ISSN 2347 - 3983

Volume 9. No. 8, August 2021

International Journal of Emerging Trends in Engineering Research

Available Online at http://www.warse.org/IJETER/static/pdf/file/ijeter10982021.pdf

https://doi.org/10.30534/ijeter/2021/10982021

A Survey on Trending Topics of Microservices

Garvit Jain[1], Urjita Thakar[2], Vandan Tewari[3], Sudarshan Varma[4]

[1]Research Scholar, Department of Computer Engineering, Shri Govindram Seksaria Institute of Technology and Science, Indore, India, ajmeragarvit06@gmail.com

[2]Professor, Department of Computer Engineering, Shri Govindram Seksaria Institute of Technology and Science, Indore, India, urjita@rediffmail.com

[3]Assosciate Professor, of Computer Engineering, Shri Govindram Seksaria Institute of Technology and Science, Indore, India, vandantewari@gmail.com

[4]Technical Lead Director, Advanz101 systems Pvt. Ltd., Corporate House, 309, B Block RNT Marg, Indore, India, sudarshan@advanz101.com

ABSTRACT

Microservice is an architectural style and a software development methodology. Microservices are used as a group of independent units with narrowly specified responsibilities that interact through well-described REST APIs. It is observed that the most challenging part in developing any application using microservice architecture is decomposing it into the correct level of granularity at design and run time, which requires good skills and domain knowledge. In this paper, some prominent topics in microservices analysis, such as determining the size and boundaries of microservices using various decomposition approaches and extraction of microservices from large monolithic applications have been discussed and analysed. Work pertaining to essential quality metrics required for a microservices-based system has also been surveyed. In this survey paper, we have identified how these topics are correlated and proposed some steps that might be beneficial in the transformation of monolithic applications into microservices.

Key words: Microservice, Architecture, REST, API, Decomposition, Monolithic, Metrics

1. INTRODUCTION

Service orientation, agile deployment and implementation paradigms have most recently resulted in a special flavor of software development called microservices [1]. Microservice is the current trend in the design, implementation, and production of software services. A microservice implements a software and systems architecture strategy that builds on the existing modularization concept while also highlighting technical borders [2]. As microservices are mainly modeled around business domains, they elude the difficulties of traditional tiered architectures. It also combines new technologies and techniques that had appeared over the past decade, helping them to alleviate the pitfalls of many Service-Oriented Architecture (SOA) implementations [3].

Microservices may also be declared with several functionality levels, and "the size of this functionality is commonly referred to as its granularity" [4]. Though there is lot of hype and increased interest in microservices shown by the software industry, there is still a general lack of formal methods to model microservices architecture decisions, including the decision of optimal microservice boundaries, and optimum level of microservice granularity [5]. The aim is to obtain the most important division when identifying model boundaries and granularity for microservices for which a great skill and domain expertize is required [6].

A widely used pattern for designing business applications is the monolithic architecture pattern. For small applications, it works fairly well: it is relatively easy to design, test and deploy small monolithic applications. For large applications, it makes more sense to use a microservice architecture that breaks down the complex functionalities into a series of services [7]. Extracting microservices from monoliths is the next version of the initial problem of decomposing software structures, and has become a vital branch of software engineering practice[8]. A banking domain experience report shows how restructuring a monolithic architecture into microservices will enhance software agility and enhance scalability as each microservice becomes an independent development, implementation, versioning, operations and scaling entity [2]. Issues unique to monolithic systems can be solved by migrating to microservice architecture. These include reducing connectivity in framework and increasing maintainability. Microservice Architecture is becoming a more common alternative for addressing current problems as it improves device maintainability and speeds up the delivery of software products [9].

A survey is carried out to classify the key trending areas of microservice research in this paper. The literature review is divided into four sections: a) The granularity of microservices, b) Identification of microservices using decomposition techniques, c) Extraction of microservices from the monolithic application and, d) Essential quality metrics required for the microservice-based system. Work related to granularity is discussed in next section.

2. GRANULARITY OF MICROSERVICES

Granularity refers to the size of a microservice, and it is important since microservice granularity affects the quality of service (QoS) [10]. Gouigoux, J.P., et. al., suggested that "granularity should be driven by the balance between the costs of Quality Assurance and the cost of deployment" [11]. A number of researchers have discussed the issue of determining the right size of the microservices however it is still an open issue [2], [12], [13]. Several factors affect the decision process for microservice granularity, these are

1) the manner in which a microservice interacts with other microservices.
2) mechanism to publish or bind to the API Gateway protocol
3) communication of microservices with the database [14].

