An Extensible Data-Driven Approach for Evaluating the Quality of Microservice Architectures

Mario Cardarelli
University of L'Aquila
L'Aquila, Italy
mario.cardarelli@gmail.com

Ludovico Iovino
Gran Sasso Science Institute
L'Aquila, Italy
ludovico.iovino@gssi.it

Paolo Di Francesco
Gran Sasso Science Institute
L'Aquila, Italy
paolo.difrancesco@gssi.it

Amleto Di Salle
University of L'Aquila
L'Aquila, Italy
amleto.disalle@univaq.it

Ivano Malavolta
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
i.malavolta@vu.nl

Patricia Lago
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
p.lago@vu.nl

ABSTRACT

Microservice architecture (MSA) is defined as an architectural style where the software system is developed as a suite of small services, each running in its own process and communicating with lightweight mechanisms. The benefits of MSA are many, ranging from an increase in development productivity, to better business-IT alignment, agility, scalability, and technology flexibility. The high degree of microservices distribution and decoupling is, however, imposing a number of relevant challenges from an architectural perspective. In this context, measuring, controlling, and keeping a satisfactory level of quality of the system architecture is of paramount importance.

In this paper we propose an approach for the specification, aggregation, and evaluation of software quality attributes for the architecture of microservice-based systems. The proposed approach allows developers to (i) produce architecture models of the system, either manually or automatically via recovering techniques, (ii) contribute to an ecosystem of well-specified and automatically-computable software quality attributes for MSAs, and (iii) continuously measure and evaluate the architecture of their systems by (re-)using the software quality attributes defined in the ecosystem. The approach is implemented by using Model-Driven Engineering techniques.

The current implementation of the approach has been validated by assessing the maintainability of a third-party, publicly available benchmark system.

CCS CONCEPTS

• **Applied computing → Service-oriented architectures**; • **Software and its engineering → Domain specific languages**; **Software maintenance tools**;

KEYWORDS

Microservices, Software quality, Model–Driven, Architecture recovery

ACM Reference Format:
Mario Cardarelli, Ludovico Iovino, Paolo Di Francesco, Amleto Di Salle, Ivano Malavolta, and Patricia Lago. 2019. An Extensible Data-Driven Approach for Evaluating the Quality of Microservice Architectures. In *The 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19), April 8–12, 2019, Limassol, Cyprus.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3297280.3297400

SAC '19, April 8–12, 2019, Limassol, Cyprus

ACM ISBN 978-1-4503-5933-7/19/04…$15.00
https://doi.org/10.1145/3297280.3297400

1 INTRODUCTION

Lewis and Fowler define the microservice architecture (MSA) as an architectural style in which a system is developed as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often HTTP-based REST APIs [11]. The MSA architectural style puts emphasis on the design and development of highly maintainable and scalable software, where a large system is arranged as a set of independent lightweight services [21]. The benefits of the MSA style are various, e.g., increase of development productivity, better business-IT alignment, agility, scalability, and technology flexibility [29].

However, the MSA architectural style is also plagued by numerous challenges, such as efficient service discovery over complex networks, security assurance, performance optimization, data sharing [1]. One of the root causes of the challenges related to MSA is that MSA shifts the complexity of the system from inside a monolith to the interdependencies among the (potentially hundreds of) microservices [29]. Other relevant reported challenges of the MSA style are related to the adoption of a different mindset for developers and ineffective knowledge communication [12]. In this context, objectively measuring, controlling, and keeping at an acceptable level the quality of the system is of paramount importance for the success of the system. As of today, there is a plethora of frameworks and tools for assessing different aspects of an MSA, such as dynatrace[1]. However, each of them has its own internal quality model, level of abstraction, and system representation. This situation makes the assessment of the overall quality of an MSA terribly difficult and inefficient as developers needs to be familiar with each measurement framework and its related notation, underlying data

[1] https://www.dynatrace.com

model, and key performance indicators in order to get actionable insights about the system.

This paper proposes MicroQuality, a novel approach for the quality evaluation of MSAs. MicroQuality is based on three main principles: abstraction, reusable quality attributes, and continuous quality assessment. For what concerns **abstraction**, MicroQuality is centered around an *architecture model* of the system, allowing developers to focus explicitly on architecture-related concerns, instead of imposing the high cognitive burden due to the potentially highly heterogeneous technologies, the internal business logic of the involved microservices, their low-level details, etc. The architecture model of the system can be specified either manually or automatically via pre-existing architecture recovery techniques for MSAs (e.g., [2, 14, 15]). MicroQuality provides **reusable quality attributes** via an ecosystem of well-specified and language-independent software quality attributes for MSAs; the proposed ecosystem fosters reuse by allowing developers to (i) define their own quality attributes, (ii) share them across different systems and organizations, and (iii) reuse quality attributes defined by other developers using the ecosystem. Technically, this is achieved by taking advantage of Model Driven Engineering techniques (i.e., model transformation and model weaving), which allow MicroQuality to be independent from both (i) the modeling language used for representing the architecture of the system, and (ii) the quality attributes living in the ecosystem. Each specific quality attribute in the MicroQuality ecosystem is paired with dedicated software components called *metric providers*, whose responsibility is to compute the value of the quality attribute for the currently analyzed system. Metric providers allow MicroQuality to automatically compute the values related to the quality attributes, thus opening for the **continuous quality assessment** of MSAs, where (i) the (automatically recovered) architecture models of the system can be measured by MicroQuality at any point in time and (ii) developers get data-driven and timely insights about the architecture of the system, while (re-)using the quality attributes defined within the whole ecosystem.

The main contributions[2] of this study are the following:

- an approach for the specification, aggregation, and evaluation of software quality attributes for microservice-based systems at the architectural level;
- a Java-based implementation of the proposed approach in the context of Docker-based systems;
- the application of the proposed approach to a third-party open-source benchmark system called Acme Air.

The remainder of the paper is organised as follows. Section 2 provides background information, Section 3 gives an overview of the proposed approach, whereas its main phases are detailed in Sections 4, 5, and 6. In Section 7 we discuss the scope, benefits and limitations of the approach. Section 8 presents a comparison with related work, and Section 9 closes the paper and discusses future work.

[2]A repository containing the tool and the examples shown in this paper are available at https://github.com/gssi/QualityMicroART

2 BACKGROUND

In this section we discuss maintainability as an example of quality attribute for MSAs (Section 2.1) and we describe the running example used throughout the paper (Section 2.2).

2.1 Example – Maintainability of MSAs

As a way to exemplify how measurable quality attributes can be aggregated in order to predicate on the overall quality of a microservice-based system, in this section we describe *maintainability* as an aggregation of *coupling*, *cohesion*, and *complexity*[3].

Maintainability can be defined as the level of effort required for modifying a software product [10]. Modifications can include corrections, improvements, and adaptation of the software due to, e.g., changes in the environment or new requirements [22].

