. Besides, we verified, based on a set of activities defined in the modernization process, that the practitioners are performing their migration process according to the best literature practices.

CCS CONCEPTS

• **Software and its engineering** → **Software evolution**; **Software architectures**; • **General and reference** → *General literature*; • **Computer systems organization** → **Cloud computing**.

KEYWORDS

Software Re-engineering, Software Migration, Microservices

ACM Reference Format:
Thelma E. Colanzi, Aline M. M. M. Amaral, Wesley K. G. Assunção, Arthur C. Zavadski, Douglas Tanno, Alessandro Garcia, and Carlos Lucena. 2021. Are we speaking the industry language? The practice and literature of modernizing legacy systems with microservices. In *15th Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS '21), September 27-October 1, 2021, Joinville, Brazil.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3483899.3483904

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SBCARS '21, September 27-October 1, 2021, Joinville, Brazil
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8419-3/21/09…$15.00
https://doi.org/10.1145/3483899.3483904

1 INTRODUCTION

In a software system with a monolithic architecture, all layers such as user interface, business rules, and database management are developed as one single piece [32]. The monolithic architecture was the standard decomposition strategy in prevailing architectural styles for many years. However, at some point, monoliths often become legacy systems with decayed architectures and obsolete technologies. These factors imply that any change usually impacts a large part of the system [19]. A possible solution being increasingly considered by companies is modernizing their systems by extracting legacy features into microservices to achieve structural, functional, and data decoupling [10, 13]. Microservice-based systems are developed as a suite of small, loosely coupled, and autonomous microservices. These microservices encapsulate business capabilities, their owned data, and communicate with each other using language-agnostic APIs [32]. The modernization of legacy systems with microservices also enables digital transformation and inclusion of innovation [22].

In our context, the modernization comprehends the process starting from the identification of driving forces governing the microservice adoption. The process takes into account all the transformations required in the legacy's functionalities, the database (or any data storage schema), and those related to operations to a microservice architecture. Finally, the modernization process also considers the verification, validation, and monitoring of the system following now the microservice architecture decomposition [43].

The topic of modernizing legacy systems with microservices has been investigated from different perspectives. Some pieces of work present approaches, techniques, and methods for guiding some modernization activities [11, 12, 14, 35, 43]. They mostly focus on dealing with the structure of the source code – i.e., the legacy's modules and their dependencies. However, these existing solutions often neglect the data model structure [8], which is also of paramount importance. For instance, the data model knowledge is required for making key decisions on the microservices' decomposition.

Other studies report on the modernization with microservices in practice, usually considering only specific activities. These studies do not cover the entire process [1, 5, 17, 18, 26, 28, 29, 38]. Furthermore, some existing surveys with practitioners focus on exploratory studies [6, 7, 10, 13, 40, 42]. However, gaps between the visions and

the reality of microservices need to be empirically investigated [44] and the body of existing studies about the modernization with microservices has various unaddressed limitations. Most studies only partially investigate the modernization process, being restrict to a few activities, most often concerning the preliminary identification of potential microservices in the legacy system. A recent proposal for conducting the modernization was synthesized from existing studies [43]; however, it lacks empirical evaluation. In addition, the studies mainly deal with source code of legacies; little is known about the modernization challenges regarding the data model decoupling through the microservice decomposition [10].

Characterizing practical evidence of why and how the whole modernization process governing the system's artifacts, including data model schemes, is conducted is of paramount importance to guide new studies in the field. This knowledge would also serve as an experience report for companies willing to start the modernization of their legacies, and feed tool builders with practical aspects that need to be taken into account.

Motivated by the aforementioned limitations, the goal of this work is to characterize the modernization process [43], described in Section 2, from the perspective of practitioners from software companies, who observed the need of modernizing legacy systems by following the microservice architectural style. We inquired the practitioners about their experience with the whole process starting from the driving forces for the modernization to the monitoring of the microservices. We also asked questions to confirm which are the criteria and granularity used for splitting the legacy, technologies, and tools used in practice in comparison to the literature. Furthermore, given the few studies on data decoupling, our study dedicated a set of questions about the role of the data entities and persistence in the modernization process, in order to investigate the alignment between the practice and the literature.

The results revealed that industry and academia are speaking the same language regarding the modernization of legacy systems with microservices. More specifically, the most common driving forces match with the microservice advantages claimed in the literature. The participating companies usually plan the modernization of the legacy system before executing the migration. The way of conducting the migration process is also aligned with the recommendations of performing an incremental migration, extracting loosely coupled microservices with single-responsibility and isolated database [32]. Most companies also monitor several aspects of the microservices extracted from the legacy system. Furthermore, the results pointed out that there is room for improvement in areas such as (i) coupling between microservices induced by database structure, shared persistent libraries, and central database usage; (ii) more adequate microservice granularity by analyzing database transactions and (iii) performance level via microservice database partitioning.

The main contributions of this work are: (i) our survey provides a comprehensive characterization of the industrial practice on modernizing legacy systems with microservices; (ii) our results point out that the industrial practices for migrating to a microservice architecture are aligned with the existing research, focusing on practical needs of industry; (iii) our findings reinforce that the modernization process roadmap [43] is a good starting point for onboarding companies willing to start a migration to a microservice architecture; and, (iv) we have identified research opportunities regarding database aspects as both an input to the modernization process and a factor to reduce microservices coupling at runtime.

2 MODERNIZATION PROCESS ROADMAP

In our recent work, we present a process for modernizing legacy systems with microservices [43]. The process was distilled from 62 primary sources following a systematic mapping study, representing the theoretical knowledge produced in academia from technical reports, case studies, and surveys. The modernization process has eight activities, which are grouped in four phases, namely Initiation, Planning, Execution, and Monitoring (see Figure 1). Next we briefly describe the activities of this process.

1. Analyze the driving forces: this activity is conducted to guide companies to identify/understand the limitations faced with the legacy system. Then, all stakeholders can reason and decide whether a modernization with microservices is a feasible solution. The driving forces can be related to technical, operational, and organizational aspects. This activity is crucial to avoid unsuccessful modernization and waste of resources.

2. Understand the legacy system: this activity aims at analyzing the legacy system to understand its implementation, architecture, features, modules, how these parts interact with each other.

3. Decompose the legacy system: the goal of this activity is to split the legacy system in small and cohesive parts. These parts are the basis for the microservices.

4. Define the microservice architecture: here practitioners define a microservice architecture with the design decisions to meet functional and non-functional properties. A microservice architecture describes the microservices, their APIs, strategy of service discovery, and communications. The goal is to pack user interfaces, business logic, and databases in a single microservice.

5. Execute the modernization: in this activity, the transformation of the implementation artifacts takes place.

6. Integrate the microservices and the legacy: considering an incremental modernization (strangler pattern [13]), the goal here is to integrate the microservices and the legacy system.

