JOURNAL OF ENGINEERING MANAGEMENT AND COMPETITIVENESS (JEMC)
VOL. 11, NO. 2, 2021, 152-162

REVIEW OF METHODS FOR MIGRATING SOFTWARE SYSTEMS TO MICROSERVICES ARCHITECTURE

UDC: 004.416.3
Review Paper

Aleksandra STOJKOV[1], Zeljko STOJANOV[2]

[1]University of Novi Sad, Technical faculty "Mihajlo Pupin" in Zrenjanin, 23000 Zrenjanin, Đure Đakovića bb, Republic of Serbia
[2]University of Novi Sad, Technical faculty "Mihajlo Pupin" in Zrenjanin, 23000 Zrenjanin, Đure Đakovića bb, Republic of Serbia
E-mail: zeljko.stojanov@uns.ac.rs

Paper received: 03.10.2021.; Paper accepted: 21.11.2021.

Majority of software systems in business use, known as legacy systems, have monolithic structure hard to maintain and upgrade with new features. The most common option to overcome this situation is reengineering of existing software systems, which can be perform in different ways and with different outcomes. One of the recent most popular approaches is migration to microservices architectures, which makes distribution of software functionalities in small and independent units possible. Each unit, called microservice is self-contained and independent, which makes system manipulation and modification easier. Several methods for migration to microservice architecture have recently been proposed. This article presents a review of methods for migrating existing systems towards microservices. In addition, this article presents software artifacts affected by migration methods and used algorithms. Implications and benefits of the presented study, as well as validity issues are discussed, followed with concluding remarks and future research directions.

Keywords: Microservices; Microservices architecture; Software architecture; Reengineering; Migration methods.

INTRODUCTION

Due to the rapid change of business conditions and constant improvement of technologies, software systems constantly change to remain useful for their users (Benestad, Anda, & Arisholm, 2010), which indicates that software maintenance is very important for improvement of software quality (Z. Stojanov, J. Stojanov, & Dobrilović, 2019). All changes can harm monolithic and legacy systems because their structure is solid and hard to modify. Through uncontrolled modifications, software systems deviate from the intended architecture, which commonly results with unmanageable monoliths (Sarkar et al., 2009). The problem with maintenance of legacy software systems is that subsequent modifications lead to their increased complexity and decrease of quality, making them hard for further maintenance (Kazanavičius & Mažeika, 2019), resulting in decreased organizational ability to quickly respond to changes in the business environment (Baškarada, Nguyen, & Koronios, 2020). To avoid the stated problems, software systems should evolve or migrate to better architecture patterns, and microservices are one of them.

In last few years, many methods for migration to microservices architecture have been proposed (Ponce, Márquez, & Astudillo, 2019). Migration to microservices has become popular in industry since microservice architecture is highly scalable (Mazzara et al., 2021), and it supports agility (microservices are easy to deploy), reliability (faults in one microservice do not propagate to other microservices), and modifiability (microservices are easy to modify) (Kazanavičius & Mažeika, 2019). However, since microservices are a new architecture style, there are no general guidelines for migrating monolithic systems to microservices (Kazanavičius & Mažeika, 2019), strongly indicating that there is a need for

ISSN 2217-8147 (Online)

Available online at http://www.tfzr.uns.ac.rs/jemc

proposing new migration methods and reviewing existing methods to systematize knowledge in this contemporary technical area.

Based on the above statements, this paper aims to present the results of reviewing scientific papers on migration methods to microservice architectures. The paper presents literature review method based on guidelines proposed by (Kitchenham, 2004), as well as findings of literature review and discussion of implications and limitations of the presented study.

The reminder of the paper is structured as follows. In Background section microservices, monolith systems, reengineering and legacy systems are described. Section Related Work outlines other papers performing similar literature review analyses. In section Research Methods there is a description of the performed research. Section Findings displays review results. The paper ends with Discussion and Conclusion sections.

BACKGROUND

The research presented in this article deals with a certain type of software architecture. Fundamentals of this architecture and the terms used in this article are described in the following four subsections: Microservices, Monolith Systems, Legacy Systems and Reengineering.

Microservices

The base for microservices architecture is service-oriented architecture, and because of that both share the same features. They are flexible and can easily adapt to new challenges, which makes software application easy to maintain and extends its life cycle (Dragičević & Bošnjak, 2019). The difference between these two architectures is the number and size of services, the way of sharing resources, and reuse (Bucchiarone et al., 2020). The number of services included in microservice architecture depends on the complexity of a system and boundaries between services. The size of a microservices depends on functionality they perform (Newman, 2020). In terms of sharing, the main logic in service-oriented architecture is "share-as-much-as-possible", but in microservices architecture it is opposite "share-as-little-as-possible" (Bucchiarone et al., 2020). It means that microservices are closed in view of sharing information related to internal implementation. They use network endpoints to connect with other units (other parts of systems, other microservices).

Microservices are independent units communicating through lightweight messages, which ensures more dynamic maintenance because when the one microservice is changed it does not affect other microservices in the system (Newman, 2021). This microservice characteristic is named high cohesion and loose coupling. Cohesion relates to which level the units use the same parts (Bruce, & Pereira, 2018), while coupling indicates the level a change in one part of the system will affect a change in another (Newman, 2020).

