Modernizing Legacy Systems with Microservices: A Roadmap

Daniele Wolfart
PPGComp - Western Paraná State University, Brazil

Wesley K. G. Assunção
PPGComp - Western Paraná State University, Brazil
DI - Pontifical Catholic University of Rio de Janeiro, Brazil

Ivonei F. da Silva
PPGComp - Western Paraná State University, Brazil

Diogo C. P. Domingos
PPGComp - Western Paraná State University, Brazil

Ederson Schmeing
PPGComp - Western Paraná State University, Brazil

Guilherme L. Donin Villaca
PPGComp - Western Paraná State University, Brazil

Diogo do N. Paza
PPGComp - Western Paraná State University, Brazil

ABSTRACT

Legacy systems are long-lived applications, with obsolete technology and degraded architecture. These systems hamper digital transformation and innovation, and require a great amount of resources for maintenance. The modernization of monolithic legacy systems is a strategy to promote better evolution and maintenance, taking advantage of new technologies such as microservices. Microservice architectural style is a paradigm to develop systems as a suite of small and autonomous services, communicating through a lightweight protocol. However, the migration of legacy systems to microservices is complex. Although we can find several studies on this topic, they usually focus on specific activities, e.g., the identification of the microservice boundaries in the legacy code. Also, existing pieces of work do not cover real-world scenarios, since they do not take into account organizational, operational, and technical aspects. To overcome this limitation, in this paper we present a roadmap for modernizing monolithic legacy systems with microservices. The roadmap is distilled from the existing body of knowledge, describing common activities and input/output information. The proposed roadmap is composed of eight activities, grouped in four phases, namely initiation, planning, execution, and monitoring. The main contributions are: (i) serve as a basis for practitioners to plan, execute, and monitor the modernization process; (ii) be a reference for researchers to design new studies; and (iii) motivate tool builders to deal with existing needs.

CCS CONCEPTS

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

EASE 2021, June 21–23, 2021, Trondheim, Norway

ACM ISBN 978-1-4503-9053-8/21/06…$15.00
https://doi.org/10.1145/3463274.3463334

KEYWORDS

Software Migration, Software Evolution, Cloud Computing

ACM Reference Format:
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 *Evaluation and Assessment in Software Engineering (EASE 2021), June 21–23, 2021, Trondheim, Norway*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3463274.3463334

1 INTRODUCTION

The large majority of existing industrial systems are long-lived applications usually developed in a monolithic architecture. Due to extensive maintenance and obsolete technology, these architectures decay and degrade [45]. Legacy systems hamper digital transformation, makes innovation more difficult, and require a great amount of resources for maintenance, draining investment that otherwise could be used in system evolution, e.g., developing new features [21, 32]. To remain competitive, companies have been modernizing these monolithic legacy systems [27]. "*Software modernization attempts to evolve a legacy system, or elements of the system, when conventionally evolutionary practices, such as maintenance and enhancement, can no longer achieve the desired system properties*" [45]. During modernization, a legacy system is re-engineered [6] or migrated [25] to a modern architecture or platform. The modernization leads to benefits such as easing engineering activities, satisfying user needs, or achieving business goals [44].

Nowadays we can observe a trend on adopting microservice architectures as strategy for modernizing monolithic legacy systems [10, 27, 50]. Microservices became popular around 2014 [29], mainly due to their adoption by big players, as for example Netflix [51] and Uber [17]. Moreover, other factors have contributed to the popularity of microservices. Firstly, microservices are inspired by the service-oriented architecture (SOA) principle. According to this principle, complex applications are built as a suite of small, loosely coupled, and autonomous components that encapsulate business capabilities and communicate with each other using language-agnostic APIs [13]. Splitting large applications into microservices provides independence for agile teams and optimizes

the deployment process [36]. Secondly, some studies argue that microservices reduce maintenance and evolution effort, increase availability of services, ease the inclusion of innovation, ease the incorporation of DevOps in the development lifecycle, and facilitate scalability [48]. Finally, there are many open-source tools to aid the deploying, managing, and monitoring of the microservices, e.g., Docker, Kubernetes, Prometheus, and Grafana, to cite some [50].

Microservices emerged in industry but only recently have caught the attention of the software-engineering research community [11, 29]. We can find, in the literature, secondary studies on the topic of migrating monolithic system into microservices [11, 19, 41], classification of refactoring approaches for the modernization with microservices [20], and industrial reports/surveys with practitioners [4, 10, 19, 50]. Despite the growing interest in the modernization with microservices in both practice and research, there is still a lack of comprehensive studies on how to conduct such a modernization. Another limitation is that existing pieces of work usually do not take into account organizational and operational perspectives of the modernization, focusing mostly on technical aspects of microservices. Furthermore, existing studies mostly focus on specific activities of the process, e.g., identification of the microservices boundaries in the code. Based on this, companies lack a roadmap for real world scenarios, covering the entire modernization process and dealing with organizational, operational, and technical aspects. These limitations might hamper or make complex the proper use of microservices as a strategy to modernize legacy systems.

Based on aforementioned limitations, the goal of this paper is to present a roadmap, i.e., a process to conduct the migration, for guiding practitioners on modernizing legacy systems with microservices. For the definition of such a roadmap, we collected, analyzed, discussed, and organized the existing literature on this topic. Our study (presented in Section 2) relies on 62 papers found by conducting a systematic mapping. The resulting roadmap (described in Section 3) is composed of eight activities and covers the modernization phases of initiation, planning, execution, and monitoring. We also describe the input and output information for each activity.

The contribution of the roadmap defined in our study (highlighted in the related work, in Section 5) is: (i) for practitioners, our roadmap can serve as a basis to decide, plan, execute the modernization process, and monitor the microservices. Instead of a starting blind(folded), we provide a comprehensive set of activities encompassing existing specialized literature; (ii) for researchers, our results can serve as a starting point to design new studies, identify gaps, e.g., focusing on the activities less investigated in the literature; and (iii) for tool builders, serve as a reference to develop automated support for the migration/modernization process.

2 STUDY DESIGN AND EXECUTION

As our goal is *to define a roadmap for modernizing legacy systems with microservices, from the perspective of the existing body of knowledge in the literature*, we posed the following research question:

RQ. Why and how are monolithic legacy systems migrated to microservices? For answering this RQ, we considered the driving forces that motivates the modernization, common activities conducted when modernizing with microservices, and the input used and output generated in each activity.

Table 1: Databases used for identifying primary sources

Database	URL	Studies
ACM Digital Library	http://dl.acm.org	53
IEEE Xplore	http://ieeexplore.ieee.org	72
Web of Science	http://www.isiknowledge.com	35
Science Direct	http://www.sciencedirect.com	99
Scopus	http://www.scopus.com	52
SpringerLink	http://www.springerlink.com	111

To achieve this goal and answer the posed RQ, the methodology of our study is composed of the following steps:

(1) **Primary sources selection**: identification, screening, and validation of studies from digital libraries.
(2) **Data extraction**: reading of papers to extract relevant data to answer the posed RQ.
(3) **Data Classification and roadmap definition**: the data extracted were classified and analyzed in order to establish a comprehensive roadmap for modernizing legacy systems with microservices.