In the literature, numerous studies attempt to assess maintainability from different perspectives and with different purposes. Oman et al. [20] provide a hierarchical structure of measurable attributes that impact maintainability. The selected attributes are aggregated in order to provide a final indicator for maintainability. Among others, the attributes of coupling, cohesion and complexity are the key indicators for the overall maintainability of the system [20].

Perepletchikov et al. [22] review the literature about coupling in software systems with the aim to find a set of metrics for estimating the overall maintainability of the system. The study reports that *it has been widely accepted that high quality software should exhibit low coupling and complexity, and high cohesion.* Moreover, the authors state that *structural software attributes do not directly describe the quality of a product; rather they are used as predictors of external quality attributes such as reliability, efficiency, maintainability and portability* and that *the higher the cohesion, the less the maintainability effort required during service development.* Also Hitz et al. [16] state that the impact of coupling on the overall maintainability of a software system is widely accepted in the state of the practice. They also report that, in structured design and programming, the importance of coupling and cohesion as main attributes w.r.t. system decomposition has been well accepted by practitioners. Software engineering experts consider software designed as both more reliable and more maintainable.

Concerning cohesion, Sindhgatta et al. [26] confirm that highly cohesive designs are desirable since they are easier to analyze and test, and provide better stability and changeability, which make the whole system more maintainable. Finally, controlling and minimizing software complexity is considered as one of the most important objectives during software development as it may affect other key software qualities like reusability, reliability, testability, and maintainability [17, 25]. Complexity is one of the major factors in the cost of developing and maintaining software [19]. To summarize, we can derive the following generic formula for software maintainability.

[3]It is important to note that MicroQuality is independent from any specific quality attribute and its users can use their own definitions, according to their technical and organizational needs.

$$Maintainability = \begin{cases} HIGH, & (\text{Coupling = LOW} \wedge \text{Cohesion = HIGH}) \vee \\ & (\text{Complexity = LOW} \wedge \text{Coupling = LOW}) \vee \\ & (\text{Cohesion = HIGH} \wedge \text{Complexity = LOW}) \\ LOW, & \text{otherwise} \end{cases}$$

Intuitively, high coupling negatively impacts the overall maintainability of the system, as well as a high level of complexity; differently, high cohesion positively impacts the maintainability of the system. The *LOW* and *HIGH* thresholds in the formula have not been explicitly defined since their values depend on the specific characteristics of the MSA being considered and can vary across different projects.

The formula presented above will be considered throughout the paper as an example of quality model definition in the context of MicroQuality.

2.2 Running Example – the Acme Air System

An illustrative example of a microservice-based architecture is given in Figure 1. Acme Air[4] is an open-source implementation of a fictitious airline system whose key requirements are: the ability to scale to billions of web API calls per day, the need to develop and deploy the application in public clouds, and the need to support multiple channels for user interaction.

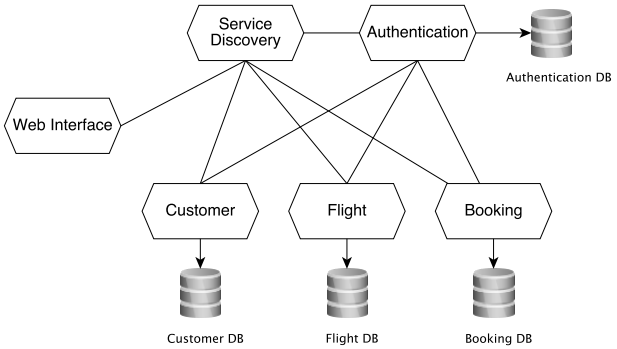

Figure 1: Acme Air architecture

The system is composed of six microservices: web interface, authentication, customer, flight, booking, and a service discovery. Each microservice is responsible to manage its own data and is allowed to communicate with the others only via RESTful calls. Services are small, independently deployable and independently scalable. The microservice-based implementation of Acme Air can support multiple data stores (i.e., MongoDB, IBM Cloudant), can run in several application modes and can support running on various runtime platforms, including stand-alone bare metal system, virtual machines, Docker containers, IBM Bluemix, and IBM Bluemix Container service [6].

Microservice-based systems must be designed to cope with failure [11], meaning that the application must be able to tolerate possible service failures either by recovering promptly or by gracefully degrading its functionalities. Acme Air can be easily equipped with Hystrix[5], a latency and fault tolerance library for distributed systems [6], which allows to monitor the system in real-time and to promptly detect and react to failures. MSAs aim to support to a great extent the agility and scalability of applications, and as new incoming requests are made through the web interface, the microservices interact with each other to serve the requested functionalities. Even though Acme Air is a simple microservice-based system, its complexity can rapidly grow for many reasons (e.g., high loads of traffic, latency issues, service failures), hence providing a suitable benchmark for our MicroQuality approach.

3 OVERVIEW OF THE APPROACH

As shown in Figure 2, the MicroQuality approach is composed of three phases, each dedicated to one of the three principles presented in the introduction, i.e., abstraction, reusable quality attributes configuration, and continuous quality assessment. Specifically, in the **architecture modeling** phase, the *Architecture Recovery Engine* (Ⓐ in the figure) automatically extracts the architecture of the MSA-based system and stores it into the *MSA model* (Ⓑ). The latter masks the complexity of the system under consideration and represents it in terms of architecturally-relevant entities, such as the microservices composing the system, their provided and required interfaces (e.g., the endpoints of their REST APIs), their interdependencies, etc. The MSA model conforms to a domain-specific language specifically tailored to represent MSAs (see Section 4). In this phase, developers can use the MSA model for understanding the overall structure of the system, without having to delve into the low-level details of each microservice (we recall that an MSA system can contain hundreds of interconnected microservices at deployment time). It is important to note that MSA models can also be manually specified and/or refined by developers; MicroQuality does not impose any constraint on how MSA models are created.

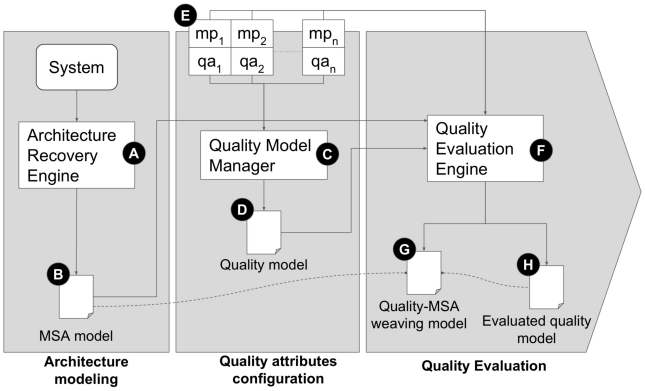

Figure 2: Overview of the approach

The goal of the **quality attributes configuration** phase is to allow developers to specify the quality model that is most suitable for the characteristics of their system. More specifically, the *Quality Model Manager* (Ⓒ) provides a domain-specific language for

