Migrating from monolithic architecture to microservices: A Rapid Review

Francisco Ponce*, Gastón Márquez*, Hernán Astudillo*
*Universidad Técnica Federico Santa María, Valparaíso, Chile
Email: francisco.ponceme@sansano.usm.cl, gaston.marquez@sansano.usm.cl, hernan@inf.utfsm.cl

***Abstract*—Microservices architecture has become enormously popular because traditional monolithic architectures no longer meet the needs of scalability and rapid development cycle, and the success of some large companies in building and deploying services is a strong motivation for others to consider making the change. However, performing the migration process is not trivial. Most systems acquire too many dependencies between their modules, and thus can't be sensibly broken apart. It is for this reason that studies that provide information associated with the migration process to practitioners are necessary. This study gathers, organizes and analyzes 20 migration techniques proposed in the literature. Results show that most proposals use approaches based on design elements as input; 90% of the proposals were applied to object-oriented software (Java being the predominant programming language); And that the main challenge is to perform the database migration.**

***Index Terms*—Microservices, Microservice Architecture, Migration to Microservices, Monolithic to Microservices**

I. Introduction

Microservices architecture has become enormously popular because traditional monolithic architectures no longer meet the needs of scalability and rapid development cycle [1], and the success of some large companies in building and deploying services is a strong motivation for others to consider making the change. Typical issues associated with monolithic architecture are technical (e.g., the system becomes highly coupled, hard to maintain, presents side effects) or business-related (e.g., long time to release new features, low productivity of developers). In some cases, migrating towards microservices architecture represents the best option for resolving existing issues and at the same time improving the system maintainability and the frequency of product releases [2].

The goal of this article is to gather, organize, and analyze the migration techniques proposed in the literature. Because the time available for this study is not compatible with a Full Systematic Reviews (see Section II-C), to achieve our goal, we conducted a Rapid Review following the protocol proposed in [3], and we complimented the Rapid Review process with the strategies presented in [4].

The results of the Rapid Review show that most proposals use approaches based on design elements as input (45%); The most predominant techniques is to generate a system dependency graph and then use a clustering algorithm to generate the candidate microservices; 70% of the proposals made the migration of a Web-based system; 90% of the proposals were applied to object-oriented software (Java being the predominant programming language); And that the main challenge is to perform the database migration.

The remainder of the article is structured as follows: Section II describes the fundamental concepts associated with the study; Section III present the research questions and the strategy used; Section IV shows the Rapid Review results and answers the research questions; In section V the most interesting topics associated with the results are discussed; Section VI addresses threats to validity; Section VII describe the related work and finally Section VIII summarizes and concludes.

II. Background

A. Monolithic architecture

In a monolithic architecture, all functionality is encapsulated into one single application, so it's modules cannot be executed independently. This type of architecture is tightly-coupled, and all your logic for handling a request runs in a single process. This allows you to use the basic features of your language to divide up the application into classes, functions, and namespaces. With some care, you can run and test the application on a developer's laptop, and use a deployment pipeline to ensure that changes are properly tested and deployed into production. You can horizontally scale the monolith by running many instances behind a load-balancer [5].

While it is a good idea to start a project using this type of architecture, because this allows you to explore both the complexity of a system and its component boundaries [6]. Once the application becomes large and the team grows in size, this architecture has some drawbacks that become increasingly significant [5] [7]:

- The application can be difficult to understand and modify. As a result, development typically slows down.
- Continuous deployment is difficult; A change made to a small part of the application requires the entire monolith to be rebuilt and deployed.
- Scaling the application can be difficult (a monolithic architecture is that it can only scale horizontally).
- Requires a long-term commitment to a technology stack.

B. Microservices architecture

The microservices architecture (MSA) style is an approach to developing a single application as a suite of small services,

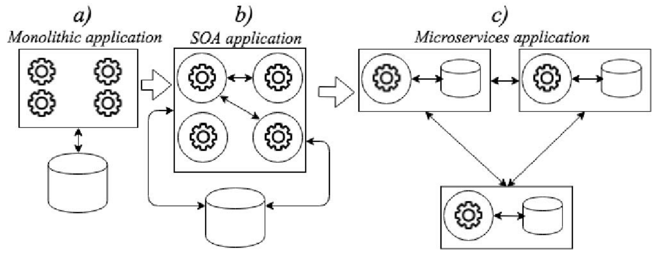


Fig. 1. From monolithic applications to microservices-based systems

each running in its own process and communicating with lightweight mechanisms (e.g. HTTP API's) [5]. Microservices are built around business capabilities and independently deployable by fully automated deployment machinery. Because of their size, they are easier to maintain and more fault-tolerant since the failure of one service will not break the whole system, which could happen with a monolithic architecture. Therefore, this style allows designing architectures that should be flexible, modular and easy to evolve [5]. Although there has not been a broad acceptance of a specific definition, the microservices approach can be seen as a well-defined implementation of Service-Oriented Architecture (SOA) [8].

