Automated Extraction of Microservice Architecture Using a Rule-Based Approach

Rizqy Ahsana Putri
Department of Informatics Engineering
Institut Teknologi Sepuluh Nopember
Surabaya 60111, Indonesia
6025231069@student.its.ac.id

Muhammad Nevin
Department of Informatics Engineering
Institut Teknologi Sepuluh Nopember
Surabaya 60111, Indonesia
6025231085@student.its.ac.id

Dwinanda Bagoes Ansori
Department of Informatics Engineering
Institut Teknologi Sepuluh Nopember
Surabaya 60111, Indonesia
6025231086@student.its.ac.id

Abdullah Faqih Septiyanto
Department of Informatics Engineering
Institut Teknologi Sepuluh Nopember
Surabaya 60111, Indonesia
7025221022@student.its.ac.id

Riyanarto Sarno
Department of Informatics Engineering
Institut Teknologi Sepuluh Nopember
Surabaya 60111, Indonesia
riyanarto@its.ac.id

2024 2nd International Conference on Technology Innovation and Its Applications (ICTIIA) | 979-8-3503-5161-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICTIIA61827.2024.10761417

***Abstract*— Microservices offer a modular approach to software architecture, enhancing flexibility and scalability. This paper introduces an automated extraction method for evaluating microservices' cohesion, complexity, and granularity. Leveraging a rule-based approach, the system identifies methods within a microservice architecture. Metrics such as Average Lack of Cohesion Metric (ALCOM), Number of Operations (NOO), and Average Service Granularity Metric (ASGM) are calculated to assess the extracted microservices. A case study in Account Payyable Microservice of ERP System illustrates the effectiveness of this method in identifying microservice variables and evaluating architecture quality. This research found that the value of ALCOM is -0.93, NOO is 3.63, and ASGM is 0.50.**

Keywords—Complexity, granularity, microservice, rule-based approach, software

I. INTRODUCTION

Microservice is an approach to software architecture in which applications are built as a series of small, independent services that communicate with each other through a clear interface [1]. Each microservice operates independently, facilitating more flexible and efficient development, deployment, and scaling of applications [2, 3]. With this approach, development teams can focus more on developing and improving a service without disrupting or impacting other services in the application [4, 5].

Much previous research has been conducted to explore the benefits and challenges of using microarchitecture. These studies have investigated various aspects including scalability, performance, data management, and security [6–9]. The results of these researches show that microservice architecture can provide advantages in terms of increasing flexibility, resilience and ease of maintenance of software applications [10, 11].

With the advantages offered by microservice architecture, developers are starting to switch to using microservice architecture. Previous research has recommended that extracting microservices from an existing monolithic code base is easier to do than building microservices from scratch [12]. The extraction process begins with leveraging a monolithic code base, followed by classifying related functionality into individual business capability groups, and then implementing each group as a service. In the industrial world, leading companies such as Netflix and Amazon have successfully extracted microservices from monolithic infrastructure, underscoring the importance of this shift in the industry's transformation towards a more modular and distributed architecture [13, 14].

In an effort to extract microservices, several studies first developed tools to assist the microservice identification process. Examples of tools include Service Cutter, which is not only used to identify microservices but also to evaluate their effectiveness [15, 16]. Additionally, Kieker has the capability to retrieve dynamic information from Java applications [17]. Furthermore, Dbeaver is utilized for extracting database tables [18, 19] and while ExploreViz is employed for the three-dimensional visualization of potential microservices, relying on statistical and dynamic data [20]. Other tools such as OpenAPI, DISCO, MSExtracter, Microservice Miner and Decomposer [21, 22] are also used to assist microservice extraction .

Apart from using tools, several studies implement rules for decomposing monolith architecture into microservices. One research that uses a rule-based approach is research conducted by Eyitemi 2020 [23]. Broadly speaking, this research consists of three stages: (1) Dynamic Analysis, (2) Analyzed Traces, and (3) Package Representation. For the third step, a total of 6 rules are used to make decisions regarding which parts can be divided into microservices. These rules are based on the level of interaction and where cuts will result in the optimal set of microservices.