2.1 Primary Sources Selection

For the selection of primary sources, we followed a methodology based on a systematic mapping, according to the process proposed by Petersen et al. [39]. A systematic mapping study is designed to provide an overview of a research field [38, 39]. From the RQ of our study, we derived the following sets of keywords: (i) "*monolith*", (ii) "*migrate, transform, refactor, decompose, extract, partition*", and (iii) "*microservices*". To define the search string[1], we composed these keywords with their lexical and syntactic alternatives (synonym, plural, gerund, etc.). For defining the search string, we also considered eight related pieces of work in the topic [11, 14, 19, 22, 33, 41, 46, 48]. The string was used for searching studies in six digital libraries, as presented in Table 1. The search on these libraries was conducted on March, 28th 2020. A total of 422 studies were retrieved from all databases. Other digital libraries could be used, however, we chose the ones frequently considered in software engineering reviews.

For the screening of the studies, we adopted four inclusion criteria (IC) and two exclusion criteria (EC): IC-1: papers written in English and peer reviewed; IC-2: the goal of the study is the migration from monolithic to microservices; IC-3: the study reports on strategies, guidelines, processes, etc., to migrate from monolithic to microservices; IC-4: full-text available; EC-1: secondary or tertiary studies; and EC-2: tutorials, keynotes, position papers.

From the 422 retrieved studies (see Table 1), after applying the ECs, 79 studies were removed (422-79 = 343). Then, by considering the ICs for the title, keywords, and abstract, 260 studies were removed (343-260 = 83). Finally, after entirely reading the studies, we removed another 21 papers (83-21 = 62). The final set of papers were composed of 62 primary sources, presented in Appendix 6.

To validate the final set of primary sources, we selected four secondary studies that completely [20, 41], or partially [11, 18],

[1]The final search string was: *((microservices OR micro-services OR "micro services") AND (refactoring OR transformation OR migration OR partition* OR granular* OR extract* OR decomposition) AND (monolith*))*

Table 2: Secondary studies for validating primary sources

#	Secondary study	Year	Studies	Overlap
1	Fritzsch et al. [20]	2018	10	10
2	Di Francesco et al. [11, 18]	2019	103	22
3	Ponce et al. [41]	2019	20	20

focus on studies on the topic of modernization/migration from monolithic to microservices. We observed the overlap between studies identified in the libraries and the ones already mapped in the secondary studies. Table 2 presents the secondary studies, together with the number of studies they discuss and the studies we identified as relevant for our work (overlap). The selection of the primary sources from the secondary studies also take into account the ICs and ECs, resulting in 42 distinct studies in the overlap.

In the four secondary studies presented in Table 2, in which [11] is an extension [18]), a total of 4, 17, and 14 relevant studies were exclusively identified from one source, namely #1, #2, and #3, respectively. In addition, 2 studies were identified from both in #1 and #3, 1 study from #2 and #3, and 4 studies among the three secondary studies. From this selection of studies, we identified 42 relevant studies. Compared with the results of our systematic mapping study, all of the 42 papers identified as relevant are among the 62 primary sources. The 20 studies not identified in the secondary studies, are mainly from 2019 and 2020. This was expected, since the secondary studies published in 2019 have not been mapped by these more recent studies. These results strongly confirm that we composed a relevant set of primary sources for our study.

2.2 Data extraction

To answer the RQ of our study, we extracted three pieces of information from each of the 62 primary sources, as follows:

- **Driving forces**: the motivations to start or that originated the modernization of the monolith legacy system with microservices.
- **Process**: which were the activities/steps described in the paper to conduct the modernization with microservices.
- **Input/Output information:** the pieces of information used as input and generated as output for each activity/step along the modernization.

The extraction was done by six authors of this paper, and validated by two experts on microservices. The authors in charge of the extraction, received a training of 20 hours about the topic of modernizing with microservices, to level their background in the topic. This training was based on the discussion of three papers [2, 5, 48].

2.3 Data Classification and roadmap definition

To analyze the driving forces and distill a roadmap, we perform a classification and a qualitative analysis of the raw data to find significant concepts and explore their relationships [7]. Three authors focused on classifying the driving forces, two authors classified the input and output information, and two authors dealt with the classification of activities/steps. For cross-validation, at least two authors inspected each primary study to collect the evidence. Also, at least other two authors discussed the data extract for refining the classification. The final set of driving forces and the definition of the roadmap were obtained by an incremental process, in refinement cycles. All this process was aided by the use of spreadsheets.

3 RESULTS AND ANALYSIS

This section describes the results and analysis of our study. To answer our research question, firstly we describe the driving forces (why) to modernize their legacy. Secondly, we focus on the activities and information used (how) for conducting the modernization.

3.1 Driving Forces

We identified 11 driving forces for modernizing legacy systems with microservices. Table 3 presents these driving forces sorted by the most mentioned ones.

Table 3: Driving forces mentioned in the primary sources

Driving Force	Total	Studies
Optimized scalability	27	S[54–56, 58–60, 62, 64, 68, 76, 78, 80, 81, 83, 86, 87, 92, 95–98, 100, 101, 104, 105, 108, 109]
Independent and automated deploy	23	S[52, 54, 59, 63–65, 68–71, 73, 74, 77, 81, 89–92, 97, 98, 102, 103, 105, 109]
Easier maintenance and evolution	17	S[64, 66, 67, 70, 73, 75, 78, 83, 85, 90, 94, 96, 104, 107, 109, 112]
Independence of teams	13	S[63, 69, 79, 81, 84, 89, 90, 97, 101, 103, 107, 111, 112]
Loosely coupled services	12	S[52, 56, 57, 69, 70, 72, 77, 80, 81, 86, 103, 106]
Cohesive services	10	S[56, 61, 69, 72, 77, 80, 82, 88, 90, 99]
Technology flexibility	9	S[53, 60, 68, 81, 90, 98, 103, 108, 113]
Infrastructure facilities	8	S[60, 62, 64, 93, 98, 102, 108, 113]
Agility enabler	7	S[62, 67, 69, 71, 82, 98, 101]
Easier reuse	3	S[59, 64, 67, 94]
Reduced time to market	3	S[68, 69, 71]

Considering *operational* aspects of the software development/delivery, scalability and deployment were mentioned in 43,55% and 37,10% studies, respectively, they are the two most common driving forces. For example, these two driving forces are related to faster S[81, 98, 103] and independent deploys S[52, 54, 65, 74, 77, 90, 92, 97, 103, 105, 109], continuous delivery S[63, 64, 73, 77, 103], software replication S[56], and increase the elasticity of large applications S[62, 78, 86, 108]. In the same category, independence of teams (20,97%) and infrastructure facilities (12,90%) are also mentioned. In the former, we can cite as example the team autonomy S[52, 69, 81, 90, 91, 97, 101, 103, 109]. For the latter, easy configuration of cloud services S[113], high reliability S[108] and availability of services S[60, 62, 64], and optimization of computational resources S[113].

From a *technical* perspective, easier maintenance and evolution (27,42%) are explicitly mentioned as motivation to adopt microservices. This driving force is a consequence of optimized modularity properties such as cohesive and loosely coupled services (together these forces are mentioned in 35,48% studies). Interestingly, only in five primary sources both coupling and cohesion are mentioned as a concern S[56, 69, 72, 77, 80]. Furthermore, technology flexibility (14,52%) and easier reuse (6,45%) are also pointed as driving forces that companies rely on to migrate to microservices.

