Attributes Assessing the Quality of Microservices Automatically Decomposed from Monolithic Applications

Michel-Daniel Cojocaru
Vrije Universiteit Amsterdam
m.cojocaru@student.vu.nl

Alexandru Uta
Vrije Universiteit Amsterdam
a.uta@vu.nl

Ana-Maria Oprescu
Universiteit van Amsterdam
a.m.oprescu@uva.nl

***Abstract*—The architectural styles in the world of software development are constantly evolving. Recently the microservice architecture is gaining more and more traction, building on concepts of Service Oriented Architecture (SOA) and steering further away from monolithic architectures. Emerged from agile communities, the microservice oriented architecture implies a number of small sized microservices independently deployable. The adoption of microservices as base for creating enterprise applications is certain, yet many companies intend to migrate from the old monolithic style instead of creating new products mainly due to cost related implications as well as challenging and complex tasks. Several tools and approaches for semi-automatic decomposition of monolithic applications to microservices have emerged, yet many of them still struggle to verify the result of such process, the architect being indispensable for assessing the output microservices. Although this area is intensely studied, no unanimously accepted and clear guidelines for defining a good microservice exist. This survey focuses on providing a comprehensive and broadly applicable set of quality assessment criteria for microservices resulted from semi-automatic migration tools or techniques. Our study aligns with industry requirements, including a case study which further validates our set of quality attributes. In the refinement step of the quality attributes set, the prospect of automating the process of validation is also discussed.**

I. Introduction

The microservice architecture is currently the most popular architecture for companies creating new applications, due to its advantages, such as agility or scalability [18]. The architecture has a crucial role in the software life-cycle, for example to ensure quality and critical attributes of the software [23]. To take advantage of the microservice-style architecture, much effort has been invested into porting legacy, monolithic applications. Automated tools [4], [20] already exist to decompose large applications into tiny services. However, there is no conceptual framework for assessing the quality of these decomposed applications. In this article, we conduct a comprehensive study of the most applied techniques to study the quality of decomposed services.

While the microservices architecture has many definitions, we choose that proposed by Martin Fowler [1]: building an application as a group of services each designed for a specific business capability, intercommunicating via lightweight mechanisms while being independently deployable.

Microservice Architectures (MSA) are built upon concepts from Service Oriented Architecture (SOA), their main focus consisting of partitioning services in relation to business capabilities. However, MSA differs from SOA through its proliferation of complete isolation between the microservices. The MSA borrows from SOA the idea of componentization as services, designed using bounded contexts. While SOA merges correlated functionalities [6] only from a business perspective, MSA also targets the size of the service. If granularity of the components is the main focus of the architecture, and keeping state in the application is not necessary, one might consider Serverless Computing [6], instead of MSA.

MSA focuses on agile development with DevOps practices for continuous integration and delivery, while keeping the data management and interaction of the services decentralized. Regarding the sharing philosophy, MSA focuses more on isolation and autonomy relying on choreography, while SOA focuses on reusability relying on orchestration [2].

Although considerable research has been conducted into microservices, some SOA-inherited challenges still persist: identifying interface boundaries, finding appropriate granularity, ensuring proper automatic deployment, utilizing testing tools. Identifying ”good” microservices does not focus only on partitioning the system for easy maintenance, but also describes the evolution of the system and its scaling capabilities.

An important design issue that has to be considered in MSA is the functionality each service possesses, that has to be carefully split/allocated with the appropriate granularity. In accordance with Conway's Law [4], organizations typically produce system architectures/designs that mirror the organization's communication patterns: the natural flow of interaction between the teams developing a software product is embedded in the calls/interaction/communication between the microservices/components.

Issues specific to the monolithic architectures, such as the system becoming highly coupled and hard to maintain, can be overcome by migrating to MSA. An increasingly adopted option for resolving existing issues (without major cost implications), MSA provides maintainability to the system and increases the rate at which software products are delivered [18].

In the following, we present the most difficult to test architectural qualities identified in scientific literature. According to existing literature [24], automated testing is the second most occurring issue regarding the migrating process. Regarding the architecture, the main issues identified are (1) high coupling (2) service boundaries identification dilemmas, and (3) system decomposition [24], in this context referring to decomposition around business capabilities.

Service composition is a recurring issue in the process of designing a MSA [3] and lacks a sound methodology to assess the quality of microservice-oriented decompositions [16], while architecture impact analysis is an unexplored area [3]. Thus, it is safe to assume that with all recent developments in automating the migration process, the validation of resulted microservices still lacks a clear set of quality criteria.

The **goal** of our study is to identify a minimum required set of quality assessment criteria applicable to any microservice resulted from migrating a monolithic application to a microservice architecture. To this end, we first identify all quality assessment criteria explored in related literature. Next, we refine this set by imposing two additional meta criteria: *meaningfulness* and *feasibility of implementation*. The former considers the applicability of a quality attribute to a microservice at component level. The latter considers if, given a quality attribute, a workload description was identified in literature.