According to Sam Newman, domain-driven design is the core basis for all granularity-based decisions, in which microservices are separated according to the application domain [3]. Different levels of granularity also has different impact on efficiency level and utilization of resources for the same workload [15].

Hassan, et al., proposed architecture-centric modeling with the concept of "aspects" for microservices which expressively capture microservitization scenarios with distinct QoS trade- offs. The research is demonstrated on an "online movie subscription based system". Analysis has been performed using the characteristics of the ADL classification framework. The results indicate that microservice environments promote analysis, mobility and location awareness which are important in adapting granularity of quality-driven microservices [5].

Bounded Context (BC) pattern that adopts a Domain Driven Design pattern known as Context Mapping, is used for modeling and determining the domain model boundaries of microservices [6]. Merson, et al., have discussed five different microservice design scenarios around Domain Driven Design aggregrates, Bounded Context (BC), domain events, and other strategies for inter-BC. Domain Driven Design comes with several advantages (a) Domain Driven Design (DDD)can help with defining microservices. (b) DDD key concepts are bounded context, aggregate, and entity. (c) A service can have the scope of an aggregate. (d) A microservice can have the scope of a bounded context (e) DDD can use domain events for inter-microservice interaction. (f) DDD can help to define the size of microservice not the Line of code (LOC) size, the size in terms of functional scope [16],[17].

Vera-Rivera, et al., suggested a genetic algorithm based Backlog model for Microservice and described a Granularity metric to determine the size of microservices. The approach used Domain Driven Design (DDD), and quasi-experiment for evaluation. The model showed better coupling and coherence, less microservices-related activities. It also enabled higher average calls from one microservice to another and a lower value for Granularity Metric. The drawback of the approach was that it resulted into coarse-grained microservices [18]. It has been observed that researchers have proposed several models for determining the correct size, similarities and boundaries of microservices at design and runtime.
In next section,the work carried out for identification of microservices by different researcher is reviewed.

3. IDENTIFICATION OF MICROSERVICES

Identifying microservices from large applications by decomposing it into suitable components using appropriate methods is another challenge in software engineering. Microservices are an autonomous unit which can be easily scaled, deployed, reuse and maintain without affecting other services. If a large application is decomposed into microservices, improvement can be seen in the quality of application and time of implementation.

Baresi, et al., explained the concept of Open API specification to identify semantic similarity based on the available functionality. It was found that about 80 percent cases obtained correct granularity and cohesiveness of microservices. The drawback of the method is that input artifacts may be mapped into very few definitions of shared vocabulary, resulting in a degradation of coarse-grained microservices [19].

Tyszberowicz, et al., introduced an approach to describe microservices depending upon specification of the use cases and functional decomposition. A KAMP tool was used to maintain the software for evaluating research on a CoCoME (Common Component Modeling Example) case study. The method ensured high coherence and low-coupling decomposition identical to manual design and within a much shorter time span [20].

Gysel, et al., proposed a systematic, consistent strategy to service decomposition based on sixteen coupling criteria extracted from literature and market practice which enabled loose coupling and better cohesion within services [21].

Abdullah, M., et al., suggested a black-box based decomposition technique to identify URL partitions through mining application logs and unsupervised machine learning. The method showed better performance, coherence and scalability of the auto-micro service compared to manual and monolithic approach. Though, advanced auto-scaling techniques were still not sufficient to achieve acceptable performance of device [22].

Zhang, Y., et al., developed an AMI genetic algorithm-based approach to optimize the identification of microservices and achieve high-cohesion-low-coupling and load balance of CPU and memory consumption while taking into account both functional and non-functional metrics. The work was evaluated on JPetStore a Legacy ERP System using Kieker Monitoring Framework and Java Visual VM. The result has shown that non-functional indicators affected the microservice recognition outcome and the AMI approach [23].

De Alwis, et al., suggested two fundamental fields of microservice functional splitting. The first is based on subtypes of artifacts (structural properties) and second was functional splitting based on common fragments of program execution (behavioral properties).The result showed that the strategies used were appropriate for migrating legacy systems to high-cohesion and low-coupling microservices and allowed greater scalability, availability, performance, and reliability [24].