Organizational driving forces are less often mentioned in the primary studies. For example, microservices are seen as agility enablers (11,29%) that are related to faster deployment cycles S[69, 71, 98]. Finally, reduced time to market (≈5%) is related to the use of microservices for different purposes/systems S[71]. Another example may be the incentive to the DevOps culture, with an emphasis on collaboration between developers and teams, ensuring shorter lead time and greater agility in software development S[69].

In summary, we can observe that several driving forces motivate the modernization. These driving forces are based on different perspectives, namely operational, technical, and organizational. Interestingly, there is not a dominant driving force that is mentioned in all papers. Yet, differently from some studies advocating mostly for technical benefits of microservices, e.g., use of different technologies [4, 5, 14, 33, 41], we can see that supporting operational tasks are the most common driving forces to adopt microservices.

3.2 Activities and Information

From the 62 primary sources, eight studies do not clearly present a modernization process, namely S[53, 64, 68, 75, 76, 82, 91, 105]. They only discuss problems with legacy systems, benefits of microservice adoption, or lessons learned with a modernization with microservice. Based on that, our analysis of activities and information of the modernization process relies on 54 primary sources. As described in Section 2, our study aims at describing how the modernization of legacy systems with microservices is done. In this sense, the information collected in the primary sources was the basis to define a roadmap for modernization.

In order to define the roadmap, we performed a comprehensive analysis of data extracted from the primary sources (see Section 2.3). Then, we defined a roadmap composed of eight activities, covering four different phases of the modernization with microservices. Figure 1 presents an overview of the phases and its activities for our proposed roadmap. The phases were proposed after we defined all the eight activities of the roadmap. When we grouped the activities, we observed a resemblance to the project management life cycle presented in the well-known Project Management Body of Knowledge (PMBOK Guide) [26, 40]. This was the basis for naming the phases of the proposed roadmap. To corroborate our analysis, Table 4 presents the studies that discuss each activity of the proposed roadmap. As expected, the first activities have received much more attention than the last ones. This indicates that the research on the topic of modernizing legacy systems with microservices is still in early stages, highlighting the need for more studies. In the following, we describe details of each activity, and discuss the observed input and output information.

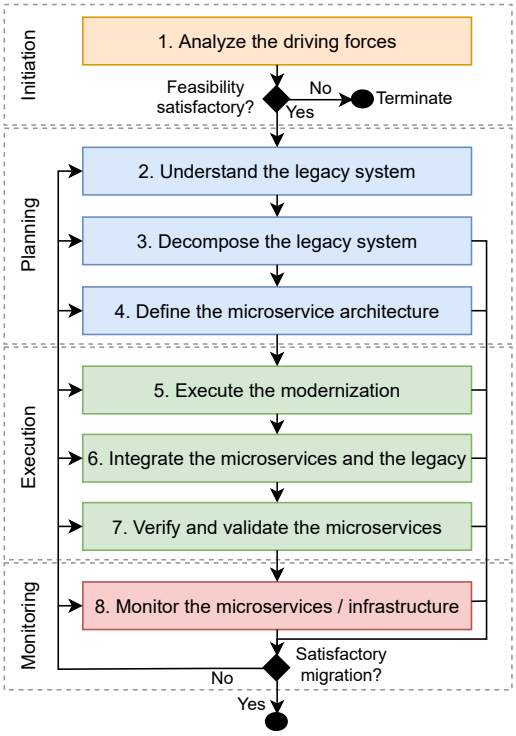

Figure 1: Proposed roadmap for modernizing legacy systems with microservices

1. Analyze the driving forces: in this first activity, companies have to clearly identify/understand the limitations faced with the legacy system. Based on such limitations, managers, engineers, developers, and, in some cases, the customers as well, can reason and decide whether a modernization with microservices is a feasible solution. At this point, these stakeholders of all perspectives (technical, operational, and organizational) can foresee the benefits of modernization and how long it will take to achieve them. It is important to highlight that the modernization of legacy systems is a long and risky process [45]. Also, this process directly impacts all perspectives of a company. Considering this, the activity of *Analyze the driving forces* is paramount to avoid unsuccessful modernization and waste of resources.

The driving forces found in the primary sources, which are related to limitations of legacy systems and expected benefits of microservices, were presented in the previous subsection. However, each company has its own scenario, leading to different needs, constraints, available resources, e.g, human, financial, and material, and expectations with the modernization. As conflicts can emerge among those driving forces, a trade-off analysis could be performed by the stakeholders. In addition to that, we recommend practitioners to have a clear view of the fundamentals of microservices, not only regarding the technical point of view but also in a broad sense [34, 35].

- **Input information**: driving forces (see Section 3.1), business goals, and resources available for conducting the modernization.

Table 4: Activities of the proposed roadmap identified/mentioned in the primary sources

#	Activity	Total	Studies
1	Analyze the driving forces	62	S[52–113]
2	Understand the legacy system	25	S[56, 62, 63, 65, 67, 69–71, 73, 74, 77, 79, 84, 85, 87, 88, 90, 92, 95, 101, 102, 104, 109, 112, 113]
3	Decompose the legacy system	46	S[52, 54–59, 61–63, 65–67, 69–73, 77, 79–81, 84–86, 88–90, 92–104, 106, 107, 109, 111, 112]
4	Define the microservice architecture	35	S[52, 58–62, 66, 69–74, 77–81, 87–90, 93–98, 101, 107–109, 111–113]
5	Execute the modernization	19	S[52, 59, 60, 69, 72, 79–81, 86, 88, 89, 94, 98–100, 106, 109, 111, 112]
6	Integrate the microservices and the legacy	7	S[60, 89, 94, 98, 106, 111, 112]
7	Verify and validate the microservices	8	S[69, 72, 73, 79, 83, 100, 109, 110]
8	Monitor the microservices / infrastructure	4	S[59, 72, 99, 111]

- **Output information**: decision on conducting the modernization of the monolithic legacy system with microservices.

2. Understand the legacy system: the goal of this activity is to analyze and explore the legacy system to understand its implementation, its features, how these features interact with each other, and how the monolithic legacy architecture is organized. This activity is fundamental to the next ones.

According to a primary source S[62], the first task to understand the legacy system is to list all the features for which the system was designed for. 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 [12, 43]. Practitioners can also use reverse engineering techniques to create high-level artifacts, e.g., UML diagrams, from low level artifacts, e.g., source code, to ease the understanding of the legacy system [6, 28]. For example, in S[73] engineers used code analysis tools and existing documentation to obtain a high-level view of system structure. Some studies describe the use of Domain-Driven Design (DDD), which recommends that the structure of the software code should match the business domain [15]. The primary sources S[69, 72, 113] use DDD to structure legacy system features in domains and subdomains with clear context and boundaries. These contexts help to understand and address complexities based on business intentions.

Regarding the information used as input to *Understand the legacy system*, studies do not make clear which artifacts they used S[56, 63, 74, 79, 87, 88]. Among the identified artifacts, source code is the most common. In addition to development artifacts, one study also mentions the use of slides, presentations, and contracts with customers, and unwritten knowledge among developers and engineers S[73]. Considering the output information, in addition to the artifacts presented below, stakeholders involved in the modernization must have a clear understanding of legacy system requirements S[79].

- **Input information**: source code S[67, 69–71, 73, 84, 85, 95, 101, 104, 109, 113]; requirements and textual documents S[65, 73, 77]; test cases S[84]; UML diagrams S[73, 95, 102]; use case specifications S[92, 112]; logs S[109, 113]; and database models/schema S[69, 73, 90].
- **Output information**: data flow graphs, execution traces, and call graphs S[56, 65, 88, 92]; usage scenario models, load profiles, and performance constraints S[84, 88]; visual representation of the system structure S[74, 88]; business capabilities and system operations S[56, 112]; feature model S[87]; and the most used functionalities, domains and subdomains, and bounded contexts S[69].