Our study is organized as follows: Section II describes our research methods; Section III describes two different decomposition techniques; Section IV presents two types of software analysis, used to label the identified validation criteria; Section V presents three decomposition approaches and constructs the minimum set of quality assessment criteria of microservices from the perspective of two self imposed meta-criteria: meaningfulness and feasibility of implementation. A case study based on an interview with an industry expert is included for ensuring alignment to industrial needs. Section VI presents related work, and we conclude in Section VII.

II. Research Methods

Our research focuses on scientific literature gathering and analysis. Selecting a complete literature corpus is crucial to a literature survey, and we explain in detail our mapping.

Mapping Strategy Our literature study survey is relying on several different research methods. The core [3], [5], [13], [17] scientific papers were proposed by the authors from their experience. From that set, we follow both forward and backward snowballing sampling approaches [24] resulting in 9 additional references. We ensure the completeness of the literature corpus by also using keywords-based unguided research, yielding 18 additional papers.

Search Domain Our literature corpus consists of 29 scientific papers published between 1998 and 2018 gathered from the following digital libraries: IEEEXplore, ACMDL, ResearchGate, Scopus, SemanticScholar, GoogleScholar.

Keywords: {microservices **and** decomposition} **or** {microservices **and** quality **and** criteria} **or** {microservices **and** metrics} **or** {microservices **and** quality **and** metrics} **or** {microservices **and** validation} **or** {microservices **and** metrics **and** validation}.

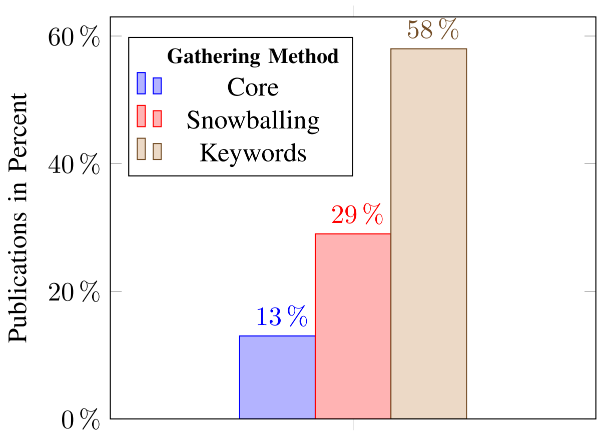

Fig. 1: Distribution of corpus across gathering methods.

Inclusion Criteria

- Studies are written in English
- Studies are accessible electronically
- Studies are published in scientific journals or conferences
- Studies are related to our research (here referring to keywords appearance in text)

III. Decomposition Tools

Although there are several decomposition approaches in literature [4], [20], we only focus on those that present a certain degree of automation. We identified two tools able to semi-automatically decompose monoliths into microservices [4], [20]. The tools do not provide correct results in 100% of the cases. Furthermore, the two approaches yield different decompositions for the same input application.

I. Service Cutter [4] is a semi-automatic decomposition tool. The concept of nanoentity is introduced for the three resource building blocks: data, operations and artifacts. "Data" represents the subset of system's data on which a microservice has ownership. "Operation" is a business rule or logic process applied on data. "Artifact" is a snapshot of un-/processed "Data" or "Operation", expressed in an unified format.

The decomposition process identifies a set of microservices to which nanoentities are assigned. There is a one-to-many relation between the microservice and the nanoentities, yet a nanoentity can be assigned to only one microservice. Grouping the nanoentities for microservice extraction uses MSA principles, e.g., coupling. The score of the coupling criteria is used as input to a clustering algorithm that finds the optimal grouping, respecting all the manually imposed constraints. The output is provided as a "service cut". A coupling criteria catalogue comprising architectural requirements such as cohesion, compatibility, constraints and communication patterns guides the semi-automatic decomposition process.

Workload Description: The Service Cutter approach was evaluated by comparison against manual decomposition according to the authors' best knowledge. The inter-service calls are not considered. The evaluation of the Service Cutter consisted of running the decomposition prototype over two applications, a fictitious "Trading System" and one sample application from literature [34], "Cargo Tracking". The resulted cuts were inspected and classified according to a self imposed four-level scoring system ranging from excellent, good, acceptable to bad. The assessment of both decompositions used a deterministic and a non-deterministic clusterization algorithm. The results for both algorithms were "good" for the "Trading System". However, for the "Tracking Cargo System" the deterministic algorithm yielded a "bad" service cut, while the non-deterministic one resulted in an "acceptable" cut.

II. Microservice Identification Through Interface Analysis [20] relies on mapping exposed interfaces to a predefined shared vocabulary (i.e., Schema.org), which can be replaced by any domain-specific ontology to extract microservice candidates from monolithic applications. It addresses quality criteria of microservices, such as granularity, coupling and cohesion. The process of mapping the concepts exposed in the interfaces specifications to the vocabulary uses statistical analysis through co-occurrence functions to match interface's terms with the input vocabulary in order to form decompositions based on semantics. By introducing a reference architecture, and not only relying on authors' best knowledge regarding the service composition, this approach proves more reliable.