7. Verify and validate the microservices: activity responsible for verifying and validating whether the microservices were developed and integrated accordingly. The legacy parts of the monolithic system can be used as oracle for the testing activity.

8. Monitor the microservices and infrastructure: this activity focuses on constantly checking the behavior of the microservices. Monitoring relies on service availability, bottlenecks, performance, use of infrastructure resources, etc. The pieces of information generated here can serve as a basis to evaluate the success of the modernization, as for example, in comparison to the driving forces.

3 STUDY DESIGN AND EXECUTION

Our empirical study consists in a survey conducted in the industrial scenario, relying on guidelines presented by Linaker et al. [25]. This survey was conceived in order to answer the research question (RQ): ***Are the practice and literature of the modernization of legacy systems with microservices aligned?***

This main RQ was divided into the three following sub-RQs, to analyze the survey's results against the literature, as follows:

- RQ1. Why do companies migrate monolithic legacy systems to microservices? To answer this sub-RQ we analyzed which are the driving forces that motivate the software companies to migrate to a microservice architecture.

- RQ2. How do companies perform the migration of monolithic legacy systems to microservices? In this question we analyzed how the migration process to a microservice architecture has been conducted by software companies in terms of the activities of the modernization process roadmap [43].

- RQ3. What are the aspects of data persistence considered in the modernization of legacy systems with microservices? To answer this question we analyzed how the database is structured in the microservice-based systems and if its structure and transactions were used as input to the modernization process.

We designed and applied the survey questionnaire using Google Forms. The questionnaire encompassed closed-ended and open-ended questions about the modernization of legacy systems using microservices. Taking into account the RQs to be answered in this work, the questionnaire[1] contained questions about (i) the participant and software company characterization, (ii) technical and organizational questions regarding the use of microservices to modernize legacy systems, answered only by participants whose companies adopt microservice architectures, and (iii) questions directed to participants whose companies do not adopt microservices, aiming at knowing their intention of using microservices to modernize legacy systems in the future. The options for the second group of questions were extracted from the literature [2, 3, 7, 10, 43].

The population defined for the study was composed of software engineers/practitioners that work at software companies. The survey was spread on social networks and by direct e-mail with individuals belonging to the population addressed that are contacts of the authors. The questionnaire circulated over one month and, we received only one answer per company.

3.1 Characterization of Participants

We received 56 answers of practitioners from different software companies. These companies are in three different countries: Brazil (54), United States of America (1), and England (1). Most of them (96.4%) are distributed across seven states of Brazil, namely Paraná, São Paulo, Santa Catarina, Distrito Federal, Espírito Santo, Rio de Janeiro, and Rio Grande do Norte. The participants are mainly software developers: 35 (62.5%). Other roles also mentioned were: software engineer 4 (7.1%), software architect 5 (8.9%), coordinator 3 (5.4%), project manager 3 (5.4%), tester 2 (3.8%), director 2 (3.8%), chief executive officer 1 (1.8%), and technology manager 1 (1.8%).

Regarding the organizational structure, 31 (55.4%) companies defined their hierarchical structure considering the business area, 23 (41.7%) used the development area, and only 2 (3.6%) considered both areas to arrange their organizational structure. Most companies (33 - 58.9%) had their teams working in different locations before COVID-19 pandemic, while 23 (41.1%) had all team members working together in the same locations. These numbers show that it is common for companies using the microservice architecture having working teams allocated in different locations, as the division of teams responsible for each microservice is facilitated, corroborating some findings from the literature [23, 39].

The questionnaire characterization section ended with a question about the use of microservices in the companies. The data collected demonstrate a positive scenario to the adoption of this architectural style with 35 (63.6%) of the companies using microservices in their projects, while 21 (36.7%) still do not use this style.

3.2 Companies not using Microservices

The survey aims at analyzing the use of microservices in software companies. Thus, Section 4 presents only results regarding companies using the microservice architectural style. In this section, we present a brief analysis of the 21 (36.7%) companies that do not use this architectural style.

We asked three questions to these participants. The first question was whether they have knowledge about microservice architecture, for which 17 (81.0%) answered "yes" and 4 (19.0%) answered "no". It is important to observe that, even not using microservices, most participants stated having knowledge about this architectural style. The participants were also asked about the architectural pattern used in their companies, the answers demonstrate that 20 (95.2%) companies use monolithic architecture, while 1 (4.8%) uses hexagonal architecture. Finally, we asked the participants about the feasibility of their companies to migrate to microservice architecture. Nine (42.9%) stated "yes" and 7 (33.3%) stated "no", demonstrating a draw for the future use of this architectural style.

4 RESULTS AND ANALYSIS

Firstly, we asked to participants which activities of the modernization process roadmap [43] are/were performed in their companies for modernizing legacy systems with microservices. Only 6 out of 35 (17.2%) participants answered that no activity was performed. However, for 5 of them it was not a modernization process, but the system has already been developed using microservices. The other participants (29 out of 35, 82.8%) selected several activities as can be seen in Figure 1, where we present (inside the circles) the number of times each activity was selected in the survey. The two most common activities refer to the modernization planning: *Understand the legacy system* and *Decompose the legacy system*. Both activities were performed by 74.3% of the software companies. Other four activities also selected by most participants were: *Integrate the microservices and the legacy* (68.6%), *Analyze the driving forces* (62.9%), *Define the microservice architecture* (60.0%), and, *Monitor the microservices/infrastructure* (60.0%).

The two least selected activities were *Verify and validate the microservices* (54.3%) and *Execute the modernization* (48.6%). For the former, it is known that testing microservice-based systems is more complex than testing monolithic systems [37]. However, we infer that for both activities there was a misunderstanding, because the participants argued that their systems have a microservice architecture, thus they have modernized their legacy systems, which is the goal of the latter activity, and companies usually perform at least unit and integration testing.

[1] Available at: https://doi.org/10.5281/zenodo.5260058

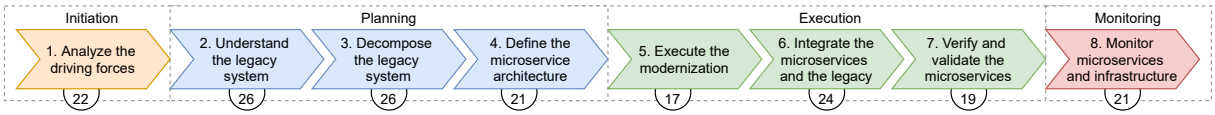

Figure 1: Roadmap Activities for modernizing legacy systems with microservices and their frequency of realization

Participants could add activities in addition to those encompassed by the modernization process. The participant P4 added an activity related to assess the migration feasibility. He argued that it is needed *"to assess if the migration is worthwhile or if building a microservice-based layer to integrate with the legacy system is enough to enable scalability and appropriate infrastructure use."*

Given the frequency of each activity, we observe that most activities of the modernization process roadmap are in fact performed by companies during the modernization of their legacy systems, reinforcing that such a roadmap can serve as a starting point and guide to the modernization process with microservices.

