From Monolith to Microservice: Measuring Architecture Maintainability

Muhammad Hafiz Hasan[1], Mohd. Hafeez Osman[2], Novia Indriaty Admodisastro[3], Muhamad Sufri Muhammad[4]
Dept. of Soft. Engineering and Information System-FSKTM, UPM, Serdang, Selangor, Malaysia[1, 2, 3, 4]

***Abstract*—The migration of monolithic applications to the cloud is a popular trend, with microservice architecture being a commonly targeted architectural pattern. The motivation behind this migration is often rooted in the challenges associated with maintaining legacy applications and the need to adapt to rapidly changing business requirements. To ensure that the migration to microservices is a sound decision for enhancing maintainability, designers must carefully consider the underlying factors driving this software architecture migration. This study proposes a set of software architecture metrics for evaluating the maintainability of microservice architectural designs for monolith to microservice architecture migration. These metrics consider various factors, such as coupling, complexity, cohesion, and size, which are crucial for ensuring that the software architecture remains maintainable in the long term. Drawing upon previous product quality models that share similar design properties with microservice, we have derived maintainability metrics that can help measure the quality of microservice architecture. In this work, we introduced our first version of structural metrics for measuring the maintainability quality of microservice architecture concerning its cloud-native characteristics. This work allows us to get early feedback on proposed metrics before a detailed evaluation. With these metrics, designers can measure their microservice architecture quality to fully leverage the benefits of the cloud environment, thus ensuring that the migration to microservice is a beneficial decision for enhancing the maintainability of their software architecture applications.**

Keywords*—*Monolith; cloud migration; software architecture; design quality; maintainability; quality metric

I. INTRODUCTION

In recent years, the demand for online applications and services has increased. Organizations and businesses with online applications perceive cloud platforms as a promising future for business strategy to remain competitive. For an organization with an existing legacy application that involves the organization's core business process, migrating the application to the cloud is more imminent to utilize cloud benefits and ensure business continuity. These applications often have monolithic software architecture, which does not consider modularity in its design principle [1], and the systems work in a silo [2]. In monolith architecture, developers develop the entire application as a single unit with a large codebase and tightly integrated components, increasing complexity and making it difficult to manage and scale [3]. These characteristics also affect its deployment approach when any minor changes to the application require a complete rebuild, leading to increased risk [4].

The motivation of the legacy application to cloud migration is to overcome the roadblocks and limitations of monolith applications [5] and to achieve cloud-native benefits such as improving application modifiability, maintainability, scalability, and deployability [6]. The cloud platform provides scalability for computing resources without worrying about the underlying infrastructure quickly and efficiently [7]. The organization also gains more flexibility and agility in responding to changing business ideas, thus increasing service innovation. However, not all migration strategies to the cloud provide mentioned benefits [8]. The *Lift-and-Shift* approach involves taking existing *as-is* on-premise applications and moving them to the cloud as a single service without architecture and design changes limiting its cloud scalability features [9].

In contrast, the microservice is a cloud architecture design pattern designed to provide better scalability and maintainability. In a microservice architecture, designers create sets of independent services that use API as their communication medium. Each microservice is responsible for specific business capabilities, strong component separation, and independent deployability execution [1], [5], [10]. Other fundamental properties of microservice architectural design are low coupling, high cohesion, and modularity [10] must be carefully considered by developers and designers [11], [12] during the design phase.

In the cloud migration context, migrated application quality should be equivalent to, or better than, legacy monolith applications. Software errors can stem devastating effects to financial loss, time delays, or even risks to life [13], [14]. Numerous frameworks for migrating from monolith to cloud have been introduced [1], [15]-[18], yet they still do not adequately address quality considerations after the migration [19]. Therefore, the migration did not accomplish its objective [5], thus introducing new product quality challenges such as application maintainability, security, reliability, and compatibility [8], [20].

From a technical standpoint, migrating monolith applications to the cloud allows for quick and effective implementation of essential software changes to meet current business needs. The relevant quality attribute is known as maintainability, which expresses the degree of effectiveness and efficiency with which an application can be changed, modified, or corrected to meet requirements [21]. Therefore, it is essential to ensure that migrated applications must be maintainable by developers to avoid accumulated waste and technical debt after the migration [22]-[24]. Thus, maintainability has become an essential quality feature [25].

However, empirical research on maintainability quality assurance remains a missing research area for microservice architecture [26]. To address this concern, the following research questions have been formulated to guide this study:

- RQ1: What are the existing structural quality metrics that relate to service-based architecture?
- RQ2: How do the existing structural metrics relate to cloud-native characteristics?
- RQ3: What are the feasible metrics for the maintainability quality model for microservice architecture?

This paper proposes the structural metrics for measuring microservice maintainability quality, focusing on microservice architecture migration. A multi-structural design metrics consisting of coupling, cohesion, complexity, and size form the basis of the maintainability measurement. These metrics help practitioners evaluate the architecture maintainability quality at the earlier migration phase to minimize post-migration technical debt, thus ensuring the achievable migration objective [6].

The remainder of this paper is organized as follows. Section II discusses related works with some comments on their limitations. Section III describes the research methodology for identifying existing service-based structural metrics. Section IV discusses how the existing structural metrics can be associated with cloud-native characteristics. Section V further discusses structural metrics from maintainability quality. Section VI briefly introduces our proposed maintainability quality model, followed by a discussion in Section VII. Finally, Section VIII concludes with a summary and outlook on potential follow-up research.

II. Related Work

The software quality model's development reflects the software architecture's progression. A robust quality model approach is necessary for measuring product architecture quality, regardless of the adopted software architecture. Thus, this work explores previous works on software quality models and monolith-to-microservice migration approaches to identify reliable and valid metrics to evaluate software design quality.

A. Software Quality Model Evolution

One of the first software product quality models introduced by [27] describes as a generic model that separates high-level quality attributes into tangible product quality properties. Due to rapidly changing and dynamic business requirements, different metrics have been proposed to meet software architecture evolutions.

In their work, Bansiya et al. [13] proposed an improved hierarchical design quality assessment model for object-oriented software architecture. This model, known as the Quality Model for Object-Oriented Design (QMOOD), builds upon Dromeys's generic quality model methodology. The QMOOD comprises four hierarchical levels: object-oriented design components, design metrics, design properties, and design quality attributes. Notably, the authors adopted most of the design quality attributes in QMOOD from the ISO/IEC 9126 standard. However, this model failed to serve simple, practical applicability as it assesses high-level design quality attributes. Hence the metrics are limited for object-oriented applications.

