Research on Quality Model and Measurement for Microservices

Jinchuan Yu
Software Engineering Institute
Shanghai Key Laboratory of Computer Software Testing & Evaluation
Shanghai Development Center of Computer Software Technology
Shanghai, China
yjc_sirius@126.com

Jianxin Ge
Software Engineering Institute
Shanghai Key Laboratory of Computer Software Testing & Evaluation
Shanghai Development Center of Computer Software Technology
Shanghai, China
gjx@sscenter.sh.cn

Jixin Sun
Software Engineering Institute
Shanghai Key Laboratory of Computer Software Testing & Evaluation
Shanghai Development Center of Computer Software Technology
Shanghai, China
29577661@qq.com

***Abstract*—In recent years, microservice architecture (MSA) has become popular. Emerging from the agile community, MSA implies a number of small, independently deployable microservices. They are characterized by low coupling, high cohesion, low complexity, and are more flexible and convenient in application, saving resource efficiency.However, there is limited research on quality models for MSA. Although MSA is a special form of service-oriented architecture, there are still some differences between the two that are hard to ignore, such as decentralization, smaller service size and encouraging technical heterogeneity. Therefore, it is difficult to directly apply the traditional quality model research of service-oriented architecture to MSA. Based on the above considerations, this paper proposes a quality model for MSA, which reflects the quality characteristics of microservices through 16 measures for each of the Functional Suitability, Flexibility, Interactivity, Performance Efficiency, Reliability and Security. At the same time, to address the shortage of theoretical validation of current research, this paper uses static analysis and dynamic analysis to validate the proposed measures and prove the rationality of its theory.**

Keywords—microservice architecture, software attribute, quality model

I. INTRODUCTION

A microservice architecture (MSA) is a variant of the Service- Oriented Architecture (SOA) structural style. As to Martin Fowler, it is to build an application as a group of services each designed for a specifific business capability, and intercommunicate via lightweight mechanisms while being independently deployable [13]. In a microservices architecture, services are fine-grained and the protocols are lightweight. MSA is currently the most popular architecture for companies creating new applications, due to its advantages, such as agility or scalability [5]. The architecture has a crucial role in the software life-cycle, for example to ensure quality and critical attributes of the software [14]. To take advantage of the microservice-style architecture, much effort has been invested into porting legacy, monolithic applications[11], [6]. Microservices is a young research area and there is very little work on comprehensive and systematic evaluation of microservice architectures(MSA). In this paper, we propose a quality model to evaluate MSA more comprehensively by mapping measures to multiple aspects of software properties.

Most of the software quality measures used in the early process-oriented paradigm of software systems were oriented towards underlying software measures, such as the number of codes, the complexity of functions or control flows, etc.After object-oriented programming became the dominant programming paradigm, existing measures and standards may not be fully applicable to object-oriented software systems, which has led researchers to reevaluate existing measures and propose new ones at the same time. Among them, the most influential ones are the C&K measure set proposed by Chidamber and Kemerer [15, 16] and the L&H measure set proposed by Li and Henry [17]. They introduce CBO (Coupling Between Object Classes), which have received a great deal of attention. With the further expansion of the system size, SOA (Service-Oriented Architecture) became more and more popular, the level of abstraction of software measure objects has been raised again. Perepletchikov et al [18] proposed two sets of measures to measure service-oriented software systems by two software properties: coupling [19] and cohesion [20]. Mario et al [4] argue that the measure values of software attributes can reflect software quality and evaluate microservice architectures in terms of cohesion, coupling, and complexity through a data-driven approach, using maintainability as an example. Yang et al [1] measured the maintainability of microservice systems by four software attributes: scale, coupling, cohesion and complexity, and established a maintainability quality model for microservice architectures. However, according to ISO/IEC 25010:2011, maintainability is only a part of the MSA quality model and no comprehensive and holistic MSA quality model has been proposed.[1] Integrating Yang et al.'s research on the quality model of MSA maintainability, Tong et al.'s research on MSA functional efficiency and others' research results, and the ISO/IEC 25010:2011 standard, this paper proposes a quality model of MSA to reflect the quality of micro The quality characteristics of microservices are reflected by 16 measures in functional suitability, flexibility, interaction, performance efficiency, reliability and security. Flexibility is a measure of the ease of secondary development, expansion and maintenance of microservice architecture, including four measures of cohesion, coupling, complexity and reusability; interactivity is a measure of the association and influence between microservice architecture and other microservices and external systems, including two measures of interoperability and coexistence. The core characteristics of microservice architecture are characterized.