[4]https://github.com/acmeair/acmeair

[5]https://github.com/Netflix/Hystrix

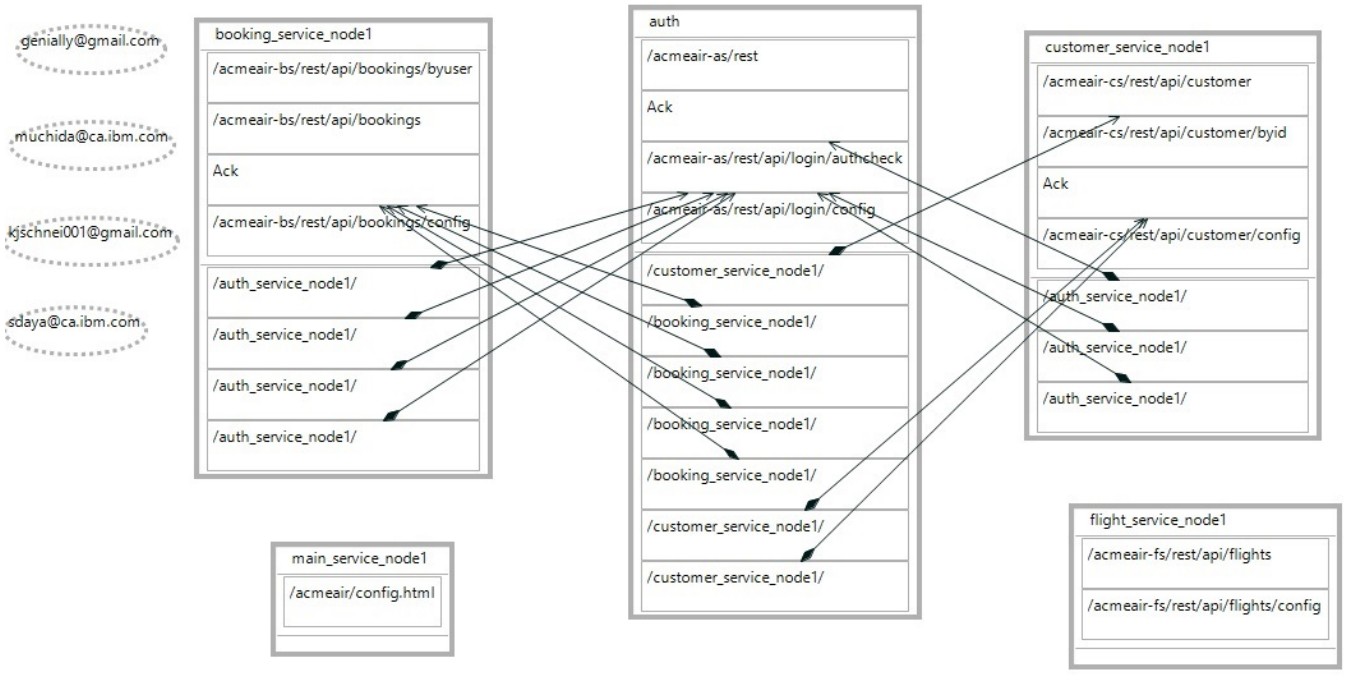

Figure 3: Acme Air MSA model

precisely specifying the *quality model* (D) in terms of the *quality attributes* it is composed of, their data types, how they are aggregated into more complex quality attributes, etc. The quality model is defined using a textual concrete syntax and refers to a set of reusable definitions of quality attributes (E). Each quality attribute qa_i in $\{qa_1 \ldots qa_n\}$ is automatically computed by MicroQuality either by means of its corresponding metric provider mp_i or by aggregating the values of other quality attributes. Metric providers are software components implemented in Java and that are (i) independent from each other, (ii) independent from the architectural language used for representing the MSA, and (iii) treated as third-party black-box data providers by MicroQuality. In this way, as we will show in the context of the running example, MicroQuality is flexible enough for allowing developers to reuse previously-defined quality attributes and for computing the values of quality attributes by using their preferred technologies and external services.

In the **quality evaluation** phase, the *MSA model* and the *quality model* are given as input to the *Quality Evaluation Engine* (F), which in turn measures the architecture of the system as based on the quality attributes defined in the *quality model*. Intuitively, for each quality attribute qa_i in the quality model, the *Quality Evaluation Engine* computes its value either by invoking its metric provider mp_i on some elements within the MSA model, or by aggregating the values of previously-computed *qa*s. The produced output consists of two models: the *Evaluated Quality Model* (H) containing all the measured quality attributes and the *Quality-Architecture Weaving model* (G), which traces the evaluated quality attributes to their corresponding entities within the MSA model of the system (e.g., a specific microservice, an interface). We opted to use a dedicated weaving model for linking the evaluated quality model and the MSA model in order to (i) keep the definitions of the *qa*s independent from the architectural language of the MSA model, thus allowing their reuse across projects and organizations, and (ii) do not constrain the developer in terms of which language should be used for creating the MSA model, thus enabling MicroQuality to use any approach for representing/extracting the architecture of the system, depending on the specific technical and organizational needs of the system.

Each model represented in Figure 2 has its own dedicated graphical/textual modeling editor, which is not depicted here for the sake of clarity. In the next sections, we detail each phase of MicroQuality by presenting its technical details, involved models, and the application to the Acme Air running example.

4 MODELING MICROSERVICE ARCHITECTURES

In this section we briefly recall the approach by Granchelli et al. [15] that we integrated in the component A for obtaining the MSA models (B), subject of the quality evaluation performed by the engine (F). This approach is a recovery mechanism for microservice architectures, composed of two phases namely *architecture recovery* and *architecture refinement*. The architecture recovery phase deals with all the activities necessary to extract an architectural model of the system starting from its source code repository and run time information (e.g., log files) that we call *MSA Model* (B). The architecture model represents the architecture of the system which is composed by all the elements of the microservices architecture. Its

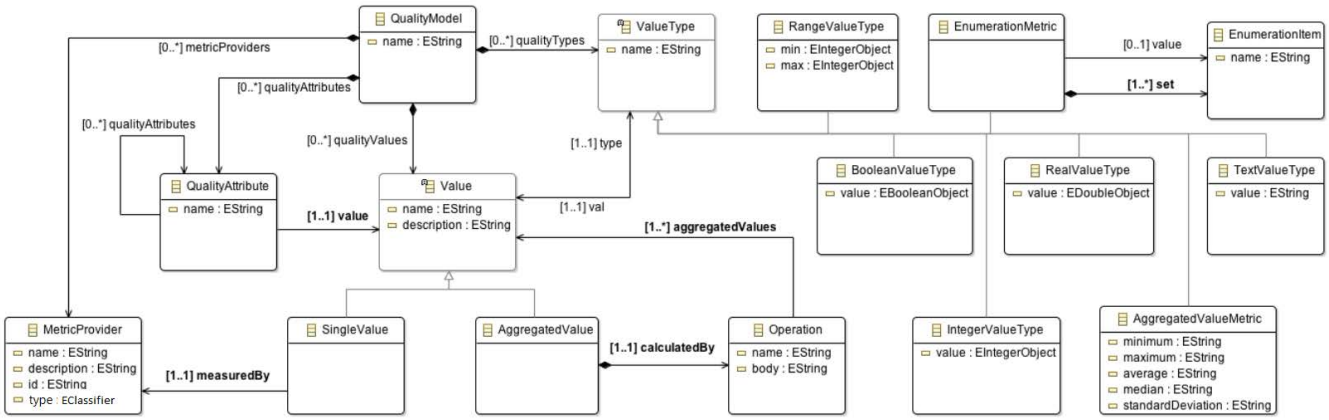

Figure 4: The customized quality metamodel from [3]