Jin, et al., proposed a Functional oriented Service Candidate Identification (FoSCI) model to distinguish service candidates from a monolithic system. A search-based functional atom grouping algorithm was used for research. The model was compared with six widely-used open-source projects and three existing methods. The results showed that FoSCI overtakes existing methods [25].

Stojanovic, et al., discussed how microservices may be defined using structural framework analysis. It defined guidelines for identifying the domain and data storage for each microservice. The concept of primitive functions, data streams, data storage and interfaces has been used along with the data dictionary, the hierarchical set of data flow diagrams and the specification of the primary functions [26].

A microservice identification method that decomposes a system using clustering technique for identification of cohesive, loosely coupled, and fine-grained microservices from a single business process, or a set of processes is also available [27].

The researchers noted that identifying microservices from a monolithic application is a difficult process and requires a good level of understanding and domain expertise. Also, from the recent researches it is observed that various decomposition techniques like clustering, open API specification, function splitting based on artifacts, use cases, and functional decomposition have been used to identify high-cohesion and low-coupling-based microservices from a monolithic application.

4. EXTRACTION OF MICROSERVICES FROM MONOLITHIC APPLICATION

Microservice is booming today and most technology industries are switching from monolithic application to microservice-based system. Industries prefer microservice-based systems as this are efficient, scalable, versatile and easy to maintain. Decomposition and identification plays an important role in shifting from a monolithic application to a microservice-based system.

Mazlami, et al., introduced a formal microservice extraction model in the refactoring and migration scenario to provide algorithmic recommendations of microservice candidates. The efficiency and consistency of the approach has been tested on 21 Java, Ruby, and Python open-source projects. All efficiency experiments have displayed satisfying performance levels for different situations. The study shows that the strategy limits the team size of the microservice to one quarter or less of the monolithic team size. Also, model depends upon strategies and graph for computing classes as an single unit that restricts the flexibility when refactored monolithic application [28].

A purified data flow-driven mechanism to test the microservice-orientated decomposition process has been proposed by Chen, et al.. It has been shown that all the effects of the decomposition and its process are rational, objective and understandable [29]. Al-Debagy, and Martinek, proposed an approach to identify API to sementically identical microservices. The approach is evaluated on Money transfer and Kanban Board Applications using affinity propagation algorithm and fast Text model. While using cohesion and complexity metrics the result shows that the algorithm may be useful in development of software architectures and developers in transition from monolithic design to microservices [30].

Taibi, and Systa, adopted a data-driven approach focused on process mining performed on log files gathered at runtime to classify microservice candidates. Approach was demonstrated on Industrial case study using DISCO tool. The method facilitated the search for alternative decomposition methods and offered a collection of measures to determine decomposition efficiency. The system may be beneficial for firms in detecting software problems which the architect did not find manually, to increase the decomposition standard of any monolithic method [31].

Carvalho, et al. analyzed the criteria for the extraction of microservices architecture with 15 experts. This survey is based on seven criterias specified in the article by researchers or practitioners. Specialists on microservice migration participated in the survey. Participants found modularity – i.e. coupling, cohesion, and reuse – criteria to be more appropriate. Author feels that existing methods, tools, and decisions on microservice extractions are reliable and inaccurate due to lack of synthesized information about the relevant criteria and their trade-offs [32].

Escobar, et al. introduced a model-centered approach to analyze and show existing configurations and relations between the business and the data layer in JEE (SISINFO) and marketplace applications using static code analysis. The results show that the understanding is significantly strengthened by the diagrams, and the modularization is the first step towards the application’s automated modernization. The approach uses a static analysis, hence authors were unable to identify dependencies between the business and data layers that are not specified through

Java-annotations [33].

In the surveyed literature, various extraction techniques have been discussed with their pros and cons. It is observed that the extraction techniques used can efficiently modularize applications into microservices by

1) maintaining loose coupling and high cohesion
2) reusing components
3) reducing team size
4) can be useful in detecting early software bugs.

In next section, the work carried out by different researcher for essential quality metrics required for microservices is presented.

5. ESSENTIAL QUALITY METRICS FOR MICROSERVICE BASED SYSTEM

All software organizations are conscious about quality attributes and metrics. In this section, the essential quality metrics are discussed. Bogner, et al., discussed the importance of the size, complexity, cohesion and coupling metrics which are helpful at the time of designing microservices [1]. Tugrul et al., discussed determination of quality of the applications while creating appropriate products based on the microservice architecture. Service size, coordination of inter-services and bad practices have been considered to quantify micro-services by also counting resources and consumers. A static analyser has been built to measure a ticket-selling application. The author argued that automatic quality software management was necessary to control software, identify bugs and bad smells by running advanced policies [34].