In recent years, many large systems evolve from self-contained *monolithic* applications built of interconnected, interdependent components (see Fig. 1-a) to collections of large services (i.e. traditional SOA), and further to collections of small, autonomous, lightweight-connected services (see Fig. 1-c). The market's high pace of demand for new application features requires changes both in the applications themselves (loose coupling and high scalability) and in the way they are built (loose team dependencies and fast deployment). Microservices address both concerns since small services can be built and deployed by independent development teams; the concomitant freedom allows teams to focus on improving each service and increase business value. Hence, in practice, DevOps (Development and Operations) and Continuous Delivery [9] are a close fit for microservice architectures.

C. Rapid Review

A rapid review (RR) is a type of knowledge synthesis in which components of the systematic review process are simplified or omitted to produce information in a short time [10].

RRs are lightweight secondary studies focused on delivering evidence promptly [3]. Some steps of Full Systematic Reviews (e.g., mapping studies, meta-analysis, or even the traditional systematic literature reviews) are deliberately omitted or simplified in RRs to achieve their proposed goal. In spite of this, there is evidence that if the RR is applied correctly and complemented by a rigorous snowball process, you can get the same results that in a Full Systematic Reviews [11] [12] [13]. Moreover, RRs should be seen as a complementary method and not a substitute for Full Systematic Reviews. Rapid Reviews have some core characteristics [10]:

- It reduces the costs of heavyweight methods.
- It delivers evidence in a timely manner.
- It reports results through appealing mediums.
- It operates in a close collaboration with practitioners.

Furthermore, one should not confuse RRs with informal literature reviews. RRs follow systematic protocols, although some methodological decisions aiming to deliver evidence in less time might introduce bias. The informal literature reviews, on the other side, do not have even a systematic protocol, being conducted adhoc.

III. Study design

In order to perform the Rapid Review, we followed the protocol proposed in [3], and we complemented the Rapid Review process with the of the strategies presented in [4] for performing systematic literature reviews. The following subsections describe in detail the study design and its execution.

A. Goals and research questions

The study's goal is to gather, organize, and analyze the migration techniques proposed in the literature. Based on this goal, we defined the following research questions:

- **RQ1**: *What are the migration techniques proposed in the literature?*
- **RQ2**: *In what types of systems have the proposed techniques been applied?*
- **RQ3**: *What type of validation do the authors of the techniques use?*
- **RQ4**: *Are there challenges associated with migration from monolith to microservices?*

B. Search Strategy

As recommended in [3] to abbreviate the search for primary studies, and conduct the Rapid Review within the available time, we used only Scopus[1] search engine, however, to reduce the possibility of missing literature, we added Google Scholar in the snowballing process.

To improve the search string, we conducted pilot searches and we excluded those keywords that did not yield new search results (for example, the *monolith** keyword). So we tested many different versions of the search String until we found one that returned relevant papers.

Based on the above, we defined the following search string: (*migrat** OR *transform** OR *extract** OR *decompos**) AND (*microservi** OR *micro-servi** OR *"micro servi"**).

C. Selection Procedure

The selection procedure was based on the following criteria:

1) The study must be in English.
2) The study must be in the context of migration from monolithic to microservices.
3) The study must be a primary study (i.e., we do not consider secondary studies).
4) The study must provide answers to at least one of the rapid review's research questions.

[1]https://www.scopus.com

We define 4 stages in which the criteria described above were applied. In the first stage, a solo researcher reads the title and keywords of the articles; In the second stage a solo researcher reviews the abstract of each article; In the third stage a solo researcher performs the complete reading of each article, and finally, a snowballing process is performed. In each of the stages a scoring system was used to determine which article goes to the next one (0 points if it does not meet the criteria, 0.5 points if there are doubts in the decision, 1 point if it meets the criteria). The process and its results can be seen in https://bit.ly/2LXRsKt.