3. Decompose the legacy system: this activity aims to break down the monolithic architecture into small and cohesive units, which will be transformed into microservices. The decomposition/identification of microservices in monolithic legacy systems is acknowledged as one of the most complex activities of the modernization process S[109]. This is one of the most investigated activities, being the focus of 85% of primary sources (46 out of 54, see Table 4). Although the topic of decomposing software system in modules has been discussed for a long time [37], the limits among modules in monolithic architectures are flexible, and the systems eventually evolve into a "big ball of mud" [16]. When these modules are microservices, the limits are physical, splitting their implementation, distribution, management, and monitoring S[61]. However, *Decompose the legacy system* is not only about partitioning the system to facilitate maintenance, but also defines how the system will be able to evolve and scale S[61]. Some authors recommend that the identification of microservices should be an incremental process S[78]. They argue that due to the complexity of this activity, practitioners will learn how to do it better for each microservice decoupled from the legacy S[78].

An important issue in the activity of identifying the microservices is the definition of the correct level of granularity, that is, the trade-off between size and number of microservices S[61]. Eliciting strong interface limits at the right level of granularity is a challenge inherited from SOA [13]. One study highlights that the proper decomposition of the legacy system into microservices, in the right granularity, is paramount to companies' sustainability S[109]. Besides, when identifying what would be the correct level of granularity, practitioners should consider a checking task regarding the *microservice smells* [47], i.e., antipatterns, such as *megaservice* and *microservice greedy*.

The starting point to decompose the legacy system is to define the responsibility for each microservice, indirectly defining the size of the microservices S[78]. Some authors recommend the microservices to be designed around business capabilities, and allow their independent development S[76, 90]. A strategy to carry out the decomposition and define the granularity of microservices is by following the Principle of Single Responsibility (SRP) S[70]. SRP states that each system unit must have no more than one reason to change [30]. This means that if we have different software components, each one must have an independent responsibility and need. In addition, to enable independent development, each microservice can be developed using technologies that best suit the characteristics of the business capability S[76]. Another strategy is the identification of microservices based on system operations and their data S[112]. The sources of information here are models that describe the system operations, i.e., the system API, and the variables that contain the information that the operations write and read. The decomposition is achieved by grouping operations so that each microservice can internally access and change only its own data. When a microservice needs to refer, i.e., read or write, to a data that belongs to another microservice, this is achieved only through the microservice API. This allows a good decomposition that guarantees low coupling, as well as high cohesion.

To aid the identification of microservices, some tools and approaches have emerged. An example is Service Cutter S[98], a tool to identify microservices that are highly cohesive and loosely coupled. An approach with the same goal is proposed in S[61], which is based on the semantic similarity of foreseen/available functionality described through OpenAPI specifications. In addition, some tools use clustering algorithms to "cut" the monolith S[67, 77, 98]. Differently, the study S[102] presents a method to identify microservices based on the analysis of UML class diagrams.

To corroborate our findings, a recent survey with practitioners indicates that defining microservice granularity is a multi-criteria task [50]. The decomposition of a legacy system into microservices should consider, for example, resource consumption, team structure, delivery cycles, business capabilities, and data access. As we discussed, this activity is related to technical, operational and organizational aspects of companies.

- **Input information**: the output of the previous activities S[54, 55, 57, 94]; desired non-functional requirements S[54, 66, 79, 80, 111]; cloud service provider requirements S[93]; and API specifications S[61].
- **Output information**: microservice candidates S[54, 55, 66, 72, 77, 81, 97, 103] or identified microservices S[57, 58, 100, 104].

4. Define the microservice architecture: this activity is responsible for defining a microservice-based architecture with the design decisions to meet functional and non-functional properties defined in previous activities. In a simple view, a microservice architecture describes the microservices, their APIs, and communications, which is usually done by using a lightweight protocol, e.g., HTTP/RESTful S[76, 87, 113]. Here, practitioners also have to define a strategy of service discovery. In this architecture, the user interface, business logic, and databases (data, logic, and behavior) are packaged in a single application/microservice and deployed on a server S[112]. In addition, despite the benefit of technology flexibility enabled by microservices (see Table 3), Wang et al. [50] identify that organizations should restrict the number of programming languages in order to facilitate maintenance and reuse.

Although the previous activity is designed to identify/decide on how to decompose the legacy system, here practitioners can refine the scope of the microservices, reasoning about the architectural role of each microservice, taking into account heterogeneity and decentralized governance S[78]. Despite the fact that our proposed roadmap has not considered stakeholders for each activity, [50] recommends the definition of owners for each microservice, with well-defined service responsibilities and boundaries.

As part of the architecture definition, practitioners should decide about having a local infrastructure, or using a cloud service provider, such as Amazon Web Services, Microsoft Azure, or Google Cloud. The infrastructure must provide resilience and high availability in microservices, allowing for scalability management and continuous deployment S[112].

- **Input information**: the output of the previous activities S[109]; non-functional requirements such as availability, security and flexibility S[87, 88]; microservices interactions, dependencies, and constraints S[87]; Data access S[94]; Facade pattern S[90, 98].
- **Output information**: microservice architecture model S[60, 79] with design decisions regarding communication protocol S[60, 72]; and cloud service provider/infrastructure S[93].

5. Execute the modernization: the goal of this activity is to conduct the modernization by re-engineering the monolithic legacy system into microservices. According to S[111], microservices reduce the development complexity in comparison to monolithic architectures. However, for the modernization that relies on existing artifacts, this can lead to more effort in comparison to greenfield projects.

Some authors reinforce the adoption of continuous integration and deployment during the migration S[59, 60, 72], using tools such as GitLab and Jenkins. Another point is automation of the configuration management, for instance, using Docker containers S[93] with Spring Boot S[59], and Artifactory for package repository S[59]. It is important to highlight that microservice architecture enables DevOps [1, 49], in this sense, integration between development and operation practices can be leveraged during the modernization [3]. Similarly to monolithic architectures, refactoring should be performed regularly and documentation is as important as the source code S[58]. In addition, when implementing microservices, code management will be necessary. Challenges like managing common code shared by multiple microservices and preventing/dealing with breaking API changes will be faced by developers. In this case, API gateways and "sidecar technology" should be considered [50].

A primary source describes a model-based approach for the automatic modernization of monoliths to microservices S[63]. This approach relies on a domain-specific language, namely JetBrains MPS. A domain-specific language is a computer language created/specialized to a particular application domain, differently from a general-purpose language, which is broadly applicable across domains [31].

- **Input information**: the output of the previous activities; available technologies and tools S[60, 86, 89, 100, 112]; developers and engineers experience S[111].
- **Output information**: microservices implemented according to the defined architecture S[59, 80]; transition documentation S[89].

6. Integrate the microservices and the legacy: this activity is responsible for the integration between microservices and the legacy system. As the modernization with microservices is an incremental process S[78], following a strangler pattern [19], practitioners have to decide on how to put together the legacy and the modernized parts of the system. This is one of the least explored and briefly discussed activities in the primary sources (see Table 4).