SOA Quality Model (SOAQM) is an extension of QMOOD to enhance architecture scalability through hierarchical abstraction and clear bottom-up relationship [28]. Bogner et al. [29] suggest that most metrics explicitly designed for SOA also apply to microservice architecture. The author then introduces the Maintainability Model for Microservices (MM4S) with five service properties: coupling, cohesion, granularity, complexity, and code maturation. Although this work is similar to ours, the authors did not consider migration scenarios, hence providing the mathematical formalization for proposed metrics.

Vera-Rivera et al. [30] conducted a systematic literature review on microservice architecture, explicitly focusing on the impact of microservice granularity on application quality. The authors employed a genetic algorithm to determine the optimal microservice granularity based on key factors such as coupling, cohesion, complexity, and resource usage. They integrated various metrics and quality attributes into their analysis with the development team's involvement for effort estimation based on user story artefacts.

Pulnil et al. [31] and Taibi et al. [32] proposed a microservice quality model that relies on microservice anti-patterns. The authors incorporated eleven microservice anti-patterns with the ISO/IEC 25010 standard as a benchmark for microservice quality attributes, while this work builds on top of microservice design principles [33]. Furthermore, the proposed quality assessment model is formulated depending on the weightage of the harmfulness level of the design properties exposing it to the biased decision by the designer.

B. Microservice

Microservice is an architecture design pattern that promises high maintainability, making it an exciting option for modernizing software during the cloud computing era [23]. Generally, microservices have been designed based on domain-driven functionality with limited business capabilities, strong component separation, and enabling automated deployment execution [1], [5], [10].

The developer and designer must carefully consider the fundamental properties of microservices, which include low coupling, high cohesion, scalability, independence, maintainability, modularity, and deployability [10]. These properties are closely related to architectural design [11], [12].

Chen et al. [34] proposed a monolith decomposition approach from a dataflow diagram viewpoint, while Fan et al. [24] suggested microservice candidate identification through domain-driven design analysis. Runtime behaviour information [35] strategy and Functionality-oriented Service Candidate Identification (FoSCI) framework introduced by [36] to identify service candidates using a search-based functional atom grouping algorithm based on recorded monolith's execution trace log. Combining the data usage with the dynamic analysis provides a better understanding of feature prioritization during the migration. The static approach based on source code [37], [38] and system structure [23] exhibits

structural information as decomposition reasoning. Meanwhile, the metric-based method, as demonstrated by previous works, uses structural properties such as coupling [37], [39], [40], service granularity [41], size [42], and cohesion [43].

Li et al. [44] introduced a method for identifying microservices based on the UML model from the source code as an input consisting of class and sequence diagrams for static and dynamic analysis. The authors then used a clustering approach to identify microservice candidates. The determination of clustering output quality relied on the utilization of functional requirements and deployment constraints. However, the authors did not consider microservice distributions to verify the architecture quality. At the same time, the ambiguity of language and contextual understanding prevents semantic-based analysis of the system requirements from guaranteeing the attainment of optimal solutions [45].

Our work extends and complements [28], [29] in the context of a structural quality model for the service-based application. Bingu et al. [28] did not consider maintainability quality in their model besides re-implementing weighted value by [13] in their quality attribute equation without necessary empirical justification. While Bogner et al. In [29] approach are beneficial for microservice maintainability design properties, the authors did not consider the migration scenario, thus limiting its applicability to the greenfield implementation. So, while the general approach from Bogner et al. is a sound foundation, this work established it to fit monolith to microservice migration scenario and enhanced it with practically collectable quality metrics for the architecture design consideration.

III. Methodology

In order to formulate the architectural maintainability quality model for monolith to microservice migration, this study followed a series of steps, as illustrated in Fig. 1. As this work highlights the monolith to microservice migration scenario, it started with a literature review process using trustable electronic journal databases for this research domain [46] consists of Scopus, SpringerLink, IEEE Xplore, and ScienceDirect to collect existing structural quality metrics.

Our initial investigation shows that the software quality model evolves laterally with software architecture advancement [29]. For this reason, pre-migration monolith object-oriented structural quality metrics [13], [47]-[49] and post-migration microservice architecture quality metrics [50]-[54] are included. Next, the object-oriented structural metrics are being mapped with microservice structural metrics based on their shared characteristics, as suggested by [47], [55]-[57]. These structural metrics help understand its relationships further, clarifying the evolution of the software quality model. Section IV explains this step's detailed approach and findings, thus answering RQ1.

The selection of structural maintainability quality metrics is guided by ISO/IEC 25010 – Software Product Quality. This product quality model comprises of hierarchical structure with maintainability quality attributes consisting of its sub-characteristics such as modularity, reusability, analyzability, modifiability, and testability [21]. Detailed methodology for this step is described in Section V, hence answering RQ2.

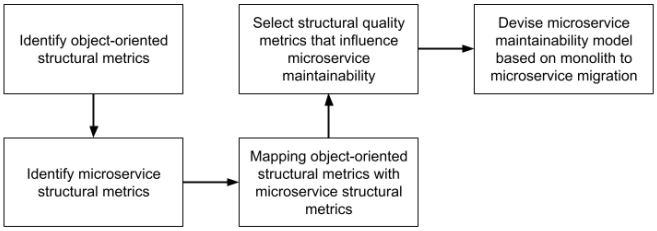

Fig. 1. The methodology of the monolith to microservice architectural maintainability quality model development.

Furthermore, software quality is still a vague and multifaced perception, which means different things to a diverse audience [13]. Therefore, referring to the procedure in Section VI, an empirical microservice maintainability quality model (*RQ3*) was devised based on selected quality metrics for monolith to microservice migration regulated by defined selection criteria in Table I. These selection criteria ensure that the selection process is within the research scope and objective.

TABLE I. Criteria for Metric Selection

Selection criteria
- Applicable to microservice architecture - Must be related to cloud-native design properties - Automatically collectable from the structural property and practically applicable in object-oriented to microservice migration scenario - Influence on ISO/IEC 25010 maintainability characteristic or sub-characteristics

IV. Service-based Structural Quality Metrics (RQ1)

Migrating monolith applications to the cloud involves transforming software architecture to exploit the distributed environment. Due to this factor, the designer must measure software structural quality during the early migration stage. It is cheaper and less time-consuming than evaluating it during operation time [43]. Developers can predict software quality by measuring structural attributes influencing software's external quality, such as maintainability. Besides, measuring structural metrics is more extensive than component-level metrics [43].

Based on the literature review, the most collective existing structural design property metrics for service-based architecture are summarised (as described in Table II) as the following:

A. Coupling

This design property is the most considered metric for measuring software architecture quality. The graph theory of design properties enables the direct analysis of coupling properties. Thirteen metrics have been proposed for measuring microservice coupling [35], [43], [44], [54], [59], [60], while fourteen metrics for service-oriented architecture (SOA) [28], [51], [58], [61]. Expressively, four of the proposed microservice metrics by Bogner et al. [29] were derived from SOA metrics [53] in the context of microservice architecture.