D. Extraction Procedure

In this step, we extracted all relevant data that could help to answer any of the research questions. The extraction process was performed by 2 authors. We used a worksheet to tabulate and organize data. The information extracted from the papers and their relation with the research questions can be observed in https://bit.ly/31lUkEO.

IV. Results

The RR was performed part-time from 06-May-2019 until 14-June-2019, executing the phases mentioned in subsection III-C.

Fig. 2 summarizes the Rapid Review process executed in this study. The search in Scopus returned 261 studies. In the first round, a solo researcher analyzed the studies' titles and keywords and excluded those that did not meet the criterion's defined, resulting in 142 studies. In the second round, a solo researcher analyzed the studies' abstract, resulting in 44 studies. In the third round, a solo researcher analyzed the entire paper's content and excluded those that could not answer any of the research questions, resulting in 17 studies. In the fourth round, solo research proceeded to perform a snowballing process, as recommended in [14], adding 3 more studies that meet all the criteria. To avoid possible missing literature, the snowballing process was performed using the database in which the article is hosted in conjunction with Google Scholar. Thus, we ended up with 20 selected primary studies.

RQ1: *What are the migration techniques proposed in the literature?*

When analyzing the selected studies, we identify different migration approaches, so to organize the proposals, we classify the different approaches as follows:

- **Model-Driven (MD)**, approaches that use design elements as input (e.g., business capabilities, business objects, domain entities, functional and non-functional requirements, data flow diagrams from business logic). **Example**: In [15] they employ an iterative process. In phase 1, they analyze the interval system architecture using Domain-Driven Design (DDD). With this process the extracted the candidate microservices. In phase 2 they analyze the database schema to verify if it is consistent with the candidate microservices. They end this phase filtering out of inappropriate candidates. Finally, in phase 3, they extract and organize the code related to the service candidates.
- **Static analysis (SA)**, approaches that require the source code as input.
  Example: In [16] they use a Graph-based microservice clustering approach using static and evolutionary coupling between software classes. This approach uses the project repository as input. They generated two modules; A static analysis module that extracts static couplings on the final revision of an investigated project by parsing abstract syntax trees. An evolutionary analysis module that extracts and saves software change information; Extracting the evolutionary coupling by detecting changes between consecutive commits.
- **Dynamic analysis (DA)**, approaches that analyze the system functionalities at runtime.
  Example: In [17] they use execution traces to guide the grouping of source code entities that are dedicated to the same functionality. They use an execution-oriented clustering method towards grouping similar functionalities as a service (execution traces are generated using the given test cases).

These approaches are not exclusive, they can be used together without problems.

As can be seen in Fig. 3 the predominant approach is to use only MD (45%, 9/20). Followed by using only SA (30%, 6/20). In general, the most predominant technique is to