Al-Debagy [24] in his 2020 research proposed a new set of metrics to measure the quality of microservices architecture. Proposed metrics include the Service Granularity Metric (SGM), the Lack of Coherence Metric (LCOM), and the Number of Operations (NOO). These metrics are used to measure the granularity, coherence, and complexity of each microservice by analyzing the application programming interface (API) [25]. These metrics are useful for illuminating the overall quality of microservices application design. In his study, Al-Debagy examined the suggested metrics across five applications varying in size and business contexts. Findings indicate that the SGM metric's optimal range falls between 0.2 and 0.6 and the LCOM metric ideally should not exceed 10 and should range from 0 to 0.8.

The aim of this research is to identify the variables that exist in microservice architecture. This research proposes a new approach to extracting microservice variables, namely a rule-based approach. There are several sections in this paper: Section 2 presents the literature review. Section 3 explains in detail the methods used in this research. Section 4 explains

the research results. The final section contains the conclusions of this paper.

II. Literature Review

Various metrics, such as LCOM, SGM (Service Granularity Metric), and NOO (Number of Operations), have been devised to evaluate the quality of microservices applications.

A. *The Lack of Cohesion Metric (LCOM)*

LCOM assesses the degree of coherence, or put differently, the extent of interconnectedness among functions within a service concerning their functionality [26]. It calculates by evaluating how often a microservice employs a specific parameter of an operation, divided by the total number of operations multiplied by the count of distinct parameters, as illustrated in Eq. (1).

$$LCOM = 1 - \frac{\sum_{i=1}^{n} MF}{M \times F} \quad (1)$$

where MF denotes the frequency of a parameter, M stands for the total number of operations, and F represents the number of unique parameters within the microservice.

The lower the LCOM value, the stronger the cohesion and coherence among operations, signifying a more efficient and well-structured microservice. To assess the overall lack of cohesion across all microservices within the application, the average LCOM (ALCOM) is calculated using the Eq. (2) that provided below.

$$ALCOM = \frac{\sum_{i=1}^{n} LCOM}{NS} \quad (2)$$

where NS represents the quantity of microservices within the application.

B. The Service Granularity Metric (SGM)

The Data Granularity of a Service (DGS) and the Functional Granularity of a Service (FGS) are two separate evaluation criteria that make up the SGM metric, which is used to gauge the granularity of services inside a microservices application [24]. First, DGS evaluates the volume of input and output data in a given microservice, computing DGS for each activity in the microservice. Eq. (3) demonstrates how this metric operates.

$$DGS = \frac{IPR}{\sum_{i=1}^{n} FP} + \frac{OPR}{\sum_{i=1}^{n} CP} \quad (3)$$

where the total number of parameters in a microservice is indicated by FP, the number of parameters within an operation is indicated by Input Parameters (IPR), the number of output parameters within an operation is indicated by Output Parameters (OPR), and the total number of output parameters in a microservice is indicated by CP. When the DGS number is close to 1, it means that the microservice has coarse-grained data; when it is close to 0, it indicates that the data is fine-grained. Second, the functional granularity of processes within a microservice is evaluated using the Functional Granularity of a Service (FGS) measure. Every operation has different logic or competence levels. Eq. (4) illustrates the functioning of this metric.

$$FGS = \frac{OT}{\sum_{i=1}^{n} O} \quad (4)$$

Here, "OT" refers to the weight allocated to each individual operation inside a microservice, while "O" stands for the total of all weights inside that specific microservice. Lastly, for every activity in the microservices application, the Service Granularity Metric (SGM) takes into account both the DGS and FGS metrics in order to assess the overall granularity of operations. The formulation of SGM is given in Eq. (5).

