Analyzing the Impact of Service Design on Maintainability Factor in Microservices Architecture

Gintoro, Ford Lumban Gaol, Haryono Soeparno and Yulyani Arifin

January 29, 2024

Analyzing the Impact of Service Design on Maintainability Factor in Microservices Architecture

Gintoro
Computer Science Department,
BINUS Graduate Program –
Doctor of Computer Science,
Bina Nusantara University
Jakarta, Indonesia 11480
gintoro@binus.ac.id

Ford Lumban Gaol
Computer Science Department,
BINUS Graduate Program –
Doctor of Computer Science,
Bina Nusantara University
Jakarta, Indonesia 11480
fgaol@binus.edu

Haryono Soeparno
Computer Science Department,
BINUS Graduate Program –
Doctor of Computer Science,
Bina Nusantara University
Jakarta, Indonesia 11480
haryono@binus.edu

Yulyani Arifin
Computer Science Department,
BINUS Graduate Program –
Doctor of Computer Science,
Bina Nusantara University
Jakarta, Indonesia 11480
yarifin@binus.edu

Abstract— **Applications designed utilizing Microservices Architecture (MSA) provide the desirable trait of good maintainability. To ensure optimal maintainability, it is important to provide services that are suitable and adhere to prescribed rules. Multiple aspects must be taken into account while designing services to ensure optimal maintainability. The objective of this study is to examine the elements that impact the capacity to sustain and improve maintainability in service design, ultimately resulting in an application that possesses strong maintainability. The Systematic Literature Review (SLR) will be utilized to identify variables and strategies for their enhancement, by examining pertinent publications on the subject. After examining 45 publications, the study discovered 8 elements and 14 solutions that can enhance the highlighted parameters throughout the services design process. The outcomes of this systematic literature review (SLR) are anticipated to give valuable insights to application developers, empowering them to generate service designs that exhibit commendable maintainability for the developed applications.**

Keywords—microservices, maintainability, service design, change request

I. Introduction

Application architecture is an essential consideration in the current environment of application development, requiring careful planning from the beginning. Utilizing a suitable application architecture is a critical determinant of success when creating apps that can effectively fulfill user requirements [1]. Microservices Architecture (MSA) is a highly popular application development architecture that is widely used in the field of application development [2]. Application developers typically accept this architecture both during the original construction and while migrating monolithic-based apps to MSA. Developers do this task to use the potential of MSA in the construction or upkeep of applications for future purposes [2].

One compelling argument for application developers to embrace or transition to MSA is its inherent potential for maintainability [2]. The importance of maintainability is paramount given the fast progression of the corporate landscape, which necessitates prompt modifications in existing or ongoing application development [3]. The rapid adoption of modification requests may be utilized as a strategic advantage in the business sector to provide a competitive edge for the company [4].

In order to achieve this capacity to be easily maintained, it is imperative to have a well-defined and accurate service architecture inside the Microservices Architecture (MSA). Flawed design, also known as anti-patterns, can result in problems or difficulties that develop in the program later on, thereby reducing the maintainability of an application designed using MSA [2], [5].

The objective of this work is to examine the elements that might impact the maintainability of a service design in systems based on Microservices Architecture (MSA) and propose strategies to improve these characteristics during the service design phase. The deliberation will be carried out employing the Systematic Literature Review (SLR) approach on articles pertaining to the subject matter, released during the timeframe of 2018 to 2023.

Previous studies have extensively researched the maintainability factor of Microservices Architecture (MSA), with a focus on its characteristics and measurement. However, this study specifically examines the quality of the maintainability factor in service design.

This study will be organized into four sections to thoroughly examine the elements that impact application maintainability. Section two will provide an in-depth analysis of the SLR technique utilized and its resulting effects. Section three will analyze and report the results of the systematic literature review (SLR), while section four will provide a concise summary of these findings.

II. Research Methodology

This study utilizes the Systematic Literature Review (SLR) approach to discover the parameters that affect the maintainability characteristics of an application created using MSA and to determine how to improve these factors. The goal is to enable the efficient management of change requests.

According to Kitchenham's works, the Systematic Literature Review (SLR) research technique consists of a structured series of steps for gathering, identifying, and thoroughly examining existing research papers [6]. The subsequent section will elucidate the steps entailed in performing a Systematic Literature Review (SLR), referencing Kitchenham's elucidations.

A. *Research Questions*

In order to initiate the Systematic Literature Review (SLR) process, two research questions will be formulated. These questions will specifically target the aims of this study and will serve as the fundamental basis for the next stages of the SLR process. The following are the two research inquiries:

RQ1: What are the variables that influence the maintainability of MSA service design?