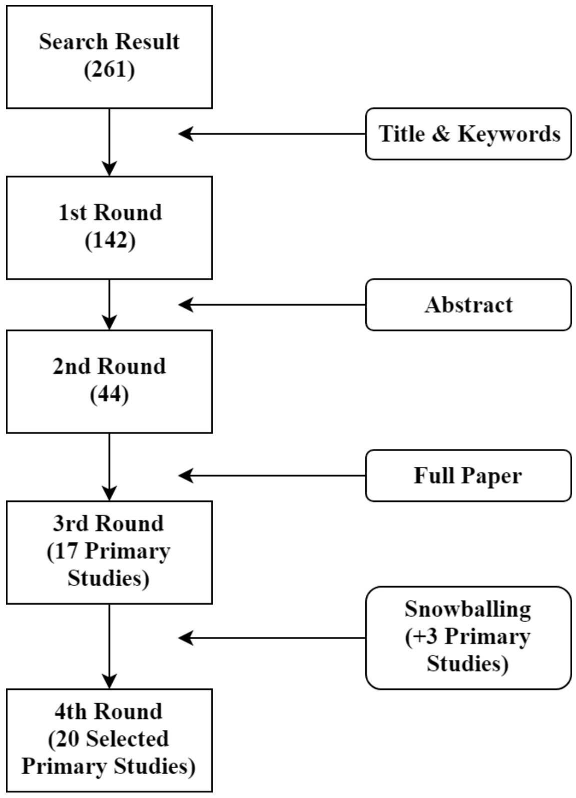


Fig. 2. Selection Procedure Results

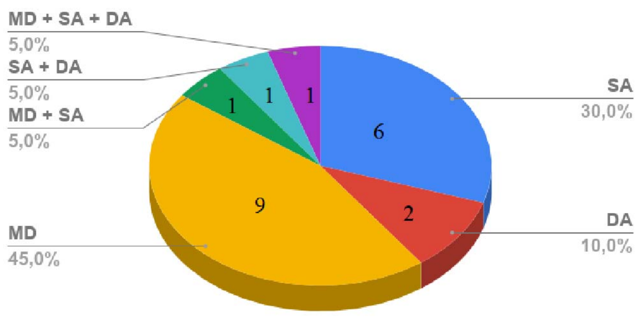


Fig. 3. Distribution of the approach used by the proposals

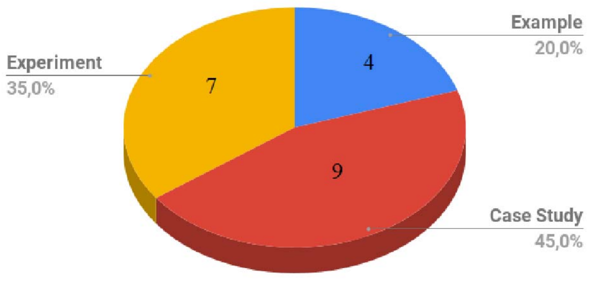


Fig. 4. Distribution of validation types

generate a system dependency graph and then use a clustering algorithm to generate the candidate microservices.

All the proposed migration techniques can be observed in Table I. Fig. 5 shows the distribution of proposals by year and type of publication. 85% (17/20) of the proposals were published in a Conference, 10% (2/20) in a Journal and 5% (1/20) in a Workshop.

RQ2: *In what types of systems have the proposed techniques been applied?*

70% (14/20) of the proposals made the migration of a Web-based system. We consider a system as Web-based if it is developed with Web technologies, hosted on remote servers, served via standard protocols (e.g., HTTP), or accessed via a unique URL.

The distribution of the programming languages with which the systems used by the proposals were developed can be seen in Table II. 90% (18/20) of the proposals were applied to object-oriented software, Java is the predominant programming language.

Only the 2/20 proposals [31] [34] made the migration of a system developed with a structured programming language (C and COBOL respectively).

RQ3: *What type of validation do the authors of the techniques use?* As can be seen in Fig. 4, most of the proposals were validated through a case study (9/20, 45%). 35% (7/20) were validated through experiments, generally associated with system performance (comparison monolithic architecture versus microservices). Finally, 20% (4/20) illustrated the proposal through an example.

RQ4: *Are there challenges associated with migration from monolith to microservices?*

Not all the proposals collected describe the challenges they faced during the process, however, the most remarkable information we could collect is listed below.

- Database migration.
- Divide business capabilities in sub-business capabilities.
- Candidate microservices could still need expert judgment before being developed in practice.
- Distributed system development needs skilled developers.
- Resources management.
- Complex environment setting, many automation tools are needed.
- Transactions are notoriously difficult to implement.
- The organization must adapt to the new way of working.

The challenge that is transversal to most of the proposals is the migration of the database. This topic is addressed in more detail in Section V-A.

V. Discussion

A. Database migration

Of the 20 proposals, only 2 of them talk about the decomposition of the monolithic system database. In [15] it is stated that Foreign keys can be marked as microservice candidates. While in [26] it is stated that to partition the data of a monolithic application, the pattern of the data access knowledge should be the basement of the partition procedure.

This result is really interesting since in [2] they carried out a survey of 18 practitioners that provides quantitative information about migrations towards MSA. In this survey, more than half of the participants reported that the data existing during the migration was not migrated. The authors state that not migrating the database may hinder the ability to evolve services independently, but also the possibility to scale up both services and their data.

Since the information obtained in our study is consistent with that obtained from the practitioners in [2], it is possible to affirm that database migration is still an open problem.