$$SGM = \sum_{i=1}^{n} DGS \times FGS \quad (5)$$

C. Number of Operations Per Microservice Metric (NOO)

The NOO metric gauges operational functionality by tallying the quantity of operations contained within the microservice [27]. Therefore, a greater number of operations within a microservice corresponds to increased complexity [28]. Eq. (6) illustrates the functioning of this metric.

$$NOO = \sum_{i=1}^{n} M \quad (6)$$

where M represents the quantity of operations per microservice. The ideal threshold for the Number of Operations metric is set at 10 or below. If the value exceeds this suggested range, it indicates that the microservice should be further decomposed into two or more separate microservices.

III. Proposed Method

This section discussed the proposed approach for identifying microservice variables on a microservice architecture. Fig. 1 illustrates the experimental design of this research.

In this research, an automated system for extracting variables and parameters in a microservice will be created. The input used is a microservice architecture for a payment account. In developing automation, the first stage is Find all methods, followed by find parameter. To find and count the number of parameters for each method in microservice, the system will utilize regular expression (regex). Finally, the results of microservice variable extraction will be evaluated using the metrics explained in section IV. Below we will explain the research stages in more detail.

1. Extract Method

At this stage, the system will look for methods in the microservice architecture that have been input previously. The method search will be carried out using Regex. The output from this stage is a collection of methods that have been found.

2. Extract Input and Output Parameters

After the system has successfully extracted the method, the next stage is find the input and output parameters. Just like the previous stage, this stage also uses Regex. The output from this stage is a collection of input and output that were successfully found.

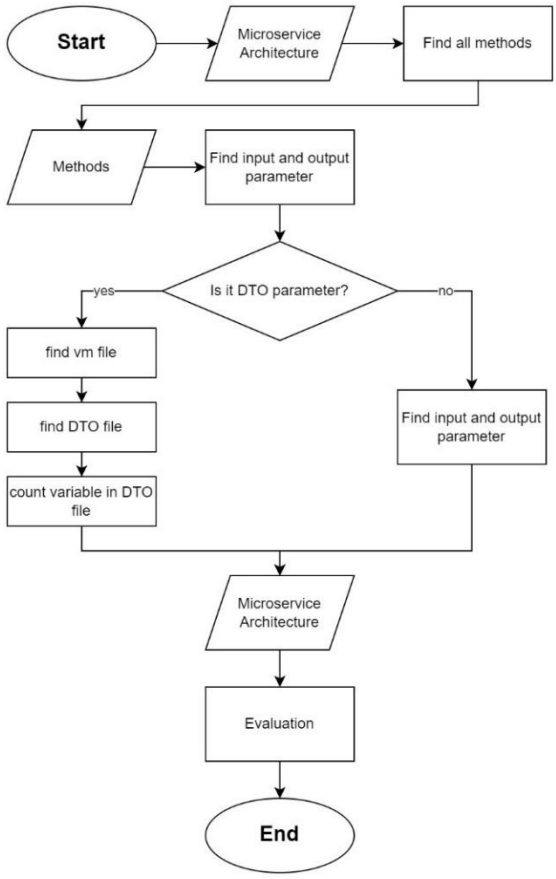

Fig. 1. Proposed Method

3. Extract DTO Parameter

The input and output found are then checked whether they are DTO parameters or not. If it is DTO, the automation system will look for the VM file, then look for the DTO file, and finally calculate the variables in the DTO file. However, if its is not DTO parameters, the system will immediately calculate the existing variables. The final stage is to combine all the parameters that have been found.

4. Regular Expression (Regex)

To count the number of parameters for each method and microservice, the system will employ regular expressions (regex), which are essentially a sequence of characters defining a particular pattern for searching. Each regex can be customized based on references from public forums to align with the system's requirements and the programming language being utilized. The regular expressions used in this research to search for and calculate the number of parameters can be seen in TABLE I.