The initial inquiry will concentrate on determining the elements that might impact the service design in an MSA-based application in order to improve its maintainability.

RQ2: How may the maintainability of service design be enhanced by improving relevant factors?

Once these characteristics have been recognized, the subsequent inquiry will center on endeavors to improve them, aiming to generate service designs with commendable maintainability.

B. *Search Strategy and Process*

The process of searching for papers will commence by defining keywords that can address the two research questions previously defined above. The identification of keywords will be carried out for both of the aforementioned research questions, along with the identification of words that are synonyms of the identified keywords. Next, from these keywords, a search string will be constructed to search for a list of papers from the specified paper repositories. The following is the search string that will be used for the search: *TITLE-ABSTR-KEY(("Microservices" OR "Microservice" OR "Microservices Architecture") AND ("Change Request" OR "Change Impact" OR "Software Update" OR "Maintainability" OR "Maintainability Metrics" OR "Software Design Quality Metrics"))*.

The search is focused on four major paper repositories commonly used in Software Engineering research, with the search criteria limited to papers published between 2018 and 2023. Only papers from conferences and journals are included, and they must be in English. Based on the search criteria, a search string was used to locate papers. The results included 440 papers, out of these 202 were selected as potential candidate papers after reviewing their titles and keywords, Table I below displays these search results according to search string usage and selected criteria:

TABLE I. DIGITAL LIBRARY SEARCH RESULT

No.	*Digital Library*	*Found*	*Candidate*	*Selected*
1.	ACM Digital Library *(http://portal.acm.org)*	20	9	3
2.	IEEE Digital Library (*http://ieeexplore.ieee.org*)	169	100	32
3.	Science@Direct (*http://www.sciencedirect.com*)	29	29	4
4.	Springer Link (*http://link.springer.com*)	222	64	6
Total Papers Selected		440	202	45

C. *Study Selection*

The selection of papers for further research from the list of papers found in the digital library will be based on selection criteria as shown in Table II below:

TABLE II. SELECTION CRITERIA

Inclusion Criteria	*Exclusion Criteria*
I1. Studies related to or mentioning factors that impact the maintainability of services design. I2. Studies related to how to enhance those factors to produce service designs with good maintainability.	E1. Studies are not written in English. E2. Studies are an editorial, book, article, opinion, or technical report. E3. Studies only focus on microservices and do not consider their impact on maintainability.

From the selection results using the criteria in Table II, out of the 202 papers identified in the abstracts above, 45 papers were further selected for in-depth research to find answers to both research questions. Table I presents the further selection results from each digital library.

D. *Data Extraction*

In order to maintain consistency in the data obtained from the chosen publications, the research procedure for the 45 studies will utilize certain criteria for data collection, as outlined in Table III. The data gathering criteria are associated with solving the pre-established study topics.

TABLE III. DIGITAL EXTRACTION FORMS

Data Field	*Notes*
F1. Factors that can influence the service design in an MSA-based application to enhance its maintainability	Related with RQ1
F2. How the improve the factor	Related with RQ2

E. *Data Synthesis*

The synthesis of outcomes will rely on the results produced from the prior data extraction phase. In order to determine the elements in service design that impact the maintainability of applications, each of these elements will be correlated with the papers that specifically address them, one by one. The conversation is classified as either direct discussion or solely derived from the article. A factor will be awarded a score of 1 for explicit discussion, whilst a value of 0.5 will be assigned for discussion that is indirectly mentioned in the work. By summing up the values for each element, we can identify the 8 factors with the greatest values. These factors are regularly covered throughout the publications. The following elements will be examined thoroughly, and subsequently, solutions will be sought for the second research question on how to improve these characteristics in the service design process based on the chosen articles.

III. Result and Discussion

A. *Variables That Influence the Maintainability of Microservices Architecture (MSA) Service Design*

Upon completing the examination of the documents and conducting the process of data synthesis, the outcomes have been acquired and displayed in Table IV. Among the 45 publications that underwent examination, 36 of them addressed the elements that influence maintainability during service design, whether directly or implicitly. The remaining 9 studies did not address these factors, however, they did offer insights on enhancing them. Table IV presents the distribution of these parameters based on the data collected from the 36 studies. The symbol "●" denotes that the paper expressly addresses the factors, whereas the symbol "○" implies that the document does not clearly mention them. The use of the "-" symbol signifies the omission of these issues in the paper's discussion. Research from the study suggests that the frequency of discussion of a certain issue in the examined studies reflects its level of effect.

According to the weighting computations, the examined publications identified coupling as the most influential component, with a weight of 17.5. The ranking of the factors, in descending order, is as follows: modifiability (16), modularity (13), complexity (11), testability (8), cohesion (6.5), size (5.5), and reusability (2.5).