As we discussed in previous activities, architectural choices need to facilitate continuous delivery and deployment during the modernization [23]. A strategy to aid the activity of *Integrate the microservices and the legacy* is the use of feature toggles [24] and backward/forward compatibility [42]. Feature toggles is a technique that supports rollback, canary testing, and A/B testing.

- **Input information**: the output of the previous activities S[60, 89]; workflows design and service interface S[111]; legacy scripts S[106].
- **Output information**: integrated services S[106, 111], Rest API S[112], API Gateway S[98].

7. Verify and validate the microservices: this activity focuses on verifying and validating whether the microservices were developed properly and the integration with the legacy was performed accordingly. This activity of *Verify and validate the microservices* can use the legacy system as oracle for testing tasks.

Considering a continuous integration environment, unit tests are expected to be created during the implementation of microservices, but in addition, black box integration tests are also required S[109]. According to a primary source S[100], automated tests must be performed to verify/validate the interoperability of the message exchanges, and check whether microservices are in accordance with the specified architecture. In the case of inadequate definition and implementation of microservices, or other bugs, developers can go back to the previously proposed activities and refine or redefine the problematic part. As the modernization is an incremental process, regression testing and continuous integration should be used to ensure that new bugs are not introduced S[83]. Furthermore, when adopting continuous deployment, practitioners must consider user's feedback as part of the modernization verification and validation S[83].

- **Input information**: the output of the previous activities; usability constraints S[100]; improvements and new features S[100]; non-functional requirements S[79]; and feedback from users S[100].
- **Output information**: unit and integration tests S[69, 73, 100, 110] and black-box/functional testing S[109] reports; metrics about satisfaction of non-functional requirements S[79, 83]; and microservices ready to deploy S[72].

8. Monitor the microservices / infrastructure: this activity is devoted to constantly check the behavior of the deployed microservices. *Monitor the microservices / infrastructure* should focus on service availability, bottlenecks, performance, use of infrastructure resources, and so on. Here, practitioners are able to evaluate and analyze information of the modernized system, in comparison to the driving forces that motivated the modernization (see Activity 1 - *Analyze the driving forces* and Section 3.1). For example, if we observed that scalability is the most common driving force for modernization, then, in this monitoring activity, the performance of the microservices can be observed. In addition to checking system health, monitoring can also support decision-making S[59]. For example, which microservices are the most used by customers, and should receive more attention from the development experts.

In the recent survey with practitioners working with microservice architectures, all 58 participants agreed to the importance of a robustness logging and monitoring framework, to achieve a mature level of microservices development [50]. In addition, 90% of the participants believe that logging and monitoring should be set up as early as the project starts. In fact, "early" in our proposed roadmap means to define, configure, and execute automated frameworks and tools also in the previous activities, such as Activity *4. Define the microservice architecture* and Activity *5. Execute the modernization* [50]. However, these actions can be postponed to a later phase when the project shows failures and delays [50]. To support practitioners on conducting the activity of *Monitor the microservices / infrastructure*, Dave and Degioanni describe five microservice monitoring principles [9]: (i) monitor containers and what runs inside them, (ii) use orchestration systems, (iii) prepare for elastic, multi-country services, (iv) monitor APIs, and (v) map monitoring to the organizational structure.

- **Input information**: the output of the previous activities; microservices deployed S[72], team structures, development and operation processes S[59].
- **Output information**: logs and metrics S[72]; change in team structures S[59].

4 THREATS TO VALIDITY

In this section we present the threats to validity of our study, and how we mitigated/reduced them.

Internal Validity: The first threat to internal validity concerns the terms used in the search string. To address this threat we composed a set of keywords based on secondary studies [11, 14, 19, 22, 33, 41, 46, 48], to make sure of covering the important terms. The second internal threat is the selection of the digital libraries to search for primary sources. In addition to selecting well-known libraries, we also validated the returned results comparing to primary sources of existing secondary study [11, 18, 20, 41], which confirmed the reliability of the libraries. The third threat is related to incorrect classification of primary sources. To reduce this threat, the authors in charge of the classification received training on the topic and extensively discussed the extracted data and performed cross-validation of the data classification. To further mitigate internal threats, we compared our results with existing secondary studies, describing similarities and differences found. In addition,

we performed a validation of the proposed roadmap in comparison to a seminal modernization planning process.

External Validity: Our roadmap is a set of high-level activities designed to serve as a starting point for the modernization with microservices. In our perspective, this roadmap can be easily adopted and aid the conduction of the modernization in different contexts, proving its external validity. Once we performed a comprehensive analysis of existing primary studies that describe the modernization process, the external validity should be appropriate. However, we believe that validation of the roadmap in real-world scenarios and with the participation of practitioners are needed.

5 RELATED WORK

This section describes related work on the topic of modernizing monolithic legacy systems with microservices and highlights existing differences to our study.

Ponce et al.[41] performed a study relying on 20 primary studies on modernization with microservices. They conducted a rapid review and identified three techniques, namely model-driven, static analysis, and dynamic analysis. The most of the techniques use elements of design as input to identify microservice candidates. This correlates to some studies identified for the Activity *3. Decompose the legacy system* of our roadmap. Fritzsch et al. [19] performed a survey with experts that adopted microservice to identify migration strategies. For example, they identify different ways to decompose monolithic systems into microservices. They mentioned the use of the *strangler pattern* and *functional decomposition* as strategies. Carvalho et al. [4] performed a survey with experienced practitioners that adopted microservices. They investigated the criteria used for supporting decision-making along microservice extraction. The practitioners mentioned that they commonly consider at least four criteria to identify microservices in legacy systems. Wang et al. [50] also performed a survey and interviews with practitioners. In this study they investigated the best practices learned by practitioners that adopted microservices. These authors also investigated the challenges faced by practitioners and how these challenges were overcome. These four studies differ from ours, as they focus on specific topics and issues of the migration, rather than how the modernization process is conducted as a whole.

Fritzsch et al. [20] describes a guide for decision making that classifies the decomposition techniques to aid the architects during the migration from monolithic to microservices. They discuss refactoring activities to support the modernization with microservices. They mention that decomposition of monolithic into services with appropriate granularity is the main challenge during the refactoring process. Similarly, Francesco et al. [10] identify and classify approaches to modernize with microservices. They found that most existing approaches focus on handling the trade-off between flexibility and complexity in the microservice architecture. Differently from our study, the goal of these studies is not the definition of a roadmap for aiding the conducting of the modernization process.

The work most similar to ours is presented by Silva et al. [8], in which the authors present a modernization process based on lessons learned from two studies (a pilot and a case study). The main drawback of their process is its validity, since it relies only on authors' experience. Our roadmap presents a more robust process based on a systematic mapping of the literature and on the results obtained by several researchers. In addition, we present two additional phases/activities compared to Silva et al. [8].

There are other secondary studies that explore the microservice architectural style and its relation to important concepts/technologies such as, devops [49], granularity [22], SOA [14], microservice gains and pains [33], and microservice smells [46]. However, their motivations are different from our study. Although these studies provide useful insights to understand the migration process, they do not focus on defining a generic roadmap. In this context, to the best of our knowledge, our paper is the first to describe recommendations based on 62 studies about migration.