Microservices are collection of separate connected services communicating through the network. Because the services do not share all information and implementation details are not essential for good communication and working of the system, each service can be developed using different technologies (Wolff, 2017). Because of its adaptability and easy maintenance, microservices are a common choice today. They can easily adapt to new trends and enable existing software systems to continue their life cycle moving to microservices architecture through reengineering.

Monolith Systems

Monolith architecture is a traditional way for building software applications. Systems built in monolithic architecture are written and deployed as a single block (Chawla & Kathuria, 2019). In the early development stage, it is easier to work with monolith systems because they are not complex. As the application grows and becomes more complex, the development process slowdowns because all parts of code are interconnected, and it becomes more difficult for maintenance (Kalske, 2017). Changes to any part of application may affect many system components and even disable the entire system (Chawla & Kathuria, 2019). This is the reason for migrating these types of systems to microservices.

Legacy Systems

Legacy systems are old systems built in outdated technologies but still in everyday use. Because of monolith architecture they are difficult to maintain and upgrade. In addition, they cannot be easily replaced because they have supported business processes for years and gradually upgraded their functions. These systems contain a significant

amount of data, and their replacement would take a lot of time and money. The best way to continue their life cycle is to migrate to new technologies. One option is to reengineer and move to a microservices architecture.

Reengineering

Reengineering is a process in which an old system changes its architecture to a new architecture pattern but keeps all functionalities. It should begin when the old system becomes too difficult to be maintained due to outdated and complex architecture that cannot longer be upgraded with new features. The process ensures that software continues to be used with better quality, and with lower costs of maintenance (Singh et al., 2019).

Migrating existing software system to microservices is an option that can ensure further system use. This is not an easy process and there are no precisely defined rules for migration. The main task in this process is identification of functionalities of a system and their migration to microservices (Velepucha & Flores, 2021). The methods and techniques to be used for migration process depend on goals, system artifacts and dependencies between them. The process usually begins with manual inspections of application structure. There are also automatic approaches, with software tool support, which identify candidates for microservice migration. The working principle of these methods is based on grouping similar elements, and proposing microservices (Kirby et al., 2021). The last step is checking the results and migrating the system to new architecture.

RELATED WORK

Microservices are an approach for architecting distributed software systems using independent and fine-grained services, which has gained attention by researchers and practitioners from industry in the last decade (Newman, 2021). This has resulted in a large number of case studies reporting practical experiences, as well as in secondary studies reporting reviews of published case studies. Identification of relevant studies and systematization of empirical knowledge introduced evidence-based software engineering that aims to contribute to improvement of practical experience of software engineers (Dyba, Kitchenham, & Jorgensen, 2005; Zhang, Babar, & Tell, 2011). Literature review in software engineering can be conducted as a Systematic Literature Review (SLR) (Kitchenham, 2004; Kitchenham et al., 2009), a Systematic Mapping Study (SMS) (Petersen et al., 2008; Petersen, Vakkalanka, & Kuzniarz, 2015), or as an informal literature review (Niazi, 2015). A literature review of studies dealing with reviews of microservice architectures is presented in this section.

Based on the literature review of 62 empirical studies, Wolfart et al. (2021) proposed a roadmap for modernization of an existing legacy system with microservices, which contains the following typical activities: analyze the driving forces, understand the legacy system, decompose the legacy system, define the microservice architecture, execute the modernization, integrate the microservices and the legacy, verify and validate the microservices, and monitor the microservices (infrastructure).

Taibi et al. (2018) conducted a systematic mapping study with 85 papers on the use of micro services, aimed at identification of common patterns and principles. The authors systematized mostly used patterns in organization of microservices, their advantages and disadvantages. The architecture patterns are categorized in sense of orchestration and coordination-oriented architecture patterns, deployment patterns, and patterns reflecting data management. Di Francesco et al. (2019) reported a systematic mapping study with 103 primary studies aimed at identification, classification, and evaluation of microservice architectures from the perspectives of publication trends, research focus, and potential for adoption in software industry.

Waseem et al. (2020) presented a systematic mapping study on use of microservices architectures in DevOps. This study included 47 primary studies published in the period from January 2009 to July 2018, based on which the authors identified the following key themes: (1) microservices development and operations in DevOps, (2) approaches and tool support for microservices architectures in Devops, and (3) microservice architecture migration experiences. Migration experience relates to identification of migration motivators, challenges, and patterns in migration process. Bushong et al. (2021) presented a systematic mapping study aimed at reviewing approaches and techniques for analyze of microservice systems, as well as evolution of microservice based systems. The study is based on reviewing 55 primary studies and provides review

of approaches and tools for microservice analysis, review of migration to microservices architectures, review of software architecture reconstruction and quality attributes, and review of microservices evolution.

Ponce et al. (2019) conducted a study that gathers, organizes and analyses 20 migration techniques proposed in the literature on microservices. The study results revealed that majority of techniques were used for migration of object-oriented software systems, while the most important challenges in migration are: migration of database, decomposition of business capabilities in smaller pieces suitable for microservices, expert judgement on the microservices candidates, distribution of work to developers, and resources management.