4.1 Modernization Process Roadmap Activities

In what follows, we present the findings extracted from the survey answers organized by activity.

1. Analyze the driving forces. In a multiple-choice question, the participants were asked about which were the driving forces to the modernization of their legacy systems with microservices. The 11 driving forces identified in the modernization process roadmap were chosen by the participants. No additional driving force was cited in the open field Others. Table 1 presents these driving forces sorted by the most mentioned ones.

Results point out that several types of driving forces motivate the modernization under different perspectives, namely operational, technical, and organizational (last column of Table 1). Also, some participants cited driving forces from different perspectives. There is not a dominant perspective that motivates the modernization with microservices, what points out that software companies aim at the benefits of microservices related to technical, operational and organizational aspects of software development.

Interestingly, the three most common driving forces are the same identified in the literature [13, 43]: *Easier maintenance and evolution*, *Optimized scalability*, and *Independent and automated deploy*. These forces were cited by more than 70% of participants. *Technology flexibility* and *Reduced time to market* were also pointed by more than 50% of participants as motivation to adopt microservices.

Table 1: Driving forces

Driving Force	Total (%)	Perspective
Easier maintenance and evolution	30 (85.7%)	Technical
Optimized scalability	26 (74.3%)	Operational
Independent and automated deploy	25 (71.4%)	Operational
Technology flexibility	23 (65.8%)	Technical
Reduced time to market	18 (51.4%)	Organizational
Loosely coupled services	16 (45.7%)	Technical
Independence of teams	14 (40.0%)	Operational
Agility enabler	12 (34.3%)	Organizational
Cohesive services	12 (34.3%)	Technical
Easier reuse	8 (22.9%)	Technical
Infrastructure facilities	7 (20.0%)	Operational

In summary, it is noticeable that supporting operational tasks are the most common driving forces to adopt microservices, in addition to technical benefits of microservices, e.g., easier maintenance and evolution task and use of different technologies [7, 9, 31, 35, 45].

2. Understand the legacy system. One of the most cited driving forces is optimized scalability as presented in Table 1. In addition, this criterion and other such as requirements, reusability and coupling (see Figure 2) are the most used when companies decide to extract microservices. It is important to observe that to use these criteria is mandatory to understand the requirements and how they were located in the code. Hence, identifying the features is a task realized by most companies. These results are corroborated by studies as [4], which stated that the first task to understand the legacy system is to list all the features of the system. Besides, according to [2, 36], the following task is the identification of the implementation artifacts of each feature, to subsequently decouple/extract them as microservices, which is known as feature location. Several techniques are used to perform these tasks as presented in the modernization process roadmap, an example was mentioned by the participant P43, which uses the Domain Driven Design (DDD) to identify the main features of the system.

3. Decompose the legacy system. One important decision to migrate to microservices is the strategy adopted by the software companies. Considering that, we asked to the participants how this migration occurred, 18 (51.4%) companies performed an incremental migration, where some projects or system modules were implemented using this architecture, in accordance with the literature recommendation [20]. Besides, 15 (42.9%) companies migrated their entire systems immediately to microservices, and 2 (5.7%) use both strategies, according to their needs.

In another question, we asked the participants how their legacy systems were decomposed into microservices. Twenty (57.1%) companies extracted only one microservice from the legacy system at a time. For instance, P27 stated that they perform an incremental domain-driven decoupling of the legacy system applying the strangler pattern [13], which allows an incremental transformation of the monolithic legacy system by replacing a particular functionality with a new microservice. In addition, 18 (51.4%) companies have decomposed their systems by business capability, defining the responsibility of the microservice. Finally, 7 (20.0%) companies decomposed their systems based on system operations and their data, whereas 2 (5.71%) did it based on UML diagrams.

We also asked participants about the criteria used when deciding to extract microservices. The results showed that 85.7% of companies considered scalability as a relevant criterion, probably because a good level of system scalability is enabled by improving and expanding a microservice without the need to modify other parts of the system, accelerating processes and saving costs, time and work [30]. Requirements and reusability were also identified as

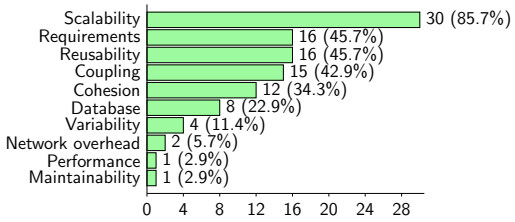

Figure 2: Criteria used for the microservice extraction.

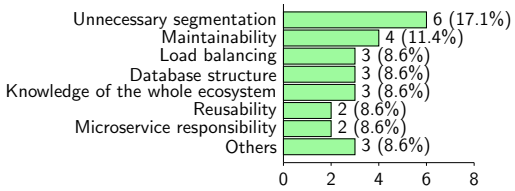

Figure 3: Concerns on the microservice granularity.

important criteria by 45.7% of companies, followed by coupling with 42.9%, in accordance with the findings of Carvalho et al. [7]. Figure 2 presents the number of occurrences of each criterion.

An important issue related to the microservice identification is the definition of the correct level of granularity, that is, the trade-off between size and number of microservices [43]. Results of an open-ended question of our survey show that the companies are concerned on the microservice granularity. Figure 3 presents their main concerns, for which the unnecessary segmentation was the most cited (17.1 %). This can be observed in answers of participants P28: "*we are careful to not create nano microservices*" and P40: "*main concern is to create unnecessary microservices*". The figure presents other responses, such as maintainability (11.4%), load balancing (8.6%), database structure (8.6%), knowledge of the whole ecosystem (8.6%), reusability (8.6%), and microservice responsibility (8.6%). The least popular concerns, grouped in the item Others, such as orchestration (2.9%), testability (2.9%), and integration (2.9%).

4. Define the microservice architecture. As a result of the driving forces related to technology flexibility and easier maintenance and evolution, the majority (64.5%) of the software companies that answered this survey uses more than one programming language. 11 (35.5%) companies use two programming languages, 6 (19.4%) use three languages, 1 (3.2%) uses four languages, 2 (6.5%) vary according to the system. The remaining 11 (35.5%) companies use only one programming language.

Aligned with the literature recommendation [42], 20 (57.1%) companies define owners for each microservice. The "owner" is a developer/engineer who is primarily responsible for investigating and solving problems related to a specific microservice. The other 15 (42.9%) companies do not define owners by microservice.

Regarding cloud infrastructure, 34 (97.1%) of the participating companies use cloud infrastructure services. Only 1 company does not use such a service. Figure 4 presents the cloud services used by companies. The Amazon Web Services (AWS) is the most used cloud infrastructure, being cited by 26 (74.3%) companies. 5 (14.3%) companies use their own cloud infrastructure.