TABLE I. REGULAR EXPRESSION TO FIND PARAMETER IN ALL METHODS AND MICROSERVICE

Pattern of Regular Expression	Function
(\w+)\s*\(([\s\S]*?)\): \s*Promise<(\w+)>	To find for all existing methods with Promise as output
vm:\s*(\w+)	To find for the body parameter which is usually marked with "vm:TypeName"

TABLE II. PSEUDOCODE TO READ FILES IN THE SYSTEM

```
FUNCTION to_analyze_directory
 parameter in : directory path
  for all files in directory
    read file
    call extract_method_info with parameter file
    extend method info
 parameter out : all methods
END FUNCTION
```

TABLE III. PSEUDOCODE TO COUNT INPUT AND OUTPUT PARAMETER

```
FUNCTION extract_method_info
 parameter in : file
    for code lines
      if a method starting with the word "async" is found
        count variable
        if vm variable
          call count_dto_variable with parameter vm variable
        input_parameter = variable + dto_variable
      if return of method
        count variable
        if vm variable
          call count_dto_variable with parameter vm variable
        output_parameter = variable + dto_variable
      print method name, input parameter, output parameter
 parameter out : method info
END FUNCTION
```

TABLE IV. PSEUDOCODE TO COUNT DTO VARIABLE

```
FUNCTION count_dto_variable
 parameter in : vm variable
  move to vm file based on path in “import” keyword
    if dto variable
      move to dto file based on path in “import” keyword
      count variable in dto file
 parameter out : dto_variable
END FUNCTION
```

The first function is to_analyze_directory. This function is used to read files within the microservice folder. The pseudocode for the to_analyze_directory function can be seen in TABLE II. Next, in TABLE III, there is pseudocode for the second function, namely the extract_method_info function. The extract_method_info function is used to obtain information about each method. The third function is the count_dto_variable function in Table IV. What this function does is it navigates to the VM and DTO files, then calculates the total number of variables in the DTO file.

IV. RESULT & DISCUSSION

1. Microservice Architecture

In this case study, a back-end program will be used in an Account Payyable Microservice of ERP (Enterprise Resource Planning) System related to Account Payable. This system is built with typescript language Node.js architecture approach.

There are a number of folders that will explain their respective functions. An explanation of the Account Payyable Microservice of ERP System architecture structure is shown in TABLE V.

Within this structure, the business-layer directory is expected to contain the core business logic of the application. It includes a services subdirectory, where the business operations are abstracted into services. Two TypeScript files, purchaseDownPaymentInvoice.service.ts and purchaseInvoice.service.ts, suggest methods handling the logic for processing purchase down payment invoices and purchase invoices, respectively. Both methods have several functions that will be counted as parameters. A detailed description of the method and the number of parameters can be seen in TABLE VI.

Data interactions are managed through the data-access directory, which would typically interface with databases or other persistence mechanisms. The helpers directory usually stores utility functions that provide support across the application, facilitating code reuse and modular design.

The dtos (Data Transfer Objects) directory houses TypeScript files like purchaseDownPaymentInvoice.dto.ts and purchaseInvoice.dto.ts, which define the data structures for the transfer of data across different layers of the application. These objects help to encapsulate data and ensure type safety while data is being transported, for instance, from the database to the client-facing side of the application.

Enumeration values that represent a fixed set of constants are placed in the enum directory, improving readability and type safety. Any application-wide messages, possibly including error messages and system notifications, are defined within the messages folder. The utility directory typically contains general utility functions, which are leveraged throughout the application to perform common tasks in a consistent manner.

View models, represented by the files in the view-models directory such as purchaseDownPaymentInvoice.vm.ts, purchaseInvoice.vm.ts, and result.vm.ts, contain data structures designed for presentation logic, dictating how data will be sent to the frontend for display to the user. Infrastructure-related code, which might include server configurations, middleware, and other foundational code, is found within the infrastructure folder. The node_modules directory is a standard inclusion in Node.js projects, containing all the third-party modules and packages installed for the application.