Aksakalli et al. (2021) presented a systematic literature review on common deployment and communication patterns in microservice architectures based on 38 selected primary studies. The authors identified three types of deployment approaches: serverless deployment, service instance per VM, and service instance per container. The following communication patterns are identified: synchronous communication, publish/subscribe communication, combination of HTTP and message queue, communication using message-oriented middleware, asynchronous communication, point-to-point communication, and communication using binary protocols. Velepucha and Flores (2021) presented a literature review of 37 papers on migration problems and related challenges from monolithic architecture to microservices. The problems relate to team organization, selection of suitable tools, incorporation of new technologies, completeness of migration process, identification and design of microservices, and information consistency when moving from one to multiple databases.

Li et al. (2021) presented a systematic literature review on quality attributes of microservices architectures based on 72 selected primary studies. Analysis of literature revealed the following most important quality attributes: scalability, performance, availability, monitorability, security, and testability. For all quality attributes, the most suitable tactics to address them are identified.

Analysis of published literature reviews on microservice architectures revealed that this is very interesting and promising research area, but also that there are a lot of space to perform additional research. This article intends to present review of literature dealing with methods for migration of existing software systems to microservice architecture.

RESEARCH METHODS

The research follows guidelines for performing systematic literature reviews in software engineering (Kitchenham, 2004; Kitchenham et al., 2009). The research is conducted in some steps that are common for literature reviews, which is presented in Figure 1.

The first step is determination of keywords related to the selected research subject. The words that are selected as the most suitable are classified into two groups due to future combination in search strings. The first group contains words: reengineering, migration, identification. The second group contains only word microservices. Combining the selected keywords, the following search strings are formed:

"reengineering" and "microservices"
"migration" and "microservices"
"identification" and "microservices"

The second step is searching for papers in digital libraries. The best results are found on Google Scholar ScienceDirect, IEEE Xplore and Springer libraries.

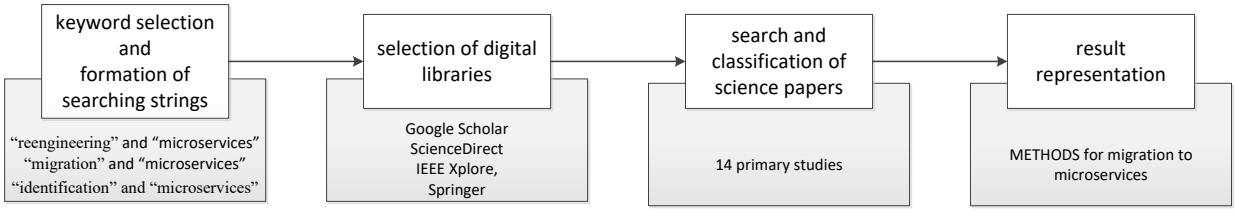

Figure1: Literature review method

Table 1: Primary studies

no.	Reference
ps1	Bucchiarone, A., Dragoni, N., Dustdar, S., Larsen, S., Mazzara, M. (2017). *From Monolithic to Microservices: An experience report*. https://doi.org/10.13140/RG.2.2.34717.00482.
ps2	Baresi, L., Garriga, M., & De Renzis, A. (2017). *Microservices Identification Through Interface Analysis*. In: De Paoli F., Schulte S., Broch Johnsen E. (eds) Service-Oriented and Cloud Computing. ESOCC 2017. Lecture Notes in Computer Science, vol 10465. Springer, Cham. https://doi.org/10.1007/978-3-319-67262-5_2
ps3	Fan, C., & Ma, S. (2017). Migrating Monolithic Mobile Application to Microservice Architecture: An Experiment Report. *2017 IEEE International Conference on AI & Mobile Services (AIMS)*, 2017, 109-112. https://doi.org/10.1109/AIMS.2017.23.
ps4	Safa, H,, Xiaodong, L., & Zhiyuan, T. (2018). An Approach to Evolving Legacy Enterprise System to Microservice-Based Architecture through Feature-Driven Evolution Rules. *International Journal of Computer Theory and Engineering, 10*(5), 164-169. https://doi.org/10.7763/IJCTE.2018.V10.1219
ps5	Eski, S., & Buzluca, F. (2018). An automatic extraction approach: transition to microservices architecture from monolithic application. *XP '18: Proceedings of the 19th International Conference on Agile Software Development: Companion*, May 2018, 25, 1–6. https://doi.org/10.1145/3234152.3234195
ps6	De Alwis A.A.C., Barros A., Fidge C., & Polyvyanyy A. (2018) Discovering Microservices in Enterprise Systems Using a Business Object Containment Heuristic. In: Panetto H., Debruyne C., Proper H., Ardagna C., Roman D., Meersman R. (eds) *On the Move to Meaningful Internet Systems. OTM 2018 Conferences*. OTM 2018. Lecture Notes in Computer Science, vol 11230. Springer, Cham. https://doi.org/10.1007/978-3-030-02671-4_4
ps7	Kamimura, Yano, K., Hatano, T., & A. Matsuo, A. (2018). Extracting Candidates of Microservices from Monolithic Application Code. *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, 2018, 571-580, https://doi.org/10.1109/APSEC.2018.00072.
ps8	Ren, Z., Wang, W., Wu, G., Gao, C., Chen,W., Wei, J., & Huang, T. (2018). Migrating Web Applications from Monolithic to Microservices Architecture. In Proceedings of *Internetware'18*, Beijing, China, September 16, 2018, 10. https://doi.org/10.1145/3275219.3275230
ps9	Zirkelbach, C., Krause, A., & Hasselbring, W. (2018). On the Modernization of ExplorViz towards a Microservice Architecture. *4th Collaborative Workshop on Evolution and Maintenance of Long-Living Software Systems (EMLS)*, 6th February 2018, Ulm, Germany.
ps10	Saidani, I., Ouni, A., Mkaouer, M.W., & Saied, A. (2019). Towards Automated Microservices Extraction Using Muti-objective Evolutionary Search. In: Yangui S., Bouassida Rodriguez I., Drira K., Tari Z. (eds) *Service-Oriented Computing. ICSOC 2019. Lecture Notes in Computer Science*, vol 11895. Springer, Cham. https://doi.org/10.1007/978-3-030-33702-5_5.
ps11	Zhang, Y., Liu, B., Dai, L., Chen, K., Cao, X. (2020). Automated Microservice Identification in Legacy Systems with Functional and Non-Functional Metrics. *2020 IEEE International Conference on Software Architecture (ICSA)*, 2020,135-145. https://doi.org/10.1109/ICSA47634.2020.00021.
ps12	Daoud, M., El Mezouari, A., Faci, F., Benslimane, D., Maamar, Z., & El Fazziki, A. (2020). Towards an Automatic Identification of Microservices from Business Processes. *2020 IEEE 29th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, 2020, 42-47. https://doi.org/10.1109/WETICE49692.2020.00017.
ps13	Gomes, M., Barbosa, H., & Maia P. H. M. (2020). Towards Identifying Microservice Candidates from Business Rules Implemented in Stored Procedures. *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*, 2020, 41-48, https://doi.org/10.1109/ICSA-C50368.2020.00015.
ps14	Al-Debagy, O., & Martinek, Pe. (2021). A Microservice Decomposition Method Through Using Distributed Representation of Source Code. *Scalable Computing: Practice and Experience, 22*(1), 39–52. https://doi.org/10.12694/scpe.v22i1.1836.