The remainder of this paper is organized as follows. Section II analyzes and quantitatively expresses the MSA,

This work is supported by National High Technology Research and Development Program of China(2018YFB1403400) and the Hebei Provincial Key Research and Development Program (Grant No. 21312102D).

Section III proposes an MSA-oriented quality model, Section IV validates the quality model results, and the summary and future work are in Section V.

II. MSA Model Analysis

A. Formal representaton of MSA

In order to clearly and accurately describe the MSA and define the measures proposed subsequently, this subsection abstracts the microservice architecture of interest by applying the approach of Yang et al. [1] to this paper and formalizes it accordingly. The conceptual software architecture of the entire microservice system can be represented in the form of Figure 2.1. Based on Yang et al.'s study, we added the relationship between the microservice system and external programs as an attribute .

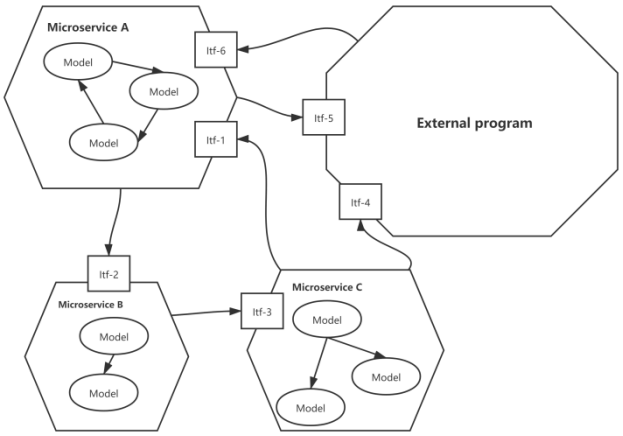

Figure 2.1: Conceptualizing software architecture for microservice systems

For a single microservice in a microservice system, formalize each microservice S_i in the following representation.

$$S_i = (M_{S_i}, ITF_{S_i}, MR_{S_i})$$

M_{S_i} denotes a collection of modules in a microservice S_i.

$$M_{S_i} = \{m_j | j \in N_0\}$$

N_0 means natural number.

ITF_{S_i} denotes the set of interfaces in a microservice S_i.

$$ITF_{S_i} = \{itf_i | j \in N_0\}$$

$$itf_i = (name_j, P_j)$$

MR_{S_i} represents the dependency relationship between modules in Microservices S_i.

$$MR_{S_i} = \{mr_j | j \in N_0\}$$

$$mr = (smr, dmr), where\ mr.smr, mr.dmr \in M_{S_i}$$

smr is the module that depends on other modules in the corresponding dependency, and dmr is the module on which other modules depend in the corresponding dependency. The whole microservice system can be formalized as a collection of microservices, and dependencies between microservices. Integrated system(IS) refers to the sum of microservice system and external programs

$$IS = (S, IR, EPR)$$

IR represents dependencies between microservices and other microservices.

$$IR = \{ir_i | i \in N_0\}$$

$$ir = (sir, dir, itf), where\ ir.sir, ir.dir\ \in S;\ ir.itf \in ITF$$

sir is the microservice that depends on other microservices in the corresponding dependency, dir is the microservice on which other microservices depend in the corresponding dependency, and itf is the interface involved in the corresponding dependency.

EPR represents the relationship between the microservice system and external programs.

B. MSA Attributes Analysis

MSA consists of one microservice unit, each of which has the ability to perform functions independently; at the same time, microservices show the characteristics of low coupling and high cohesion, making the whole MSA more flexible, and each module and each microservice unit has the ability to be reused; in addition, due to the low complexity and small scale of microservice system, the resources required for execution are lower and more efficient. To address the advantages and characteristics of MSA, we developed a quality model of MSA by combining the research of Michel-Daniel et al.[1]

III. Quality Model For MSA

A. Software Quality Model Study

Software quality models can help us to better propose and apply measures, and usually measure models describe the entities, attributes, and relationships of measures, and currently common measure models in the field of software engineering include the GQM (Goal Question Measure) model [21] and the QMOOD (Quality Model for Object-Oriented Design) model.