Lastly, the web-main directory could potentially contain the entry point for the web server or the main application file that kicks off the web application. Without access to the actual TypeScript files, the specific methods within the purchaseDownPaymentInvoice and purchaseInvoice mehods cannot be ascertained. Typically, these services would include methods for creating, processing, and managing invoices, with input parameters and return types defined to ensure consistent data handling and operation execution. TypeScript enriches JavaScript by adding static types and object-oriented features, which enhances code quality and maintainability in large-scale applications like an ERP system.

TABLE V. ARCHITECTURE STRUCTURE OF ACCOUNT PAYYABLE MICROSERVICE OF ERP SYSTEM

Directory	Description
business-layer	Contains business logic of the application.
services	Holds service classes for specific operations like handling purchase invoices.
data-access	Manages interactions with databases or data storage.
helpers	Stores utility functions supporting various application parts.
dtos	Data Transfer Objects directory, defining data structures for data transfer.
enum	Contains enumerations for constants improving code readability and safety.
messages	Defines system-wide messages such as errors or notifications.
utility	General utility functions used across the application.
view-models	View models for data presentation, typically for the front end.
infrastructure	Contains foundational code like server configurations and middleware.
node_modules	Node.js packages installed for the application.
web-main	Likely holds the main application file for the web server.

TABLE VI. ARCHITECTURE DEESCRIPTION OF METHODS IN ACCOUNT PAYYABLE MICROSERVICE OF ERP SYSTEM

Method Name	Function Name	Description	Function Input Parameter	Function Output Parameter
purchase-Down-Payment-Invoice	find-All-Invoices	responsible for retrieving all invoices related to down payments on purchases	0	0
	find-Invoice-By-IDE	searches for a specific down payment invoice by its unique identifier	1	1
	find-Invoices-By-Criteria	allows for searching invoices based on a set of criteria, such as purchase code or invoice number	2	1
	create-Invoice	This operation is used to create a new down payment invoice with the provided details	17	30
	calculate-Down-Payment	responsible for calculating the down payment amount for a purchase invoice	18	3
purchase-Invoice	find-All-Invoices	responsible for retrieving all invoices related to down payments on purchases	0	0
	find-Invoice-By-ID	searches for a specific down payment invoice by its unique identifier	1	1
	find-Invoices-By-Criteria	allows for searching invoices based on a set of criteria, such as purchase code or invoice number	2	1
	invoice-Exists-By-Purchase-Order-Code	this method checks if an invoice already exists for a given purchase order code	1	1
	create-Invoice	This operation is used to create a new down payment invoice with the provided details	14	24

TABLE VII. CALCULATION PROCESS OF ASGM

Method Name	IPR	OPR	MF	M	F	ALCOM	FP	CP	DGS	OT	O	FGS	DGS*FGS	ASGM	NOO
purchase-Down-Payment-Invoice	38	35	116	2	30	-0,93	56	62	0,618	4	8	0,5	0,309	0,5	3,63
purchase-Invoice	18	27							0,318	4		0,5	0,190		

2. Analysis of Microservice Methods

The proposed method will automatically extract the number of input parameters and output parameters for each method in each microservice. Furthermore, 3 variables ALCOM (Average LCOM), ASGM (Average SGM), and NOO will be generated. Metrics such as input and output parameters indicate the complexity of the method interfaces, providing insight into the level of interaction that each method has with other components of the system. Each calculation obtained can be seen in TABLE VII.