Based on the formed search strings, the search for papers was performed in the selected digital libraries. Through analysis of collected papers' titles, abstracts, and the reported findings, 14 primary studies are selected. Details about collected studies are systematized in Excel tables, which contain detailed bibliographical data of each study, and specific data extracted based on the proposed review objective (migration method, used algorithm, affected software artifacts, domain of software use, and software types). In Table 1 primary studies (ps) are listed used for more detailed analysis and construction of the findings.

FINDINGS

Numerous reasons for the migration of existing software systems to microservices are mentioned in the analyzed literature. The main identified reasons are improvement of software elasticity, more controlled transformation and evolution, and better scaling of existing software for new requirements. Migration also helps to reduce size of existing systems and makes it easier to upgrade and maintain them. This is very important since several industrial studies reported increased complexity of maintained software systems, which

leads to unmanageable evolution of existing systems.

To improve stated disadvantages of existing software systems, the authors of the selected primary studies proposed methods for migration to microservices. The methods, artifact used as input for microservices migration methods, and used algorithms are shown in Table 2.

Table 2: Methods for migration to microservices

ps	Method	Artifact	Algorithm
ps1	Direct conversations, interviews and discussions with the FX Core team, and manually inspecting the source code.	source code, databases and services	
ps2	Matching the terms used in the OpenAPI specifications supplied as input against a reference vocabulary to suggest possible decompositions	OpenAPI specification and reference vocabulary	Decomposition Algorithm, Semantic Assessment Algorithm
ps3	First the system architecture is analyzed then with domain driven design candidates are proposed. The last step is comparation of candidates with database and their selection.	system requirements and the database	Domain-Driven Design
ps4	Application of set of transformation rules for substitution of each part of legacy application into services	source code	
ps5	Application of the graph clustering technique on system relationships and couplings between the classes represented as a graph.	static codes and software repositories	Fast Community graph clustering algorithm
ps6	Categorization of identified business objects into different categories and graph creating based on relationships between business objects	business object relationships and their execution patterns	NSGA II
ps7	Dependent programs related to each entry point, which is the interaction point between the system and the user are collected and used for making list of programs (program groups) and data from the source code.	relationship between program groups and data	SArF software clustering algorithm
ps8	Combination of static and dynamic analysis in order to get knowledge about application. From static aliases function call graph are made and dynamic analysis cluster application.	source code	Determination of microservices boundary
ps9	The application is divided into two parts front-end and back-end. Then both of parts become new microservices.	codebase as frontend and backend	
ps10	Each class of system is added to one empty microservice and optimal solutions are required with algorithm	structural dependencies in the source code (set of classes)	NSGA-II
ps11	Finding invocations of object during execution logs and generate object matrix that evolve in class matrix. With algorithm microservices are proposed.	execution and performance logs	AMI algorithm
ps12	From Business process model dependencies between activities are extracted and clustering algorithm are used for identified candidate to microservices	business processes	Collaborative clustering algorithm
ps13	System requirements are found stored procedure analyzed and microservices are proposed.	system requirement and database - business rules implemented in stored procedures	
ps14	Use classes from source code and cluster them based on semantical similarity	source code	Affinity Propagation algorithm

Methods used for migration vary from interview and conversation-based methods, methods that analyze different software artifacts (source code, classes, method invocation during execution, methods call graphs, etc.), clusters them and propose microservice candidates, to methods focused on analysis of business cases.