The following are factors that affect maintainability when designing services:

1. Coupling refers to the degree of connection between modules, indicating the level of interdependence among them. In the context of MSA, coupling refers to the interconnections between different services. Increased coupling between microservices can lead to more complexity, resulting in reduced testability and modularity [7], [8].
2. Modifiability is a sub-characteristic of maintainability, according to the ISO 25010 software quality standard. It pertains to the ability to modify goods or systems without introducing flaws or reducing the quality of the current product. Modifiability refers to the degree of simplicity with which a system may be adjusted or altered to accommodate new requirements, difficulties, or to address them [9].
3. Modularity is a design paradigm that entails dividing systems into smaller, autonomous components, known as modules, each of which carries out a distinct purpose. Subsequently, each of these modules may be individually designed, tested, and updated, resulting in enhanced maintainability, modifiability, and reusability of the entire system [4], [9].
4. Complexity pertains to the extent of interconnectedness and interdependencies across different microservices inside a system. This encompasses factors such as the quantity of services, their interconnections, and the efficient management and coordination of all of them. The heightened intricacy of a system can render it more challenging to comprehend, alter, and sustain, while also amplifying the likelihood of errors and system downtime hazards [10].
5. Testability refers to the capacity of a system to reveal its flaws through testing that involves executing it [2]. The presence of high testability allows for the effective detection and resolution of problems within its framework, leading to enhanced dependability and ease of maintenance [5], [11].
6. In the context of MSA, cohesion refers to the degree to which the components inside a microservice are interconnected and how tightly their tasks are related to each other. Attaining strong cohesiveness inside a microservice in a microservices architecture (MSA) is advantageous as it results in enhanced modularity and maintainability [8].
7. The size of a service can have a significant impact on its maintainability. Smaller services are often more comprehensible, modifiable, and testable for administrators and developers, hence improving their maintainability [2], [12].
8. Reusability pertains to the capacity of a microservice to be employed in different contexts or applications. This implies that the design and implementation of the system should be done in such a way that it may be used again in various sections of an application, and possibly in separate applications as well [9].

As discussed earlier, coupling is an essential aspect in service design to ensure the maintainability of an MSA-based application. However, it is not the only factor that needs to be considered in the service design process. Seven other factors are also crucial, as they are interrelated and contribute to developing an application that can quickly accommodate change requests.

Therefore, it is important to factor in all these aspects during the service design process.

B. *How May the Maintainability of Service Design be Enhanced by Improving Relevant Factors?*

Following discussion on the factors of service design that can impact maintainability, the discussion will move on to the second research question, which is how to enhance these factors in order to produce service designs with good maintainability. According to the review conducted on papers related to the second research question, here are strategies that can be done to improve those 8 factors:

1. Design for Single Responsibility, each microservice should be created to do one task well and this reduces dependency between services, making them simpler to maintain and less likely to be affected by changes elsewher

TABLE IV. VARIABLES THAT INFLUENCE THE MAINTAINABILITY OF MICROSERVICES ARCHITECTURE (MSA) SERVICE DESIGN

Studies	CL	MY	MD	CX	TT	CH	SZ	RS
[8]	●	●	●	-	-	-	●	-
[14]	●	-	●	●	-	●	-	○
[9]	●	●	●	-	-	-	-	-
[16]	-	-	-	●	-	-	●	-
[18]	○	●	○	●	-	-	-	-
[20]	●	●	-	-	-	●	-	-
[22]	-	●	-	-	-	-	-	-
[24]	-	-	-	-	●	-	-	-
[26]	●	-	●	-	-	-	-	-
[28]	-	-	●	○	-	-	-	-
[30]	●	●	-	-	●	●	-	-
[31]	-	●	-	-	-	●	-	-
[32]	●	-	-	-	○	-	-	-
[34]	●	-	●	●	●	-	-	-
[36]	-	-	○	-	-	-	-	-
[38]	-	●	-	-	-	-	●	-
[40]	○	-	-	-	-	-	-	-
[42]	●	●	-	-	-	-	-	-
[13]	-	-	-	-	●	-	-	-
[3]	●	●	-	●	-	●	●	-
[15]	●	-	-	●	-	○	-	-
[17]	-	●	-	-	-	-	-	-
[19]	-	-	○	-	-	-	-	-
[21]	-	-	●	●	●	-	○	-
[23]	●	●	-	-	-	○	●	●
[25]	-	-	●	-	-	-	-	-
[27]	●	●	-	-	○	-	-	-
[29]	-	-	●	●	-	-	-	-
[4]	●	-	●	-	-	-	-	-
[5]	-	-	-	●	●	-	-	-
[33]	-	●	-	-	-	-	-	-
[35]	○	-	-	-	-	-	-	-
[37]	-	●	●	○	-	-	-	-
[39]	-	-	-	-	●	-	-	-
[41]	●	-	-	-	-	○	-	●
[1]	●	●	●	●	-	-	-	-
Total Weight	17.5	16	13	11	8	6.5	5.5	2.5