Workload Description: The evaluation uses the sample application "Cargo Tracking System" without achieving the expected result [20]. The evaluation used two more applications, "Money Transfer" and "Kanban Board". The first achieved 80% correctness, while the second scored 77% correctness.

IV. Validation Criteria

Metric-based evaluation is compulsory for assessing the quality attributes of microservices. A vast number of quality criteria measuring a variety of aspects concerning microservices exist [27]. However, evaluation approaches are scarce [11], while assessment methods for semi-automatic decompositions are entirely missing [29]. We first define the types of analyses required by a quality attribute. Next, we describe each identified quality attribute and its workload description.

The validation of semi-automatic decomposition methods should take into consideration the type of analysis required for each quality attribute. Therefore, in the following, static analysis together with dynamic analysis are briefly described.

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

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

A. Static Analysis

Next, we present the list of identified criteria that can be assessed via static analysis. Each metric is described and their respective workload descriptions are presented together with guidelines identified in literature.

TABLE I: Quality Criteria Overview. The presence of workload description is indicated by $\oplus$, its absence by $\ominus$.

	Quality Attributes																							
	Static Analysis					Dynamic Analysis																		
Citations	Granularity	LOC	Open Interface	High Cohesion	Loose Coupling	Execution Cost	Response Time	Availability	Succ. Exec. Rate	Usage Frequency	Scalability	Independence	Maintainability	Deployment	Health Management	Modularity	Manageability	Performance	Reusability	Tech. Heterogeneity	Agility	Security	Load Balancing	Org. Alignment
[9]						$\oplus$	$\ominus$	$\ominus$	$\oplus$	$\ominus$	$\ominus$													
[10]			$\ominus$		$\ominus$						$\ominus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\ominus$	$\oplus$	$\ominus$	$\oplus$	$\ominus$	$\ominus$	$\oplus$	$\ominus$	$\ominus$
[11]	$\oplus$		$\oplus$	$\oplus$	$\oplus$						$\oplus$	$\oplus$	$\ominus$	$\ominus$	$\ominus$	$\ominus$	$\oplus$	$\oplus$		$\ominus$		$\ominus$		$\ominus$
[12]		$\oplus$		$\oplus$																				$\ominus$
[13]	$\oplus$	$\oplus$			$\oplus$	$\ominus$	$\oplus$							$\oplus$	$\oplus$			$\oplus$						$\ominus$
[14]	$\oplus$											$\ominus$		$\ominus$						$\ominus$				$\ominus$
[15]													$\ominus$	$\oplus$		$\ominus$	$\oplus$					$\oplus$	$\ominus$	
[16]	$\oplus$			$\ominus$	$\ominus$						$\ominus$	$\oplus$		$\ominus$						$\ominus$				
[17]				$\oplus$	$\oplus$						$\ominus$	$\ominus$	$\ominus$	$\ominus$	$\ominus$	$\ominus$	$\ominus$	$\ominus$	$\ominus$			$\ominus$	$\ominus$	$\ominus$
[18]	$\ominus$			$\ominus$	$\ominus$							$\ominus$	$\ominus$	$\ominus$				$\ominus$			$\ominus$			$\ominus$
[19]			$\ominus$		$\ominus$		$\oplus$	$\ominus$			$\oplus$			$\oplus$	$\ominus$			$\oplus$				$\oplus$	$\ominus$	

Criteria	**Meaning**
Granularity	Size of microservice
LOC	Lines of code
Open Interface	Accessible and standardized interface
Succ. Exec. Rate	Rate of successful executions
Tech. Heterogeneity	Heterogeneity regarding the programming language
Org. Alignment	Alignment with organization's communication patterns

Granularity: The most controversial quality attribute of a microservice is its granularity, also referred as size. The controversy stems from the lack of a generally-accepted definition for the target size of a microservice. From a *workload description* perspective, it can be measured using Lines of Code (LOC) [27] or the number of exposed interfaces [11].

LOC: a metric defined by the number of lines of code written for the software entity in question. Although most of the times granularity is associated with the number of "Lines of Code" [27], it is not a very descriptive metric per se. In industry, the size of a code base represented as LOC ranges from less than 100 LOC to more than 10000 LOC [12], making this metric rather informative than reliable. From a *workload description* perspective, a more useful application of this metric is determining a relative size between microservices, i.e., use statistics to identify components that are relatively "big" inside the same system [27].

Open Interfaces: a metric borrowed from SOA which describes granularity of microservices via the number of open/exposed interfaces. From a *workload description* perspective, the size of a microservice is quantifiable by investigating the number of operations available through the exposed interfaces [11]. The operations can be weighted based on their granularity or number of parameters [27].