We can calculate the value of IPR and OPR values based on the total number of parameters. It can be seen that the MF value is the sum of the IPR and OPR values for each method so that we get

$$MF = \sum_{i=1}^{n} IPR_i + \sum_{i=1}^{n} OPR_i = 116$$

N he value of M is obtained from the number of methods in the system, namely 2 methods including 'purchase-Down-Payment-Invoice' and 'purchase-Invoice'. Based on automatic extraction, there are 30 unique parameters that will be included as the F value. Furthermore, Based on Equation (1), the calculation can be made the value of ALCOM is

$$ALCOM = 1 - \frac{116}{2 \times 30} = -0{,}93$$

Other calculations are Number of Operations (NOO) and Average Service Granularity Metric (ASGM). Based on Eq. (2), the NOO value can be calculated through the average of the number of methods in each microsevice (M) so that the calculation is

$$NOO = \frac{MF}{M+F} = \frac{116}{2+30} = 3{,}63$$

Furthermore, for calculating ASGM, it can be seen through the process in Eq. (3) – Eq. (5). The ASGM calculation can be seen in TABLE VII.

3. Comparison with Previous Research

This research also compares both the metrics and extraction processes with several similar papers published previously. Previous studies have employed various metrics, including response time, number of calls, COSMIC function points, ALOCM, ASGM, and NOO. However, the extraction processes in papers [29–31] were conducted manually. A notable contribution of this paper is the implementation of an automated extraction process. A more detailed comparison is presented in Table VIII.

TABLE VIII. COMPARISON WITH PREVIOUS RESEARCHES

Ref	Metric	Extraction Process
Shadija, Rezai, and Hill [29]	Response time, Number of calls	Manual
Vural, Koyuncu, and Misra [30]	COSMIC function points	Manual
Al-Debagy and Martinek [31]	ALOCM, ASGM, NOO	Manual
This Paper	ALOCM, ASGM, NOO	Automated

V. CONCLUSION

The presented rule-based approach offers a systematic method for evaluating microservices within their own architecture. By automating the identification of methods and parameters, developers can assess the cohesion, complexity, and granularity of microservices. Metrics such as ALCOM, NOO, and ASGM provide valuable insights for architecture evaluation. The case study demonstrates the applicability of the approach, highlighting its potential to improve software design practices. This result show that the evaluation value of ALCOM is -0.93, NOO is 3.63, and ASGM is 0.50. Future work may focus on exploring additional metrics to further enhance the evaluation process and optimize microservice quality.

ACKNOWLEDGMENT

This research was funded by Institut Teknologi Sepuluh Nopember (ITS) under Penelitian Dana Departemen Program, Penelitian Keilmuan, Penelitian Flagship, Penelitian Kolaborasi Pusat and under the Project Scheme of the Publication Writing and Intellectual Property Rights (IPR) Incentive (Penulisan Publikasi dan Hak Kekayaan Intelektual/PPHKI) Program 2024.

REFERENCES

[1] V. Velepucha and P. Flores, "A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges," *IEEE Access*, vol. 11, pp. 88339–88358, 2023, doi: 10.1109/ACCESS.2023.3305687.

[2] H. Calderón-Gómez, L. Mendoza-Pittí, M. Vargas-Lombardo, J. M. Gómez-Pulido, D. Rodríguez-Puyol, G. Sención, and M.-L. Polo-Luque, "Evaluating Service-Oriented and Microservice Architecture Patterns to Deploy eHealth Applications in Cloud Computing Environment," *Applied Sciences*, vol. 11, no. 10, p. 4350, May 2021, doi: 10.3390/app11104350.

[3] J. Zaki, S. M. R. Islam, N. S. Alghamdi, M. Abdullah-Al-Wadud, and K.-S. Kwak, "Introducing Cloud-Assisted Micro-Service-Based Software Development Framework for Healthcare Systems," *IEEE Access*, vol. 10, pp. 33332–33348, 2022, doi: 10.1109/ACCESS.2022.3161455.

[4] G. Marquez, C. Taramasco, H. Astudillo, V. Zalc, and D. Istrate, "Involving Stakeholders in the Implementation of Microservice-Based Systems: A Case Study in an Ambient-Assisted Living System," *IEEE Access*, vol. 9, pp. 9411–9428, 2021, doi: 10.1109/ACCESS.2021.3049444.