TABLE II. PUBLICATIONS ON QUALITY METRICS FOR A SERVICE-BASED ARCHITECTURE

Source	Authors	Architecture	Focus
[28]	Bingu et al.	Service-Oriented	Effectiveness, understandability, flexibility, reusability, and discoverability based on QMOOD metrics: coupling (1), cohesion (1), complexity (1), size (1), and service granularity (1)
[42]	Taibi et al.	Microservice	Coupling (2) and size (2)
[43]	Panichella et al.	Microservice	Maintainability based on coupling (1), size (1)
[44]	Li et al.	Microservice	Coupling (1) and cohesion (1)
[51]	Mohammed et al.	Service-Oriented	Coupling (3), cohesion (2), and complexity (3) metrics
[53]	Rud et al.	Service-Oriented	Applicability from OOP, CBSE, and Web Domains - complexity (3), reliability (4), and performance (4) metrics
[54]	Bogner et al.	Microservice	Maintainability – coupling (4), cohesion (4), complexity (3), size (1)
[58]	Hofmeister et al.	Service-Oriented	Complexity using coupling (2) metrics
[59]	Santos et al.	Microservice	Complexity consists of cohesion (2) and coupling (2) metrics
[60]	Vera-Rivera et al.	Microservice	Complexity based on cohesion (1), coupling (3)
[61]	Perepletchikov et al.	Service-Oriented	Maintainability by extending OO coupling (8) metrics

B. Size

Four metrics were proposed for microservice [35], [43], [54], and one for service-oriented [28] architecture. The reason for considering fewer size metrics for SOA than for microservice is that the two architectural styles operate at a different level of service granularity. This design property is essential to microservice architecture that promotes smaller atomic functionality than SOA in its design principle.

C. Cohesion

Highly cohesive architecture refers to the strength between operations of services. Cohesion in microservice is more meaningful than for SOA and object-oriented architecture to minimize external dependencies [62] that negatively influence product quality. Eight metrics for microservice cohesion [44], [54], [59], and three metrics for service-oriented [28], [51] were proposed in previous works.

D. Complexity

From the literature review, three metrics for microservices [54] and seven metrics for service-oriented architecture [28], [51], [53] were identified pertaining to this design property. Due to the common structural complexity characteristics for SOA and microservice, Bogner et al. applied three complexity metrics originally proposed for SOA to the microservice architecture.

V. QUALITY METRICS FOR CLOUD-NATIVE ARCHITECTURE (RQ2)

This work defines cloud-native architecture as a distributed, elastic, and horizontally scalable application composed of microservices [63], [64]. Thus, casting the existing legacy application to the cloud as a virtualized environment cannot be demanded as a valid cloud-native application [65]. Moreover, a microservice is a self-contained deployment unit designed according to cloud-focused design principles such as IDEALS [33] to gain full cloud benefits.

Regarding architecture quality, previous design property quality metrics from RQ1 are mapped with IDEALS design principles as in Table III to justify the selection of quality metrics for cloud-native architecture based on defined criteria in Table I.

Instead of designing microservice for a new greenfield scenario, this work focuses explicitly on migrating monolith applications to microservices. This process involves three main phases: pre-migration, migration, and post-migration [67]. Therefore, to answer the following research question RQ3, this work considers monolith quality metrics in the maintainability model, thus devising related metrics based on its common structural characteristics. Identifying related quality metrics during the early migration phase helps the designer to make an informed decision to propose quality microservice architecture design before moving to the cloud environment.

VI. MAINTAINABILITY QUALITY MODEL FOR MICROSERVICE (RQ3)

This paper distinguished existing service-based architecture quality metrics in RQ1. Collected metrics are then aligned with cloud-native characteristics (RQ2) to funnel the microservice architecture-related quality metrics findings.

From the application architectural perspective, a service-based design pattern can be perceived as a higher abstraction layer for object-oriented architecture [68], [69]. One could consider the interaction of methods in object-oriented programming as a form of class interaction in microservices at an abstract level. In contrast, object-oriented class interactions can be understandable at a higher abstraction level as interactions of clusters of classes known as microservice. With this insight, we propose a set of quality metrics to measure microservice structural maintainability described in Fig. 2.

A. Coupling

Coupling is the degree to which the elements in a design are connected or express the strength of interdependencies and interconnections of service with other services [70]. From the quality perspective, these metric impacts system quality, such as maintainability and testability. The findings indicate that incorporating structural coupling can be highly significant for developers who wish to monitor the decomposition quality of their services [43]. A high level of structural coupling resulted in more frequent bug occurrences and propagated changes within modules of systems. Therefore, a successful decomposition should produce minimized coupling between microservices and maximized cohesion. A small number of couplings positively influence product maintainability.

TABLE III. RELATED QUALITY METRICS FOR IDEALS DESIGN PRINCIPLES

Design Principle	Design Property	Related Quality Metrics
Interface segregation	Complexity	Total Response of Service (TRS) [29]
		Service Support for Transactions (SST) [29]
		Measure of Functional Abstraction (MFA) [13]
	Size	Number of Operations [28]
		Non-Extreme Distribution (NED) [55]
		Component Balance [54]
	Cohesion	Service Interface Data Cohesion (SIDC) [29]
		Service Interface Usage Cohesion (SIUC) [29]
		Total Service Interface Cohesion (TSIC) [29]
Deployability	Coupling	Service Interdependence in the System (SIY) [29]
		Coupling Between Microservice (CBM) [35]
		Structural Coupling [43]
		Coupling of Service (COS) [58]
	Complexity	Number of Versions per Service (NVS) [29]
		Number of Hierarchies (NOH) [13]
		Density of Aggregation (DOA) [58]
Event-driven	Coupling	Absolute Dependence of the Service (ADS) [29]
Availability	Coupling	Coupling Between Microservice (CBM) [35]
		Absolute Criticality of the Service (ACS) [29]
		Absolute Dependence of the Service (ADS) [29]
		Service Interdependence in the System (SIY) [29]
	Size	Numer of Operations [28]
	Cohesion	Service Interface Data Cohesion (SIDC) [29]
		Service Interface Usage Cohesion (SIUC) [29]
Loose coupling	Coupling	Service Interdependence in the System (SIY) [29]
		Absolute Importance of the Service (AIS) [29]
		Absolute Dependence of the Service (ADS) [29]
		Absolute Criticality of the Service (ACS) [29]
		Direct Class Coupling (DCC) [13]
		Coupling of Service (COS) [58]
		Structural Coupling [43]
		Coupling Between Microservice (CBM) [35]
Single responsibility	Cohesion	Activity Cohesion (AC) [66]
		Service Cohesion (SC) [66]
		Service Design Cohesion (SDC) [66]