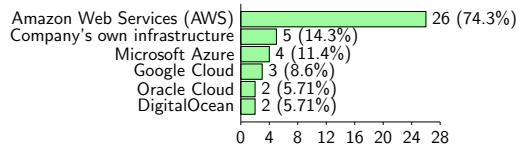

Figure 4: Cloud infrastructure used.

We also asked whether the software companies follow any strategy for the definition of new microservices. If so, the participants might describe the strategy. Eight (22.8%) participants explicitly answered that they did not define any formal strategy for doing that. Participants cited four different strategies used to define new microservices, as follows: (i) new business functionalities (11.4%), (ii) bottleneck identification through performance issues (11.4%), (iii) areas of interest of the application layer (2.9%), and (iv) process instance based on the Open Application Model (OAM) (2.9%).

When defining the microservice architecture, practitioners need to decide how to deal with reuse needs. Then, in another question, we asked whether there is duplicate code in different microservices. A number of 27 (77.1%) participating companies answered that there is no duplication of source code in their systems. The participant P4, who works in one of these companies, stated that *"We do not duplicate source code. However, when some component is shared by more than one team, its maintenance is also shared"*. On the other hand, in 8 (22.9%) companies there is code duplication in different microservices, with the goal of (i) enabling the customization of some functionality or (ii) performing similar operations in different microservices, usually related to infrastructure issues. 2 participants stated that the code duplication occurs in the beginning of the migration process, to allow the incremental migration to the microservice architecture.

We observe that each company deals with duplication in different ways, probably due to the lack of formal approach to guide companies to explore reuse of source code in the context of microservice architecture [6]. The companies that have code duplication pointed different ways to deal with the duplication, they are: (i) building libraries shared between services for allowing reuse (4 answers, 11.4%); (ii) Schema Registry used to prevent duplication of data transfer object used in messages (1 answer, 2.9%); (iii) simultaneous changes and simultaneous delivery (1 answer, 2.9%); (iv) leave the code only in the microservice which is more appropriate to the functionality, resulting in a coupling between microservices (1 answer, 2.9%); and (v) when they judge that duplication is necessary, there is no duplicate code traceability or documentation, to prevent the coupling generated between microservices (1 answer, 2.9%).

We also asked the participants about the communication interface used to integrate microservices in their companies. Figure 5 presents the results of this question, demonstrating a clear predominance of the REST APIs (80.0%) followed by Messaging (20.0%), Simple HTTP (14.3%), gRPC (8.6%), Apache Kafka (5.75%), Queues (5.75%), and Service Mesh (5.75%). This result is similar to studies as [1] that discusses that microservices aim at transforming software systems in packages of independent small services, developed in different platforms, executing their own process through communication interfaces, such as REST APIs or HTTP API.

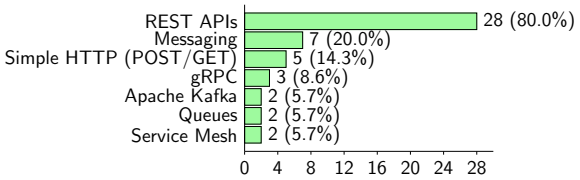

Figure 5: Communication tools used.

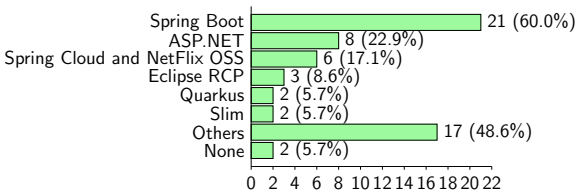

Figure 6: Frameworks used.

In addition to the questions whose results were presented here, the database structure also impacts the definition of microservice architectures. However, as the survey has ten questions regarding the database structure and management, the results about this subject are presented in Section 4.1.

5. Execute the modernization. The questionnaire also asked questions related to technologies used to automate the process of re-engineering the legacy systems using microservices. Figure 6 presents the frameworks mentioned and the number of companies that use them. The most popular frameworks are: Spring Boot (60.0%), ASP Net (22.9%), Spring Cloud and NetFlix OSS (17.1%), Eclipse RCP (Rich Client Platform) (8.6%), Quarkus (5.7 %), and Slim (5.7%). The least popular frameworks were cited once, grouped in the Others item (17 answers, 46.6%). Two companies (5.7%) stated that they do not use frameworks.

Figure 7 shows the continuous integration tools used by the companies. The most used tools are Jenkins (60.0%), followed by Azure (17.1 %), Gitlab CI (17.1 %) and Circle CI (17.1 %). The other tools, grouped in the Others item, include AWS Pipeline, BildKite, Envoyer, Droid and TeamCity, which were cited once (2.9%). Five companies (14.3%) stated not using tools.

Regarding the automation of the configuration management, 32 (91.4%) companies use containerization, such as Docker, to manage microservices, whereas only 3 (8.6%) do not use containerization.

6. Integrate the microservices and the legacy. This activity refers to the decisions on how to put together the legacy and the modernized parts of the system [43]. As the modernization of the legacy with microservices is usually an incremental process, we investigated whether the participating companies needed to reactivate or reuse some functionality already migrated from the legacy due to a problem in the resulting microservice. Only 5 (14.3%) companies answered that yes.

When asked about the reason for this reactivation of the legacy functionality, we received 5 different answers. Three reasons are related to development issues: (i) bugs in the new implementation that were not identified by the quality assurance team; (ii) integration error due to a problem in the build of one microservice; and

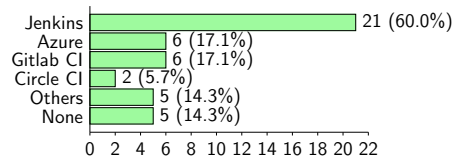

Figure 7: Continuous integration tools used.

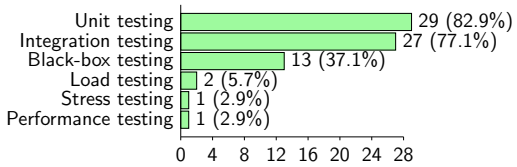

Figure 8: Types of test applied to verify microservices.

(iii) design decisions to reactivate the legacy functionality because synchronizing data on different databases was not feasible. The other remaining reasons are related to infrastructure: (iv) the system performance was not good enough in the production environment, and (v) problem with integration with third-party systems.

Interestingly, the participant P28, who answered that its company never reactivated one microservice, stated that *"if you are migrating to microservices and you feel like going back to the monolithic architecture, you are doing something wrong. There are classical migration mistakes: 1. you do nano services, 2. you duplicate data, 3. you destroy the network, 4. you do not scale the system but scale the invoice; thus, what should solve your problems creates a lot of them."* So P28 corroborates the Richardson and Smiths [37] statement that *"There are no silver bullets. Like every other technology, the microservice architecture pattern has drawbacks."*

7. Verify and validate the microservices. Despite only 19 participants have selected the activity of verifying and validating the microservices, when asked about which type of testing they perform to verify the microservices, only 4 (out 35) stated that none type of testing is applied in their respective software companies, corroborating our inference that there was a misunderstanding during the answering of the question about the activities of the modernization process roadmap.