CL: Coupling; MY: Modifiability; MD: Modularity; CX: Complexity; TT: Testability; CH: Cohesion; SZ: Size; RS: Reusability

●: Mentioned by the studies (weight 1) ○: Not explicitly mentioned by the studies (weight 0.5) -: Not mentioned by the studies (weight 0)

2. Decentralize Data Management, microservices should utilize their own private databases in order to ensure data consistency and reduce synchronization needs between services, making maintenance simpler. By eliminating dependencies between services, coupling can be reduced drastically making them simpler to maintain [24].
3. Use Published Interfaces, microservices should communicate with one another using clearly delineated, published interfaces that make the relationship between them simpler by only needing know about their interface rather than its inner workings [4].
4. Avoid Over-Coupling, when microservices depend on each other for their functionality, they become over-coupled. This makes maintenance difficult as changes to one service may affect others; to prevent this issue from arising, design your services so they are as independent of one another as possible [4].
5. Reduce Business Coupling Between Services, when the business logic of two services are tightly interdependent, any problems in either one could cause difficulties in others. To reduce this risk, design services to be as independent from one another in terms of their business logic as possible [26].
6. Consider the Size of Microservices, a microservice's size can greatly influence its ability to interact with other services. Larger microservices often have more interactions between themselves and other services, which makes maintenance harder. Consider breaking large microservices down into more manageable chunks [4].
7. Use Connectivity Patterns, the way in which service members connect can have a major effect on its maintainability, so use connectivity patterns that reduce coupling to make service maintenance simpler [4], [44].
8. Separate Persistency, when each microservice has its own database, modularity and modifiability increase [9].
9. Appropriate Service Relationships, such as using separate databases for each microservice, increase both modularity and modifiability [9].
10. Right Cuts, having appropriate cuts promotes modularity as functionalities are properly divided and not spread across various microservices, meaning changes will only impact one or two services when implemented, making maintenance and scaling of systems easier [9].
11. Separate Libraries, separating libraries promotes modularity because microservices become more autonomous. Modifiability is increased significantly while fault tolerance increases as misbehaving libraries won't impact other microservices [9].

TABLE V. MAPPING VARIABLES AND IMPROVEMENT STRATEGY

		Factors that Impact Maintainability in Service Design							
No.	**Strategy to Improve Those Factor**	**CL**	**MY**	**MD**	**CX**	**TT**	**CH**	**SZ**	**RS**
1	**Design for Single Responsibility**	V	V	V	V	V	V	V	-
2	**Decentralize Data Management**	V	V	V	V	V	V	V	-
3	**Use Published Interfaces**	V	V	-	V	V	-	-	-
4	**Avoid Over-Coupling**	V	-	-	V	-	-	V	-
5	**Reduce Business Coupling Between Services**	V	V	-	V	-	-	V	-
6	**Consider the Size of Microservices**	V	V	-	V	-	-	-	-
7	**Use Connectivity Patterns**	V	-	-	V	-	-	-	-
8	**Separate Persistency**	-	V	V	-	-	-	-	-
9	**Appropriate Service Relationship**	-	V	V	-	-	-	-	-
10	**Right Cuts**	-	-	V	-	-	-	-	V
11	**Separate Libraries**	-	-	V	-	-	-	-	V
12	**Non-ESB Microservices**	-	V	V	-	-	-	-	V
13	**Domain-Driven Design**	-	-	-	-	-	V	V	-
14	**Bounded Context**	-	-	-	-	-	V	V	-

CL: Coupling; MY: Modifiability; MD: Modularity; CX: Complexity; TT: Testability; CH: Cohesion; SZ: Size; RS: Reusability

V: Strategy to improve the factor   - : Improvement strategy not related with the factor

12. Non-ESB Microservices, not having ESB microservices makes the system less complex and enhances modularity. An ESB microservice can be a single point of failure, and hence fault tolerance is higher when having less number of ESB microservices [9], [45].
13. Domain-Driven Design, or DDD, involves developing microservices around business domains for optimal cohesion. All functionality within each microservice will relate directly to its purposeful business domain [2], [46].
14. Bounded Context, is a central pattern in Domain-Driven Design that defines the scope within which models apply and provides consistency of meaning and language use across models within that context. This approach increases cohesion as all functionality within microservices relates back to one specific environment [2].