Cohesion: represents the degree to which the operations provided by a microservice cater to only one functionality. Although recurrently mentioned in literature, this property is difficult to measure automatically due to its semantic essence. From a *workload description* perspective, cohesion can be measured via the exposed interfaces by asserting the similarity between the parameters' data types [27]. Another approach is considering the invocation behavior of the operations provided by the interfaces, where a ratio is calculated from the number of operations per client and the number of clients times the number of available operations [27]. Average values may be used to determine the components with low cohesion, which are candidates for decomposition [27].

Coupling: represents the dependencies and connections of a microservice to another. A clear guideline is that circular dependencies should be avoided [28]. Hence, when microservices are identified to circularly reference one another, they should became candidates for a merger. From a *workload description* perspective, the degree of coupling can be calculated as the number of calls to a microservice divided by the number of invocations that the microservice is making towards others [27]. Average values may be used to identify microservices that have relative tight coupling (containing redundant invocations), which are candidates for refactorization [27]. A popular method for testing the coupling between microservices is using Docker Swarm, which provides total isolation and decoupling to the containers [13], together with a network protocol analyzer (i.e. WireShark) for identifying the invocations and dependencies.

B. Dynamic Analysis

Next, we present the list of identified criteria that can be assessed via dynamic analysis. Each metric is described and their respective workload descriptions are presented together with guidelines identified in the literature.

Execution cost: Although the cost implications of executing a certain microservice are seldom mentioned in literature, we consider this aspect equally important to others, especially for industry [8], [10]. From a *workload description* perspective, this is an additive attribute, computed as the monetary cost of the resources used for running the microservice [9]. It depends on the deployment platform chosen by the DevOps team. The cost can be estimated at design time and corrected after the implementation is evaluated at execution time.

Response time: the anticipated delay between the time when a request to a microservice is issued and the time when the result is delivered [9]. From a *workload description* perspective, it is a max-operator attribute and its measure unit is seconds [9]. It only considers the execution time, excluding the network delay time (which is dependent on the geo-location of the client). When measuring synchronous calls, it considers the longest response time, while for asynchronous calls, it considers the average time spent [11].

Availability: the proportion of time the microservice is accessible within a required time period and from a *workload description* perspective is usually quantified as a percentage [9]. Most of the contemporary cloud platforms Service Level Agreements (SLA) guarantee availability of 99.9999% or above. This is a multiplicative quality attribute [9], thus a slight decrease in the availability of one microservice that has high coupling is dramatically decreasing the overall availability of the system.

Successful execution rate: the ability of a service provider to successfully fulfil the requests within a given period of time [9]. From a *workload description* perspective, it is measured as a number between 0 and 1 or a percentage [9] calculated as the ratio between successful requests and the total number of requests.

Usage Frequency: the ratio between the requests made to the assessed microservice and all the requests made in the entire system. This is an additive quality attribute [9]. From a *workload description* perspective, we consider it measurable as a number between 0 and 1 or a percentage, although we did not identified a formal description [9].

Scalability: the ability of a microservice to function correctly (as designed) irrespective of the changes in size (amount of resources) without inquiring performance penalties [19].

- Horizontal scalability: replicate the microservice.
- Vertical scalability: increase the amount of resources allocated to a microservice.

From a *workload description* perspective, the degree of scalability can be calculated by analyzing the distribution of synchronous requests provided by the exposed interfaces, a high diversity of requests indicating poor scalability [19].

Independence: has multiple connotations, the most important being the isolation of the microservice with respect to others with which it should only communicate via the exposed interfaces [11]. From a *workload description* perspective, the independence of a microservice can be synthesized via the coupling attribute or calculated via dependency graphs [10]. Other connotations are independence in the technology used for development and independence of the teams developing the microservice [12], [15].

Maintainability: is a broader concept described by a variety of quality criteria such as granularity, technology heterogeneity, coupling and cohesion. From a *workload description* perspective, the maintainability can be represented via multiple metrics such as the number of interfaces exposed by the microservice, number of technologies used, or the number of asynchronous dependencies [11]. Hence, a microservice is considered maintainable if it presents high degree of cohesion, low degree of coupling [27] and homogeneous technology stack.

Deployment: The ease of deployment of microservices is a very desirable quality attribute for industry, usually relying on cloud infrastructure due to its automatic elasticity and on-demand resource provisioning [10]. A typical scenario involves fully automated deployment of containers each running a microservice. From a *workload description* perspective, the quality attribute of deployment is usually tested via execution timelines and instance graphs together with use-case and sequence UML diagrams [10]. The deployment quality attribute is also linked to lowering the execution cost [13] via shortening the development cycles and increasing the microservice maintainability.