1) GQM model:

The GQM model is the "Goal-Problem-Measure" model, which is one of the common measures models in software engineering practice, and is based on the idea that measures are measured by answering specific questions about the goal of the measure. The GQM model consists of three main layers: the conceptual layer, the operational layer, and the data layer. The conceptual layer is the goal to be measured, and the operational layer decomposes the abstracted goal into concrete questions. The data layer will give specific measures for these questions. By deriving the measure values of the measures and thus answering the questions, the specific measure of the target is finally obtained. The framework of the approach is shown in Figure 3.1.

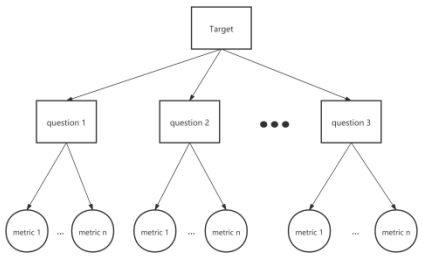

Figure 3.1: GQM Model Framework

2) QMOOD model:

The QMOOD model is a four-level hierarchical quality model proposed by Bansiya and Davis [22]. Originally applied to the quality assessment of object-oriented software systems, it has been migrated to other types of software systems [23]. The QMOOD model is a hierarchical model consisting of quality attributes and software features that reflect the quality attributes.The model consists of four layers.The first layer is the quality attributes that are the goals of the assessment; the second layer is more specific software attributes such as cohesion, coupling, comprehensibility etc.; the third layer is the specific measures that can be used to evaluate the second layer of software attributes; and the last layer is the software components that these measures focus on, such as classes, components, interfaces, etc. Figure 3.2 illustrates the four-layer quality model framework of QMOOD.

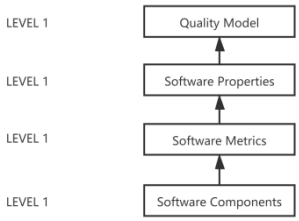

Figure 3.2: QMOOD Model Framework

B. MSA Quality Model

The final quality model for the microservice-oriented architecture is shown in Figure 3.3. The first layer is the sum of the quality attributes to be evaluated in this paper. The second layer is the software attributes that can reflect the characteristics of the microservice architecture, which are Functional Suitability, Flexibility, Interactivity, Performance Efficiency, Reliability and Security. The third layer is the specific measures, which are 16 in total. These 16 measures each of the six software attributes in different aspects. The fourth layer is the object to be measured by the proposed measures. Considering that the microservice architecture advocates technical heterogeneity, the selected measures are to a certain extent independent of the programming language and technical implementation, this paper selects modules, interfaces, microservices and their related relationships as the measures of microservices themselves based on the characteristics of microservices. At the same time, we add the external system as the measure object because we have to consider the interaction between microservice system and other systems.

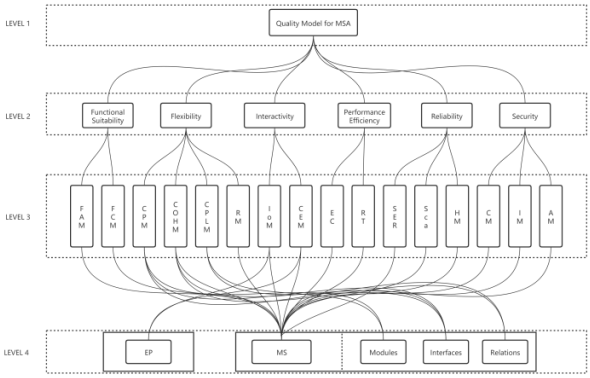

Figure 3.3: Quality model for MSA

C. Study On measures Of MSA Quality Model

This section measures MSA by Functional Suitability, Flexibility, Interactivity, Performance Efficiency, Reliability and Security. Each of the six software attributes and their measures will be described below.

1) Functional Suitability:

This attribute measures degree to which MSA provides functions that meet stated and implied needs when used under specified conditions.

a) FAM(Functional appropriateness measures): This measure measures degree to which the functions facilitate the accomplishment of specified tasks and objectives. The user is counted to demonstrate the steps necessary to complete a task as well as any unnecessary steps. The ratio of the former to the latter is the measure for this section.

b) FCM(Functional correctness measures): This measure measures degree to which MSA provides the correct results with the needed degree of precision.