The Table V shows how each strategy can be utilized to enhance the identified factors. By analyzing the mapping of these strategies and factors, the second research question can be answered. This involves increasing the factors using the strategies that have been obtained from the review paper. To produce a service design with good maintainability, it is essential to consider two significant strategies - design for single responsibility and decentralized data management. These strategies are highly effective in improving seven out of eight influential factors. Thus, it is highly recommended to employ these strategies while designing services. The contribution of this study is to give new insight about factors and strategies to improve maintainability in designing the services in Microservices Architecture (MSA). With this insight, service designers can design applications with a good level of maintainability, while for academics this insight will be able to provide a basis for the research of good service design by paying attention to the factors that affect the maintainability of an application.

IV. CONCLUSION

With the rapid development of today's business world, the demand for change requests in software is one of the most frequent in software development activities. Microservices Architecture (MSA) adoption is one of the answers to help application developers respond quickly to change requests. Proper and good service design factors in applications built with Microservices Architecture (MSA), will help improve the maintainability of the application. The paper conducts systematic literature review (SLR) study to identify factors that can influence the service design in a Microservices Architecture (MSA)-based application to enhance its maintainability and the strategy to improve those factors.

Based on review in the 45 papers from the systematic literature review (SLR) process, conducted with Kitchenham stages, 8 factors that can influence the service design were obtained. The service designer in Microservices Architecture (MSA) needs to focus on studying and improving eight important factors, which are coupling, modifiability, modularity, testability, cohesion, size, and reusability. In this study, strategies to enhance these factors during the service design process have been discussed. One such strategy is the implementation of the Single Responsibility Principle, which aims to ensure that each service has only one function. This helps keep the size of the services small and less complex. However, one limitation of this paper is that it does not involve practical service designers or companies that have already used service design in Microservices Architecture (MSA). Therefore, the proposed strategies and factors may still need improvement to make them more relevant.

Due respect to another paper, this research seeks new novelty in finding factors and strategies to enhance those factors, that influence the maintainability of Microservices Architecture (MSA) service design. For the future study from this paper, besides involving Microservices Architecture (MSA) service design practitioner, some measurement techniques for each factor can be deployed to measure each factor, before the services design is implemented in the software development stage.

REFERENCES

[1] R. Mishra, N. Jaiswal, R. Prakash, and P. N. Barwal, "Transition from Monolithic to Microservices Architecture: Need and proposed pipeline," in *2022 International Conference on Futuristic Technologies (INCOFT)*, Belgaum, India: IEEE, Nov. 2022, pp. 1–6. doi: 10.1109/INCOFT55651.2022.10094556.

[2] S. Li *et al.*, "Understanding and addressing quality attributes of microservices architecture: A Systematic literature review," *Information and Software Technology*, vol. 131, p. 106449, Mar. 2021, doi: 10.1016/j.infsof.2020.106449.

[3] M. H. Hasan, Mohd. H. Osman, N. I. Admodisastro, and M. S. Muhammad, "From Monolith to Microservice: Measuring Architecture Maintainability," *IJACSA*, vol. 14, no. 5, 2023, doi: 10.14569/IJACSA.2023.0140591.

[4] C. Zhong, H. Zhang, C. Li, H. Huang, and D. Feitosa, "On measuring coupling between microservices," *Journal of Systems and Software*, vol. 200, p. 111670, Jun. 2023, doi: 10.1016/j.jss.2023.111670.

[5] T. Schirgi and E. Brenner, "Quality Assurance for Microservice Architectures," in *2021 IEEE 12th International Conference on Software Engineering and Service Science (ICSESS)*, Beijing, China: IEEE, Aug. 2021, pp. 76–80. doi: 10.1109/ICSESS52187.2021.9522227.

[6] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering – A systematic literature review," *Information and Software Technology*, vol. 51, no. 1, pp. 7–15, Jan. 2009, doi: 10.1016/j.infsof.2008.09.009.

[7] R. C. Amantha Hutapea, A. P. Wahyudi, and Suhardi, "Design Quality Measurement for Service Oriented Software on Service Computing System: a Systematic Literature Review," in *2018 International Conference on Information Technology Systems and Innovation (ICITSI)*, Bandung - Padang, Indonesia: IEEE, Oct. 2018, pp. 375–380. doi: 10.1109/ICITSI.2018.8696092.

[8] R. Yilmaz and F. Buzluca, "A Fuzzy Quality Model to Measure the Maintainability of Microservice Architectures," in *2021 2nd International Informatics and Software Engineering Conference (IISEC)*, Ankara, Turkey: IEEE, Dec. 2021, pp. 1–6. doi: 10.1109/IISEC54230.2021.9672417.