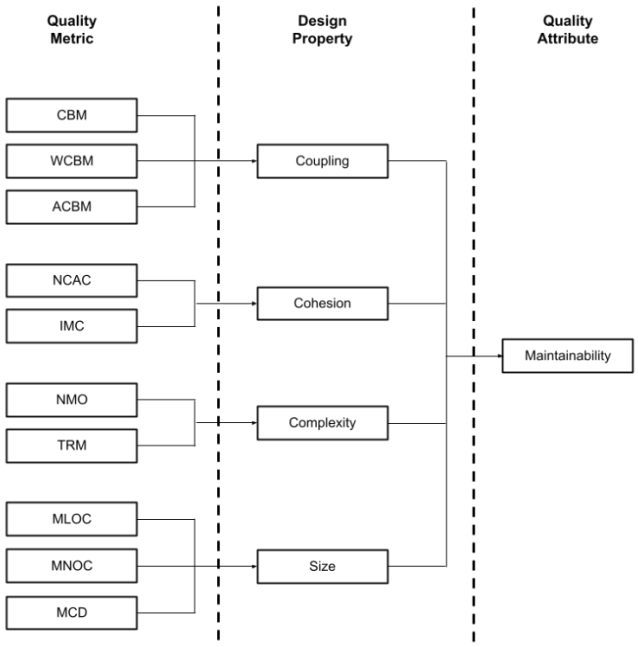

Fig. 2. The structural metric for microservice maintainability.

1) Coupling Between Microservice (CBM): CBM is the number of other microservices that the microservice coupled with [35], [37]. The inspiration for this coupling derives from the widely recognized Coupling Between Objects (CBO) metric proposed by [71]. CBO counts several types of interactions, including method calls, parameter types, references, and return types. However, CBM only counted for each unique class interaction, excluding its frequencies and bi-directional relationship. To calculate the relative CBM for each microservice as follows:

$$M=\{m_i \ldots m_n\} \text{ is a set of microservice} \quad (1)$$

$$R=\{r_i \ldots r_n\} \text{ is a set of microservice interaction} \quad (2)$$

$$\text{TotalofInteraction(r,m)=Number of occurrence r in m} \quad (3)$$

$$\text{CBM(r,m)}=\begin{cases}1 & \text{TotalofInteraction}>0\\ 0 & \text{Otherwise}\end{cases} \quad (4)$$

2) Weighted Coupling between Microservice (WCBM): WCBM is the frequency of other microservice that the microservice *m* is coupled with, i.e., the number of microservices where *m* has to interact once [29], [58]. As microservice holds clusters of classes, interactions with other microservices are more expensive than intra-microservice. Thus, frequencies for external microservice coupling need to be considered as interaction weightage. To formulate WCBM for each microservice as follows:

$$\text{WCBM(r,m)=TotalofInteraction(r,m)} \quad (5)$$

Absolute Coupling between Microservice (ACBM): The total number of bi-directional coupling frequencies between microservices where microservice *m1* interacts with microservice *m2* and *m2* also interacts with *m1* [29]. This metric helps represent inter-microservice dependencies and

how strongly they interact. ACBM for a microservice is represented as:

$$B=\{b_i \ldots b_n\} \text{ is a set of bi-directional interactions} \quad (6)$$

$$\text{TotalofBidirectional(b,m)=Number of occurrence b in m} \quad (7)$$

B. Cohesion

The degree to which the elements in a microservice design unit are logically related or connected. A high degree of cohesion is a sign of *"togetherness"* where classes within the microservice provide similar behavior to produce specific services or responsibilities. This characteristic matches the ideal cloud-native design pattern, where each microservice should contain a single responsibility for better maintainability.

1) Normalize Cohesion among Classes of Microservice (NCAM): The NCAM for Microservice is an adaptation of the object-oriented metric Cohesion Among Methods of Class (CAM) [13]. A CAM cluster directly influences the microservice *m* cohesion in migration to microservice architecture. Thus, *m* cohesion can be served with an average CAM for a particular *m*. To represent NCAM for a microservice as follows:

$$\text{NCAM}(m_i)= 1\text{-Avg}(\text{CAM}(m_i)) \quad (8)$$

2) Intra Microservice Coupling (IMC): IMC is the frequency of internal microservice coupling. In compliance with [13], microservice architecture obtains higher object-oriented design abstraction. Therefore, strong relatedness and interactions between classes within the microservice *m* indicate strong *m* cohesion. This strong relatedness demonstrates that each class in a microservice is working together to serve specific functions. IMC for a microservice is represented as follows:

$$C=\{c_i \ldots c_n\} \text{ is a set of classes in } m \quad (9)$$

$$P=\{p_i \ldots p_n\} \text{ is a set of class interaction} \quad (10)$$

$$TotalClassInteraction(p,m)=\text{Number of occurrence } p \text{ in } m \quad (11)$$

$$IMC(m_i)=TotalClassInteraction(p,m) \quad (12)$$

C. Complexity

Complexity is the degree of connectivity between elements of a microservice. This metric is also concerned with the dependencies, microservice operations, and the number of requests with other microservices. Santos et al. [59] derived that complexity architecture has a negative impact on the system's maintainability as it is difficult to make changes and decreases software understanding.

1) Number of Microservice Operations (NMO): NMO is the number of total operations for the microservices [28]. In migrating from monolith to microservice architecture, the total number of microservice operations encompasses all operations across all classes of the microservice. A high number of internal microservice operations may result in a complex design that requires maintenance. With an increase in the number of clusters, the structural complexity of the microservice also grows. NMO for a microservice is represented as follows:

$$O=\{o_i \ldots o_n\} \text{ is a set of operation in } c \quad (13)$$

$$TotalOperation(o,c)=\text{Sum of } o \text{ in } c \quad (14)$$

$$NMO(m_i)=TotalOperation(o,c) \text{ in } m \quad (15)$$

2) Total Response for Microservice (TRM): TRM is the total requests for operation O values of microservice [29]. This work employed an adapted version of the Response for Class (RFC) metric [71] from object-oriented design to the context of microservice design. Each microservice exposes its interface mi for other microservices, increasing its dependencies and negatively impacting its complexity. TRM for a microservice can be expressed as:

$$\text{TRM}(m_i)= \sum \text{RFC}(m_i) \quad (16)$$

D. Size

This metric measures the size of structural design elements consisting of the number of classes and microservice lines of code. The more extensive and granular the microservice, the more challenging it is to maintain due to the possibilities of multiple responsibilities to the microservice. Size metrics are crucial design attributes in software estimation before executing migration [72].