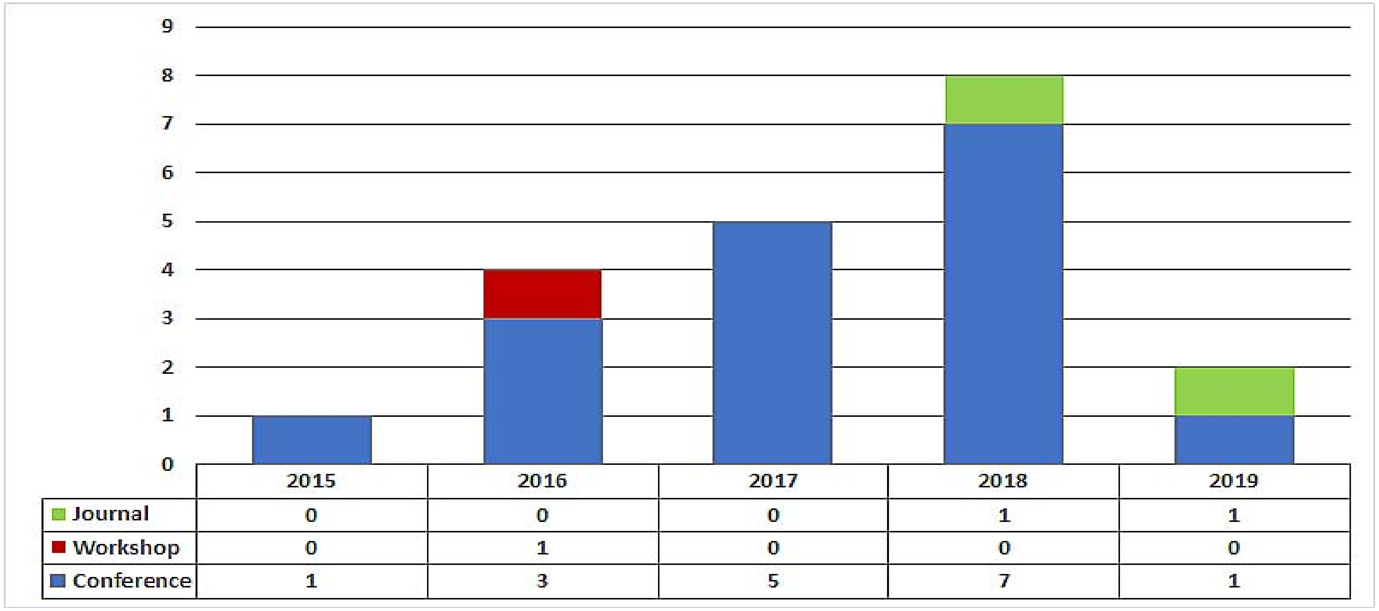
B. Migration techniques

It is interesting to see how most of the proposals (85%, 17/20) focus on using only one of the approaches (MD, SA or DA); Since it is possible to combine them without problems, we believe that more evidence of mixed proposals is needed. For example, none of them use the MD + DA combination. However, it should be noted that SA and DA approaches are generally associated with a particular technology (due to the type of tools needed for the process); This may be an indicator of why Java is the predominant language of the proposals.

On the other hand, since the MD approach uses design elements as input, it is avoided to link the technique to a

TABLE I
LIST OF SELECTED PRIMARY STUDIES AND APPROACH USED

[18]	A Probabilistic Approach For Obtaining An Optimized Number Of Services Using Weighted Matrix And Multidimensional Scaling	MD
[16]	An Automatic Extraction Approach - Transition to Microservices Architecture from Monolithic Application	SA
[19]	Discovering Microservices in Enterprise Systems Using a Business Object Containment Heuristic	MD+SA+DA
[20]	Extraction of Microservices from Monolithic Software Architectures	SA
[21]	From Monolith to Microservices: A Dataflow-Driven Approach	MD
[22]	From Monolithic Systems to Microservices: A Decomposition Framework based on Process Mining	DA
[17]	Functionality-oriented Microservice Extraction Based on Execution Trace Clustering	DA
[23]	Identifying Microservices Using Functional Decomposition	MD
[24]	Microservices Identification Through Interface Analysis	SA
[15]	Migrating Monolithic Mobile Application to Microservice Architecture: An Experiment Report	MD
[25]	Migrating to Cloud-Native Architectures Using Microservices: An Experience Report	MD
[26]	Migrating Web Applications from Monolithic Structure to Microservices Architecture	SA+DA
[27]	Object-aware Identification of Microservices	MD
[28]	Re-architecting OO Software into Microservices A Quality-Centred Approach	SA
[29]	Requirements Reconciliation for Scalable and Secure Microservice (De)composition	MD
[30]	Service Cutter: A Systematic Approach to Service Decomposition	MD
[31]	Towards a Technique for Extracting Microservices from Monolithic Enterprise Systems	MD
[32]	Towards the Understanding and Evolution of Monolithic Applications as Microservices	SA
[33]	Unsupervised learning approach for web application auto-decomposition into microservices	SA
[34]	Using Microservices for Legacy Software Modernization	MD+SA