Detailed analysis of methods revealed that they use different system artifact as main input for decomposition and microservices identification. The most commonly used artifacts are: source code - classes and their interaction, software execution traces and logs, code bases of application frontend and backend, databases – tables, business rules and stored procedures. Some methods use system

requirements, relationship between program groups and data or business processes. Analysis indicate that majority of methods use different source code elements because there are strict language grammars for writing code, as well as tools that can analyze code, which is essential for automating some steps in migration process.

Algorithms identified in primary studies are listed in Table 2. Primary studies marked as ps1, ps4, ps9 and ps13 do not mention the use of any algorithm and because of that there are empty cells in rows associated to these primary studies. Algorithms used in primary studies are:

- *Decomposition Algorithm.* This is an algorithm for microservice proposition using OpenAPI specification and reference vocabulary for input and creating links between the input operation and the description of operation in the best way. Operations with similar concept represent one group and then become candidates for microservice. [ps3]
- *Semantic Assessment Algorithm*. This algorithm is a part of decomposition algorithm analyzing each operation in software system. [ps3]
- *Fast Community graph clustering algorithm.* This algorithm represents a software system as a graph and then clusters it to detect microservices candidates. [ps6]
- *NSGA II*. Nondominated Sorting Genetic Algorithm sorts and compares all solutions to detect microservices. The algorithm is described by Deb et al. (2002) and used in [ps6, ps10].
- *SArF software clustering algorithm*. This algorithm collects all software resources into clusters for microservices, without human interaction. The algorithm is described by Kobayashi et al. (2012) and used in [ps7].
- *Determination of microservices boundary*. This algorithm generates a graph from the classes and their functions, while interaction algorithm clusters graph and forms application behavior characteristic interaction matrix. [ps8]
- *AMI algorithm*. An Automated Microservices Identification algorithm discovers key objects and determines connection between them and classes. Based on identified objects and connections it divides a software system into a set of microservices. [ps11]
- *Collaborative Clustering algorithm.* This algorithm uses matrix for storage dependencies between each couple of activities and clusters them based on shared activities to propose microservices. [ps12]
- *Affinity Propagation algorithm.* This algorithm performs clustering based on measurement of data similarity, which is described by Frey and Dueck (2007) and used in [ps14].

Most of the methods described algorithms they use for microservices extraction. Although they have different name, the basic principle is the same. The algorithms differ in the input parameters, and whether they are automatic, semi-automatic or manual. Based on the analysis of methods, affected software artifacts, and used algorithms in migration to microservices, a general migration process is proposed and presented in Figure 2. The output of the proposed process are microservices candidates.

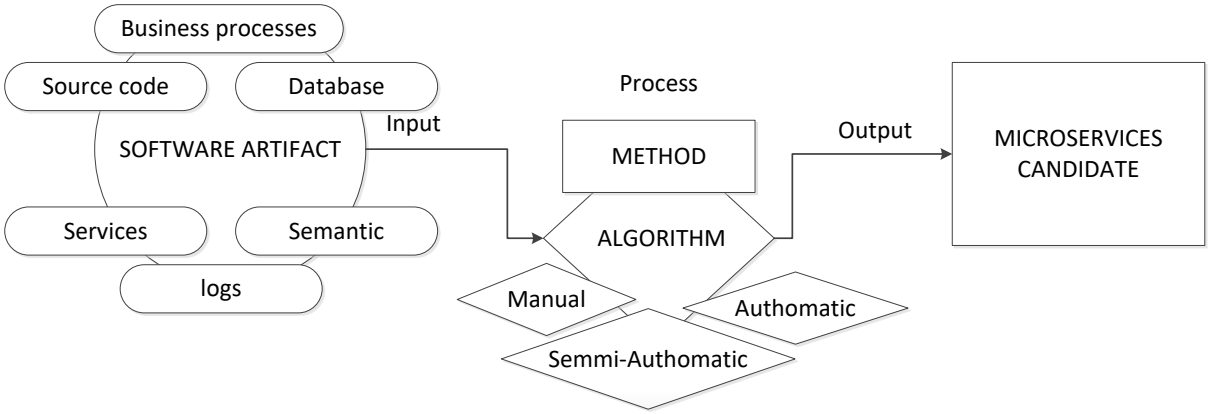

Figure 2: General migration process

Identified migration methods are tested on some specific software applications. Domains of applications use, software types and used migration methods are presented in Table 3.

The most used application is web store named JPetStore [ps8, ps10, ps11, ps14], which is a legacy ERP system with monolithic architecture. Analysis of domain of use revealed that migration was performed in variety of domains, such as banking systems, learning applications, business and booking software, blogs, and forums.