3) Microservice Line of Code (MLOC): The number of all non-empty, non-commented lines of the microservice body. This classic Line of Code (LOC) metric helps understand and overview microservice size. For a too-big microservice, there might be a sign that a technical debt problem exists. MLOC for a microservice is expressed as follows:

$$\text{MLOC}(m_i)= \sum (\text{NE \&\& NC})m_i \quad (17)$$

where *(NE & NC)* are non-empty NE and non-commented NC lines of codes within a microservice m_i.

4) Microservice Number of Classes (MNOC): The number of classes within a microservice [35] can measure how big the microservice is and identify if there are microservices that are too big. The number of classes should be minimized to keep microservice more independent of changes. MNOC for a microservice represents as:

$$\text{MNOC}(m_i)= \sum \text{C}(m_i) \quad (18)$$

where $C(m_i)$ are sets of classes within a microservice m_i.

5) Microservice Class Distribution (MCD): MCD is the number of class sizes in microservice candidates distribution, with a desire that microservice may not contain too many or too few classes. This structural metric is an adaptation of the Non-extreme Distribution (NED) metric by [55]. Therefore, we measure how evenly distributed the sizes for each generated microservice candidate are. Our work improvised this approach by using standard deviation to understand the average of scattered microservice clusters instead of the mean

value that is heavily influenced by outliers value in determining the bound of non-extreme value for the number of classes within microservices. For better interpretability, measuring *1-MCD*, with a lower value demonstrates a better microservice distribution. To express MCD for a microservice as follows:

$$\mathrm{MCD}(m_i)=\frac{\sum_{n=1,\ n\ \text{not extreme}}^{N} c_n}{|C_i|} \quad (19)$$

where c_n is the number of classes in microservice m_i, and C_i is the set of classes of microservice m_i. n is not extreme if its size is within the bounds of *{mean of classes for all microservices ±std deviation}* . This work measures its normalized value with *1-MCD(m_i)* for better interpretability, and lower values are recommended.

VII. Discussion

Measuring architecture quality for migrated monolith applications to microservice is crucial to ensure migration to the cloud achieves the migration objective. Despite various migrations approaches, less attention was given to the post-migration architecture quality. This work starts by identifying existing structural metrics for measuring service-based architecture quality. Our first contribution in this work is reporting the most applicable design property metrics for service-based architecture, including its influence on architecture quality (Section IV). This design property catalogue is a reference for other researchers in understanding how software architecture evolution influences the characteristics of its design properties.

Another state-of-the-art contribution of this paper is that it maps service-based quality metrics with the cloud-native design principles [33]. In contrast, previous quality metrics [13], [28] focus on the structural characteristics without considering architecture quality. From the structural quality perspective, this mapping is essential to ensure the designated cloud architecture pattern benefits from the cloud environment.

The proposed maintainability quality model for microservice architecture is the main contribution to this work. Ten structural quality metrics for measuring microservice architecture maintainability quality enable software designers and developers to assess the designed microservice candidate's quality before executing the migration, thus minimizing post-migration quality concerns and ensuring achievable migration objectives [6], [77], [78]. Despite relying on single design property in measuring structural maintainability quality, this approach promotes multiple design properties to give better accuracy and consistent result [79].

While this work pointed out several quality metrics related to microservice design properties, this work is still exposed to construct validity as we may not be able to cover all design properties [47] that influence monolith-to-microservice migration architecture quality [73]. However, this work covers various design properties than previous work [61], [74], [75] on architectural maintainability quality. Our approach is based on ISO/IEC 25010 [21] and additional structural design properties that influence maintainability quality measurement.

Regarding external validity, some of the metrics devised from existing work [13], [28], [29], [35], [55], [58], [71] are based on shared structural characteristics. This work ensured the soundness of the selected metrics by exclusively considering reliable peer-reviewed sources and established authors. Hence, our selection is adequate to initiate an exploration for microservice maintainability quality when migrating from monolith architecture.

This method relies on the monolith application as the source before the migration execution. This work focuses on migration instead of greenfield implementation. Thus, the selection of the quality metrics is heavily influenced and devised by the existing application architecture characteristics. Even though other works proposed various quality metrics for measuring product quality, the complexity of the metrics hindered the applicability of the approach by the industries [76]. As a result, our proposed quality metrics are more practical for industrial practice.

The limitation of this paper is that we did not adopt a more rigorous methodology for this paper, such as conducting a systematic or multivocal literature review. These procedures could have offered a more solid empirical basis for selecting publications. Moreover, a more rigorous process could have been employed to identify the metric candidates presented in this study to minimize any potential subjective bias. Additionally, certain digital libraries were excluded from the search process due to time limitations.

VIII. Summary and Conclusion

To measure architecture maintainability quality when migrating monolith applications to microservice architecture, this paper proposed a set of metrics related to coupling, cohesion, complexity, and size design property. These metrics were derived from cloud-native architectural design principles to utilize cloud benefits. The proposed metrics allow migration designers and developers to measure software architecture maintainability quality for microservice during design time. Additionally, this work included the mathematical formalization of the proposed metrics. Moreover, applying multiple design properties for measuring microservice architecture maintainability quality is an adaptation of state-of-the-art in this research domain.

As part of this work evaluation process, we intend to assess the metrics through case studies and extend their application to real-world industrial projects to evaluate their efficacy. These forthcoming efforts encompass the development of a tooling approach aimed at promoting a structured and rational migration process and providing practical illustrations of metric implementation throughout the migration process to evaluate the architecture quality of microservice candidates.

References

[1] A. Megargel, V. Shankararaman, and D. K. Walker, "Migrating from Monoliths to Cloud-Based Microservices: A Banking Industry Example," no. August, pp. 85–108, 2020, doi: 10.1007/978-3-030-33624-0_4.

[2] A. S. Ganesan and T. Chithralekha, "A Survey on Survey of Migration of Legacy Systems," *ACM Int. Conf. Proceeding Ser.*, vol. 25-26-Augu, 2016, doi: 10.1145/2980258.2980409.

[3] F. De Angelis and A. Polini, "Evaluation of cloud portability of legacy applications," *Proc. - 11th IEEE/ACM Int. Conf. Util. Cloud Comput. Companion, UCC Companion 2018*, pp. 232–237, 2019, doi: 10.1109/UCC-Companion.2018.00061.

[4] S. A. Maisto, B. Di Martino, and S. Nacchia, "From Monolith to Cloud Architecture Using Semi-automated Microservices Modernization," *Lect. Notes Networks Syst.*, vol. 96, pp. 638–647, 2020, doi: 10.1007/978-3-030-33509-0_60.

[5] H. Knoche and W. Hasselbring, "Using Microservices for Legacy Software Modernization," *IEEE Softw.*, vol. 35, no. 3, pp. 44–49, 2018.