6 CONCLUSION

Modernization of monolithic systems with microservices is a trend, which has called the attention of the software engineering research community. However, despite the existence of many studies on this topic, most of them focus on specific points of the modernization. We observed that there is a lack of a comprehensive study investigating the modernization of legacy systems with microservices as a whole. To fill this gap, this paper presented a study based on a systematic mapping method to answer the research question: *Why and how are monolithic legacy systems migrated to microservices?*

The answer for this question is that, based on 62 studies, we find 11 different driving forces that motivate the modernization. These driving forces are related to technical, operational, and organizational aspects. By an extensive analysis of 54 studies, we could observe that the modernization is composed of eight activities, covering the phases of initiation, planning, execution, and monitoring, which can be accomplished in an incremental manner during the migration process. The identified activities were the basis to define a comprehensive roadmap to contribute to practitioners, researchers and tool builders.

As future work, we consider the validation of our roadmap in industrial settings. More studies focusing the activities less investigated, namely execution of the modernization, integration of microservices and legacy, verification and validation, and monitoring of the modernized system and infrastructure. We also see the opportunity for the development of tools to support the modernization.

ACKNOWLEDGMENTS

This work is supported by the FAPERJ PDR-10 program under Grant No. 202073/2020 and CNPq under Grant No. 408356/2018-9.

REFERENCES

[1] 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.

[2] Armin Balalaie, Abbas Heydarnoori, Pooyan Jamshidi, Damian A. Tamburri, and Theo Lynn. 2018. Microservices migration patterns. *Software Prac. Experience* 48, 11 (2018), 2019–2042.

[3] Len Bass, Ingo Weber, and Liming Zhu. 2015. *DevOps: A software architect's perspective.* Addison-Wesley Professional.

[4] 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 International Workshop on Conducting Empirical Studies in Industry and 6th International Workshop on Software Engineering Research and Industrial Practice*. IEEE, 22–29.

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

[6] E. J. Chikofsky and J. H. Cross. 1990. Reverse engineering and design recovery: a taxonomy. *IEEE Software* 7, 1 (1990), 13–17.

[7] Juliet Corbin and Anselm Strauss. 2014. *Basics of qualitative research: Techniques and procedures for developing grounded theory.* Sage publications.

[8] Hugo S. da Silva, Glauco Carneiro, and Miguel Monteiro. 2019. Towards a Roadmap for the Migration of Legacy Software Systems to a Microservice based Architecture. In *9th International Conference on Cloud Computing and Services Science*. SciTePress.

[9] Apurva Dave and Loris Degioanni. 2016. *The Five Principles of Monitoring Microservices.* https://thenewstack.io/five-principles-monitoring-microservices/

[10] Paolo Di Francesco, Patricia Lago, and Ivano Malavolta. 2018. Migrating towards microservice architectures: an industrial survey. In *International conference on software architecture*. IEEE, 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 (2019), 77–97.

[12] Bogdan Dit, Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. 2013. Feature location in source code: a taxonomy and survey. *Journal of software: Evolution and Process* 25, 1 (2013), 53–95.

[13] Thomas Erl. 2005. *Service-Oriented Architecture: Concepts, Technology, and Design.* Prentice Hall e, Upper Saddle River, NJ.

[14] Tom Černý, Michael Donahoo, and Michal Trnka. 2018. Contextual understanding of microservice architecture: current and future directions. *ACM SIGAPP Applied Computing Review* 17 (01 2018), 29–45.

[15] Eric Evans. 2004. *Domain-driven design: tackling complexity in the heart of software.* Addison-Wesley Professional.

[16] Brian Foote and Joseph Yoder. 1997. Big ball of mud. *Pattern languages of program design* 4 (1997), 654–692.

[17] Susan J. Fowler. 2016. *Production-Ready Microservices: Building Standardized Systems Across an Engineering Organization.* O'Reilly.

[18] P. D. Francesco, I. Malavolta, and P. Lago. 2017. Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption. In *International Conference on Software Architecture*. 21–30.

[19] Jonas Fritzsch, Justus Bogner, Stefan Wagner, and Alfred Zimmermann. 2019. Microservices Migration in Industry: Intentions, Strategies, and Challenges. In *International Conference on Software Maintenance and Evolution*. IEEE, 481–490.

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

[21] A. Sivagnana Ganesan and T. Chithralekha. 2016. A Survey on Survey of Migration of Legacy Systems. In *International Conference on Informatics and Analytics* (India). ACM, New York, NY, USA.

[22] Sara Hassan, Rami Bahsoon, and Rick Kazman. 2020. Microservice transition and its granularity problem: A systematic mapping study. *Software: Practice and Experience* (06 2020).

[23] Wilhelm Hasselbring. 2018. Software architecture: Past, present, future. In *The Essence of Software Engineering*. Springer, Cham, 169–184.

[24] Pete Hodgson. 2017. *Feature Toggles (aka Feature Flags).* https://martinfowler.com/articles/feature-toggles.html

[25] Anca Daniela Ionita, Marin Litoiu, and Grace Lewis. 2012. *Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments* (1st ed.). IGI Global, USA.

[26] Harold Kerzner. 2017. *Project management: a systems approach to planning, scheduling, and controlling.* John Wiley & Sons.

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

[28] Rainer Koschke. 2003. Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey. *Journal of Software Maintenance and Evolution: Research and Practice* 15, 2 (2003), 87–109.

[29] James Lewis and Martin Fowler. 2014. *Microservices: a Definition of This New Architectural Term.* https://www.martinfowler.com/articles/microservices.html

[30] Robert C Martin. 2002. The single responsibility principle. *The principles, patterns, and practices of Agile Software Development* (2002), 149–154.

[31] Marjan Mernik. 2017. Domain-specific languages: A systematic mapping study. In *International Conference on Current Trends in Theory and Practice of Informatics*. Springer, 464–472.

[32] Jason Miller. 2018. *Spending on legacy IT continues to grow, but there is light at the end of the tunnel.* https://federalnewsnetwork.com/ask-the-cio/2018/08/spending-on-legacy-it-continues-to-grow-but-there-is-light-at-the-end-of-the-tunnel/.

[33] 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, 1-2 (sep 2019), 3–15.

[34] Sam Newman. 2015. *Building Microservices* (1st ed.). O'Reilly Media.

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

[36] Helena Holmström Olsson and Jan Bosch. 2014. Climbing the "Stairway to Heaven": evolving from agile development to continuous deployment of software. In *Continuous software engineering*. Springer, 15–27.

[37] David L Parnas. 1972. On the criteria to be used in decomposing systems into modules. In *Pioneers and Their Contributions to Software Engineering*. Springer, 479–498.

[38] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. 2008. Systematic Mapping Studies in Software Engineering. In *12th International Conference on Evaluation and Assessment in Software Engineering* (Italy) *(EASE)*. British Computer Society, Swinton, UK, 68–77.

[39] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology* 64 (2015), 1–18.

[40] PMI. 2017. *A guide to the project management body of knowledge (PMBOK guide)* (6th ed.). Project Management Institute.

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

[42] Michael Pratt. 2020. *Ensuring backwards compatibility in distributed systems.* https://stackoverflow.blog/2020/05/13/ensuring-backwards-compatibility-in-distributed-systems/

[43] Julia Rubin and Marsha Chechik. 2013. A survey of feature location techniques. In *Domain Engineering*. Springer, 29–58.

[44] Robert Seacord, Santiago Comella-Dorda, Grace Lewis, Patrick Place, and Daniel Plakosh. 2001. *Legacy System Modernization Strategies.* Technical Report CMU/SEI-2001-TR-025. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=5729

[45] Robert C. Seacord, Daniel Plakosh, and Grace A. Lewis. 2003. *Modernizing Legacy Systems: Software Technologies, Engineering Process and Business Practices.* Addison-Wesley Longman Publishing Co., Inc., USA.