	2015	2016	2017	2018	2019
🟩 Journal	0	0	0	1	1
🟥 Workshop	0	1	0	0	0
🟦 Conference	1	3	5	7	1

Fig. 5. Year and publication type distribution

TABLE II
PROGRAMMING LANGUAGES

Programming Language	Number of proposals
Java	12
C	1
Cobol	1
PHP	1
Python	1
Ruby	1

particular technology, but the important elements are neglected during the analysis (e.g., system performance or resource consumption). Therefore, the best option is to complement the different types of approaches.

C. Programming languages

Only one proposal uses the SA approach in non-object-oriented programming languages [34], and no proposals were found that use DA in non-object-oriented programming languages. This may be due to various reasons (availability of tools to perform the analysis for example); However, we believe that it is because these programming languages are generally associated with a particular type of systems (e.g., banking systems), and since for this type of systems performance and transactions are of vital importance, Microservices may not be the answer (this conclusion was obtained after the talk of Len Bass in the International Software Architecture Ph.D. School (iSAPS) 2019).

VI. Threats to validity

In this section, we proceed to discuss the threats to the validity [35] of our study. As exposed in [3] Rapid Reviews usually present more threats to validity than other secondary studies due to its lightweight methodology.

Threats to internal validity describe factors that could affect the results obtained from the study. (1) Only one search engine was used, which may limit the number of primary studies found. To mitigate this, the snowballing process was performed using the database in which the article is hosted in conjunction with Google Scholar. (2) The entire selection procedure was conducted by a solo researcher, which may introduce selection bias. To mitigate this, we use a scoring system in each of the phases, and the information gathering process was carried out by 2 researchers. However, it is not possible to mitigate this threat completely.

VII. Related work

In [36] discusses the notion of architectural refactoring and subsequently compares 10 existing refactoring approaches recently proposed in the academic literature. The approaches are classified by the underlying decomposition technique and visually presented in the form of a decision guide for quick reference. The review yielded a variety of strategies to break down a monolithic application into independent services. With one exception, most approaches are only applicable under certain conditions.

In [2] they report about an empirical study on migration practices towards the adoption of microservices in industry. Specifically, they designed and conducted a survey targeting practitioners involved in the process of migrating their applications and they collected information (utilizing interviews and questionnaires) on (i) the performed activities, and (ii) the challenges faced during the migration.

In [37] they elaborate on challenges and technics of legacy software migration from a monolithic architecture to microservice architecture. Different migration methods and techniques are reviewed, and their benefits and drawbacks are analyzed.

In [38] they report a set of migration and rearchitecting design patterns that they have empirically identified and collected from industrial-scale software migration projects. These migration patterns can help information technology organizations plan their migration projects toward microservices more efficiently and effectively. Also, the proposed patterns facilitate the definition of migration plans by pattern composition.

VIII. Conclusions

Microservices architecture has become enormously popular because traditional monolithic architectures no longer meet the needs of scalability and rapid development cycle, and the success of some large companies in building and deploying services is a strong motivation for others to consider making the change.

However, performing the migration process is not a trivial process. Most systems acquire too many dependencies between their modules, and thus can't be sensibly broken apart. It is for this reason that studies that provide information associated with the migration process to practitioners are necessary.

This study yields 20 unique articles associated with migration techniques. Results show that most proposals use approaches based on design elements as input (9/20); The most predominant techniques is to generate a system dependency graph and then use a clustering algorithm to generate the candidate microservices; 70% of the proposals made the migration of a Web-based system; and 90% of the proposals were applied to object-oriented software (Java being the predominant programming language). On the other hand, since the information obtained in our study is consistent with that obtained from the practitioners in [2], it is possible to affirm that database migration is still an open problem.

Finally, since the 3 approaches used to perform the migration (MD, SA, and DA) can be used together without problems, we believe that more evidence of mixed proposals is needed. For example, none of the proposals use the MD + DA combination.

Acknowledgments

This work was partly supported by Comisión Nacional de Investigación Científica (CONICYT) through grants PCHA/Doctorado Nacional/2019-21191132, PCHA/Doctorado Nacional/2016-21161005, Dirección de Postgrado y Programas and Universida Técnica Federico Santa María.