Health Management: also known as resiliency to failure [10], is a quality attribute describing the ability of a microservice to cope with failures. A microservice complies to this property by saving the internal state, and restarting automatically while loading the most up-to-date state prior to the failure. This term addresses both compute resiliency (the process can restart) and data consistency (no data loss). From a *workload description* perspective, it is a binary attribute with ”yes” or ”no” values and it is verified via instance graphs or type graphs [10].

Modularity: a quality attribute relying mostly on cohesion through the rule of ”single responsibility” [10]. It is also influenced by the complexity of the system, referring to the granularity of the microservice and the separation of business concerns. Thus, from a *workload description* perspective, is quantifiable via the number of exposed interfaces.

Manageability: is a quality attribute referring to the degree of centralization, where none is best. Additionally, a manageable microservice should be audible and self-managed (or with minimum centralized control) [10]. Similarly to the maintainability, a microservice is considered manageable if it presents a high degree of cohesion and a low degree of coupling. From a *workload description* perspective, it may be calculated by inspecting the dependencies of the microservice [10].

Performance: has different meanings depending on the context [10]. For microservices, performance is usually regarded as a combination of response time and throughput [14]. The throughput of a microservice is the amount of requests that can be processed in one time unit [19]. It is dependent on the technology used for development and heavily related to the degree of internal optimality of the microservice, being measurable at run-time. From a *workload description* perspective, it may be computed as the longest synchronous call or the average size of message, for asynchronous calls [11].

Reusability: is a very desirable quality attribute for industry [4], due to its major cost reduction prospects. Although a microservice development might imply high costs to achieve pluggability, the resulting reusability might in fact save development cost. Moreover, the microservice can be repurposed so that with little changes can be used outside of its design time domain. From a *workload description* perspective, the reusability is shown via deployment UML diagram [10].

Technology Heterogeneity: refers to agnosticism regarding the programming language used to develop the microservice [10]. No formal *workload description* was identified.

Agility: is bound to the context of development and ease of adaptation to minor changes (usually iterative and incremental) [10], which is in turn relying on the deployment platform and its continuous delivery capabilities. From a *workload description* perspective, we consider agility related to the granularity of the microservice.

Security: is a recurrently mentioned quality attribute in the literature. However, we identified little security testing methodologies for microservices. From a *workload description* perspective, manual inspection of execution timelines together with UML diagrams [10] are used in assessing the security of a microservice. This might be caused by the security vulnerabilities being strictly dependent on the domain, implementation or utilized technologies of the microservice.

Load Balancing: represents the workload distribution among microservices and only has sense in the horizontal scalability context [19]. We identified no *workload description* for this quality attribute at a microservice level.

Organizational Alignment: is a quality attribute of a microservice which represents the degree to which, according to Conway’s Law [5], the dependencies between the microservices mirror the communication pattern of the organization that is developing them. The alignment to the microservice paradigm should decrease the overhead between different teams developing different microservices, while maximizing the internal communication efficiency and cohesion between developers implementing the same microservice [10]. We identified no *workload description* for this quality attribute.

V. Discussion

In order for the criteria assessing conformance to the quality attributes to have meaning, they must be discussed in a decomposition context. Therefore, we briefly describe three decomposition methods. Next, we discuss the quality attributes in their respective contexts followed by a case study conducted in collaboration with an industry practitioner. This case study helps validate our findings from an industry perspective.

As the migration process evolved, researchers started to address the problem of architecture decomposition in a more systematic manner. Richardson proposed four partitioning strategies [25]: (1)”Decomposition by business capability” and (2)”Decomposition by sub-domain”, which constitute the most abstract patterns requiring the architect’s intervention for decision making [26]; (3)”Decomposition by verbs or use cases” and (4)”Decomposition by nouns or resources”, which lend better to automation if predefined criteria are considered. ”Decomposition by sub-domain” is referred in the literature also as Domain Driven Design (DDD). Although DDD is one of the most used techniques for shaping microservice architectures, other methods have been used in this area also, such as Architecture-centric Modeling [14] and Data Flow Diagrams (DFD) .

A. Decomposition Methods

The following decomposition methods were considered when selecting the minimum set of attributes due to their high recurrence in the literature as references.

Domain Driven Design. The microservices architecture mainly rely on domain-driven design, a technique in which each microservice must serve a specific and limited functionality that is mirroring the specified business capability via clear service boundaries. The composition has to align with consistency constraints, while the microservices designed from bounded contexts usually are aligned with the domain model boundaries or communication patterns inside the organization [4]. The components (microservices) are created following the isolation principle, being independent between each other and the technology used (including the programming language), the deployment process and the team developing it for the entire lifecycle of the microservice.

Architecture-Centric Modeling. This decomposition method relies on extending the microservices with "ambients" that implement primitives which can be regarded as adaptable boundaries in the business modelling process [14]. The ambients contain aspects that describe the adaptation behaviour required for dynamically altering the granularity, therefore separating divergent business concerns and scoping the boundaries of the microservices.