[46] Jacopo Soldani, Damian Tamburri, and Willem-Jan Heuvel. 2018. The Pains and Gains of Microservices: A Systematic Grey Literature Review. *Journal of Systems and Software* 146 (09 2018).

[47] Davide Taibi and Valentina Lenarduzzi. 2018. On the Definition of Microservice Bad Smells. *IEEE Software* vol 35 (05 2018).

[48] 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.

[49] Davide Taibi, Valentina Lenarduzzi, and Claus Pahl. 2019. Continuous Architecting with Microservices and DevOps: A Systematic Mapping Study. In *Communications in Computer and Information Science*. Springer, 126–151.

[50] Yingying Wang, Harsha Kadyala, and Julia Rubin. 2020. Promises and Challenges of Microservices: an Exploratory Study. *Empirical Software Engineering* (2020), 1–45.

[51] Coburn Watson, Scott Emmons, and Brendan Gregg. 2015. *A Microscope on Microservices.* http://techblog.netflix.com/2015/02/a-microscope-on-microservices.html

A PRIMARY SOURCES

[52] Muhammad Abdullah, Waheed Iqbal, and Abdelkarim Erradi. 2019. Unsupervised learning approach for web application auto-decomposition into microservices. *Journal of Systems and Software* 151 (2019).

[53] Stephen Abrams, Patricia Cruse, John Kunze, and David Minor. 2010. Curation micro-services: A pipeline metaphor for repositories. In *5th International Conference on Open Repositories*.

[54] Mohsen Ahmadvand and Amjad Ibrahim. 2016. Requirements Reconciliation for Scalable and Secure Microservice (De)composition. In *IEEE 24th International Requirements Engineering Conference Workshops (REW)*.

[55] Omar Al-Debagy and Peter Martinek. 2019. A New Decomposition Method for Designing Microservices. *Periodica Polytechnica Electrical Engineering and Computer Science* 63, 4 (2019).

[56] Adambarage Anuruddha Chathuranga De Alwis, Alistair Barros, Colin Fidge, and Artem Polyvyanyy. 2018. Discovering Microservices in Enterprise Systems Using a Business Object Containment Heuristic. In *Lecture Notes in Computer Science*.

[57] Mohammad Javad Amiri. 2018. Object-Aware Identification of Microservices. In *IEEE International Conference on Services Computing (SCC)*.

[58] Martín Arévalo, Carlos Escobar, Pascal Monasse, Nelson Monzón, and Miguel Colom. 2017. The IPOL Demo System: A Scalable Architecture of Microservices for Reproducible Research. In *Reproducible Research in Pattern Recognition*.

[59] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. 2016. Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. *IEEE*