[5] B. Harjo, R. Sarno, and S. Rochimah, "Automatic Web Service Composition for SaaS Business Intelligence," *International Journal of Intelligent Engineering and Systems*, vol. 13, no. 2, pp. 286–298, Apr. 2020, doi: 10.22266/ijies2020.0430.28.

[6] M. Söylemez, B. Tekinerdogan, and A. Kolukısa Tarhan, "Challenges and Solution Directions of Microservice Architectures: A Systematic Literature Review," *Applied Sciences*, vol. 12, no. 11, p. 5507, May 2022, doi: 10.3390/app12115507.

[7] Y. Wang, H. Kadiyala, and J. Rubin, "Promises and challenges of microservices: an exploratory study," *Empir Softw Eng*, vol. 26, no. 4, p. 63, Jul. 2021, doi: 10.1007/s10664-020-09910-y.

[8] R. Laigner, Y. Zhou, M. A. V. Salles, Y. Liu, and M. Kalinowski, "Data Management in Microservices: State of the Practice, Challenges, and Research Directions," Feb. 2021.

[9] A. Hannousse and S. Yahiouche, "Securing microservices and microservice architectures: A systematic mapping study," *Comput Sci Rev*, vol. 41, p. 100415, Aug. 2021, doi: 10.1016/j.cosrev.2021.100415.

[10] M. Ali Temoor and B. Muzaffar Ali, "Architecture for Microservice Based System. A Report," 2020, doi: 10.13140/RG.2.2.17340.16004/1.

[11] A. Razzaq and S. A. K. Ghayyur, "A systematic mapping study: The new age of software architecture from monolithic to microservice architecture—awareness and challenges," *Computer Applications in Engineering Education*, vol. 31, no. 2, pp. 421–451, Mar. 2023, doi: 10.1002/cae.22586.

[12] A. F. A. A. Freire, A. F. Sampaio, L. H. L. Carvalho, O. Medeiros, and N. C. Mendonça, "Migrating production monolithic systems to microservices using aspect oriented programming," *Softw Pract Exp*, vol. 51, no. 6, pp. 1280–1307, Jun. 2021, doi: 10.1002/spe.2956.

[13] S. McAleese, J. C. McLaughlin, F. Detyna, A. Murashev, M. Yilmaz, and P. M. Clarke, "Serverless Software Engineering – and How to Get There.," 2022, pp. 75–90. doi: 10.1007/978-3-031-15559-8_6.

[14] A. Henry and Y. Ridene, "Migrating to Microservices," in *Microservices*, Cham: Springer International Publishing, 2020, pp. 45–72. doi: 10.1007/978-3-030-31646-4_3.

[15] D. Bajaj, A. Goel, and S. C. Gupta, "GreenMicro: Identifying Microservices From Use Cases in Greenfield Development," *IEEE Access*, vol. 10, pp. 67008–67018, 2022, doi: 10.1109/ACCESS.2022.3182495.

[16] X. Sun, S. Boranbaev, S. Han, H. Wang, and D. Yu, "Expert system for automatic microservices identification using <scp>API</scp> similarity graph," *Expert Syst*, vol. 41, no. 5, May 2024, doi: 10.1111/exsy.13158.

[17] N. Lapuz, P. Clarke, and Y. Abgaz, "Digital Transformation and the Role of Dynamic Tooling in Extracting Microservices from Existing Software Systems," 2021, pp. 301–315. doi: 10.1007/978-3-030-85521-5_20.

[18] D. Bajaj, U. Bharti, A. Goel, and S. C. Gupta, "A Prescriptive Model for Migration to Microservices Based on SDLC Artifacts," *Journal of Web Engineering*, Jun. 2021, doi: 10.13052/jwe1540-9589.20312.