References

[1] S. Newman, "Building microservices: designing fine-grained systems," *O'Reilly Media, Inc.*, 2015.

[2] P. Di Francesco, P. Lago, and I. Malavolta, "Migrating towards microservice architectures: An industrial survey," in *2018 IEEE International Conference on Software Architecture (ICSA)*, April 2018, pp. 29–2909.

[3] P. G. Cartaxo, B. and S. Soares, "The role of rapid reviews in supporting decision-making in software engineering practice," in *Proceedings of the 22Nd International Conference on Evaluation and Assessment in Software Engineering 2018*, ser. EASE'18. New York, NY, USA: ACM, 2018, pp. 24–34. [Online]. Available: http://doi.acm.org/10.1145/3210459.3210462

[4] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," 2007.

[5] J. Lewis and M. Fowler, "Microservices. A definition of this new architectural term," https://martinfowler.com/articles/microservices.html, [Online; accessed May 27, 2019].

[6] M. Fowler, "Monolithfirst," https://martinfowler.com/bliki/MonolithFirst.html, [Online; accessed May 29, 2019].

[7] C. Richardson, "Pattern: Monolithic architecture," https://microservices.io/patterns/monolithic.html, [Online; accessed May 29, 2019].

[8] O. Zimmermann, "Microservices tenets," *Computer Science - Research and Development*, vol. 32, no. 3, pp. 301–310, Jul 2017. [Online]. Available: https://doi.org/10.1007/s00450-016-0337-0

[9] S. J. Fowler, *Production-ready Microservices: Building Standardized Systems Across an Engineering Organization.* O'Reilly Media, Inc., 2016.

[10] A. C. e. a. Tricco, "A scoping review of rapid review methods," *BMC Medicine*, 2015.

[11] B. L. et al, "Rapid and responsive health technology assessment: The development and evaluation process in the south and west region of england," *Journal of Clinical Effectiveness*, 1997.

[12] P. Corabian and C. Harstall, "Rapid assessments provide acceptable quality advice," *Annu Meet Int Soc Technol Assess Health Care*, 2002.

[13] S. e. a. Taylor-Phillips, "Comparison of a full systematic review versus rapid review approaches to assess a newborn screening test for tyrosinemia type 1," *Research Synthesis Methods*, 2017.

[14] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '14. New York, NY, USA: ACM, 2014, pp. 38:1–38:10. [Online]. Available: http://doi.acm.org/10.1145/2601248.2601268

[15] C. Fan and S. Ma, "Migrating monolithic mobile application to microservice architecture: An experiment report," in *2017 IEEE International Conference on AI Mobile Services (AIMS)*, June 2017, pp. 109–112.

[16] S. Eski and F. Buzluca, "An automatic extraction approach: Transition to microservices architecture from monolithic application," in *Proceedings of the 19th International Conference on Agile Software Development: Companion*, ser. XP '18. New York, NY, USA: ACM, 2018, pp. 25:1–25:6. [Online]. Available: http://doi.acm.org/10.1145/3234152.3234195

[17] W. Jin, T. Liu, Q. Zheng, D. Cui, and Y. Cai, "Functionality-oriented microservice extraction based on execution trace clustering," in *2018 IEEE International Conference on Web Services (ICWS)*, July 2018, pp. 211–218.

[18] A. Sayara, M. S. Towhid, and M. S. Hossain, "A probabilistic approach for obtaining an optimized number of services using weighted matrix and multidimensional scaling," in *2017 20th International Conference of Computer and Information Technology (ICCIT)*, Dec 2017, pp. 1–6.

[19] A. A. C. De Alwis, A. Barros, C. Fidge, and A. Polyvyanyy, "Discovering microservices in enterprise systems using a business object containment heuristic," in *On the Move to Meaningful Internet Systems. OTM 2018 Conferences*, H. Panetto, C. Debruyne, H. A. Proper, C. A. Ardagna, D. Roman, and R. Meersman, Eds. Cham: Springer International Publishing, 2018, pp. 60–79.

[20] G. Mazlami, J. Cito, and P. Leitner, "Extraction of microservices from monolithic software architectures," in *2017 IEEE International Conference on Web Services (ICWS)*, June 2017, pp. 524–531.

[21] R. Chen, S. Li, and Z. Li, "From monolith to microservices: A dataflow-driven approach," in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, Dec 2017, pp. 466–475.