representation is based on a domain-specific language (DSL) for microservice-based systems, that we reused in our approach. We briefly recall its underlying metamodel with the intent of presenting an example of an MSA Model. The metamodel is composed of seven metaclasses where `Product` is the root node of the system being designed. `MicroService` represents the microservices composing the system. `Interface` represents a communication endpoint and it is attached to specific microservices, for which it represents either an input or output port. `Link` connects two interfaces together, thus representing the communication among them. `Team` is composed of one or more developers. Each microservice is *owned* by only one team. The metaclass `Developer` represents a software developer that participates to the system. `Cluster` is a logical abstraction for grouping together specific microservices.

This DSL is developed around the microservice needs and characteristics [11], and it is kept minimal in order to support the design and description of multiple microservice-based systems. The *architecture refinement* phase aims to refine the initial extracted architectural model into one or more *refined architectural models* by means of refinement incremental steps. The software architect can decide to enhance the generated architectural model in order to recover an architecture more suitable for its needs, e.g., removing unnecessary details, perform model analysis, architectural change impact analysis, overall understanding of the system, and finally decide when the refined architectural model is ready to be finalized. Every model generated by the Architecture Recovery Engine or refined by the architect is an instance of this DSL metamodel. For instance, Figure 3 shows the model, extracted from the given repository, representing the refined MSA model of the running Acme Air system. This model, realized with the provided editor appositely conceived for the approach, can easily be inspected to identify the extracted services and the connections among services. For instance, the $/auth\_service\_node1/$ of the $customer\_service\_node1$ component is connected to the $/acmeair-cs/rest/api/login/authcheck$ node of the $auth$ component. This means, that the component $customer\_service\_node1$ uses the $/acmeair-cs/rest/api/login/authcheck$ operation of the $auth$ component to verify if the user is authorized to perform their profile editing operation offered by $/acmeair-cs/rest/api/customer/config$.

5 DEFINING QUALITY ATTRIBUTES FOR MICROSERVICE ARCHITECTURES

Another important component of MicroQuality is the integration of the Quality Metamodel presented by Basciani et al. [3], that we customized for representing quality definition in the field of MSA. This approach plays a key role since it enables the specification of quality models according to user requirements for different software artefacts and domains, like MSA. The considered metamodel is shown in Figure 4, it consists of a number of constructs as explained in the following. `QualityModel` is the root element consisting of `QualityAttributes`, `ValueTypes`, and `MetricProviders`. A `QualityAttribute` represents a quality aspect that is considered to be relevant for contributing to the quality assessment of a given artefact, in our case the MSA architectural model and all its elements. A quality attribute, like *maintainability* can be an aggregation of other attributes, like the *complexity* and the *coupling*, as detailed in the remaining of this section. Thus, each quality attribute indicates how to compose the contained attributes in order to provide an overall quality evaluation for the considered attribute. Each quality attribute has a `Value` representing its calculated value by the evaluation engine. `Value` can be `SingleValue` or `AggregatedValue`, where `SingleValue` represents the value obtained by the application of a given `MetricProvider`, which refers to a software component able to calculate the value of the related quality attribute. A `MetricProvider` is labelled with the attribute `id`, used to identify and retrieve the actual software component implementing the considered metric, that is part of the overall architecture of the proposed approach, labeled with Ⓔ in Figure 2. This metamodel has been extended in order to enable the evaluation of internal elements and not only an overall evaluation as proposed in [3]. This has been enabled with the attribute *type* inside `MetricProvider` metaclass, allowing to associate the model element to be considered by the metric calculation. `AggregatedValue` indicates a composition of different values and it is specified by means of an OCL expression. Each `Value` has a reference with a `ValueType` element, which defines its type, enabling the specifications of categories (like *LOW*, *MEDIUM, HIGH*), ranged values (e.g., from 0 to 5), textual, boolean, integer, and real values. This approach has been applied to specify

Quality Model for MSA, labeled with Ⓓ in Figure 2, where the software architect can specify relevant quality attributes for its domain and perspective.

In this section we report on an excerpt of the quality model we processed in the running example for the Acme Air evaluation. Figure 5 shows the quality model containing a set of considered quality attributes as defined in Section 2.1. The model is represented in the left hand side with a concrete syntax used by the architect to easily define the concepts and on the right the model-based representation that will be actually processed by the engine. For instance, in the defined model the *maintainability* is defined with a bottom-up process, where first we define the Enumeration type to hold the evaluation result of the quality metric. As defined in the maintainability formula in Section 2.1, the result of the estimation is an enumeration of 3 possible values 'HIGH', 'MEDIUM', 'LOW' (see Figure 5 lines 12-14) [6]. Then the aggregated value that holds the enumeration is defined in lines 22-32. Inside the aggregated value we create the operation that compute the maintainability value (lines 23-31). In this operation the values to be aggregated to estimate the maintainability, can be specified as shown in lines 25-26, together with the OCL operation aggregating the values (lines 27-30).

OCL [32] is the object constraint language to express the formulas that the quality attributes use for navigating the MSA models. In lines 3-8 one of the Metric Provider named *CouplingCalculator* has been defined with its relative ID and its type, defining which metric component will calculate the coupling of the *MicroService* model element.

6 EVALUATING THE QUALITY OF MICROSERVICE ARCHITECTURES

In this section, given as input an MSA model Ⓑ, we show how this architecture can be measured with respect to the quality definition (Ⓓ), for example the one shown in Figure 5. The evaluation engine, labeled with Ⓕ, is a software component that is able to evaluate the quality attributes defined in the quality model. By applying the declared metrics in the model and invoking the related $\{mp_1 \dots mp_n\}$ metric providers, it is able to generate a model called *Evaluated Quality Model* (labeled with Ⓗ) together with the *Quality-Architecture Weaving Model* (Ⓖ).