Testing a microservice-based system is much more complex than testing a monolithic system [37]. To test a class of a microservice, we need to launch that service and any services that it depends upon, or at least configure stubs for those services. Higher levels of test require similar configuration. Despite this complexity, 88.5% of the participating companies execute testing tasks. Figure 8 presents the type of test applied to verify microservices. Unit testing is applied by 82.9% of companies, followed by integration testing (77.1%). Load testing, stress testing and performance testing were cited by 5.7%, 2.9% and 2.9% of the participants, respectively. In addition, 37.1% of participants stated they execute black-box testing.

8. Monitor the microservices and infrastructure. The use of microservice-based systems implies in constantly monitoring the behavior of the deployed microservices. This monitoring includes, for example, to check how the most used microservices are performing and scaling, the level of coupling between different

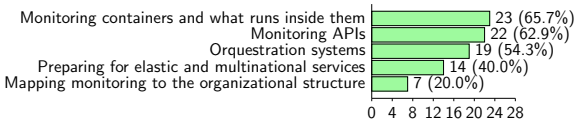

Figure 9: Principles of monitoring microservices used.

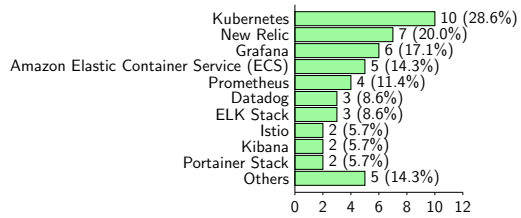

Figure 10: Monitoring tools used for microservices.

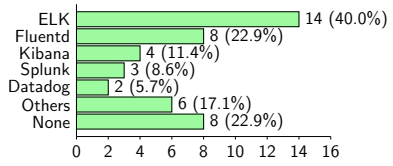

Figure 11: Log management tools used.

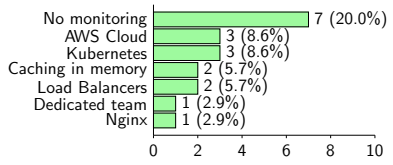

Figure 12: Management of network overhead.

microservices, and the use of infrastructure resources. Mazlami et al. [27] present a study with practitioners working with microservice architectures. In this research all participants agreed with the importance of robust logging and monitoring tools, to achieve a mature level of microservice development. In addition, 90% of the participants in their study believe that logging and monitoring should be set up as early as the project starts.

In our study, most of the software companies (80.0%) use automated system monitoring mechanisms. For these companies, we inquired what principles of monitoring microservices they use. Figure 9 shows the answers, where we observe that more than 60% of the participants monitor both containers and what runs inside them (65.7%) and APIs (62.9%). Orchestration systems are also used by 54.3% of the participants. Figure 10 shows which tools these software companies use for monitoring microservices. The most popular tools are Kubernetes (28.6%), New Relic (20.0%), Grafana (17.1%), Amazon Elastic Container Service (14.3%), and Prometheus (11.4%). The least popular tools were grouped in the Others item in the figure. This item encompasses five tools that were cited once (2.8%): Fluentd, Graylog, Splunk, Swagger, Zabbix.

Another reason that increases the need for monitoring microservices and infrastructure is when the company makes APIs available for other companies to consult information in the system. About this subject, 21 (60.0%) participating companies argued that they do not make it, whereas 14 (40.0%) of them offer microservices migrated from the legacy system for use by third parties.

Regarding the use of log management tools, Figure 11 presents the answers about this subject. The companies used a great diversity of tools, and the most cited are ELK (40.0%), Fluentd (22.9%), Kibana (11.4%) and Datalog (5.7%). The other tools listed in the figure are cited only once, such as Cloudwatch, Graylog, Log4J, New Relic, SLF4J and Database event records. An interesting result observed in the data collected is that a relevant number of companies, namely 8 (22.9%), does not use management log tools.

Monitoring the network overhead is also an indispensable task to adequately and efficiently use the microservice architecture. We inquired the survey participants about how their companies support this task (see Figure 12). Seven (20.0%) companies do not present concerns related to network overhead. Other companies present strategies to deal with this task such as: perform caching in memory (5.7%), using load balancers (5.7%), and having a dedicated team (2.9%). In addition, the answers include the use of tools such as: AWS Cloud (8.6%), Kubernetes (8.6%), and Nginx (2.9%).

Misalignment. The results show that 14.3% of companies do not use integration continuous tools. This represents a gap between the literature and the practice. Some authors, such as [2], highlight the fundamental importance of continuous integration to the quality of the system as a whole. Furthermore, 22.9% of companies do not perform automatic log control, which demonstrates a low level of maturity related to this subject. Wang et al. [42] discuss the importance of a robust logging and monitoring framework, to achieve a mature level of microservice development. Studies on customization of microservices could support the companies which deal with code duplication (22.9%).

4.2 Data Persistence in Microservices

Monolithic applications are usually designed around a single database that manages all data and transactions, while microservice architectures move towards data decentralization in which each microservice manages its (potentially heterogeneous) persistence mechanism [24]. The first architectural style is very simple to deploy and brings the benefit of atomic transactions: all concurrency and consistency aspects of the data operations are managed by the database server as long as they occur inside a transaction, which will be atomic (either entirely committed or canceled) [32].

Despite being convenient, a centralized database has the drawback of increasing the coupling in a microservice architecture since it is a strong runtime dependency shared by all microservices, reducing its autonomy [32]. Modernizing a monolithic system to an architecture of loosely coupled, independent (micro)services involves refactoring the centralized database into multiple smaller databases, reducing the coupling. However, it requires concurrency and consistency be managed by the application code, instead of the central database manager [15]. As presented in Figure 13, the decentralized persistence model was chosen by 20 (57.1%) participants, and just 9 (25.7%) opted for a centralized database.

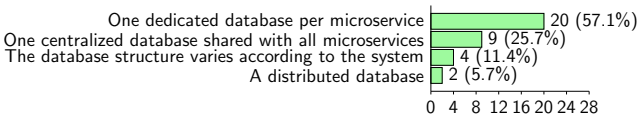

Figure 13: Type of database structure used.

To characterize how persistence aspects affected the usage of a microservice architecture, we asked 10 specific questions regarding such concern, which were answered by 26 participants.

Two questions were formulated to assess if the database and its structures are sources of coupling. Eleven (45.8%) answered that there were situations where changes in the database structure of one microservice affected another microservice, whereas 15 (54.2%) have not reported such an issue. Nine (34.6%) reported that some database tables are involved in transactions of more than just one microservice. These answers indicate that the database, its structures and transactions are a source of coupling in microservice-based systems. Nunes et al. [33] include such concerns in their approach, which takes the application transactional contexts as an input in the process of identifying microservice candidates that best correlate to business capabilities.