2) Flexibility:

This attribute is an upgraded version of maintainability, which reflects the concept of high maintainability of microservice systems ---- coupling, cohesion and complexity, and also reflects the ability of microservice systems to be developed twice, which is the concept of reusability we proposed.

a) CPM(Coupling measures): The coupling measure measures the degree of dependency between elements within a microservice and between a microservice and other microservices. Loose coupling has a positive impact on flexibility.The measures proposed in this paper quantify the coupling at two main levels, the microservice implementation element level and the microservice level, respectively. The main concerns include the dependency relationships between modules belonging to the same microservice and the invocation relationships between microservices generated through interfaces.

- Internal Microservices Coupling of Model，IMSCM. The IMSCM calculates the sum of the dependencies between the modules inside the microservice, i.e., the connections between modules [24]. In fact, this type of coupling measure has been validated in studies of service-oriented architectures [12] and is considered to be directly related to coupling.

$$IMSCM = \left|MR_{S_i}\right|$$

- Weighted Microservices Coupling of Interface， WMSCI. WMSCI measures the degree of dependency between microservices and microservices in a system, and in this measure, this paper abstracts the coupling to the microservice level, and the specific measures include the invoking and invoked relationships. In this paper, we refer to the study of Kulesza et al [25], which argues that both incoming and outgoing coupling are very important and that these couplings contribute to the decision of whether to refactor or not.The WMSCI measure is equal to the sum of the weighted coupling value of the microservice's dependency on other microservices and the weighted coupling value of the microservice's dependency on other microservices, which is calculated as follows.

$$WMSCI = |\{ir \in IR | ir.sir = S_i \cup ir.dir = S_i\}|$$

b) COHM(Cohesion measures): A design with high cohesion significantly enhances the understandability and testability of a software system, while improving its stability and modifiability, which in turn affects the flexibility of the software system. However, compared to coupling, cohesion is difficult to analyze quantitatively and measure automatically, and more often than not cohesion relies on semantic or subjective evaluation.

- Microservices Cohesion of Interface Data，MSCID. MSCID quantifies the cohesiveness of a given microservice by measuring the degree of similarity of the parameters passed in the interfaces exposed by its microservices; a microservice is highly cohesive if all interfaces work on the same type of input parameters.

$$MSCID = \frac{2 * \sum_{a=1}^{n} \sum_{b=a+1}^{n} \left| \frac{P_a \cap P_b}{P_a \cup P_b} \right|}{n(n-1)}$$

- Microservices Cohesion of Interface Usage，MSCIU. MSCIU quantifies the cohesiveness of a given microservice by measuring the invocations of that interface by other microservices. A microservice is considered highly cohesive when each user of the microservice (microservice consumer) invokes all the public interfaces of the microservice, which can be considered highly relevant for implementing a certain functionality.

$$MSCIU = \frac{|\{ir \in IR | ir.dir = S_i\}|}{\left|ITF_{S_i}\right| * \left|CS_{S_i}\right|}$$

- Microservices Cohesion of Model， MSCM. This measure abstracts the methods and properties in a class to the module level in a microservice. If the number of connectivity graphs formed by all modules and dependencies of a microservice is 1, then the microservice is cohesive to a certain extent. However, considering that many microservice architectures manage the operations related to interfaces through a unified module, which often leads to the module becoming the hub of an otherwise unconnected graph, the module and its dependencies are removed from the connected graph when using this measure for cohesion measurement.

$$MSCM = \frac{2 * \sum_{a=1}^{n} \sum_{b=a+1}^{n} IsBg(M_a, M_b)}{n(n-1)}$$

M_a and M_b are modules in the microservice, $IsBg(M_a, M_b)$ is to calculate whetherM_a and M_b belong to the same connectivity graph, and its return value is 1 or 0.

c) CPLM(Complexity measures): Complexity primarily measures the complexity of the microservice implementation and execution functions, and more broadly includes the ease with which developers can perceive, understand, and modify them. High complexity can have a negative impact on flexibility.In this paper, complexity is measured by the following two measures, taking into account the complexity of the microservice implementation itself and the complexity of developer awareness and understanding.