[9] S. Pulnil and T. Senivongse, "A Microservices Quality Model Based on Microservices Anti-patterns," in *2022 19th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, Bangkok, Thailand: IEEE, Jun. 2022, pp. 1–6. doi: 10.1109/JCSSE54890.2022.9836297.

[10] M. Garriga, "Towards a Taxonomy of Microservices Architectures," in *Software Engineering and Formal Methods*, vol. 10729, A. Cerone and M. Roveri, Eds., in Lecture Notes in Computer Science, vol. 10729. , Cham: Springer International Publishing, 2018, pp. 203–218. doi: 10.1007/978-3-319-74781-1_15.

[11] T. Vassiliou-Gioles, "A simple, lightweight framework for testing RESTful services with TTCN-3," in *2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, Macau, China: IEEE, Dec. 2020, pp. 498–505. doi: 10.1109/QRS-C51114.2020.00089.

[12] F. Fahmi, P.-S. Huang, and F.-J. Wang, "Detecting Artifact Anomalies in Microservice-Based Financial Applications," in *2020 IEEE International Conference on Services Computing (SCC)*, Beijing, China: IEEE, Nov. 2020, pp. 418–421. doi: 10.1109/SCC49832.2020.00061.

[13] F. Osses, G. Márquez, and H. Astudillo, "Exploration of academic and industrial evidence about architectural tactics and patterns in microservices," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, Gothenburg Sweden: ACM, May 2018, pp. 256–257. doi: 10.1145/3183440.3194958.

[14] K. Sellami, M. A. Saied, and A. Ouni, "A Hierarchical-DBSCAN Method for Extracting Microservices from Monolithic Applications." arXiv, Jun. 14, 2022. Accessed: Oct. 12, 2023. [Online]. Available: http://arxiv.org/abs/2206.07010

[15] L. Cao and C. Zhang, "Implementation of Domain-oriented Microservices Decomposition based on Node-attributed Network," in *2022 11th International Conference on Software and Computer Applications*, Melaka Malaysia: ACM, Feb. 2022, pp. 136–142. doi: 10.1145/3524304.3524325.

[16] D. Guo and H. Wu, "A Review of Bad Smells in Cloud-based Applications and Microservices," in *2021 International Conference on Intelligent Computing, Automation and Systems (ICICAS)*, Chongqing, China: IEEE, Dec. 2021, pp. 255–259. doi: 10.1109/ICICAS53977.2021.00059.

[17] M. H. Hasan, M. H. Osman, N. I. Admodisastro, and M. S. Muhammad, "Legacy systems to cloud migration: A review from the architectural perspective," *Journal of Systems and Software*, vol. 202, p. 111702, Aug. 2023, doi: 10.1016/j.jss.2023.111702.

[18] M. I. Joselyne, G. Bajpai, and F. Nzanywayingoma, "A Systematic Framework of Application Modernization to Microservice based Architecture," in *2021 International Conference on Engineering and Emerging Technologies (ICEET)*, Istanbul, Turkey: IEEE, Oct. 2021, pp. 1–6. doi: 10.1109/ICEET53442.2021.9659783.

[19] A. Krause, C. Zirkelbach, W. Hasselbring, S. Lenga, and D. Kroger, "Microservice Decomposition via Static and Dynamic Analysis of the Monolith," in *2020 IEEE International Conference on Software Architecture Companion (ICSA-C)*, Salvador, Brazil: IEEE, Mar. 2020, pp. 9–16. doi: 10.1109/ICSA-C50368.2020.00011.

[20] R. Capuano and H. Muccini, "A Systematic Literature Review on Migration to Microservices: a Quality Attributes perspective," in *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, Honolulu, HI, USA: IEEE, Mar. 2022, pp. 120–123. doi: 10.1109/ICSA-C54293.2022.00030.

[21] C.-Y. Li, S.-P. Ma, and T.-W. Lu, "Microservice Migration Using Strangler Fig Pattern: A Case Study on the Green Button System," in *2020 International Computer Symposium (ICS)*, Tainan, Taiwan: IEEE, Dec. 2020, pp. 519–524. doi: 10.1109/ICS51289.2020.00107.

[22] H. R. Lima, K. C. Souza, L. V. De Paula, L. M. C. E Martins, W. F. Giozza, and R. T. De Sousa, "Acceptance Tests over Microservices Architecture using Behaviour-Driven Development," in *2021 16th Iberian Conference on Information Systems and Technologies (CISTI)*, Chaves, Portugal: IEEE, Jun. 2021, pp. 1–6. doi: 10.23919/CISTI52073.2021.9476643.

[23] R. A. Schmidt and M. Thiry, "Microservices identification strategies : A review focused on Model-Driven Engineering and Domain Driven Design approaches," in *2020 15th Iberian Conference on Information Systems and Technologies (CISTI)*, Sevilla, Spain: IEEE, Jun. 2020, pp. 1–6. doi: 10.23919/CISTI49556.2020.9141150.