Business persistent classes (entities) are shared between microservices in 6 (23.1%) of the cases. The participants P12, P9, and P16 answered an open question about the approach adopted to deal with this situation. They answered, respectively: "*Maven shared library*", "*common package*" and "*a library containing the entities*". These answers reinforce the existence of coupling in such scenarios.

The consistency of database operations involving multiple microservices is managed in the application logic in 17 (65.4%) of the companies, indicating a high level of adherence to the concept of microservice persistence autonomy. Seven (26.9%) answered that they use atomic transactions, which is expected in centralized database scenarios. Strangely, 4 (15,4%) of these companies rely on atomic transactions in a decentralized scenario. This may indicate that there is no business transaction scattered between microservices or that some hidden consistency scheme is in place. Finally, 2 (3.8%) participants have not provided an answer about this subject.

Database transactions were an input to the architecture design process in 19 (73.1%) companies, of which 13 (50.0%) applied database refactoring techniques to allow better microservice modularization. Five (19.2%) opted to keep the transactions as they were, even that affected the modularization level. Six (23.1%) have not analyzed the database transactions at all. This result indicates that analyzing database transactions can be more explored as a source of information for migrating to a microservice architecture, which can be addressed by further research.

Database performance and scalability were a concern to 15 (57.7%) participants, and all had in common the decentralized database model. They reported many different strategies to deal with this concern: Amazon auto-scaling, Command Query Responsibility Segregation (CQRS), load monitoring, distributed cache, cloud-based database with frequent monitoring, partitioning, load testing, and events and asynchronous processing. This variety of responses show that there is room for improvement that further research can explore. Surprisingly, 11 (42.3%) of the participants have not considered database performance and scalability for the modernization, which also can be investigated to determine whether the database is not a bottleneck or was simply neglected, which could hurt the goal of achieving a better system performance.

Finally, we asked the participants using a centralized database what are complicating factors to adopt a decentralized model. The answers included: database licensing cost, the separation of contexts, proprietary database managed by a third party company, data maintenance and integrity, the database is shared with the legacy and BI applications, and resistance from the database administrator. Except for this latter, most of these issues are addressed exactly by partitioning the database and adopting a lighter, potentially open-source, commodity database mechanism per microservice: licensing costs could go down, third-party dependency mitigated, better isolation from the legacy, and business intelligence.

These findings show that data management plays a significant role in designing a microservice architecture and are aligned with the findings of Laigner et al. [34], that cite challenges regarding how to deal with aspects of data consistency and separation of contexts as there are many patterns to address this matter and the current database management systems are not adapted to this scenario.

4.3 Answering RQs

In this section we answer the posed sub-RQs.

RQ1. Why do companies migrate monolithic legacy systems to microservices?

The migration to microservice architecture is motivated by driving forces from operational, technical, and organizational perspectives. In addition, we observe that the three most common driving forces are the same identified in the literature modernization process roadmap: *Easier maintenance and evolution*, *Optimized scalability*, and *Independent and automated deploy*. These three forces were cited by more than 70% of companies.

RQ2. How do companies perform the migration of monolithic legacy systems to microservices?

The results of the survey show that every activity of the modernization process roadmap is realized by most companies. Around 50% of them performed an *incremental migration* to microservices decomposing the legacy system by *business capabilities*. The main criteria considered to define the microservice candidates are *scalability* (85.7%), *requirements* (45.7%), and *reusability* (45.7%). Most companies (64.5%) use *more than one programming language* per project. *Unit* and *integration testing* are executed by at least 77% of the companies. Only 22.9% of the participating companies deal with *code duplication* and they have not a standard policy to address the customization source code needs. The communication interface to integrate microservices is mainly *API REST*. Almost all companies (97.1%) use *cloud infrastructure services*, mainly the Amazon Web Services. *Spring Boot*, *Jenkins*, *Docker*, *Kubernetes* and *ELK* are the most cited technologies for infrastructure configuration, continuous integration, containerization, microservice monitoring, and logging management, respectively. This panorama shows *the practice and the literature of modernizing legacy systems with microservices do speak the same language*. The driving forces that motivate the migration to microservices are the same pointed

by the literature as the main advantages of microservices as well as the strategy and the criteria taking into account to the migration process are also aligned with the literature. Furthermore, the use of at least two programming languages in a project reinforces the technology flexibility provided by microservices.

RQ3. What are the aspects of data persistence considered in the modernization of legacy systems with microservices?

Most of the companies reports *decentralizing the database* (57.1%), 73.1% *analyzes the database transactions* to reason about the new architecture, and 50.0% *refactored the database* to obtain better results in the migration to microservices. Data consistency in database operations involving multiple microservices is managed in the *application logic* (65.4%) and the *performance of the database* is a concern to 57.7% of the companies. This scenario shows that data persistence is an important concern in the industry when developing microservices, and that the practice and the literature are also speaking the same language regarding this subject.

5 THREATS TO VALIDITY

We present here the threats to validity and their mitigations.

Internal Validity: one potential threat is the diversity of participants and companies. Although our sample has not a regular distribution over Brazilian states, we cover the states that have higher concentration of software companies. As expected, most participants work in Brazilian companies, which is in accordance with the scope defined for our study. Another threat is concerned to the modernization process [43] chosen to represent the theoretical knowledge produced in academia. If this process is changed, the companies' experience will be the same. Thus, the terminology or number of activities in the modernization process could be different, but the meaning of the final results would be similar.

External Validity: we cannot argue that the survey sample represents all types of companies, considering size, problem domain, localization, among other company characteristics, what can impact on the generalization of the results. However, we mitigated this threat by making the questionnaire available in different media and social networks, to reach the largest number of participants, regardless of geographic location and company profile.

Construct Validity: several questions in the survey included a list of predefined responses, so the participants may have opted to choose one of these responses, rather than to fill in an open text field, describing other options more suited to their reality. However, we complemented our questionnaire with options collected from grey literature. These options were validated in the pilot study conducted with an experienced developer. The decision of using a high number of closed-ended questions was to reduce the complexity of filling out the questionnaire (the time to complete it was 10 minutes in the pilot study), and thereby reducing the participants' fatigue. In addition, the interpretation of open-ended questions performed by the authors might have been influenced by some type of bias.

6 RELATED WORK

There are many pieces of work reporting approaches, techniques, and methods to modernize legacy systems with microservices, as discussed in secondary studies [11, 12, 14, 35, 43]. All these secondary studies brings contributions to understand different aspects of the modernization, as for example, the modernization process roadmap proposed in [43]. However, they do not validate their findings in practice, which is our focus. Other studies investigate the state of the practice on using microservices, but without focusing on the modernization process of legacy systems [3, 16, 21, 41, 44].