[6] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, *Cloud Computing Patterns*. Vienna: Springer Vienna, 2014.

[7] M. Armbrust *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4. pp. 50–58, Apr. 2010, doi: 10.1145/1721654.1721672.

[8] M. Shuaib, A. Samad, S. Alam, and S. T. Siddiqui, "Why Adopting Cloud Is Still a Challenge?—A Review on Issues and Challenges for Cloud Migration in Organizations," *Adv. Intell. Syst. Comput.*, vol. 904, pp. 387–399, 2019, doi: 10.1007/978-981-13-5934-7_35.

[9] A. K. Kalia *et al.*, "Mono2Micro: An AI-based toolchain for evolving monolithic enterprise applications to a microservice architecture," *ESEC/FSE 2020 - Proc. 28th ACM Jt. Meet. Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, no. December, pp. 1606–1610, 2020, doi: 10.1145/3368089.3417933.

[10] J. Lewis and M. Fowler, "Microservices," 2014. https://www.martinfowler.com/articles/microservices.html (accessed Mar. 30, 2022).

[11] D. Taibi, V. Lenarduzzi, and C. Pahl, "Architectural patterns for microservices: A systematic mapping study," *CLOSER 2018 - Proc. 8th Int. Conf. Cloud Comput. Serv. Sci.*, vol. 2018-Janua, pp. 221–232, 2018, doi: 10.5220/0006798302210232.

[12] M. Grieger, M. Fazal-Baqaie, G. Engels, and M. Klenke, "Concept-based engineering of situation-specific migration methods," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2016, vol. 9679, pp. 199–214, doi: 10.1007/978-3-319-35122-3_14.

[13] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Trans. Softw. Eng.*, vol. 28, no. 1, pp. 4–17, 2002, doi: 10.1109/32.979986.

[14] M. Zhivich and R. K. Cunningham, "The real cost of software errors," *IEEE Secur. Priv.*, vol. 7, no. 2, pp. 87–90, 2009, doi: 10.1109/MSP.2009.56.

[15] A. Selmadji, A. D. Seriai, H. L. Bouziane, R. Oumarou Mahamane, P. Zaragoza, and C. Dony, "From monolithic architecture style to microservice one based on a semi-automatic approach," *Proc. - IEEE 17th Int. Conf. Softw. Archit. ICSA 2020*, no. Section III, pp. 157–168, 2020, doi: 10.1109/ICSA47634.2020.00023.

[16] B. Althani, S. Khaddaj, and B. Makoond, "A Quality Assured Framework for Cloud Adaptation and Modernization of Enterprise Applications," *Proc. - 19th IEEE Int. Conf. Comput. Sci. Eng. 14th IEEE Int. Conf. Embed. Ubiquitous Comput. 15th Int. Symp. Distrib. Comput. Appl. to Business, Engi*, pp. 634–637, 2017, doi: 10.1109/CSE-EUC-DCABES.2016.251.

[17] K. Sabiri, F. Benabbou, and A. Khammal, "Model driven modernization and cloud migration framework with smart use case," *Lect. Notes Networks Syst.*, vol. 37, pp. 17–27, 2018, doi: 10.1007/978-3-319-74500-8_2.

[18] I. Pigazzini, F. Arcelli Fontana, and A. Maggioni, "Tool support for the migration to microservice architecture: An industrial case study," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11681 LNCS, pp. 247–263, 2019, doi: 10.1007/978-3-030-29983-5_17.

[19] M. H. Hasan, M. H. Osman, N. I. Admodisastro, and M. S. Muhammad, "Legacy systems to cloud migration: A review from the architectural perspective," *J. Syst. Softw.*, p. 111702, Apr. 2023, doi: 10.1016/j.jss.2023.111702.

[20] A. Patel, N. Shah, D. Ramoliya, and A. Nayak, "A detailed review of Cloud Security: Issues, Threats Attacks," in *Proceedings of the 4th International Conference on Electronics, Communication and Aerospace Technology, ICECA 2020*, Nov. 2020, pp. 758–764, doi: 10.1109/ICECA49313.2020.9297572.

[21] ISO/IEC, "ISO/IEC 25010:2010, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models," vol. 1991. 2010.

[22] V. Lenarduzzi, F. Lomio, N. Saarimäki, and D. Taibi, "Does migrating a monolithic system to microservices decrease the technical debt?," *J. Syst. Softw.*, vol. 169, p. 110710, 2020, doi: 10.1016/j.jss.2020.110710.

[23] D. Taibi, V. Lenarduzzi, and C. Pahl, "Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation," *IEEE Cloud Comput.*, vol. 4, no. 5, pp. 22–32, 2017, doi: 10.1109/MCC.2017.4250931.

[24] C. Y. Fan and S. P. Ma, "Migrating Monolithic Mobile Application to Microservice Architecture: An Experiment Report," *Proc. - 2017 IEEE 6th Int. Conf. AI Mob. Serv. AIMS 2017*, pp. 109–112, 2017, doi: 10.1109/AIMS.2017.23.

[25] T. Coulin, M. Detante, W. Mouchère, and F. Petrillo, "Software Architecture Metrics: a literature review," 2019, [Online]. Available: http://arxiv.org/abs/1901.09050.

[26] Y. Li, C. Z. Wang, Y. C. Li, J. Su, and C. H. Chen, "Granularity Decision of Microservice Splitting in View of Maintainability and Its Innovation Effect in Government Data Sharing," *Discret. Dyn. Nat. Soc.*, vol. 2020, no. 39, 2020, doi: 10.1155/2020/1057902.

[27] R. G. Dromey, "A model for software product quality," *IEEE Trans. Softw. Eng.*, vol. 21, no. 2, pp. 146–162, 1995, doi: 10.1109/32.345830.

[28] S. Bingu, C. Siho, K. Suntae, and P. Sooyong, "A design quality model for service-oriented architecture," *Neonatal, Paediatr. Child Heal. Nurs.*, pp. 403–410, 2008, doi: 10.1109/APSEC.2008.32.

[29] J. Bogner, S. Wagner, and A. Zimmermann, "Automatically measuring the maintainability of service- and microservice-based systems - a literature review," *ACM Int. Conf. Proceeding Ser.*, vol. Part F1319, no. October, pp. 107–115, 2017, doi: 10.1145/3143434.3143443.

[30] F. H. Vera-Rivera, C. Gaona, and H. Astudillo, "Defining and measuring microservice granularity—a literature overview," *PeerJ Comput. Sci.*, vol. 7, p. e695, 2021, doi: 10.7717/peerj-cs.695.

[31] S. Pulnil and T. Senivongse, "A Microservices Quality Model Based on Microservices Anti-patterns," *2022 19th Int. Jt. Conf. Comput. Sci. Softw. Eng. JCSSE 2022*, 2022, doi: 10.1109/JCSSE54890.2022.9836297.