Table 3: Domain of use and software types in migration methods

Domain of use	Software type	ps
Banking system	monolithic system	ps1
Cargo Tracking System		ps2
industrial application		ps7
renting bikes		ps12
Learning System	monolith mobile application	ps3
web store	monolithic web application	ps8, ps10, ps11, ps14
software quality measurement and visualization tool	monolithic enterprise systems	ps5
web crawling project		ps5
business system	legacy enterprise system	ps8
ferry booking system		ps6
web-based monitoring and visualization tool	legacy system	ps9
IT company	legacy web system	ps13
Spring Boot Pet Clinic	open-source application	ps7
Forum	messaging boards application	ps14
Blog	blogging website	ps10, ps14

Variety of software types are identified in primary studies, but common to all of them is that they are monolithic, and therefore, they most often migrate to microservices to increase their performances (availability, scalability, reliability, and maintainability).

DISCUSSION

Results of literature review indicate that several methods have been used for migration of existing software systems to microservices architectures. These methods use variety of software artifacts as a starting point in analysis and implement different algorithms for decomposition of existing systems and identification of microservices candidates. In this section, implications of the presented study, and limitations that threat validity are discussed.

Implications

This paper presents a preliminary literature review designed on the systematic literature review guidelines (Kitchenham, 2004; Kitchenham et al., 2009), but with some simplifications related to the selection of keywords and databases for search of studies. Due to the stated simplifications, the study resulted in a smaller number of selected primary studies. Nevertheless, presented findings indicate that this type of preliminary literature reviews can be valuable evidence for a larger audience from academia and industry.

This study presents details on tailored guidelines for conduct of preliminary literature review, which can be of great benefits for PhD students and young researchers (Pickering & Byrne, 2014). This study can be used as a model for conducting preliminary literature reviews in PhD research, and later for extension of reviews to achieve systematic review of the relevant literature.

Researchers in the field of software architectures and reengineering of software systems can use this study findings as a starting point for inquiry of different methods and algorithms for migration to microservice architectures, as well for decisions on methods suitable for implementation in specific domains and for specific software types.

Finding of this study reported experiences from the selected case studies can help software architects and maintenance experts in selection of optimal migration methods for their domains and software systems, which can help avoiding typical obstacles and problems in reengineering existing software systems.

Limitations and Validity

Although presented findings and discussed benefits indicate that the presented study is useful for both researchers and experts from industry, there are some limitations that treat its validity and should be discussed (Wright, Kim, & Perry, 2010). In addition, literature review studies should discuss some specific validity issues, such as construction of search strings, selection of sources, and role of bias in selection and classification of studies (Ampatzoglou et al., 2019).

The first limitation relates to selection of keywords and composition of search strings used for finding the relevant studies. The search strings are listed in

the main section of this article, but identification of potential synonyms for the selected keywords can provide improved search and potential identification of a larger number of relevant studies. For example, keyword "identification" can be used together with synonyms "discovery" and "finding" using logical OR operator in search strings, which can lead to discovery of more studies related to migration methods. The second limitation relates to use of more digital libraries for searching. Publisher such as Wiley or ACM list a large number of articles, which should be included in further literature reviews. Additional problem relates to access to articles, since some publishers do not allow access without subscription. These limitations will be considered in further, more detailed, and systematic literature reviews.

CONCLUSIONS

Microservices are contemporary architectural patterns for structuring software systems, with increased scalability, availability, reliability, and maintainability compared to older patterns. These characteristics of microservices attract many organizations to migrate their old systems. Although several methods have been proposed for migration of old systems to microservice architecture in the last ten years, there is no general guideline. This article provides systematization of methods for migration of existing software systems to microservices based on a literature review. The findings of this paper provide detailed insight into methods used for migration of software systems to microservice architectures, accompanied with detailed overview of software artifacts used in the methods, domains of use and software types transformed during migration process. The findings of this study are valuable for experts from software industry during reengineering of old systems since they can find information about migration methods connected to affected software artifacts, domain of use and software types. Researcher from academia can use this review study as a starting point for their studies or can use and adapt the review method presented in the study.

Future work will be pursued in two directions. The first one is conduct of a systematic literature review in which stated limitation of this study will be addressed. The second one is development of a method and a tool for migration of web applications in complex technical systems to microservices architecture.

ACKNOWLEDGEMENT

The Ministry of Education, Science and Technological Development of the Republic of Serbia supports this research under the project "The Development of Software Tools for Business Process Analysis and Improvement", project number TR32044.

REFERENCES

Aksakalli, I.K., Çelik, T., Can, A.B., & Tekinerdoğan, B. (2021). Deployment and communication patterns in microservice architectures: A systematic literature review. *Journal of Systems and Software*, *180*, 111014. https://doi.org/10.1016/j.jss.2021.111014.

Ampatzoglou, A., Bibi, S., Avgeriou, P., Verbeek, M., & Chatzigeorgiou, A. (2019). Identifying, categorizing and mitigating threats to validity in software engineering secondary studies. *Information and Software Technology*, *106*, 201-230. https://doi.org/10.1016/j.infsof.2018.10.006.

Baškarada, S., Nguyen, V., & Koronios, A. (2020). Architecting Microservices: Practical Opportunities and Challenges. *Journal of Computer Information Systems*, *60*(5), 428-436). https://doi.org/10.1080/08874417.2018.1520056.