[22] D. Taibi. and K. Systä., "From monolithic systems to microservices: A decomposition framework based on process mining," in *Proceedings of the 9th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER,*, INSTICC. SciTePress, 2019, pp. 153–164.

[23] S. Tyszberowicz, R. Heinrich, B. Liu, and Z. Liu, "Identifying microservices using functional decomposition," in *Dependable Software Engineering. Theories, Tools, and Applications*, X. Feng, M. Müller-Olm, and Z. Yang, Eds. Cham: Springer International Publishing, 2018, pp. 50–65.

[24] L. Baresi, M. Garriga, and A. De Renzis, "Microservices identification through interface analysis," in *Service-Oriented and Cloud Computing*, F. De Paoli, S. Schulte, and E. Broch Johnsen, Eds. Cham: Springer International Publishing, 2017, pp. 19–33.

[25] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Migrating to cloud-native architectures using microservices: An experience report," in *Advances in Service-Oriented and Cloud Computing*, A. Celesti and P. Leitner, Eds. Cham: Springer International Publishing, 2016, pp. 201–215.

[26] Z. Ren, W. Wang, G. Wu, C. Gao, W. Chen, J. Wei, and T. Huang, "Migrating web applications from monolithic structure to microservices architecture," in *Proceedings of the Tenth Asia-Pacific Symposium on Internetware*, ser. Internetware '18. New York, NY, USA: ACM, 2018, pp. 7:1–7:10. [Online]. Available: http://doi.acm.org/10.1145/3275219.3275230

[27] M. J. Amiri, "Object-aware identification of microservices," in *2018 IEEE International Conference on Services Computing (SCC)*, July 2018, pp. 253–256.

[28] A. Selmadji, A.-D. Seriai, H. L. Bouziane, C. Dony, and R. O. Mahamane, "Re-architecting oo software into microservices," in *Service-Oriented and Cloud Computing*, K. Kritikos, P. Plebani, and F. de Paoli, Eds. Cham: Springer International Publishing, 2018, pp. 65–73.

[29] M. Ahmadvand and A. Ibrahim, "Requirements reconciliation for scalable and secure microservice (de)composition," in *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, Sep. 2016, pp. 68–73.

[30] M. Gysel, L. Kölbener, W. Giersche, and O. Zimmermann, "Service cutter: A systematic approach to service decomposition," in *Service-Oriented and Cloud Computing*, M. Aiello, E. B. Johnsen, S. Dustdar, and I. Georgievski, Eds. Cham: Springer International Publishing, 2016, pp. 185–200.

[31] A. Levcovitz, R. Terra, and M. T. Valente, "Towards a technique for extracting microservices from monolithic enterprise systems," *CoRR*, vol. abs/1605.03175, 2016. [Online]. Available: http://arxiv.org/abs/1605.03175

[32] D. Escobar, D. Cárdenas, R. Amarillo, E. Castro, K. Garcés, C. Parra, and R. Casallas, "Towards the understanding and evolution of monolithic applications as microservices," in *2016 XLII Latin American Computing Conference (CLEI)*, Oct 2016, pp. 1–11.

[33] M. Abdullah, W. Iqbal, and A. Erradi, "Unsupervised learning approach for web application auto-decomposition into microservices," *Journal of Systems and Software*, vol. 151, pp. 243 – 257, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121219300408

[34] H. Knoche and W. Hasselbring, "Using microservices for legacy software modernization," *IEEE Software*, vol. 35, no. 3, pp. 44–49, May 2018.

[35] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, "Experimentation in software engineering," *Springer Science and Business Media*, 2012.

[36] J. Fritzsch, J. Bogner, A. Zimmermann, and S. Wagner, "From monolith to microservices: A classification of refactoring approaches," in *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, J.-M. Bruel, M. Mazzara, and B. Meyer, Eds. Cham: Springer International Publishing, 2019, pp. 128–141.

[37] J. Kazanavičius and D. Mažeika, "Migrating legacy software to microservices architecture," in *2019 Open Conference of Electrical, Electronic and Information Sciences (eStream)*, April 2019, pp. 1–5.

[38] A. Balalaie, A. Heydarnoori, P. Jamshidi, D. A. Tamburri, and T. Lynn, "Microservices migration patterns," *Software: Practice and Experience*, vol. 48, no. 11, pp. 2019–2042, 2018. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2608