[32] D. Taibi, V. Lenarduzzi, and C. Pahl, "Microservices Anti-patterns: A Taxonomy," in *Microservices*, Cham: Springer International Publishing, 2020, pp. 111–128.

[33] P. Merson, "Principles for Microservice Design: Think IDEALS, Rather than SOLID," *InfoQ*, pp. 1–11, Sep. 2020, Accessed: Jun. 09, 2021. [Online]. Available: https://www.infoq.com/articles/microservices-design-ideals/.

[34] R. Chen, S. Li, and Z. Li, "From Monolith to Microservices: A Dataflow-Driven Approach," *Proc. - Asia-Pacific Softw. Eng. Conf. APSEC*, vol. 2017-Decem, pp. 466–475, 2017, doi: 10.1109/APSEC.2017.53.

[35] D. Taibi and K. Systä, "From monolithic systems to microservices: A decomposition framework based on process mining," *CLOSER 2019 - Proc. 9th Int. Conf. Cloud Comput. Serv. Sci.*, no. Closer, pp. 153–164, 2019, doi: 10.5220/0007755901530164.

[36] W. Jin, T. Liu, Y. Cai, R. Kazman, R. Mo, and Q. Zheng, "Service Candidate Identification from Monolithic Systems based on Execution Traces," *IEEE Trans. Softw. Eng.*, pp. 1–1, 2019, doi: 10.1109/tse.2019.2910531.

[37] S. Eski and F. Buzluca, "An automatic extraction approach - Transition to microservices architecture from monolithic application," *ACM Int. Conf. Proceeding Ser.*, vol. Part F1477, pp. 1–6, 2018, doi: 10.1145/3234152.3234195.

[38] J. Kazanavičius and D. Mažeika, "Analysis of legacy monolithic software decomposition into microservices," in *CEUR Workshop Proceedings*, 2020, vol. 2620, pp. 25–32.

[39] M. Gysel, L. Kölbener, W. Giersche, and O. Zimmermann, "Service cutter: A systematic approach to service decomposition," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes*

Bioinformatics), vol. 9846 LNCS, pp. 185–200, 2016, doi: 10.1007/978-3-319-44482-6_12.

[40] G. Mazlami, J. Cito, and P. Leitner, "Extraction of Microservices from Monolithic Software Architectures," *Proc. - 2017 IEEE 24th Int. Conf. Web Serv. ICWS 2017*, pp. 524–531, 2017, doi: 10.1109/ICWS.2017.61.

[41] A. A. C. De Alwis, A. Barros, A. Polyvyanyy, and C. Fidge, "Function-splitting heuristics for discovery of microservices in enterprise systems," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11236 LNCS, pp. 37–53, 2018, doi: 10.1007/978-3-030-03596-9_3.

[42] D. Taibi and K. Systä, "A Decomposition and Metric-Based Evaluation Framework for Microservices," *Commun. Comput. Inf. Sci.*, vol. 1218 CCIS, pp. 133–149, 2020, doi: 10.1007/978-3-030-49432-2_7.

[43] S. Panichella, M. Rahman, and D. Taibi, "Structural Coupling for Microservices," pp. 280–287, 2021, doi: 10.5220/0010481902800287.

[44] J. Li, H. Xu, X. Xu, and Z. Wang, "A Novel Method for Identifying Microservices by Considering Quality Expectations and Deployment Constraints," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 32, no. 3, pp. 417–437, 2022, doi: 10.1142/S021819402250019X.

[45] S. A. Salloum, R. Khan, and K. Shaalan, "A Survey of Semantic Analysis Approaches," in *Advances in Intelligent Systems and Computing*, vol. 1153 AISC, 2020, pp. 61–70.

[46] A. Cavacini, "What is the best database for computer science journal articles?," *Scientometrics*, vol. 102, no. 3, pp. 2059–2071, 2015, doi: 10.1007/s11192-014-1506-1.

[47] L. Ardito, R. Coppola, L. Barbato, and D. Verga, "A Tool-Based Perspective on Software Code Maintainability Metrics: A Systematic Literature Review," *Sci. Program.*, vol. 2020, 2020, doi: 10.1155/2020/8840389.

[48] R. Bharathi and R. Selvarani, "A framework for the estimation of OO software reliability using design complexity metrics," *Int. Conf. Trends Autom. Commun. Comput. Technol. I-TACT 2015*, 2016, doi: 10.1109/ITACT.2015.7492648.

[49] S. M. Yacoub, H. H. Ammar, and T. Robinson, "Dynamic metrics for object oriented designs," *Int. Softw. Metrics Symp. Proc.*, pp. 50–61, 1999, doi: 10.1109/metric.1999.809725.

[50] K. Qian, J. Liu, and F. Tsui, "Decoupling metrics for services composition," *Proc. - 5th IEEE/ACIS Int. Conf. Comput. Info. Sci., ICIS 2006. conjunction with 1st IEEE/ACIS, Int. Work. Component-Based Softw. Eng., Softw. Arch. Reuse, COMSAR 2006*, vol. 2006, pp. 44–47, 2006, doi: 10.1109/ICIS-COMSAR.2006.30.

[51] A. A. Mohammed Elhag and R. Mohamad, "Metrics for evaluating the quality of service-oriented design," *2014 8th Malaysian Softw. Eng. Conf. MySEC 2014*, no. September, pp. 154–159, 2014, doi: 10.1109/MySec.2014.6986006.

[52] A. K. Kalia, J. Xiao, R. Krishna, S. Sinha, M. Vukovic, and D. Banerjee, "Mono2Micro: a practical and effective tool for decomposing monolithic Java applications to microservices," no. January, pp. 1214–1224, 2021, doi: 10.1145/3468264.3473915.

[53] D. Rud and A. Schmietendorf, "Product metrics for service-oriented infrastructures," *IWSM/MetriKon*, no. May, 2006, [Online]. Available: http://www.cs.uni-magdeburg.de/~rud/papers/Rud-07.pdf.

[54] J. Bogner, S. Wagner, and A. Zimmermann, "Towards a practical maintainability quality model for service and microservice-based systems," *ACM Int. Conf. Proceeding Ser.*, vol. Part F1305, pp. 195–198, 2017, doi: 10.1145/3129790.3129816.

[55] U. Desai, S. Bandyopadhyay, and S. Tamilselvam, "Graph Neural Network to Dilute Outliers for Refactoring Monolith Application," 2021, [Online]. Available: http://arxiv.org/abs/2102.03827.