A fragment of the Java code of the evaluation engine is shown in Listing 1. The evaluation engine for each value (lines 4 to 15), depending on its nature, single or aggregated, invokes the related service responsible for computing it. In particular, in case of single value the evaluation engine invokes the related *mp* metric provider (line 6-8), whereas, in case of aggregated value, it executes the OCL operation by using the OCL evaluator (line 12).

```
1  class EvalQuality implements EvaluationEngine {
2  public msaQualityRelationships evalQuality() {
3    QualityModel qm = s.getQualityModel();
4    for (Value v: qm.getQualityValues()) {
5     if (v instanceof SingleValue) {
6       MetricProvider mp = ((SingleValue)v).
            getMeasuredBy();
7       MetricCalculator mc = mcs.get(mp.getId());
8       RelationGroup rg = mc.calculate(msa.
            getMicroserviceModel(), v);
9      ws.addRelationGroup(rg);
10    } else if (v instanceof AggregatedValue) {
11         AggregatedValue av = (AggregatedValue)v;
12         RelationGroup rg = os.evaluateOCL(av);
13         ws.addRG(rg);
14    }
15   }
16   s.saveQM();
17   ws.saveWeaving();
```

Listing 1: Excerpt of the Evaluation Engine

Each metric provider is implemented as a Java class for each single value. Listing 2 shows a snippet of code related to the Complexity Metric Provider.

```
1  class ComplexityCalculator implements
       MetricCalculator {
2  ...
3  public RelationGroup calc(Product p, Value v) {
4   RelationGroup rg = ws.createRG(v.getVarName());
5   ListValue lv = qm.createListValue();
6   for (EObject o : p.eAllContents()) {
7    if (o instanceof v.getMeasuredBy().type) {
8     int sum = 0;
9     if (o.getExpose() != null)
10     sum += o.getExpose().size();
11    if (o.getRequire() != null)
12     sum += o.getRequire().size();
13    IntegerValueType el = qm.createInt(sum);
14    Relations r = ws.createRRL(o, el);
15    rg.getReferTo().add(r);
16    lv.getElements().add(el);
17   }
18  }
19  v.setValueType(lv);
20  return rg;
21 }
```

Listing 2: Metric Provider for complexity quality attribute

Line 7 checks if the type specified in the quality model is instance of the processed element of the MSA Model in the iteration; if so, it applies the code responsible for computing the value of the metric on the current artefact (lines 8-17). Once computed, it creates the relation in the resulting weaving model, to link the right artefact of the MSA model with the calculated value in the quality model (lines 14-15).

As already said, the results of the quality evaluation phase are multiple. The first generated model is the *Evaluated Quality Model*. It is an instance of the declared and executed quality model (Ⓒ in Figure 2), where the calculated values for the quality attributes, relative to the subject MSA model, are stored. An example of the evaluated quality model for the Acme Air system is shown in Figure 7 right panel (labelled with Ⓗ).

The second generated model is the *Quality-MSA weaving model*, labelled with Ⓖ in Figure 2. In order to compose this model we conceived a Weaving Metamodel shown in Figure 6.

Weaving models are special kinds of models that set relations across different models in order to create a wider representation of the domain. The model weaving is considered as the operation

[6] The thresholds associated with the ranges can be customized by the architect

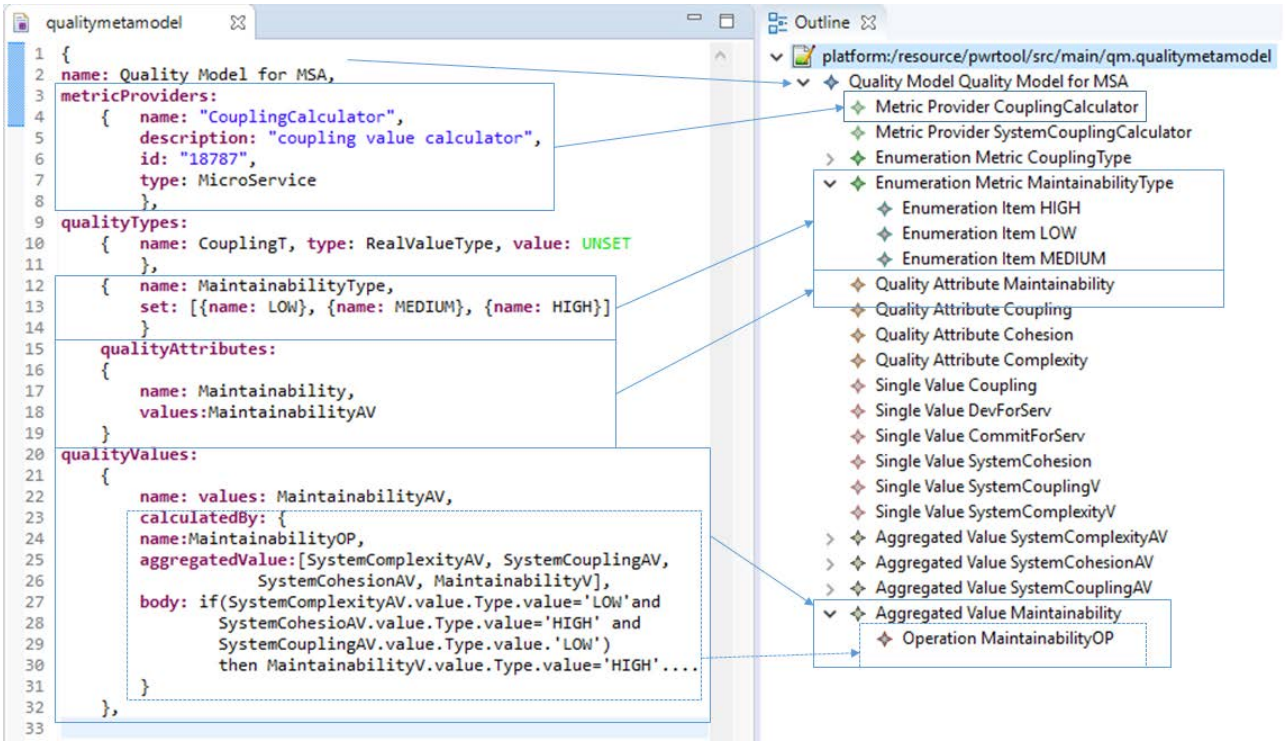

Figure 5: Excerpt of the Quality Model for MSA

for setting fine-grained relationships between models or metamodels and executing operations on them based on the semantics of the weaving associations specifically defined for the considered application domain [9, 24].

In the MicroQuality approach the operation of model weaving is used to assign a specific evaluated metric value from the evaluated quality model, to the MSA architecture target of the computation. This allows to realize both a higher separation of concerns as well as a higher level of abstraction. A metric value, that impacts a quality attribute of the architecture, can be related to each element of the MSA architecture model, e.g., cluster, microservice, product, interface. For this reason the *type* attribute of the metaclass `MetricProvider` in the Quality Metamodel has been set to type

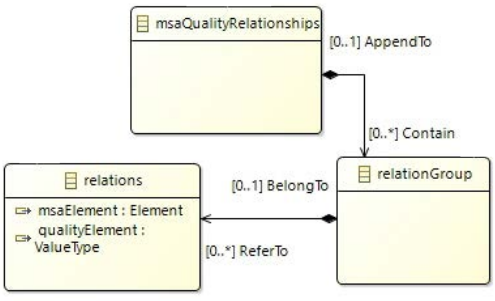

Figure 6: Quality-architecture weaving metamodel

`EClassifier`, in order to allow the developer to refer to each possible element in the application domain. These enable the architect to specify custom quality attributes and target them on elements of the MSA architecture model without changing or refactoring the models.