Benestad, H.C., Anda, B., & Arisholm, E. (2010). Understanding cost drivers of software evolution: a quantitative and qualitative investigation of change effort in two evolving software systems. *Empirical Software Engineering*, *15*(2), 166 – 203. https://doi.org/10.1007/s10664-009-9118-8.

Bruce, M., & Pereira, A. P. (2018) *Microservices in Action.* Manning. ISBN: 1617294454, 9781617294457

Bucchiarone, A., Dragoni N., Dustdar, S., Lago, P., Mazzara, M., Rivera, V., & Sadovykh, A. (2020) *Microservices Science and Engineering.* Springer. ISBN: 978-3-030-31648-8

Bushong, V., Abdelfattah, A.S., Maruf, A.A., Das, D., Lehman, A., Jaroszewski, E., Coffey, M., Cerny, T., Frajtak, K., Tisnovsky, P., & Bures, M. (2021). On Microservice Analysis and Architecture Evolution: A Systematic Mapping Study. *Applied Sciences*, *11*(17), 7856. https://doi.org/10.3390/app11177856.

Chawla, H., & Kathuria, H. (2019). *Evolution of Microservices Architecture. In: Building Microservices Applications on Microsoft Azure*. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-4828-7_1

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *Transactions on Evolutionary Computation, 6*(2*)*, 182–197. https://doi.org/10.1109/4235.996017

Di Francesco, P., Lago, P., & Malavolta, I. (2019). Architecting with microservices: A systematic

mapping study. *Journal of Systems and Software*, *150*, 77–97. https://doi.org/10.1016/j.jss.2019.01.001.

Dragićević, Z., & Bošnjak, S. (2019). Harmonizing business and digital enterprise strategy using soa middle-out and service-based approach. *Journal of Engineering Management and Competitiveness*, *9*(2), 97-112.

Dyba, T., Kitchenham, B., & Jorgensen, M. (2005). Evidence-based software engineering for practitioners, *IEEE Software*, *22* (1),158–165. https://doi.org/10.1109/MS.2005.6.

Frey, B.J., & Dueck, D. (2007). Clustering by passing messages between data points. *Science 315*, 5814, 972–976. https://doi.org/10.1126/science.1136800

Kalske, M. (2017). *Transforming monolithic architecture towards microservice architecture*. M.Sc. Thesis University of Helsinki, Department of Computer Science

Kazanavičius, J., & Mažeika, D. (2019). Migrating Legacy Software to Microservices Architecture. *In Proceedings of 2019 Open Conference of Electrical, Electronic and Information Sciences (eStream)* (pp. 1-5). https://doi.org/10.1109/eStream.2019.8732170.

Kirby, L., Boerstra, E., Anderson, Z., & Rubin J.,= (2021). Weighing the Evidence: On Relationship Types in Microservice Extraction. *Proceedings of the IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*, pp. 358-368. https://doi.org/10.1109/ICPC52881.2021.00041

Kitchenham, B.A. (2004). *Procedures for Undertaking Systematic Reviews, Joint Technical Report*. Computer Science Department, Keele University (TR/SE-0401) and National ICT Australia Ltd. (0400011T.1). Keele, UK.

Kitchenham, B., Brereton, O.P., Budgen, D., Turner, M., Bailey, J., & Linkman, S. (2009). Systematic literature reviews in software engineering - A systematic literature review. *Information and Software Technology*, *51*(1), 7-15. https://doi.org/10.1016/j.infsof.2008.09.009.

Kobayashi, K., Kamimura, M., Kato, K., Yano, K., & Matsuo, A. (2012). Feature-gathering dependency-based software clustering using dedication and modularity. *Proceedings of the 28th IEEE International Conference on Software Maintenance (ICSM)*, 462-471.

Li, S., Zhang, H., Jia, Z., Zhong, C., Zhang, C., Shan, Z., Shen, J., & Babar, M.A. (2021). Understanding and addressing quality attributes of microservices architecture: A Systematic literature review. *Information and Software Technology*, *131*, 106449. https://doi.org/10.1016/j.infsof.2020.106449.

Mazzara, M., Dragoni, N., Bucchiarone, A., Giaretta, A., Larsen, S.T., & Dustdar, S. (2021). Microservices: Migration of a Mission Critical System. *IEEE Transactions on Services Computing*, *14*(5), 1464-1477. https://doi.org/10.1109/TSC.2018.2889087.

Newman, S. (2021). Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol, CA, USA: O'Reilly Media. ISBN: 978-1-492-03402-5

Newman, S. (2020) Monolith To Microservices Evolutionary Patterns to Transform Your Monolith, O'Reilly, ISBN: 978-1-492-07554-7

Niazi, M. (2015). Do Systematic Literature Reviews Outperform Informal Literature Reviews in the Software Engineering Domain? An Initial Case Study. *Arabian Journal for Science & Engineering*, *40*(3), 845-855. https://doi.org/10.1007/s13369-015-1586-0.

Petersen, K., Feldt, R., Mujtaba, S., & Mattsson, M. (2008). Systematic mapping studies in software engineering. *In Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE'08)*, 68-77, Swinton, UK.