Data Flow Diagram. This decomposition method follows a top-down approach in which dataflow diagram is created by the architect and the client, that contains the business requirements of the project, later on being iteratively refined until the detailed dataflow diagram that meets the business logic is obtained. An aggregation algorithm, then fuse similar operations that posses the same type of output into a virtual dataflow that is later partitioned by the formal algebra algorithm into microservice candidates.

B. Meta Criteria

Additionally to the quality attributes identified in literature, we propose two meta criteria: meaningfulness and feasibility of implementation. The first considers whether each of the quality attributes is applicable to the microservice at component level, independent of its applicability at system-level. The second considers the existence in the literature of a workload description for each metric. The inclusion is relying on logical **and** between the two meta-criteria.

C. Selecting the minimum set of quality attributes

In the following we challenge each quality attribute identified in the literature and decide its inclusion in the minimum subset of quality attributes based on the two meta-criteria and the attribute's applicability to the decomposition methods.

Granularity: Although no consensus was reached over the de-facto metric for assessing granularity, we do recommend using the two identified. From *feasibility of implementation* perspective, "Lines of Code" is usable only in context of relativity compared to the other microservices of the same system. Furthermore, the number of operations available via exposed interfaces of a microservice is easy to extract from available documentation. Both metrics should be used to determine a relative size value compared to the rest of microservices by using statistics. From a *meaningfulness* perspective, granularity relates to the costs involved in deployment of the microservice and its quality assurance costs. This metric is applicable to all three decomposition techniques with a particularly high flexibility for the Architecture-centric Modeling, which allows variations in granularity.

Cohesion: Being a core principle of MSA, cohesion is included in our minimum recommended quality attribute set. From *feasibility of implementation* perspective, it is measurable by asserting the similarity between the names or parameters' data types of the exposed interfaces. From *meaningfulness* standpoint, the highly cohesive microservices respect the principle of "single responsibility" from MSA. This metric is applicable to all three decomposition techniques.

Coupling: Low coupling, being a core principle of MSA is included in our minimum recommended quality attribute set. From *feasibility of implementation* perspective, it is measurable via calculating the ratio between the number of dependencies other microservices have to it, divided by the number of dependencies it has towards other microservices of the same system. We recommend avoiding circular dependencies and use Docker Swarm for testing the degree of coupling between the components. This metric is applicable to all three decomposition techniques.

Execution Cost: At least for industry practitioners, an important aspect of the software development is represented by costs, therefore from a *meaningfulness* perspective, the execution cost is included in the recommended quality attributes set. From a *feasibility of implementation* perspective, the execution cost is calculated via estimations of cost at design time, based on the resources' price of chosen deployment platform. This metric is orthogonal to all three decomposition techniques.

Response time: From a *meaningfulness* perspective the response time is crucial for ensuring responsiveness/performance of the microservice resulted from migrations. It should be compared to the response time of the monolith for the same request. From a *feasibility of implementation* perspective, response time is measured via programmatically monitoring the elapsed time between issuing a request and receiving a response. The caller of the microservice, calculates the response time (also named latency) as the sum between the network latency and the execution time. Following, the execution time is measurable from inside microservice's code. This metric is applicable to all three decomposition techniques.

Availability: From a *meaningfulness* perspective, the availability is more related to the chosen deployment platform, rather than the microservice as a component, thus it is not included in our minimum quality attribute set. This metric is orthogonal to all three decomposition techniques.

Successful execution rate: From a *meaningfulness* perspective, the successful execution rate is not part of our minimum set of quality attributes, as its composition encompasses already included attributes, namely response time. This metric is orthogonal to all three decomposition techniques.

Usage Frequency: From a *meaningfulness* perspective, the usage frequency is not part of our minimum set of quality attributes. This is due to the fact that it is measurable after the implementation, and concerns other stakeholders. This metric is orthogonal to all three decomposition techniques.

***Scalability**:* Although scalability has meaning in the context of the entire system and not only at microservice's level, we include it in the minimum quality attribute set due to its importance in distributed systems. From a *feasibility of implementation* perspective, it is measurable by assessing the distribution of dependencies and diversity of synchronous requests made by the microservice. This metric is applicable to all three decomposition techniques.

***Independence**:* From a *meaningfulness* perspective, independence of a microservice is synthesizeable from the coupling attribute, thus it is not included in our minimum quality attribute set. This metric is applicable to all three decomposition techniques.

***Maintainability**:* From *feasibility of implementation* perspective, this quality can be extracted from others already included in the minimum set. Fine granularity, decreased level of system's technology heterogeneity and good scalability indicate good maintainability. This metric is applicable to all three decomposition techniques.

***Deployment**:* From *meaningfulness* perspective, the deployment is related to the chosen platform and concerns the entire system, therefore it is not included in our minimum set. From a *feasibility of implementation* perspective, it is tested via execution timelines and instance graphs together with use-case and sequence UML diagrams. This metric is orthogonal to all three decomposition techniques.