Figure 7 shows the resulting weaving model of the Acme Air system. This permits the architect to inspect the relationships among evaluated quality attributes and the MSA model given as input. The left panel (labelled with Ⓑ) shows the MSA model of the Acme Air system (subject of the evaluation), extracted by the Architecture Recovery Engine. The selected element in the model is the `Product`, identifying the whole software product Acme Air.

The central panel (labelled with Ⓖ), is the relation connector between the evaluated quality model and the MSA model, that can be navigated in order to read the evaluation result. This panel renders the weaving model conforming to the weaving metamodel in Figure 6. The evaluation result can be seen in the right panel, where the selected relation drives the reading to the maintainability of the whole system resulted *HIGH*.

The proposed approach allows to inspect the single elements composing the MSA model and navigate the related quality evaluation for the executed quality model given as input (for instance the one proposed in Figure 5).

The result of the evaluation for the quality model in Figure 5 on the whole Acme Air system is summarized in Table 1 since the model in Figure 7 cannot be exploded to show all the evaluations covered in the paper. The Acme Air system resulted LOW in Coupling with a result of 0.5, LOW in Cohesion (0.64) and LOW in

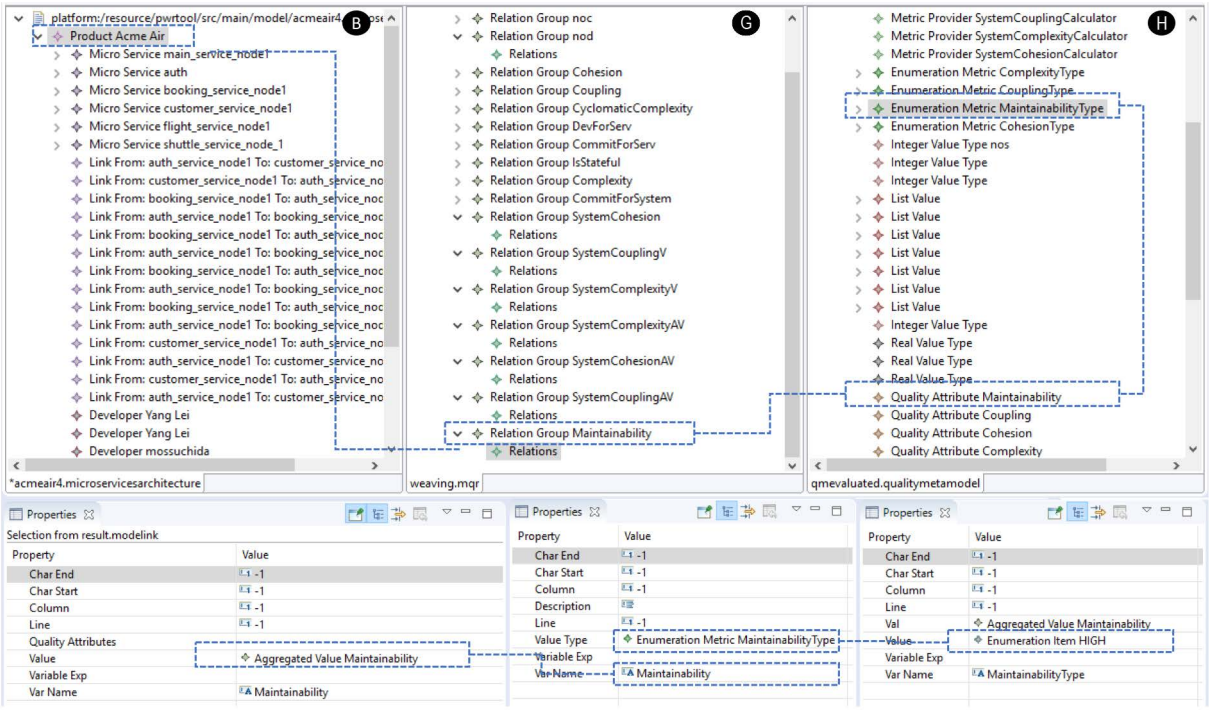

Figure 7: Weaving Quality between the Acme Air MSA model (left) and the evaluated quality model (right)

complexity (5), confirming to have a HIGH level of maintainability (see formula in Section 2.1).

Table 1: Quality Assessment of the Acme Air architecture

Quality Attribute	Evaluation
Coupling	LOW
Cohesion	LOW
Complexity	LOW
Maintainability	**HIGH**

7 DISCUSSION

The MicroQuality approach has been designed specifically for microservice-based systems. It combines the advantages of the approach for recovering microservice-based systems presented in [14] with the advantages of the extensible and generic approach assessing the quality of modelling artefacts published by Basciani et al. [3].

On the one hand, a limitation of the scope of our approach to MSA lies in the fact that the used recovery approach relies on specific sources of information (e.g., Docker files) typically available only in microservice-based systems. On the other hand, the integration of MicroQuality with automatic architecture recovery allows MicroQuality to support continuous quality assessment of MSA systems –a quite powerful feature– as the architectural models can be automatically recovered and measured at any point in time.

MicroQuality addresses a growing interest of the community around the continuous quality assessment of MSA systems, as also confirmed by Bogner et al.: *the potentially very large number of services makes the automatic gathering of metrics all the more important* [4]. Moreover, the same authors report that well established metrics for MSA systems have not been established yet. From this perspective, the flexibility of MicroQuality is of great value as it allows to define customised quality attributes for MSA systems.

To the best of our knowledge, well established metrics for MSA are not yet defined. To address this problem, the same authors in [4] created a repository of possible metrics for MSA as a first step towards the identification of suitable metrics in the field. Therefore, these metrics can be used by architects and developers for their projects and can be easily integrated into the MicroQuality approach (we recall that MicroQuality provides the possibility for its users to define their own quality models and metric providers). This makes the overall approach more user-centric since the same users (i.e., architects and developers) can define, use, and share their own quality specification models.

Inevitably, the current implementation of MicroQuality inherits the same limitations of the architecture recovery technique presented in [14]. Even though the recovery approach aims to be generic, it still relies on specific sources of information (e.g., Docker files or similar), which, if not present, makes the architecture recovery technique challenging or difficult to be applied.

As already mentioned in Section 2.2, we have validated our approach using the Acme Air benchmark. Currently, this seems to be the only open-source test system available for benchmarking microservice-based systems [8]. We aim to further improve the

validation of our MicroQuality approach on a larger MSA system with a higher number of microservices.