- Microservices Model Propagation Cost， MSMPC. MSMPC refers to the visibility theory from the study [26], which measures the extent to which changes in a single module lead to potential changes in other modules within the microservice. This is expressed as the possible information flow and dependencies between modules, which can be obtained through the passing of dependencies. If a module is directly or indirectly dependent on a large number of other modules, the more likely it is that changes to this module will affect other modules. MSMPC is calculated as follows.

$$MSMPC = \sum_{a=1}^{n} \sum_{b=1}^{n} IsCt(M_a, M_b, T_c)$$

where Ma and Mb are modules in the microservice, and Is Ct(Ma, Mb, TC) is the calculation of whether Ma and Mb can be connected by the passing of dependencies, i.e., whether the value of the passing closure matrix at their locations is not zero.

- Microservices Parameter Count， MSPC. MSPC mainly measures the number of data structures appearing in the microservice, and in this paper we mainly consider the number of parameters of the interface, which includes its own interface and the interface of invoking other microservices. MSPC is calculated as follows.

$$MSPC = \bigcup_{j=1}^{|ITFC|} P_j, itf_j \in ITFC$$

d) RM (Reusability measures): Reusability is a very desirable quality measure for industry [27], due to its major cost reduction prospects. Moreover, the microservice can be repurposed so that with little changes can be used outside of its design time domain.

- The degree of reuse of microservices in secondary development. Counts the number of times a microservice module is reused in the execution of different services to characterize its degree of reuse.

- Time efficiency of reuse of microservices in secondary development. Characterize the reuse time efficiency of a microservice module by counting the time it consumes to reuse it during the execution of different services.

3) Interactivity:

This section examines the interaction between the microservice system and the external programs. The microservices system has to cooperate with the external programs to achieve the function, but not to affect the normal operation of the external programs.

a) IOM (Interoperability measures): The extent to which microservices systems exchange information with external programs.

b) CEM (Co-existence measures): The microservice system does not affect the execution of its own functions when interacting with external programs, i.e., no exceptions occur in either program during the execution of the test.

4) Performance Efficiency: performance relative to the amount of resources used under stated conditions. Resources include monetary resources and time resources.

a) EC (Execution cost): this measure is computed as the monetary cost of the resources used for running the

microservice. The cost can be estimated at design time and corrected after the implementation is evaluated at execution time.[2]

b) RT (Response time): the anticipated delay between the time when a request to a microservice is issued and the time when the result is delivered.The average response time of all requests over a period of time is measured, and the smaller the average response time, the faster the processing speed and the more efficient the service.[3]

$$X = \sum_{i=1}^{n} (A_i)/n$$

A_i=Time taken by the system to response to a specific user task or system task at i-th measurement.
n=Number of responsed measures

5) Reliability: degree to which a system, product or component performs specified functions under specified conditions for a specified period of time.

a) SER (Successful execution rate): the ability of a service provider to successfully fulfil the requests within a given period of time. It is measured as a number between 0 and 1 or a percentage calculated as the ratio between successful requests and the total number of requests.

b) Sca (Scalability): the ability of a microservice to function correctly (as designed) irrespective of the changes in size (amount of resources) without inquiring performance penalties. the degree of scalability can be calculated by analyzing the distribution of synchronous requests provided by the exposed interfaces, a high diversity of requests indicating poor scalability.

c) HM (Health management): a quality attribute describing the ability of a microservice to cope with failures. A microservice complies to this property by saving the internal state, and restarting automatically while loading the most up-to-date state prior to the failure. It is a binary attribute with ”yes” or ”no” values and it is verified via instance graphs or type graphs [8].

6) Security: degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization.

a) CM (Cofidentiality measures): degree to which a product or system ensures that data are accessible only to those authorized to have access.

$$CM = 1 - A/B$$

A=Numbers of cofidentiality data items that can be accessed without authorization
B=Number of data items that require access control

b) IM (Integrity measures): degree to which a system, product or component prevents unauthorized access to, or modification of, computer programs or data.

$$IM = A/B$$

A=Number of data corruption prevention methods actually implemented
B=Number of data corruption prevention methods available and recommended

IV. Validation of Quality Models

measure-based evaluation is compulsory for assessing the quality attributes of microservices. A vast number of quality criteria measuring a variety of aspects concerning microservices exist [9]. However, evaluation approaches are scarce [7], while assessment methods for semi -automatic decompositions are entirely missing [10]. Therefore, we have designed a combination of static and dynamic analysis. In the following, we introduce the static and dynamic analysis, and then specific attributes will be selected for analysis.

A. Static Analysis:

It is a technique that does not require execution of the analyzed software. Scanning the code of a microservice before being linked to other microservices can aid the identification and correction of vulnerabilities without incurring the costs of running the code.