Candela, et al. conducted two studies aiming at the objective and subjective exploration of the components by the coupling and software modularization. The result showed that the modularization of open source software was far from optimal in terms of both architectural and logical cohesion/coupling. Subjectively, 83 percent of developer findings claimed that only high cohesion and low coupling recommenders may not be enough to suggest re-modularization solutions [35].

In another work, empirical study to investigate the interactions between various cohesion and coupling metrics in object- oriented system has been presented. It was observed that the relationship between cohesion and coupling is inversely associated with each other [36].

Li, et al. suggested a decision-making criteria and methods for the resolution of microservice granularity from the maintainability perspective. The results show that the maintenance metric of each microservice has dramatically enhanced the scalability and maintenance of the system [37].

Rud, et al. suggested metrics that can be used to assess complexity, performance and reliability of compound services and service oriented systems. For implementation, web service metadata and WS-BPEL has been used to automatically assess the quality of distributed business process both at design time and run time [38].

Cojocaru, et al. proposed a set of quality criteria based on two meta-criteria: meaningfulness in the decomposition context and feasibility of implementation. These quality criteria were applied on both Microservice Architecture and Service Oriented Architecture. They proposed that the quality of microservices derived from the semi-automatic application may be useful to provide a complete solution for migrating monolithic applications to Microservice Architecture [39].

Hirzalla, proposed a SOA metrics framework which included both service level and SOA-wide metrics to measure design and runtime qualities of a SOA solution. The SOA-wide metrics predicted the overall complexity, agility and health status of SOA solutions, while service level metrics focused on the fundamental building blocks of SOA i.e. services [40].

From the reviewed literature it was observed that metrics such as maintenance, coupling, cohesion, complexity are widely used for enhancing the quality of microservices in terms of maintainability, scalability, availability, reliability and performance. Authors have also suggested several other criteria to measure the quality of application such as meaningfulness in the decomposition context, feasibility of implementation, calculation of service size, coordination of inter-services, and bad practices.

6. CONCLUSION

In this paper, the earlier research carried out on prominent issues on microservices have been surveyed. Available work has been categorized into four sections such as

1) granularity of microservices
2) identification of microservices using decomposition techniques
3) extracting microservices from monolithic application
4) essential quality metrics needed for microservice based system.

Microservice Ambient approach is useful for determining optimal level of granularity at design time and it also supports run time analysis. As Microservice Backlog genetic algorithm has lower granularity metrics, lesser average calls and higher cohesion value for identifying microservices as compared to Service cutter, MITIA, AMI algorithm, and DDD. DDD is found to be the best decomposition approach but has some critical failure points which are overcome by Microservice Backlog genetic algorithm.

Domain Driven approach has potential for providing decomposition solution but they are not validated by expert and are not yet established in practice. MITIA and evaluation metric algorithm may be useful for software architects and developers to help them transition from monolithic design to micro-services.

To maintain the quality of the microservice-based system, advanced software policies and metrics are required. It is observed that high cohesion and low coupling has inverse relationship. Other metrics like product, complexity etc. are also required for designing and re-modularizing the software system.

The researchers followed a common methodology for migrating any monolithic application to a microservice-based system which is defined step by step as follows

1) Identify the number of microservices that can be extracted by decomposing the application.
2) Use a decomposition technique to extract out the identified microservices.
3) Calculate the granularity and other quality metrics for each microservice, as size and quality metrics are useful in assessing the aspects of microservice-based applications.

Since Microservices are being adopted very fast, research is still in progress in different related field.

REFERENCES

[1]Justus Bogner, Stefan Wagner, and Alfred Zimmermann: "Automatically measuring the maintainability of service- and microservice-based systems." In: Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement on - IWSM Mensura 2017, Association for Computing Machinery, New York, NY, USA, pp. 107–115,2017.

[2]P. Jamshidi, C. Pahl, N. C. Mendonc,a, J. Lewis, and S. Tilkov, "Mi- croservices: The journey so far and challenges ahead," IEEE Software, vol. 35, no. 3, pp. 24–35, 2018.

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

[4]D. Shadija, M. Rezai, R. Hill, Microservices: Granularity vs. Perfor- mance. In UCC 2017 Companion - Companion Proceedings of the 10th International Conference on Utility and Cloud Computing. Association for Computing Machinery, Inc. 2017. pp. 215-220.