[56] A. Prajapati, A. Parashar, and J. K. Chhabra, "Restructuring Object-Oriented Software Systems Using Various Aspects of Class Information," *Arab. J. Sci. Eng.*, vol. 45, no. 12, pp. 10433–10457, 2020, doi: 10.1007/s13369-020-04785-z.

[57] T. Engel, M. Langermeier, B. Bauer, and A. Hofmann, "Evaluation of Microservice Architectures: A Metric and Tool-Based Approach," vol. 2, Springer International Publishing AG, 2018, pp. 74–89.

[58] H. Hofmeister and G. Wirtz, "Supporting service-oriented design with metrics," *Proc. - 12th IEEE Int. Enterp. Distrib. Object Comput. Conf. EDOC 2008*, pp. 191–200, 2008, doi: 10.1109/EDOC.2008.13.

[59] N. Santos and A. Rito Silva, "A complexity metric for microservices architecture migration," *Proc. - IEEE 17th Int. Conf. Softw. Archit. ICSA 2020*, pp. 169–178, 2020, doi: 10.1109/ICSA47634.2020.00024.

[60] F. H. Vera-Rivera, E. G. Puerto-Cuadros, H. Astudillo, and C. M. Gaona-Cuevas, "Microservices Backlog - A Model of Granularity Specification and Microservice Identification," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 12409 LNCS, pp. 85–102, 2020, doi: 10.1007/978-3-030-59592-0_6.

[61] M. Perepletchikov, C. Ryan, K. Frampton, and Z. Tari, "Coupling metrics for predicting maintainability in service-oriented designs," *Proc. Aust. Softw. Eng. Conf. ASWEC*, pp. 329–338, 2007, doi: https://doi.org/10.1109/ECBS.1998.10027.

[62] S. Kramer and H. Kaindl, "Coupling and cohesion metrics for knowledge-based systems using frames and rules," *ACM Trans. Softw. Eng. Methodol.*, vol. 13, no. 3, pp. 332–358, 2004, doi: 10.1145/1027092.1027094.

[63] N. Kratzke and P. C. Quint, "Understanding cloud-native applications after 10 years of cloud computing - A systematic mapping study," *J. Syst. Softw.*, vol. 126, pp. 1–16, 2017, doi: 10.1016/j.jss.2017.01.001.

[64] R. Lichtenthäler, M. Prechtl, C. Schwille, T. Schwartz, P. Cezanne, and G. Wirtz, "Requirements for a model-driven cloud-native migration of monolithic web-based applications," *Software-Intensive Cyber-Physical Syst.*, vol. 35, no. 1–2, pp. 89–100, 2020, doi: 10.1007/s00450-019-00414-9.

[65] D. Bajaj, U. Bharti, A. Goel, and S. C. Gupta, "Partial Migration for Re-architecting a Cloud Native Monolithic Application into Microservices and FaaS," in *Communications in Computer and Information Science*, 2020, vol. 1170, pp. 111–124, doi: 10.1007/978-981-15-9671-1_9.

[66] M. Daghaghzadeh, A. B. Dastjerdi, and H. Daghaghzadeh, "A Metric for Measuring Degree of Service Cohesion in Service Oriented Designs," *Int. J. Comput. Sci. Issues*, vol. 8, no. 5, pp. 83–89, 2011.

[67] B. Althani and S. Khaddaj, "The Applicability of System Migration Life Cycle (SMLC) Framework," *Proc. - 2017 16th Int. Symp. Distrib. Comput. Appl. to Business, Eng. Sci. DCABES 2017*, vol. 2018-Septe, pp. 141–144, 2017, doi: 10.1109/DCABES.2017.38.

[68] M. Perepletchikov, C. Ryan, and K. Frampton, "Comparing the Impact of Service-Oriented and Object-Oriented Paradigms on the Structural Properties of Software," 2005, pp. 431–441.

[69] Y. I. Mansour and S. H. Mustafa, "Assessing Internal Software Quality Attributes of the Object-Oriented and Service-Oriented Software Development Paradigms: A Comparative Study," *J. Softw. Eng. Appl.*, vol. 04, no. 04, pp. 244–252, 2011, doi: 10.4236/jsea.2011.44027.

[70] M. Savić, M. Ivanović, and M. Radovanović, "Analysis of high structural class coupling in object-oriented software systems," *Computing*, vol. 99, no. 11, pp. 1055–1079, 2017, doi: 10.1007/s00607-017-0549-6.

[71] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Trans. Softw. Eng.*, vol. 20, no. 6, pp. 476–493, 1994, doi: 10.1109/32.295895.

[72] S. Wanjala Munialo, G. Muchiri Muketha, and K. Kabeti Omieno, "Size Metrics for Service-Oriented Architecture," *Int. J. Softw. Eng. Appl.*, vol. 10, no. 2, pp. 67–83, 2019, doi: 10.5121/ijsea.2019.10206.

[73] J. Estdale and E. Georgiadou, "Applying the ISO/IEC 25010 Quality Models to Software Product," *Commun. Comput. Inf. Sci.*, vol. 896, no. January, pp. 492–503, 2018, doi: 10.1007/978-3-319-97925-0_42.

[74] M. Perepletchikov, C. Ryan, and K. Frampton, "Cohesion metrics for predicting maintainability of service-oriented software," *Proc. - Int. Conf. Qual. Softw.*, no. Qsic, pp. 328–335, 2007, doi: 10.1109/QSIC.2007.4385516.

[75] J. Ludwig, S. Xu, and F. Webber, "Static software metrics for reliability and maintainability," *Proc. - Int. Conf. Softw. Eng.*, pp. 53–54, 2018, doi: 10.1145/3194164.3194184.

[76] J. A. Valdivia, A. Lora-González, X. Limón, K. Cortes-Verdin, and J. O. Ocharán-Hernández, "Patterns Related to Microservice Architecture: a

Multivocal Literature Review," *Program. Comput. Softw.*, vol. 46, no. 8, pp. 594–608, 2020, doi: 10.1134/S0361768820080253.

[77] J. Kazanavicius and D. Mazeika, "Migrating Legacy Software to Microservices Architecture," *2019 Open Conf. Electr. Electron. Inf. Sci. eStream 2019 - Proc.*, 2019, doi: 10.1109/eStream.2019.8732170.

[78] R. Khadka *et al.*, "Does software modernization deliver what it aimed for? A post modernization analysis of five software modernization case studies," in *2015 IEEE 31st International Conference on Software Maintenance and Evolution, ICSME 2015 - Proceedings*, Sep. 2015, pp. 477–486, doi: 10.1109/ICSM.2015.7332499.

[79] A. Mishra, R. Shatnawi, C. Catal, and A. Akbulut, "Techniques for calculating software product metrics threshold values: A systematic mapping study," *Appl. Sci.*, vol. 11, no. 23, 2021, doi: 10.3390/app112311377.