[24] S. S. De Toledo, A. Martini, P. H. Nguyen, and D. I. K. Sjoberg, "Accumulation and Prioritization of Architectural Debt in Three Companies Migrating to Microservices," *IEEE Access*, vol. 10, pp. 37422–37445, 2022, doi: 10.1109/ACCESS.2022.3158648.

[25] J. Bogner, J. Fritzsch, S. Wagner, and A. Zimmermann, "Microservices in Industry: Insights into Technologies, Characteristics, and Software Quality," in *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, Hamburg, Germany: IEEE, Mar. 2019, pp. 187–195. doi: 10.1109/ICSA-C.2019.00041.

[26] X. Zuo, Y. Su, Q. Wang, and Y. Xie, "An API gateway design strategy optimized for persistence and coupling," *Advances in Engineering Software*, vol. 148, p. 102878, Oct. 2020, doi: 10.1016/j.advengsoft.2020.102878.

[27] P. Bacchiega, I. Pigazzini, and F. A. Fontana, "Microservices smell detection through dynamic analysis," in *2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Gran Canaria, Spain: IEEE, Aug. 2022, pp. 290–293. doi: 10.1109/SEAA56994.2022.00052.

[28] V. Lakhai, O. Kuzmych, and M. Seniv, "An improved approach to the development of software with increased requirements for flexibility and reliability in terms of creating small and medium-sized projects," in *2022 IEEE 17th International Conference on Computer Sciences and Information Technologies (CSIT)*, Lviv, Ukraine: IEEE, Nov. 2022, pp. 474–477. doi: 10.1109/CSIT56902.2022.10000787.

[29] K. Sooksatra, R. Maharjan, and T. Cerny, “Monolith to Microservices: VAE-Based GNN Approach with Duplication Consideration,” in *2022 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, Newark, CA, USA: IEEE, Aug. 2022, pp. 1–10. doi: 10.1109/SOSE55356.2022.00007.

[30] J. Gong and L. Cai, “Analysis for Microservice Architecture Application Quality Model and Testing Method,” in *2023 26th ACIS International Winter Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD-Winter)*, Taiyuan, Taiwan: IEEE, Jul. 2023, pp. 141–145. doi: 10.1109/SNPD-Winter57765.2023.10223960.

[31] M. G. Moreira and B. B. N. De França, “Analysis of Microservice Evolution using Cohesion Metrics,” in *Proceedings of the 16th Brazilian Symposium on Software Components, Architectures, and Reuse*, Uberlandia Brazil: ACM, Oct. 2022, pp. 40–49. doi: 10.1145/3559712.3559716.

[32] S. Soares De Toledo, A. Martini, A. Przybyszewska, and D. I. K. Sjoberg, “Architectural Technical Debt in Microservices: A Case Study in a Large Company,” in *2019 IEEE/ACM International Conference on Technical Debt (TechDebt)*, Montreal, QC, Canada: IEEE, May 2019, pp. 78–87. doi: 10.1109/TechDebt.2019.00026.

[33] J. A. Valdivia, X. Limon, and K. Cortes-Verdin, “Quality attributes in patterns related to microservice architecture: a Systematic Literature Review,” in *2019 7th International Conference in Software Engineering Research and Innovation (CONISOFT)*, Mexico City, Mexico: IEEE, Oct. 2019, pp. 181–190. doi: 10.1109/CONISOFT.2019.00034.

[34] Y. Zhang, B. Liu, L. Dai, K. Chen, and X. Cao, “Automated Microservice Identification in Legacy Systems with Functional and Non-Functional Metrics,” in *2020 IEEE International Conference on Software Architecture (ICSA)*, Salvador, Brazil: IEEE, Mar. 2020, pp. 135–145. doi: 10.1109/ICSA47634.2020.00021.

[35] S. Speth, “Semi-automated Cross-Component Issue Management and Impact Analysis,” in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Melbourne, Australia: IEEE, Nov. 2021, pp. 1090–1094. doi: 10.1109/ASE51524.2021.9678830.

[36] A. Steffens, H. Lichter, and J. S. Döring, “Designing a next-generation continuous software delivery system: concepts and architecture,” in *Proceedings of the 4th International Workshop on Rapid Continuous Software Engineering*, Gothenburg Sweden: ACM, May 2018, pp. 1–7. doi: 10.1145/3194760.3194768.