[5]S. Hassan, N. Ali, and R. Bahsoon, "Microserviceambients: An archi- tectural meta-modelling approach for microservice granularity," in Proc. ICSA 2017. IEEE, pp. 1–10, April 2017.

[6]Identify domain-model boundaries for each microservice. Accessed on: Jan 25, 2021. Available at: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/identify-microservice-domain-model-boundaries

[7]C. Richardson , Microservices: Decomposing Applications for Deployability and Scalability, 2014 Available: https://www.infoq.com/articles/microservices-intro (2014),Accessedon: Jan 25,2021.

[8]G. Mazlami, "Algorithmic extraction of microservices from monolithic code bases," Master Thesis, Software Evolution and Architecture Lab, Department of Informatics, University of Zurich, 2017.

[9]P. Di Francesco, P. Lago and I. Malavolta, "Migrating Towards Microservice Architectures: An Industrial

[10]F. Rademacher, S. Sachweh, and A. Zundorf, "Differences between model-driven development of service-oriented and microservicear- chitecture," in Int. Conf. on Software Architecture Workshop Proc. (ICSAW).IEEE, 2017.

[11]J.-P. Gouigoux and D. Tamzalit, "From monolith to microservices: Lessons learned on an industrial migration to a web oriented architec- ture," in 2017 IEEE International Conference on Software Architecture Workshops (ICSAW). IEEE, pp. 62–65,2017.

[12]S. Hassan, R. Bahsoon, and R. Kazman, "Microservice transition and its granularity problem: A systematic mapping study". Software: Practice and Experience, 2019.

[13]Mohammad Sadegh Hamzehloui, Shamsul Sahibuddin, and Ardavan Ashabi, "A Study on the Most Prominent Areas of Research in Microservices," International Journal of Machine Learning and Computing, vol. 9,no.2,pp.242-247,2019.

[14]I. J. Munezero, D. Mukasa, B. Kanagwa and J. Balikuddembe, "Partitioning Microservices: A Domain Engineering Approach," 2018 IEEE/ACM Symposium on Software Engineering in Africa (SEiA), Gothenburg, Sweden, 2018, pp. 43-49.

[15]O. Mustafa and J. Marx G ´omez, "Optimizing economics of microservices by planning for granularity level," in Proceedings of the ProWeb 2017 Programming Technology for the Future Web, Brussels Belgium, April 2017.

[16]P. Merson and J. Yoder, " Modeling Microserviceswith DDD". Presentation at SATURN 2019. Available at saturn2019.sched.com/event/LY5d/modelling-microservices-with-ddd. May2019.

[17]P. Merson and J. Yoder, "Modeling Microservices with DDD", IEEE International Conference on Software Architecture Companion (ICSAC), Salvador, Brazil, 2020, pp. 7-8.

[18]F.H. Vera-Rivera, E.G. Puerto-Cuadros, H. Astudillo, C.M. Gaona- Cuevas, "Microservices Backlog - A Model of Granularity Specification and Microservice Identification." In: Wang Q., Xia Y., Seshadri S., Zhang LJ. (eds) Services Computing – SCC 2020. SCC 2020. Lecture Notes in Computer Science, vol. 12409, Springer, Cham (2020).

[19]L. Baresi, M. Garriga, A. De Renzis: Microservicesidentification through interface

analysis. In: De Paoli, F., Schulte, S., Broch Johnsen, E. (eds.) ESOCC 2017. LNCS, vol. 10465, pp. 19–33, Springer, Cham (2017).

[20]S. Tyszberowicz, R. Heinrich, B. Liu, Z. Liu: Identifying microservices using functional decomposition. In: Feng,X., Müller-Olm, M.,Yang, Z. (eds.) SETTA 2018. LNCS, vol. 10998, pp. 50–65, Springer, Cham (2018).

[21]M. Gysel, L. Kolbener, W. Giersche, and O. Zimmermann: Service cutter: a systematic approach to service decomposition. In: Aiello, M., Johnsen, E.B., Dustdar, S., Georgievski, I. (eds.) ESOCC 2016. LNCS, vol.9846,pp.185–200,Springer,Cham(2016).

[22]M. Abdullah, W. Iqbal, and A. Erradi, "Unsupervised learning approach for web application auto-decomposition into microservices," Journal of Systems and Software, vol. 151, pp. 243 – 257, 2019.