Software 33, 3 (2016).
[60] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. 2016. Migrating to Cloud-Native Architectures Using Microservices: An Experience Report. In *Communications in Computer and Information Science*.
[61] Luciano Baresi, Martin Garriga, and Alan De Renzis. 2017. Microservices Identification Through Interface Analysis. In *Service-Oriented and Cloud Computing*.
[62] Amina Boubendir, Emmanuel Bertin, and Noemie Simoni. 2017. A VNF-as-a-service design through micro-services disassembling the IMS. In *20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*.
[63] Antonio Bucchiarone, Kemal Soysal, and Claudio Guidi. 2020. A Model-Driven Approach Towards Automatic Migration to Microservices. In *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*.
[64] Luiz Carvalho, Alessandro Garcia, Wesley K. G. Assunção, Rodrigo Bonifácio, Leonardo P. Tizzei, and Thelma Elita Colanzi. 2019. EExtraction of configurable and reusable microservices from legacy systems: An exploratory study. In *23rd International Systems and Software Product Line Conference) - volume A*.
[65] Rui Chen, Shanshan Li, and Zheng Li. 2017. From Monolith to Microservices: A Dataflow-Driven Approach. In *24th Asia-Pacific Software Engineering Conference*.
[66] Andreas Christoforou, Lambros Odysseos, and Andreas Andreou. 2019. Migration of Software Components to Microservices: Matching and Synthesis. In *14th International Conference on Evaluation of Novel Approaches to Software Engineering*.
[67] Michel Cojocaru, Alexandru Uta, and Ana-Maria Oprescu. 2019. MicroValid: A Validation Framework for Automatically Decomposed Microservices. In *IEEE International Conference on Cloud Computing Technology and Science*.
[68] Hugo S. da Silva, Glauco Carneiro, and Miguel Monteiro. 2019. Towards a Roadmap for the Migration of Legacy Software Systems to a Microservice based Architecture. In *9th International Conference on Cloud Computing and Services Science*.
[69] Hugo S. da Silva, Glauco Carneiro, and Miguel Monteiro. 2019. Towards a Roadmap for the Migration of Legacy Software Systems to a Microservice based Architecture. In *9th International Conference on Cloud Computing and Services Science*.
[70] Daniel Escobar, Diana Cardenas, Rolando Amarillo, Eddie Castro, Kelly Garces, Carlos Parra, and Rubby Casallas. 2016. Towards the understanding and evolution of monolithic applications as microservices. In *42nd Latin American Computing Conference*.
[71] Sinan Eski and Feza Buzluca. 2018. An Automatic Extraction Approach - Transition to Microservices Architecture from Monolithic Application. In *19th International Conference on Agile Software Development Companion (XP)*.
[72] Chen-Yuan Fan and Shang-Pin Ma. 2017. Migrating Monolithic Mobile Application to Microservice Architecture: An Experiment Report. In *IEEE International Conference on AI & Mobile Services (AIMS)*.
[73] Paolo Di Francesco, Patricia Lago, and Ivano Malavolta. 2018. Migrating Towards Microservice Architectures: An Industrial Survey. In *International Conference on Software Architecture*.
[74] Arvind Gopu, Soichi Hayashi, Michael D. Young, Ralf Kotulla, Robert Henschel, and Daniel Harbeck. 2016. Trident: scalable compute archives: workflows, visualization, and analysis. In *Software and Cyberinfrastructure for Astronomy IV*, Gianluca Chiozzi and Juan C. Guzman (Eds.).
[75] Jean-Philippe Gouigoux and Dalila Tamzalit. 2017. From Monolith to Microservices: Lessons Learned on an Industrial Migration to a Web Oriented Architecture. In *International Conference on Software Architecture Workshops*.
[76] Dong Guo, Wei Wang, Jingxuan Zhang, Qiao Xiang, Chenxi Huang, Jinda Chang, and Liqing Zhang. 2016. Cloudware: An emerging software paradigm for cloud computing. In *8th Asia-Pacific Symp. on Internetware*.
[77] Michael Gysel, Lukas Kölbener, Wolfgang Giersche, and Olaf Zimmermann. 2016. Service Cutter: A Systematic Approach to Service Decomposition. In *Service-Oriented and Cloud Computing*.
[78] Sara Hassan, Nour Ali, and Rami Bahsoon. 2017. Microservice Ambients: An Architectural Meta-Modelling Approach for Microservice Granularity. In *International Conference on Software Architecture*.
[79] Sara Hassan and Rami Bahsoon. 2016. Microservices and Their Design Trade-Offs: A Self-Adaptive Roadmap. In *IEEE International Conference on Services Computing (SCC)*.
[80] Wilhelm Hasselbring and Guido Steinacker. 2017. Microservice Architectures for Scalability, Agility and Reliability in E-Commerce. In *International Conference on Software Architecture Workshops*.
[81] Alexis Henry and Youssef Ridene. 2019. Migrating to Microservices. In *Microservices*. Springer, 45–72.
[82] Baskaran Jambunathan and Y Kalpana. 2016. Multi cloud deployment with containers. *International J. of Engineering and Technology* 8, 1 (2016).
[83] Andrea Janes and Barbara Russo. 2019. Automatic Performance Monitoring and Regression Testing During the Transition from Monolith to Microservices. In *International Symp. on Software Reliability Engineering Workshops*.
[84] Wuxia Jin, Ting Liu, Qinghua Zheng, Di Cui, and Yuanfang Cai. 2018. Functionality-Oriented Microservice Extraction Based on Execution Trace Clustering. In *IEEE International Conference on Web Services (ICWS)*.
[85] Manabu Kamimura, Keisuke Yano, Tomomi Hatano, and Akihiko Matsuo. 2018. Extracting Candidates of Microservices from Monolithic Application Code. In *25th Asia-Pacific Software Engineering Conference*.
[86] Gabor Kecskemeti, Attila Csaba Marosi, and Attila Kertesz. 2016. The ENTICE approach to decompose monolithic services into microservices. In *2016 International Conference on High Performance Computing & Simulation*.
[87] Sander Klock, Jan Martijn E. M. van der Werf, Jan Pieter Guelen, and Slinger Jansen. 2017. Workload-Based Clustering of Coherent Feature Sets in Microservice Architectures. In *International Conference on Software Architecture*.
[88] Holger Knoche. 2016. Sustaining Runtime Performance while Incrementally Modernizing Transactional Monolithic Software towards Microservices. In *7th International Conference on Performance Engineering*.
[89] Holger Knoche and Wilhelm Hasselbring. 2018. Using Microservices for Legacy Software Modernization. *IEEE Software* 35, 3 (2018).
[90] Alessandra Levcovitz, Ricardo Terra, and Marco Tulio Valente. 2015. Towards a Technique for Extracting Microservices from Monolithic Enterprise Systems. In *3rd Brazilian Workshop on Software Visualization, Evolution and Maintenance*.
[91] Frank Leymann, Johannes Wettinger, Sebastian Wagner, and Christoph Fehling. 2016. Native Cloud Applications - Why Virtual Machines, Images and Containers Miss the Point!. In *6th International Conference on Cloud Computing and Services Science*.
[92] Shanshan Li, He Zhang, Zijia Jia, Zheng Li, Cheng Zhang, Jiaqi Li, Qiuya Gao, Jidong Ge, and Zhihao Shan. 2019. A dataflow-driven approach to identifying microservices from monolithic applications. *J. of Systems and Software* 157 (2019).
[93] Jyhjong Lin, Lendy Chaoyu Lin, and Shiche Huang. 2016. Migrating web applications to clouds with microservice architectures. In *International Conference on Applied System Innovation (ICASI)*.
[94] David S. Linthicum. 2016. Practical Use of Microservices in Moving Workloads to the Cloud. *IEEE Cloud Computing* 3, 5 (2016).
[95] Prabal Mahanta and Suchin Chouta. 2020. Translating a Legacy Stack to Microservices Using a Modernization Facade with Performance Optimization for Container Deployments. In *Workshops On the Move to Meaningful Internet Systems (OTM)*.
[96] Salvatore Augusto Maisto, Beniamino Di Martino, and Stefania Nacchia. 2019. From Monolith to Cloud Architecture Using Semi-automated Microservices Modernization. In *Advances on P2P, Parallel, Grid, Cloud and Internet Computing*.
[97] Genc Mazlami, Jurgen Cito, and Philipp Leitner. 2017. Extraction of Microservices from Monolithic Software Architectures. In *IEEE International Conference on Web Services (ICWS)*.
[98] Alan Megargel, Venky Shankararaman, and David K. Walker. 2020. Migrating from Monoliths to Cloud-Based Microservices: A Banking Industry Example. In *Computer Communications and Networks*.
[99] Ola Mustafa, Jorge Marx Gómez, Mohamad Hamed, and Hergen Pargmann. 2017. GranMicro: A Black-Box Based Approach for Optimizing Microservices Based Applications. In *Progress in IS*.
[100] Petru Nicolaescu and Ralf Klamma. 2015. A Methodology and Tool Support for Widget-Based Web Application Development. In *Engineering the Web in the Big Data Era*.
[101] Luís Nunes, Nuno Santos, and António Rito Silva. 2019. From a Monolith to a Microservices Architecture: An Approach Based on Transactional Contexts. In *Software Architecture*.
[102] Joonseok Park, Mikyeong Moon, and Keunhyuk Yeom. 2019. Approach to Identify Microservices based on Analysis Class Model. *International J. of Advanced Science and Technology* 28, 4 (2019).
[103] Ilaria Pigazzini, Francesca Arcelli Fontana, and Andrea Maggioni. 2019. Tool Support for the Migration to Microservice Architecture: An Industrial Case Study. In *Software Architecture*.
[104] 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 *10th Asia-Pacific Symp. on Internetware*.
[105] Anika Sayara, Md. Shamim Towhid, and Md. Shahriar Hossain. 2017. A probabilistic approach for obtaining an optimized number of services using weighted matrix and multidimensional scaling. In *20th International Conference of Computer and Information Technology (ICCIT)*.
[106] Walter Scarborough, Carrie Arnold, and Maytal Dahan. 2016. Case Study: Microservice Evolution and Software Lifecycle of the XSEDE User Portal API *(XSEDE16)*. New York, NY, USA.
[107] Anfel Selmadji, Abdelhak-Djamel Seriai, Hinde Lilia Bouziane, Christophe Dony, and Rahina Oumarou Mahamane. 2018. Re-architecting OO Software into Microservices - A Quality-Centred Approach. In *Service-Oriented and Cloud Computing*.
[108] Atsushi Shimoda and Tsubasa Sunada. 2018. Priority Order Determination Method for Extracting Services Stepwise from Monolithic System. In *7th International Congress on Advanced Applied Informatics (IIAI-AAI)*.

[109] Davide Taibi and Kari Systä. 2019. From Monolithic Systems to Microservices: A Decomposition Framework based on Process Mining. In *9th International Conference on Cloud Computing and Services Science*.

[110] Tomohiro Takeda, Masakazu Takahashi, Tsuyoshi Yumoto, Satoshi Masuda, Tohru Matsuodani, and Kazuhiko Tsuda. 2019. Applying Change Impact Analysis Test to Migration Test Case Extraction Based on IDAU and Graph Analysis Techniques. In *International Conference on Software Testing, Verification and Validation Workshops*.

[111] Dmitri Tchoubraev and Daniel Wiczynski. 2015. Swiss TSO integrated operational planning, optimization and ancillary services system. In *2015 IEEE Eindhoven PowerTech*.

[112] Shmuel Tyszberowicz, Robert Heinrich, Bo Liu, and Zhiming Liu. 2018. Identifying Microservices Using Functional Decomposition. In *Dependable Software Engineering. Theories, Tools, and Applications*.

[113] Zhiping Luo UU, Michel Korpershoek, and AnaMaria Oprescu VU. 2015. Towards a MicroServices Architecture for Clouds. (2015).