[19] L. Jackson, "Persisting Our Data," in *The Complete ASP.NET Core 3 API Tutorial*, Berkeley, CA: Apress, 2020, pp. 113–166. doi: 10.1007/978-1-4842-6255-9_7.

[20] I. Oumoussa and R. Saidi, "Evolution of Microservices Identification in Monolith Decomposition: A Systematic Review," *IEEE Access*, vol. 12, pp. 23389–23405, 2024, doi: 10.1109/ACCESS.2024.3365079.

[21] R. Sarno, Y. A. Effendi, and F. Haryadita, "Modified Time-Based Heuristics Miner for Parallel Business Processes," *International Review on Computers and Software (IRECOS)*, vol. 11, no. 3, p. 249, Mar. 2016, doi: 10.15866/irecos.v11i3.8717.

[22] S. Huda, T. Ahmad, R. Sarno, and H. A. Santoso, "Identification of process-based fraud patterns in credit application," In: *2014 2nd International Conference on Information and Communication Technology (ICoICT)*, May 2014, pp. 84–89. doi: 10.1109/ICoICT.2014.6914045.

[23] F.-D. Eyitemi and S. Reiff-Marganiec, "System Decomposition to Optimize Functionality Distribution in Microservices with Rule Based Approach," In: *2020 IEEE International Conference on Service Oriented Systems Engineering (SOSE)*, Aug. 2020, pp. 65–71. doi: 10.1109/SOSE49046.2020.00015.

[24] O. Al-Debagy and P. Martinek, "A Metrics Framework for Evaluating Microservices Architecture Designs," *Journal of Web Engineering*, Jun. 2020, doi: 10.13052/jwe1540-9589.19341.

[25] Y. Dian Harja and R. Sarno, "Determine the best option for nearest medical services using Google maps API, Haversine and TOPSIS algorithm," In: *2018 International Conference on Information and Communications Technology (ICOIACT)*, Mar. 2018, pp. 814–819. doi: 10.1109/ICOIACT.2018.8350709.

[26] E. N. H. Kirgil and T. E. Ayyildiz, "Analysis of Lack of Cohesion in Methods (LCOM): A Case Study," In: *2021 2nd International Informatics and Software Engineering Conference (IISEC)*, Dec. 2021, pp. 1–4. doi: 10.1109/IISEC54230.2021.9672419.

[27] O. Al-Debagy and P. Martinek, "Extracting Microservices' Candidates from Monolithic Applications: Interface Analysis and Evaluation Metrics Approach," In: *2020 IEEE 15th International Conference of System of Systems Engineering (SoSE)*, Jun. 2020, pp. 289–294. doi: 10.1109/SoSE50414.2020.9130466.

[28] N. Y. Setiawan and R. Sarno, "Multi-Criteria Decision Making for Selecting Semantic Web Service Considering Variability and Complexity Trade-Off," *J Theor Appl Inf Technol*, vol. 86, no. 2, pp. 316–326, 2016, [Online]. Available: https://api.semanticscholar.org/CorpusID:53700238

[29] D. Shadija, M. Rezai, and R. Hill, "Microservices: granularity vs. performance," In: *Companion Proceedings of the10th International Conference on Utility and Cloud Computing*, Dec. 2017, pp. 215–220. doi: 10.1145/3147234.3148093.

[30] H. Vural, M. Koyuncu, and S. Misra, "A Case Study on Measuring the Size of Microservices," *Vural, H., Koyuncu, M., & Misra, S. (2018). A Case Study on Measuring the Size of Microservices. Communication Systems and Applications.*, pp. 454–463, 2018, doi: 10.1007/978-3-319-95174-4_36.

[31] O. Al-Debagy and P. Martinek, "A Metrics Framework for Evaluating Microservices Architecture Designs," *Journal of Web Engineering*, Jun. 2020, doi: 10.13052/jwe1540-9589.19341.