[23]Y. Zhang, B. Liu, L. Dai, K. Chen, and X. Cao, "Automated Mi- croservice Identification in Legacy Systems with Functional and Non-Functional Metrics." in 2020 IEEE International Conference on Software Architecture, (ICSA), pp. 135-145, 2020.

[24]De Alwis, A.A.C., Barros, A., Polyvyanyy, A., Fidge, C.: Function- splitting heuristics for discovery of microservices in enterprise systems. In:Pahl,C., Vukovic,M., Yin,J.,Yu,Q.(eds.) ICSOC2018. LNCS, vol. 11236, pp. 37–53. Springer, Cham (2018).

[25]Jin, W., Liu, T., Cai, Y.,Kazman, R., Mo, R., Zheng, Q.: Service candidate identification from monolithic systems based on execution traces.IEEETrans.Softw.Eng.(2019).

[26]Stojanovic, T. D., Lazarevic, S. D., Milic, M., & Antovic, I. (2020). Identifying microservices using structured system analysis. 2020 24th International Conference on Information Technology (IT) (2020).

[27]M. J. Amiri, "Object-Aware Identification of Microservices," 2018 IEEE International Conference on Services Computing (SCC), 2018, pp. 253- 256

[28]G. Mazlami, J. Cito, and P. Leitner, "Extraction of microservices from monolithic software architectures," In: 2017 IEEE InternationalConfer- ence on Web Services (ICWS). IEEE (2017), pp. 524–531, 2017.

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

[30]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) (2020), pp. 289-294, 2020.

[31]Taibi,D., Systa,K., "From monolithic systems to microservices: A decomposition framework based on processmining," in Proceedings of the9th International Conference onCloud Computing and Services percent Science - Volume 1: CLOSER, INSTICC. SciTePress, pp. 153–164,2019.

[32]L. Carvalho, A. Garcia, W. K. G. Assuno, R. de Mello and M.J. de Lima, "Analysis of the criteria adopted in industry to extract microservices", Proceedings of the Joint7th International Workshop on Conducting Empirical Studies in Industry and 6th International Workshop on Software Engineering Research and Industrial Practice ser.CESSER- IP '19, pp.22-29,2019.

[33]D.Escobar, D.Cárdenas, R.Amarillo, E.Castro, K.Garcés, C.Parra, and R.Casallas, "Towards the understanding and evolution of monolithic applications as microservices", in 2016 XLII Latin American Computing percent Conference (CLEI) ,pp.1–11,Oct2016.

[34]Tugrul Asik, "Policy Enforcement upon Software Based on Microservice Architecture. "IEEE computer society(2017),pp.283–287,2017.

[35]I. Candela, G. Bavota, B. Russo and R. Oliveto: "Using cohesion and coupling for software remodularization: is it enough?" ACM Trans. Softw. Eng. Methodol., vol. 25, no. 3, pp. 1–28, 2016.

[36]S. A. Miquirice and R. S. Wazlawick, "Relationship between cohesion and coupling metrics for object-oriented systems" in Information and Software Technologies, Cham: Springer International Publishing, pp. 424-436,2018.

[37] Li, Y., Wang, C.-Z., Li, Y., & Su, J. (2020). Granularity Decision of Microservice Splitting in View of Maintainability and Its Innovation Effect in Government Data Sharing. Discrete Dynamics in Nature and Society, 2020, 1–11(2020).

[38] Rud, D., Schmietendorf, A., Dumke, R.R.: Product Metrics for Service- Oriented Infrastructure. In: International Workshop on Software Mea-surement/Metrikon 2006(2006).

[39]M. Cojocaru, A. Uta and A. Oprescu, "Attributes Assessing the Quality of Microservices Automatically Decomposed from Monolithic Applications," 2019 18th International Symposium on Parallel and Distributed Computing(ISPDC),2019,pp.84-93(2019).

[40] Hirzalla, M., Cleland-Huang, J., Arsanjani, A.: A metrics suite for evaluating flexibility and complexity in service oriented architectures. In: Feuerlicht, G., Lamersdorf, W. (eds.) ICSOC 2008. LNCS, vol. 5472, pp. 41–52. Springer, Heidelberg (2009).

[41] Rizki, M., Fajar, A. N., & Retnowardhani, A. (2020). Microservices architecture design: Proposed for online healthcare. International Journal of Emerging Trends in Engineering Research, 8(4), 1040–1046. https://doi.org/10.30534/ijeter/2020/14842020