Several pieces of work report experiences and lessons learned of modernization conducted in industry [1, 5, 17, 18, 26, 28, 29, 38]. However, they provide a discussion considering only the specific context in which the modernization took place. Yet, they do not analyze their findings against existing literature. Our work advances in this direction, presenting results in the light of existing studies.

Other studies present surveys with practitioners. For example, Taibi et al. [40], as a result of a survey among experienced practitioners, presented a process framework based on the comparison of three different migration processes. In addition, these authors described motivations and issues that commonly take place during the migrations. Di Francesco et al. [10] conducted a survey, targeting practitioners involved in the process of migrating their applications, to collect information about the performed activities and the challenges faced. Carvalho et al. [7] also conducted a survey with practitioners that participated in the modernization of legacy systems with microservices. In an extension of this survey, these authors also investigated how reuse and customization are leveraged after the modernization with microservices [6]. With the similar goal, Wang et al. [42] performed a survey and interviews with practitioners, investigating best practices learned by practitioners that adopted microservices. Finally, Fritzsch et al. [13] report a qualitative study on intentions, strategies, and challenges in the context of migrations to microservices. Differently from our work, which focus on a confirmatory study of the entire modernization process, those studies conducted exploratory studies only considering specific modernization scenarios, activities, or issues.

Regarding the database decomposition, Newman [32] describes ways to achieve decentralization of data and transactions through many design patterns. Differently, our study does not focus on what patterns practitioners have adopted, but rather whether they were concerned about database during the modernization, whether the data is decentralized, and if they dealt with eventual consistency.

7 CONCLUSION

This paper reports on a survey to investigate whether the industry and academia are "speaking the same language" with respect to the modernization of legacy systems with microservices. Practitioners from 35 companies participated of this survey. We used a modernization process roadmap [43] to conduct and organize our analysis. First, the obtained results pointed out that modernization driving forces vary from superior to technology flexibility – the most expected benefits from microservice architecture. Related to the migration process, we observed that every activity of the modernization process roadmap is realized by most participating companies. Some misalignments were detected, which can be explored in further research. For instance, despite most companies using decentralized databases, there are some research opportunities related to data persistence that can be addressed in the future.

As a further work we plan to carry out this survey with a greater number of companies and a larger diversity of countries and Brazilian states. We are also starting a comprehensive study regarding data persistence aspects to investigate issues such as performance, usage of relaxed consistency models, and analysis of database transactions in the process of modernization with microservices.

ACKNOWLEDGMENTS

This work is supported by CNPq grants 151723/2020-6, 428994/2018-0, 408356/2018-9, 434969/2018-4, 309844/2018-5, 421306/2018-1; FAPERJ grants 22520-7/2016, 010002285/2019, and 202073/2020.

REFERENCES

[1] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. 2015. Migrating to cloud-native architectures using microservices: an experience report. In *4th European Conference on Service-Oriented and Cloud Computing (ESOCC)*. 201–215.

[2] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. 2016. Microservices architecture enables devops: Migration to a cloud-native architecture. *IEEE Software* 33, 3 (2016), 42–52.

[3] Justus Bogner, Jonas Fritzsch, Stefan Wagner, and Alfred Zimmermann. 2019. Microservices in Industry: Insights into Technologies, Characteristics, and Software Quality. In *16th Intl. Conference on Software Architecture (ICSA)*. 187–195.

[4] Amina Boubendir, Emmanuel Betin, and Noemie Simons. 2017. VNF-as-a-service design through micro-services disassembling the IMS. In *20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*. 203–210.

[5] Antonio Bucchiarone, Nicola Dragoni, Schahram Dustdar, Stephan T. Larsen, and Manuel Mazzara. 2018. From Monolithic to Microservices: An Experience Report from the Banking Domain. *IEEE Software* 35, 3 (2018), 50–55.

[6] Luiz Carvalho, Alessandro Garcia, Wesley K. G. Assunção, Rodrigo Bonifácio, Leonardo P. Tizzei, and Thelma Elita Colanzi. 2019. Extraction of Configurable and Reusable Microservices from Legacy Systems: An Exploratory Study. In *23rd Intl. Systems and Software Product Line Conference (SPLC)*. 26–31.

[7] Luiz Carvalho, Alessandro Garcia, Wesley K. G. Assunção, Rafael de Mello, and Maria Julia de Lima. 2019. Analysis of the Criteria Adopted in Industry to Extract Microservices. In *7th Intl. Workshop on Conducting Empirical Studies in Industry / 6th Intl. Workshop on Software Engineering Research and Industrial Practice (CESSER-IP)*. 22–29.

[8] Luiz Carvalho, Alessandro Garcia, Thelma Elita Colanzi, Wesley K. G. Assunção, Juliana Alves Pereira, Baldoino Fonseca, Márcio Ribeiro, Maria Julia de Lima, and Carlos Lucena. 2020. On the Performance and Adoption of Search-Based Microservice Identification with toMicroservices. In *36th Intl. Conference on Software Maintenance and Evolution (ICSME)*. 569–580.

[9] Rui Chen, Shanshan Li, and Zheng Li. 2018. From Monolith to Microservices: A Dataflow-Driven Approach. In *25th Asia-Pacific Software Engineering Conference (APSEC)*. 466–475.

[10] Paolo Di Francesco, Patricia Lago, and Ivano Malavolta. 2018. Migrating towards microservice architectures: an industrial survey. In *15th Intl. Conference on Software Architecture (ICSA)*. 29–2909.

[11] Paolo Di Francesco, Patricia Lago, and Ivano Malavolta. 2019. Architecting with microservices: A systematic mapping study. *Journal of Systems and Software* 150, 0 (2019), 77–97.

[12] P. D. Francesco, I. Malavolta, and P. Lago. 2017. Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption. In *14th Intl. Conference on Software Architecture (ICSA*. 21–30.

[13] Jonas Fritzsch, Justus Bogner, Stefan Wagner, and Alfred Zimmermann. 2019. Microservices Migration in Industry: Intentions, Strategies, and Challenges. In *35th Intl. Conference on Software Maintenance and Evolution (ICSME)*. 481–490.

[14] Jonas Fritzsch, Justus Bogner, Alfred Zimmermann, and Stefan Wagner. 2018. From monolith to microservices: a classification of refactoring approaches. In *1st Intl. Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment (DEVOPS)*. 128–141.

[15] A. Furda, C. Fidge, O. Zimmermann, W. Kelly, and A. Barros. 2018. Migrating Enterprise Legacy Source Code to Microservices: On Multitenancy, Statefulness, and Data Consistency. *IEEE Software* 35, 3 (2018), 63–72.