B. Dynamic Analysis:

It is a technique that requires the execution of the analyzed software, usually employed when the application code is not available. The most common type of dynamic analysis consists of Unit Tests and it has the benefit of validating static analysis findings or identify new flaws.

C. Validation test result Analysis:

a) Functional Suitability Validation Analysis: The analysis of Functionality Suitability is a dynamic analysis. To verify the functionality of the microservice, the program of the microservice unit is executed and then compared with the expected result, if the result meets the expectation, it means the Functionality Suitability is good.

b) Flexibility Validation Analysis: The analysis of Flexibility belongs to a combination of static and dynamic analysis. By studying the modules and interfaces of microservices and their relationships, we can determine the coupling, cohesion, and complexity of microservices. These belong to static analysis; however, when studying the reuse of microservices in secondary development, we need to run microservices in different development environments, which belongs to dynamic analysis.

c) Co-existence measures Validation Analysis: Co-existence measures means that the interaction between the microservice architecture and the external program does not affect the execution of their respective functions, and that no exceptions occur in either program during the execution of the test. Obviously, we need to make the microservices system and the external program run simultaneously, which is a dynamic analysis .

d) Confidientially measures Validation Analysis: The extent to which the microservices system ensures that only authorized people have access to the data. The detection of such measures is a dynamic analysis and requires verification that unauthorized users have access to the data.

V. Summary and Future Work

A. Summary

Microservices is a young research area, and there is still relatively little work on quality models for microservices architectures. Although microservice architectures, as a special form of service-oriented architectures, can to some extent draw on quality modeling work on service-oriented architectures, there are still some differences between the two that cannot be ignored, such as decentralization, more promotion of technical heterogeneity, and smaller service size. At the same time, the current research on quality models for

services often lacks theoretical validation, which may be more important than quantitative analysis. Based on this, this paper conducts quality modeling research for microservice architectures, and its main work and contributions include.

1) A quality model is proposed:

A quality model is proposed for the microservice architecture, which reflects the quality characteristics of microservices by measuring the six software attributes of microservices Functional Suitability, Flexibility, Interactivity, Performance Efficiency, Reliability and Security through 16 measures. The former considers the compatibility between the microservice architecture and external programs, and the latter adds the examination of the reusability of the microservice architecture on the basis of maintainability.

2) Validation of the quality model

We designed a combination of dynamic and static analysis to validate and analyze the MSA quality model with key indicators to increase the reliability of the results.

B. Future work

Although this paper establishes a quality model under microservice architecture and conducts theoretical validation to prove its effectiveness, there are still some shortcomings and areas that need further research. For example, other quality model measures will be added, such as cohesion attributes that are relatively incomplete, and subsequent research can measure cohesion through semantic analysis to achieve a more accurate assessment of flexibility. Moreover, a more comprehensive quantitative study will be conducted subsequently after more industrial data are collected to further prove the validity of the model.

REFERENCES

[1] Deyu Yang, Research on Software Maintainability Quality Model for Microservices Architecture, 2020

[2] Michel-Daniel Cojocaru, Alexandru Uta, Ana-Maria Oprescu, Attributes Assessing the Quality of Microservices Automatically Decomposed from Monolithic Applications, 2019 18th International Symposium on Parallel and Distributed Computing (ISPDC).

[3] Yexin Tong, Xinkui Qu, Research on Quality of Service Assurance in Microservice Architecture, 2019

[4] M. Cardarelli, L. Iovino, P. D. Francesco, A. D. Salle, I. Malavolta, P. Lago, An extensible data-driven approach for evaluating the quality of microservice architectures, in: C. Hung, G. A. Papadopoulos (Eds.), Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019, Limassol, Cyprus, April 8-12, 2019, ACM, 2019, pp. 1225‑1234.

[5] P. Di Francesco, P. Lago and I. Malavolta, ” Migrating Towards Microservice Architectures: An Industrial Survey”, in International Conference on Software Architecture (ICSA), Seattle, WA, 2018, pp. 29-2909.

[6] L. Baresi, M. Garriga, A. De Renzis, ” Microservices Identification Through Interface Analysis, in ” Service-Oriented and Cloud Computing”, ESOCC, 2017. Lecture Notes in Computer Science, vol 10465. Springer

[7] Engel, Thomas, Melanie Langermeier, Bernhard Bauer and Alexander Hofmann. Evaluation of Microservice Architectures: A measure and ToolBased Approach, CAiSE Forum, 2018.