Petersen, K., Vakkalanka, S., Kuzniarz, L. (2015). Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, *64*, 1-18. https://doi.org/10.1016/j.infsof.2015.03.007.

Pickering, C., & Byrne, J. (2014). The benefits of publishing systematic quantitative literature reviews for PhD candidates and other early-career researchers. *Higher Education Research & Development*, *33*(3), 534-548. https://doi.org/10.1080/07294360.2013.841651.

Ponce, F., Márquez, G., & Astudillo, H. (2019). Migrating from monolithic architecture to microservices: A Rapid Review. *In Proceedings of the 38th International Conference of the Chilean Computer Science Society (SCCC)*, 1-7. Concepcion, Chile. https://doi.org/10.1109/SCCC49216.2019.8966423.

Sarkar, S., Ramachandran, S., Kumar, G.S., Iyengar, M.K., Rangarajan, K., & Sivagnanam, S. (2009). Modularization of a Large-Scale Business Application: A Case Study. *IEEE Software*, *26*(2), 28-35. https://doi.org/10.1109/MS.2009.42.

Singh, J., Singh, Dhindsa K., & Singh, J. (2019). Reengineering Framework to Enhance the Performance of Existing Software. *International Journal of Advanced Computer Science and Applications*, *10*(5). https://doi.org/10.14569/IJACSA.2019.0100570.

Stojanov, Z., & Stojanov, J., & Dobrilović, D. (2019). A lightweight inductive method for process assessment based on frequent feedback: a study in a micro software company. *Journal of Engineering Management and Competitiveness*, *9*(2), 134-147.

Taibi, D., Lenarduzzi, & V., Pahl, C. (2018) Architectural Patterns for Microservices: A Systematic Mapping Study. *In Proceedings of the 8th International Conference on Cloud Computing and Services Science*, 221–232. Madeira, Portugal. https://doi.org/10.5220/0006798302210232.

Velepucha, V., & Flores, P. (2021). Monoliths to microservices - Migration Problems and Challenges:

A SMS. *In Proceedings of 2021 Second International Conference on Information Systems and Software Technologies (ICI2ST)* (pp. 135-142). Quito, Ecuador. 2021. https://doi.org/10.1109/ICI2ST51859.2021.00027.

Waseem, M., Liang, P., & Shahin, M. (2020). A Systematic Mapping Study on Microservices Architecture in DevOps. *Journal of Systems and Software*, *170*, 110798. https://doi.org/10.1016/j.jss.2020.110798.

Wolfart, D., Assunção, W.K.G., da Silva, I.F., Domingos, D.C.P., Schmeing, E., Donin Villaca, G.L., & do N. Paza, D. (2021). Modernizing Legacy Systems with Microservices: A Roadmap. *In Proceedings of Evaluation and Assessment in Software Engineering (EASE 2021)*,149-159, Trondheim, Norway. https://doi.org/10.1145/3463274.3463334.

Wolff, E. (2017). *Microservices Flexible Software Architecture*. Pearson Education, Inc. ISBN-13: 978-0-134-60241-7 ISBN-10: 0-134-60241-2.

Wright, H.K., Kim, M., & Perry, D.E. (2010). Validity Concerns in Software Engineering Research. *In proceedings of the FSE/SDP Workshop on Future of Software Engineering Research - FoSER '10*, 411-414. Santa Fe, New Mexico, USA. 2010. https://doi.org/10.1145/1882362.1882446.

Zhang, H., Babar, M.A., & Tell, P. (2011). Identifying relevant studies in software engineering. *Information and Software Technology*, *53*(6), 625-637. https://doi.org/10.1016/j.infsof.2010.12.010.

PREGLED METODA ZA MIGRACIJU SOFTVERSKIH SISTEMA NA MIKROSERVISNU ARHITEKTURU

Većina softverskih sistema u poslovnoj upotrebi, nazvani nasleđeni sistemi, imaju monolitnu strukturu koja je nepogodna za održavanje i nadogradnju novim funkcionalnostima. Da bi se prevazišla ova situacija najčešće se koristi reinženjering postojećih softverskih sistema, koji se može sprovesti na više različitih načina i sa različitim ishodima. Jedan od najpopularnijih pristupa u poslednje vreme je migracija na mikroservisnu arhitekturu, koja omogućuje distribuciju softverskih funkcionalnosti u male i nezavisne jedinice. Svaka jedinica, nazvana mikroservis, je samostalna i nezavisna, što omogućuje lakše rukovanje i održavanje sistema. Značajan broj metoda za migraciju na mikroservisnu arhitekturu predložen je u literaturi. Ovaj članak predstavlja pregled metoda za migraciju postojećih sistema na mikroservisnu arhitekturu. Takođe, članak prikazuje softverske artefakte obuhvaćene ovim metodama, kao i korišćene algoritme. Diskutuje se o implikacijama i koristima od prezentovane studije, kao i pitanjima validnosti. Na kraju članka su prikazana zaključna razmatranja i pravci budućih istraživanja.

Ključne reči: Mikroservisi; Mikroservisna arhitektura; Softverske arhitekture; Reinženjering; Metode za migraciju.