***Health Management**:* From a *meaningfulness* perspective the health management of a microservice is crucial for coping to failures. From a *feasibility of implementation* perspective, the health of a microservice can be implemented by throwing an error if the response time exceeds a specified threshold and restarting the container [13]. Having met both meta-criteria it is included in our minimum set. This metric is applicable to all three decomposition techniques.

***Modularity**:* From a *meaningfulness* perspective, due to the fact that modularity can be exposed via other already included quality attributes such as cohesion, granularity, scalability, it is excluded from our minimum set. This metric is applicable to all three decomposition techniques.

***Manageability**:* From a *meaningfulness* perspective, due to the fact that manageability can be exposed via other already included quality attributes such as high degree of cohesion and a low degree of coupling, it is excluded from our minimum set. This metric is applicable to all three decomposition techniques.

***Performance**:* From a *meaningfulness* perspective, it can be assessed through response time, an attribute already included in our minimum set. Thus, performance is excluded. This metric is applicable to all three decomposition techniques.

***Reusability**:* From a *meaningfulness* perspective, reusability also has beneficial cost implications, additionally to repurposing the microservice. From a *feasibility of implementation* perspective, it can be measured via deployment UML diagrams, therefore being included in our minimum quality criteria set. This metric is applicable to the first two decomposition techniques, Domain-Driven Design and Architecture-Centric Modeling, but is orthogonal to Data Flow Diagram due to its focus on data.

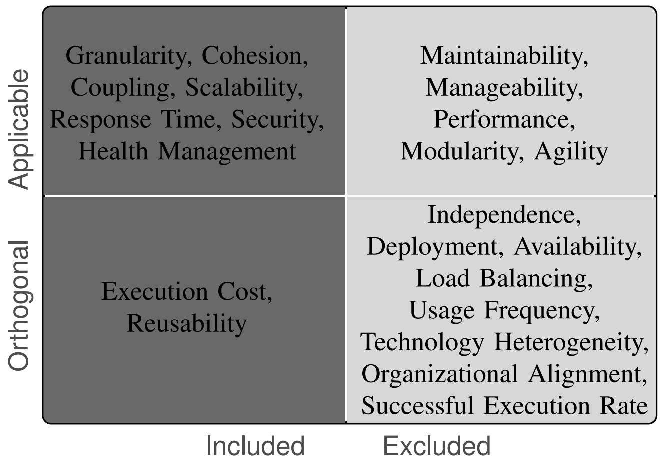
Applicable
Orthogonal
Granularity, Cohesion, Coupling, Scalability, Response Time, Security, Health Management
Maintainability, Manageability, Performance, Modularity, Agility
Execution Cost, Reusability
Independence, Deployment, Availability, Load Balancing, Usage Frequency, Technology Heterogeneity, Organizational Alignment, Successful Execution Rate
Included
Excluded

Fig. 2: Validation attribute categorization.

***Technology Heterogeneity**:* From a *meaningfulness* perspective, this attribute is more related to the chosen deployment platform, rather than to the microservice, thus it is not included in our minimum quality attribute set. This metric is orthogonal to all three decomposition techniques.

***Agility**:* From a *meaningfulness* perspective, this attribute can be expressed using granularity, an already included quality. Thus, agility is excluded from our minimum set. This metric is applicable to all three decomposition techniques.

***Security**:* From a *meaningfulness* perspective, security is a required quality attribute that every software has to posses, therefore, being included in our minimum quality attribute set. From a *feasibility of implementation* point of view, the security aspect is dependent on the domain, implementation or technologies used in development. However, as a general guideline, the security is usually assessed via inspection of execution timelines and UML diagrams. This metric is applicable to all three decomposition techniques.

***Load Balancing**:* From a *meaningfulness* perspective, this attribute is more related to the chosen deployment platform, rather than the microservice, thus it is not included in our minimum quality attribute set. Furthermore, this metric is orthogonal to all three decomposition techniques.

***Organizational Alignment**:* From a *meaningfulness* perspective, it has advantages such as decreasing the overhead of different teams developing different microservices and maximizing the internal communication efficiency. From a *feasibility of implementation* perspective, we identified no workload description, thus leading to its exclusion from our minimum quality attribute set. This metric is orthogonal to all three decomposition techniques.

Minimum set of quality attributes. From the exhaustive initial set of literature identified quality attributes, we refined a minimum set of quality attributes (Figure 2) that should be considered when assessing a microservice resulted from a monolithic architecture migration process. We based our refinement process on two self-proposed meta-criteria: meaningfulness and feasibility of implementation.

D. Case Study

Since our minimum set of quality attributes contains characteristics that concern industry practitioners, we interviewed an expert. The interviewee was directly involved in decomposition processes. By providing accurate insights from industry his opinion is valuable to our study.