[37] F. Ponce, J. Soldani, H. Astudillo, and A. Brogi, “Should Microservice Security Smells Stay or be Refactored? Towards a Trade-off Analysis,” in *Software Architecture*, vol. 13444, I. Gerostathopoulos, G. Lewis, T. Batista, and T. Bureš, Eds., in Lecture Notes in Computer Science, vol. 13444. , Cham: Springer International Publishing, 2022, pp. 131–139. doi: 10.1007/978-3-031-16697-6_9.

[38] S. Dalla Palma, M. Garriga, D. Di Nucci, D. A. Tamburri, and W.-J. Van Den Heuvel, “DevOps and Quality Management in Serverless Computing: The RADON Approach,” in *Advances in Service-Oriented and Cloud Computing*, vol. 1360, C. Zirpins, I. Paraskakis, V. Andrikopoulos, N. Kratzke, C. Pahl, N. El Ioini, A. S. Andreou, G. Feuerlicht, W. Lamersdorf, G. Ortiz, W.-J. Van Den Heuvel, J. Soldani, M. Villari, G. Casale, and P. Plebani, Eds., in Communications in Computer and Information Science, vol. 1360. , Cham: Springer International Publishing, 2021, pp. 155–160. doi: 10.1007/978-3-030-71906-7_13.

[39] C.-F. Wu, S.-P. Ma, A.-C. Shau, and H.-W. Yeh, “Testing for Event-Driven Microservices Based on Consumer-Driven Contracts and State Models,” in *2022 29th Asia-Pacific Software Engineering Conference (APSEC)*, Japan: IEEE, Dec. 2022, pp. 467–471. doi: 10.1109/APSEC57359.2022.00064.

[40] S. Kapferer and O. Zimmermann, “Domain-Driven Service Design: Context Modeling, Model Refactoring and Contract Generation,” in *Service-Oriented Computing*, vol. 1310, S. Dustdar, Ed., in Communications in Computer and Information Science, vol. 1310. , Cham: Springer International Publishing, 2020, pp. 189–208. doi: 10.1007/978-3-030-64846-6_11.

[41] J. Bogner, B. Choudhary, S. Wagner, and A. Zimmermann, “Towards a Generalizable Comparison of the Maintainability of Object-Oriented and Service-Oriented Applications,” in *Advances in Service-Oriented and Cloud Computing*, vol. 1115, M. Fazio and W. Zimmermann, Eds., in Communications in Computer and Information Science, vol. 1115. , Cham: Springer International Publishing, 2020, pp. 114–125. doi: 10.1007/978-3-030-63161-1_9.

[42] L. D. S. B. Weerasinghe and I. Perera, “Evaluating the Inter-Service Communication on Microservice Architecture,” in *2022 7th International Conference on Information Technology Research (ICITR)*, Moratuwa, Sri Lanka: IEEE, Dec. 2022, pp. 1–6. doi: 10.1109/ICITR57877.2022.9992918.

[43] G. Rodriguez, L. F. Esteberena, C. Mateos, and S. Misra, “Reducing Efforts in Web Services Refactoring,” in *Computational Science and Its Applications – ICCSA 2019*, vol. 11622, S. Misra, O. Gervasi, B. Murgante, E. Stankova, V. Korkhov, C. Torre, A. M. A. C. Rocha, D. Taniar, B. O. Apduhan, and E. Tarantino, Eds., in Lecture Notes in Computer Science, vol. 11622. , Cham: Springer International Publishing, 2019, pp. 544–559. doi: 10.1007/978-3-030-24305-0_41.

[44] G. Vale, F. F. Correia, E. M. Guerra, T. De Oliveira Rosa, J. Fritzsch, and J. Bogner, “Designing Microservice Systems Using Patterns: An Empirical Study on Quality Trade-Offs,” in *2022 IEEE 19th International Conference on Software Architecture (ICSA)*, Honolulu, HI, USA: IEEE, Mar. 2022, pp. 69–79. doi: 10.1109/ICSA53651.2022.00015.

[45] Y. Rouf, J. Mukherjee, and M. Litoiu, “Towards a Robust On-line Performance Model Identification for Change Impact Prediction,” in *2023 IEEE/ACM 18th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, Melbourne, Australia: IEEE, May 2023, pp. 68–78. doi: 10.1109/SEAMS59076.2023.00018.

[46] H. Vural, M. Koyuncu, and S. Misra, “A Case Study on Measuring the Size of Microservices,” in *Computational Science and Its Applications – ICCSA 2018*, vol. 10964, O. Gervasi, B. Murgante, S. Misra, E. Stankova, C. M. Torre, A. M. A. C. Rocha, D. Taniar, B. O. Apduhan, E. Tarantino, and Y. Ryu, Eds., in Lecture Notes in Computer Science, vol. 10964. , Cham: Springer International Publishing, 2018, pp. 454–463. doi: 10.1007/978-3-319-95174-4_36.