8 RELATED WORK

The approach presented in this paper shows distinguishing characteristics that makes it different from other existing approaches in the field. First of all, our approach specifically targets the domain of microservice-based systems. Second, it combines together (and extends) two different existing approaches: the architecture recovery approach presented by Granchelli et al. in [15] and the automated quality assessment approach proposed by Basciani et al. in [3]. To the best of our knowledge this is the first approach that provides the possibility to: (i) automatically extract a model of the architecture of a microservice-based system, (ii) support the customization of the quality attributes of the system, and (ii) produce quantitative measures for their assessments.

If we consider the steps of the approach separately, there exists a number of related works in the field. Concerning architecture recovery, many approaches exist to recover a model of an existing system by analysing its source code. In the majority of cases, these techniques are applicable only to systems developed with monolithic or service-oriented architectures (SOA), while we specifically target microservice-based systems. Two secondary studies [1, 8] have reported a possible lack of research in architecture recovery techniques specific for MSA. Two different approaches have been published so far: MicroART [15] and MiSAR [2]. MicroART is an approach for automatic architecture recovery of MSA systems and leverages its own specific domain-specific language using Model Driven Engineering techniques. MiSAR [2] is a more recent approach for architecture recovery and relies on a different set of concepts mapped on its own specific metamodel. An important difference between the two approaches is that while MicroART is automatic, MiSAR relies on a manual process for the definition of mapping rules as well as for the code inspection.

Concerning the assessment of quality attributes, in MicroQuality approach metrics are calculated directly on the (recovered) architectural model of the system. Differently, existing approaches in the literature often rely on source code analysis. For example, Stormer presents *SQUA3RE* [28], a conceptual framework providing a set of concepts and components for enabling quality attribute analysis on existing monolith systems. In this work, the definition of quality attributes is based on the QUA Model [28]. Nuraini and Widyani present a quality assessment methodology [18] based on the use of SOA-QEM as quality model. The quality attributes are mapped to basic metrics, which are evaluated on the source code with the use of an external tool. Plösch et al. propose an Evaluation Method for Internal Software Quality *(EMISQ)* [23] for systematically assessing the internal software quality of a software system. EMISQ metrics extraction is performed with tools directly on the system source code and it is not possible to create customized quality attributes and apply them to the system evaluation. Goeb and Lochmann [13] present a unifying meta-model to describe the quality of service-oriented systems as an enhancement of the Quamoco meta-model. Their approach relies on the source code analysis. The Quamoco approach [7, 30, 31] provides a tool chain to both define and assess software quality. It contains the metamodel with the related editor and an integration with the quality assessment toolkit ConQAT. The ConQAT provides the value indicators for quality attribute specified in the quality model, which are later aggregated to assess software quality. The *ISO Standard 14598* [27] provides a framework for evaluating the quality of software products as well as the requirements for software measurement and evaluation. However, the standard does not allow to create customized quality attributes and apply them to the framework. In [3] Basciani et al. presented an extensible and generic approach to assess the quality of modelling artefacts. They proposed a tool chain enabling users to specify custom quality models, and to apply them on the artefacts being analysed in order to automatically assess their quality. The presented application of the proposed approach are real metamodels and ATL transformations. We partially used the presented tool in the domain of MSA, extending the approach and redefining the metric providers for the application. The approach by Basciani et al. does not allow to inspect the artifacts internally, but we have overcome this limitation by introducing the weaving model in our approach. The advantages of focusing on a specific DSL for quality models allows the MicroQuality approach to overcome the limitation of other existing approaches. We have validated our approach focusing on the maintainability quality attribute, but the MicroQuality approach supports the definition of any quality attributes, provided that the specific metric provider components are implemented to retrieve the necessary information for the computation of the metric. This characteristics allows MicroQuality to be more flexible with respect to other approaches, as for example techniques based on UML profiles (e.g., UML Marte), which rely on predefined concepts with the result of being limited in their extensibility and customization.

A first contribution to the field of maintainability metrics for microservice-based system has been performed by Bogner et al. [4]. In [4] they present a holistic survey of maintainability metrics specifically designed for service-based systems, and provide also initial insights for the applicability of the same metrics to microservice-based systems. Authors also suggest that researchers and practitioners can use the identified metrics as a repository to identify appropriate metrics for their projects. Bogner et al. defined a maintainability quality model for service- and microservice-based systems in [5], but the quality model has not been fully validated yet.

In this section we have reported on the state of the art in architecture recovery techniques for microservice-based systems and the architectural evaluation approaches that are more closely related to our approach. However, due to the extensive amount of work in the literature, providing a complete general reference on recovery techniques and architectural evaluation approaches is unfeasible. For the interested reader, more information can be found in the secondary studies published in [1, 8, 21].

9 CONCLUSIONS AND FUTURE WORK

In this paper we have presented MicroQuality, an approach for the quality evaluation of MSAs. MicroQuality is centered around a technology-independent *architecture model* of the system, which can be specified either manually or automatically via pre-existing

architecture recovery techniques for MSAs. Model Driven Engineering techniques are used for making MicroQuality independent from both (i) the modeling language used for representing the architecture of the system, and (ii) the quality attributes computed on the model. Quality attributes live in a dedicated ecosystem of well-specified and language-independent software quality attributes for MSAs. MicroQuality has been architected so to enable the continuous quality assessment of MSAs in order to get data-driven and timely insights in the overall quality of the measured system.

MicroQuality has been evaluated in the context of Acme Air, a publicly available benchmark system for microservices. We applied MicroQuality to measure the maintainability of Acme Air, defined as an aggregation of coupling, cohesion, and complexity of its microservices.

As future work, we are planning to architect MicroQuality as a microservice-based system itself, so to make its main components independently extensible and flexible. We are also developing a web app for accessing all MicroQuality functionalities in an integrated manner, thus improving its overall usability. This will open the possibility for performing industrial case studies in which the ecosystem of quality attributes will be incrementally enriched and MicroQuality will be used for evaluating the quality of real MSAs in the context of industrial projects. In this direction, we plan to extend the quality evaluation applying metrics which can depend on run-time aspects, e.g., frequency of invocations, frequency of selection of branches in choices, frequency of services' failures and so on.

ACKNOWLEDGMENTS

This research work has been supported by the Ministry of Economy and Finance, Cipe resolution n. 135/2012 (project INCIPICT - INnovating CIty Planning through Information and Communication Technologies).

REFERENCES

[1] Nuha Alshuqayran, Nour Ali, and Roger Evans. 2016. A Systematic Mapping Study in Microservice Architecture. In *Service-Oriented Computing and Applications (SOCA), 2016 IEEE 9th International Conference on*. IEEE, 44–51.

[2] Nuha Alshuqayran, Nour Ali, and Roger Evans. 2018. Towards Micro Service Architecture Recovery: An Empirical Study. In *2018 IEEE International Conference on Software Architecture (ICSA)*. IEEE.