[8] N. Alshuqayran, N. Ali and R. Evans, ” A Systematic Mapping Study in Microservice Architecture ”, 9th International Conference on ServiceOriented Computing and Applications (SOCA), Macau, 2016, pp. 44-51.

[9] J. Bogner, S. Wagner, A. Zimmermann, ” Automatically measuring the maintainability of service and microservice-based systems - a literature review ”. in 27th International Workshop on Software Measurement, Gothenburg, Sweden, 2017, pp. 107-115.

[10] J. Bogner, S. Wagner, A. Zimmermann, ” Towards a practical maintainability quality model for service and microservice-based systems ”, in Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings, ECSA, 2017, pp. 195198.

[11] M. Gysel, L. Kolbener, W. Giersche and O. Zimmermann, ” Service Cutter: A Systematic Approach to Service Decomposition ”, in ESOCC, Vienna, 2016, pp. 185-200.

[12] M. Perepletchikov, C. Ryan, A controlled experiment for evaluating the impact of coupling on the maintainability of service-oriented software, IEEE Transactions on Software Engineering 37 (4) (2011) 449‑465.

[13] M. Fowler and J. Lewis, Microservices, 2014. [Online]. Available: https://martinfowler.com/articles/microservices.html

[14] R. Kazman, S. G. Woods, and S. J. Carriere. ” Requirements for integrating software architecture and reengineering models ”, in Proceedings of the Working Conference on Reverse Engineering (WCRE), Washington DC, 1998, pp. 154163.

[15] S. R. Chidamber, C. F. Kemerer, Towards a metrics suite for object oriented design, SIGPLAN Not. 26 (11) (1991) 197‑211.

[16] S. R. Chidamber, C. F. Kemerer, A metrics suite for object oriented design, IEEE Transactions on Software Engineering 20 (6) (1994) 476‑493.

[17] W. Li, S. Henry, Object-oriented metrics that predict maintainability, Journal of Systems and Software 23 (2) (1993) 111‑122, object-Oriented Software.

[18] M. Perepletchikov, C. Ryan, K. Frampton, Comparing the impact of service-oriented and object-oriented paradigms on the structural properties of software, in: R. Meersman, Z. Tari, P. Herrero (Eds.), On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 431‑441.

[19] M. Perepletchikov, C. Ryan, K. Frampton, Z. Tari, Coupling metrics for predicting maintainability in service-oriented designs, in: 2007 Australian Software Engineering Conference (ASWEC’07), 2007, pp. 329‑340.

[20] M. Perepletchikov, C. Ryan, K. Frampton, Cohesion metrics for predicting maintainability of service-oriented software, in: Seventh International Conference on Quality Software (QSIC 2007), 2007, pp. 328‑335.

[21] V. Basili, F. Shull, F. Lanubile, Building knowledge through families of experiments, Software Engineering, IEEE Transactions on 25 (1999) 456‑473.

[22] J. Bansiya, C. Davis, A hierarchical model for object-oriented design quality assessment, IEEE Transactions on Software Engineering 28 (1) (2002) 4‑17.

[23] B. Shim, S. Choue, S. Kim, S. Park, A design quality model for service-oriented architecture, in: Proceedings of the 2008 15th Asia-Pacific Software Engineering Conference, APSEC ’ 08, IEEE Computer Society, USA, 2008, pp. 403‑410.

[24] E. Yourdon, L. Constantine, Structured design: fundamentals of a discipline of computer program and systems design, Englewood Cliffs: Yourdon Press, 1979.

[25] U. Kulesza, C. Sant’ Anna, A. Garcia, R. Coelho, A. von Staa, C. J. P. de Lucena, Quantifying the effects of aspect-oriented programming: a maintenance study, in: 22nd IEEE International Conference on Software Maintenance (ICSM 2006), 24-27 September 2006, Philadelphia, Pennsylvania, USA, IEEE Computer Society, 2006, pp. 223‑233.

[26] D. Sharman, A. Yassine, Characterizing complex product architectures, Systems Engineering 7 (2004) 35‑60.

[27] M. Gysel, L. Kolbener, W. Giersche and O. Zimmermann, ” Service Cutter: A Systematic Approach to Service Decomposition ”, in ESOCC, Vienna, 2016, pp. 185-200.