[16] Javad Ghofrani and Daniel Lübke. 2018. Challenges of Microservices Architecture: A Survey on the State of the Practice.. In *10th Central-European Workshop on Services and their Composition (ZEUS)*. 1–8.

[17] Jean-Philippe Gouigoux and Dalila Tamzalit. 2017. From Monolith to Microservices: Lessons Learned on an Industrial Migration to a Web Oriented Architecture. In *14th Intl. Conference on Software Architecture Companion (ICSA)*. 62–65.

[18] Jean-Philippe Gouigoux and Dalila Tamzalit. 2019. "Functional-First" Recommendations for Beneficial Microservices Migration and Integration Lessons Learned from an Industrial Experience. In *Intl. Conf. on Software Architecture*. 182–186.

[19] Penny Grubb and Armstrong A. Takang. 2003. *Software maintenance: concepts and practice*. World Scientific.

[20] Sara Hassan, Nour Ali, and Rami Bahsoon. 2017. Microservice Ambients: An Architectural Meta-Modelling Approach for Microservice Granularity. In *14th Intl. Conference on Software Architecture (ICSA)*. 1–10.

[21] Holger and Hasselbring Wilhelm Knoche. 2019. Drivers and Barriers for Microservice Adoption – A Survey among Professionals in Germany. *Enterprise Modelling and Information Systems Architectures (EMISAJ) – International Journal of Conceptual Modeling* 14, 1 (2019), 1–35.

[22] H. Knoche and W. Hasselbring. 2018. Using Microservices for Legacy Software Modernization. *IEEE Software* 35, 3 (2018), 44–49.

[23] Valentina Lenarduzzi and Outi Sievi-Korte. 2018. On the Negative Impact of Team Independence in Microservices Software Development. In *19th Intl. Conference on Agile Software Development: Companion (XP)*. 1–4.

[24] James Lewis and Martin Fowler. [n.d.]. *Microservices: a Definition of This New Architectural Term*. https://www.martinfowler.com/articles/microservices.html

[25] Johan Linåker et al. 2015. *Guidelines for Conducting Surveys in Software Engineering*. Technical Report. Lund University.

[26] Welder Luz, Everton Agilar, Marcos César de Oliveira, Carlos Eduardo R. de Melo, Gustavo Pinto, and Rodrigo Bonifácio. 2018. An Experience Report on the Adoption of Microservices in Three Brazilian Government Institutions. In *32nd Brazilian Symposium on Software Engineering (SBES)*. 32–41.

[27] G Mazlami, J Cito, and P Leitner. 2017. Extraction of microservices from monolithic software architectures. In *24th Intl. Conf. on Web Services (ICWS)*. 1–8.

[28] Manuel Mazzara, Nicola Dragoni, Antonio Bucchiarone, Alberto Giaretta, Stephan T. Larsen, and Schahram Dustdar. 2018. Microservices: Migration of a Mission Critical System. *IEEE Transactions on Services Computing* (2018), 1–1.

[29] Alan Megargel, Venky Shankararaman, and David K. Walker. 2020. Migrating from Monoliths to Cloud-Based Microservices: A Banking Industry Example. In *Software Engineering in the Era of Cloud Computing*. 85–108.

[30] Rômulo Manciola Meloca, R. Ré, and André Luis Schwerz. 2018. An Analysis of Frameworks for Microservices. *Latin American Computer Conf.* (2018), 542–551.

[31] Davide Neri, Jacopo Soldani, Olaf Zimmermann, and Antonio Brogi. 2019. Design principles, architectural smells and refactorings for microservices: a multivocal review. *SICS Software-Intensive Cyber-Physical Systems* 35, 1 (2019), 3–15.

[32] Sam Newman. 2019. *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*. O'Reilly Media.

[33] Luís Nunes, Nuno Santos, and António Rito Silva. 2019. From a Monolith to a Microservices Architecture: An Approach Based on Transactional Contexts. In *13th European Conference on Software Architecture (ECSA 2019)*. 37–52.

[34] Rodrigo Nunes Laigner, Yongluan Zhou, Marcos Antonio Vaz Salles, Yijian Liu, and Marcos Kalinowski. 2021. Data Management in Microservices: State of the Practice, Challenges, and Research Directions. *Proceedings of the VLDB Endowment* 14, 13 (2021).

[35] Francisco Ponce, Gastón Márquez, and Hernán Astudillo. 2019. Migrating from monolithic architecture to microservices: A Rapid Review. In *38th Intl. Conference of the Chilean Computer Science Society (SCCC)*. 1–7.

[36] Zhongshan Ren, Wei Wang, Guoquan Wu, Chushu Gao, Wei Chen, Jun Wei, and Tao Huang. 2018. Migrating Web Applications from Monolithic Structure to Microservices Architecture. In *Asia-Pacific Symposium on Internetware*. 1–10.

[37] Chris Richardson and Floyd Smith. 2016. *Microservices - From Design to Deployment*. "NGINX, Inc.".

[38] Santonu Sarkar, Gloria Vashi, and PP Abdulla. 2018. Towards transforming an industrial automation system from monolithic to microservices. In *23rd Intl. Conference on Emerging Technologies and Factory Automation (ETFA)*. 1256–1259.

[39] Jacopo Soldani, Damian Andrew Tamburri, and Willem-Jan Van Den Heuvel. 2018. The pains and gains of microservices: A Systematic grey literature review. *Journal of Systems and Software* 146, 1 (2018), 215–232.

[40] Davide Taibi, Valentina Lenarduzzi, and Claus Pahl. 2017. Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation. *IEEE Cloud Computing* 4, 5 (2017), 22–32.

[41] Markos Viggiato, Ricardo Terra, Henrique Rocha, Marco Tulio Valente, and Eduardo Figueiredo. 2018. Microservices in practice: A survey study. In *6th Workshop on Software Visualization, Evolution and Maintenance (VEM)*. 1–8.

[42] Yingying Wang, Harshavardhan Kadiyala, and Julia Rubin. 2021. Promises and challenges of microservices: an exploratory study. *Empirical Software Engineering* 26, 4 (2021), 1–44.

[43] Daniele Wolfart, Wesley K. G. Assunção, Ivonei F. da Silva, Diogo C. P. Domingos, Ederson Schmeing, Guilherme L. Donin Villaca, and Diogo do N. Paza. 2021. Modernizing Legacy Systems with Microservices: A Roadmap. In *25th Evaluation and Assessment in Software Engineering (EASE)*. 149–159.

[44] He Zhang, Shanshan Li, Zijia Jia, Chenxing Zhong, and Cheng Zhang. 2019. Microservice Architecture in Reality: An Industrial Inquiry. In *16th Intl. Conference on Software Architecture (ICSA)*. 51–60.

[45] Tom Černý, Michael Donahoo, and Michal Trnka. 2018. Contextual understanding of microservice architecture: current and future directions. *ACM SIGAPP Applied Computing Review* 17, 1 (2018), 29–45.