[3] Francesco Basciani, Juri Di Rocco, Davide Di Ruscio, Ludovico Iovino, and Alfonso Pierantonio. 2016. A Customizable Approach for the Automated Quality Assessment of Modelling Artifacts. In *10th International Conference on the Quality of Information and Communications Technology, QUATIC 2016, Lisbon, Portugal, September 6-9, 2016*. 88–93. https://doi.org/10.1109/QUATIC.2016.025

[4] Justus Bogner, Stefan Wagner, and Alfred Zimmermann. 2017. Automatically measuring the maintainability of service-and microservice-based systems: a literature review. In *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement*. ACM, 107–115.

[5] Justus Bogner, Stefan Wagner, and Alfred Zimmermann. 2017. Towards a practical maintainability quality model for service-and microservice-based systems. In *Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings*. ACM, 195–198.

[6] Shahir Daya, Nguyen Van Duy, Kameswara Eati, Carlos M Ferreira, Dejan Glozic, Vasfi Gucer, Manav Gupta, Sunil Joshi, Valerie Lampkin, Marcelo Martins, et al. 2016. *Microservices from Theory to Practice: Creating Applications in IBM Bluemix Using the Microservices Approach*. IBM Redbooks.

[7] Florian Deissenboeck, Lars Heinemann, Markus Herrmannsdoerfer, Klaus Lochmann, and Stefan Wagner. 2011. The quamoco tool chain for quality modeling and assessment. In *Software Engineering (ICSE), 2011 33rd International Conference on*. IEEE, 1007–1009.

[8] Paolo Di Francesco, Patricia Lago, and Ivano Malavolta. 2017. Research on Architecting Microservices: trends, Focus, and Potential for Industrial Adoption. *IEEE International Conference on Software Architecture (ICSA)* (2017).

[9] Davide Di Ruscio. 2007. *Specification of model transformation and weaving in model driven engineering*. Ph.D. Dissertation. PhD thesis, Universita degli Studi dell'Aquila (February 2007) http://www. di. univaq. it/diruscio/phdThesis. php.

[10] International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC). 2001. *Software Engineering–Product Quality: Quality model*. Vol. 1. ISO/IEC.

[11] Martin Fowler and J Lewis. Last accessed: Mar 2017. Microservices a definition of this new architectural term. *URL: http://martinfowler.com/articles/microservices.html* (Last accessed: Mar 2017).

[12] Paolo Di Francesco, Patricia Lago, and Ivano Malavolta. 2018. Migrating Towards Microservice Architectures: An Industrial Survey. In *IEEE International Conference on Software Architecture, ICSA 2018, Seattle, USA, April 30 - May 4, 2018*. 29–39.

[13] Andreas Goeb and Klaus Lochmann. 2011. A software quality model for SOA. In *Proceedings of the 8th international workshop on Software quality*. ACM, 18–25.

[14] Giona Granchelli, Mario Cardarelli, Paolo Di Francesco, Ivano Malavolta, Ludovico Iovino, and Amleto Di Salle. 2017. MicroART: A Software Architecture Recovery Tool for Maintaining Microservice-based Systems. *IEEE International Conference on Software Architecture (ICSA)* (2017).

[15] Giona Granchelli, Mario Cardarelli, Paolo Di Francesco, Ivano Malavolta, Ludovico Iovino, and Amleto Di Salle. 2017. Towards Recovering the Software Architecture of Microservice-based Systems. *First International Workshop on Architecting with MicroServices (AMS)* (2017).

[16] Martin Hitz and Behzad Montazeri. 1995. Measuring coupling and cohesion in object-oriented systems. (1995).

[17] Usha Kumari and Shuchita Upadhyaya. 2011. An interface complexity measure for component-based software systems. *International Journal of Computer Applications* 36, 1 (2011), 46–52.

[18] Aminah Nuraini and Yani Widyani. 2014. Software with service oriented architecture quality assessment. In *Data and Software Engineering (ICODSE), 2014 International Conference on*. IEEE, 1–6.

[19] Edward E Ogheneovo. 2014. On the relationship between software complexity and maintenance costs. *Journal of Computer and Communications* 2, 14 (2014), 1.

[20] Paul Oman and Jack Hagemeister. 1992. Metrics for assessing a software system's maintainability. In *Software Maintenance, 1992. Proceerdings., Conference on*. IEEE, 337–344.

[21] Claus Pahl and Pooyan Jamshidi. 2016. Microservices: A systematic mapping study. In *Proceedings of the 6th International Conference on Cloud Computing and Services Science*. 137–146.

[22] Mikhail Perepletchikov, Caspar Ryan, Keith Frampton, and Zahir Tari. 2007. Coupling metrics for predicting maintainability in service-oriented designs. In *Software Engineering Conference, 2007. ASWEC 2007. 18th Australian*. IEEE, 329–340.

[23] Reinhold Plösch, Harald Gruber, A Hentschel, Ch Körner, Gustav Pomberger, Stefan Schiffer, Matthias Saft, and S Storck. 2008. The EMISQ method and its tool support-expert-based evaluation of internal software quality. *Innovations in Systems and Software Engineering* 4, 1 (2008), 3–15.

[24] Th Reiter, E Kapsammer, W Retschitzegger, and W Schwinger. 2005. Model integration through mega operations. In *Workshop on Model-driven Web Engineering*. 20.

[25] Steven D Sheetz, David P Tegarden, and David E Monarchi. 1991. Measuring object-oriented system complexity. In *Proc. 1st Workshop on information Technologies and Systems*.

[26] Renuka Sindhgatta, Bikram Sengupta, and Karthikeyan Ponnalagu. 2009. Measuring the quality of service oriented design. In *Service-Oriented Computing*. Springer, 485–499.

[27] I Standard. 1999. Information technology–software product evaluation–part 1: General overview. *ISO Standard* (1999), 14598–1.

[28] Christoph Stormer. 2007. Software Quality Attribute Analysis by Architecture Reconstruction (SQUA3RE). In *Software Maintenance and Reengineering, 2007. CSMR'07. 11th European Conference on*. IEEE, 361–364.

[29] Anne Thomas and Aashish Gupta. 2017. Innovation insights for Microservices. *https://www.gartner.com/doc/3579057/innovation-insight-microservices* (2017).

[30] Stefan Wagner, Andreas Goeb, Lars Heinemann, Michael Kläs, Constanza Lampasona, Klaus Lochmann, Alois Mayr, Reinhold Plösch, Andreas Seidl, Jonathan Streit, et al. 2015. Operationalised product quality models and assessment: The Quamoco approach. *Information and Software Technology* 62 (2015), 101–123.

[31] Stefan Wagner, Klaus Lochmann, Lars Heinemann, Michael Kläs, Adam Trendowicz, Reinhold Plösch, Andreas Seidl, Andreas Goeb, and Jonathan Streit. 2012. The quamoco product quality modelling and assessment approach. In *Proceedings of the 34th international conference on software engineering*. IEEE Press, 1133–1142.

[32] Jos B Warmer and Anneke G Kleppe. 1998. The object constraint language: Precise modeling with uml (addison-wesley object technology series). (1998).