Validation of motivation: When asked about the issues of monolithic architectures and the catalyst of architectural shift, the practitioner first mentioned the tedious process of developing monoliths. The monoliths were often difficult to understand by new programmers who recently joined the development team. New employees usually spent their first couple of months on the project digesting the architecture and data flow of the application. From a business perspective the monolithic applications usually required long periods of time to develop and debug, stretching the development cycles. Additionally, deployments challenges generated by quality assurance (QA), limited the number of releases at two per year. Intriguingly, the costs involved in deploying the application were considered minor priorities, the majority of costs being inflicted by development.

Validation of migration processes: When asked about the migration process from monoliths to microservices, the expert described the process as an incremental transition from the monolith. Firstly, functionality is extracted and converted to a microservice which is then connected to the monolith. This technique of incrementally externalizing functionality has the positive side-effect of cost effective addition of new features to the application. This is achieved simultaneously with the externalization aspect of the migration. The load is incrementally transferred from the monolith to the new microservices via A/B testing.

Validation of migration type: When asked about which is the cheaper option between a full rebuild and a migration, the expert acknowledged that for large scale applications the rebuild is not a viable option due to time and cost implications. Nonetheless, there were also a small number of cases where the rebuild was preferred due to intricacies of the legacy implementation.

Validation of granularity: When asked about metrics used in industry for assessing the *granularity* of microservices, the expert acknowledged that many industry practitioners still rely on LoC (lines of code). Another metric used is the amount of endpoints multiplied by the amount of methods to assess the granularity of microservices. There are several heuristics used in practice for determining the size of the team therefore scoping the granularity of the microservice, the most recurrent being the pizza team size. Although the use of relative sizes between components is a promising start for assessing the granularity, it is not yet considered the ”holy grail” due to it’s poor accuracy in less granular projects. Considering the situation in which the granularity is extremely low (two components) the metric would not provide useful information. Industry practitioners often combine multiple metrics that are scored in various forms for assessing a quality attribute.

Validation of cohesion: When asked about the use of semantic analysis on the interfaces’ names for assessing the *cohesion* of microservices, the industry practitioner deemed the approach very creative yet not practical for industry use. In the industry the main assessing technique of this quality attribute still relies on the subjective opinion of the architect. Usually together with the increase in size of components, practitioners start to address the possibility of isolating functionality in groups of business functionality. In such cases, following the concept of Bounded Contexts [31], the components are considered candidates for division.

Validation of coupling: When asked about practical techniques of assessing *coupling* of microservices, the expert described similar approaches to the assessment of cohesion. Additionally, in industry architects search for bidirectional communication paths in order to detect high coupling degree between components. It is considered good practice to avoid bidirectional communication altogether due to their inherent dependencies that increase the coupling.

Validation of cost implications: In regards to the *execution costs*, the industry expert presented an example of a client who run an ad-hoc simulation for predicting costs. Nonetheless, he agreed that cost estimations usually consists of a mix between intuition and experience. In some cases detailed predictions were made at design time, which was only possible in the context of detailed knowledge about the expected workload. In the industry, a back-track from deployment stage to the design stage is rarely encountered due to major costs implications. Furthermore, the execution cost is often considered negligible in the context of hiring experienced developers (which in US cost up to 200k USD/year). A conclusion of such examples is that shortening the development cycles is usually the main goal, irrespective of small additional costs that may occur.

Validation of response time: When asked about the assessment of microservices’ *response time*, the industry expert stated that latency is predominantly used and agreed that the deployment platforms usually track the endpoints themselves. In industry, automatic metric assessment is done via tools such as Prometheus (or any other metric collector).

Validation of scalability: In regards to *scalability* of microservices, the expert acknowledged that in industry there is one overused metric: whether the microservice is stateless or not. A system composed from stateful microservices is more difficult to scale. The assessment of this quality attribute is aided by the use of HTTP workload generators (e.g ApacheBench [32]).

Validation of health management: When asked about practical techniques of assessing *health management* of microservices, the industry expert stated that the common approach consists of manually or automatically producing chaos inside the system by using tools such as Netflix's ChaosMonkey [33]. Often it happens manually, the industry practitioners removing a component from the system in order to test system's resiliency.

Validation of reusability: When asked about assessing *reusability* of microservices, the expert emphasized that reusability is strictly dependent on the specificity of the application's domain. An ad-hoc produced solution was be to inspect the dependency graph of the system and detect the coupling. A service with low coupling is often deemed reusable inside the same system. However this technique does not provide insights about reusability of the service in the context of another system. Another approach is to open-source the service and monitor the number of stars (followers) it acquires on Github indicating how many systems might be reusing the service.

Validation of security: In regards to the assessment of microservice's *security*, the industry expert acknowledged that the security is not bound to the microservice itself but rather dependent on the technologies used and the particularities of the application. Nonetheless, in his opinion, the microservice architecture at least give the option of more security by providing a way to isolate the vulnerability (sensitive data such as credit card data, identities).

The interview does predominantly match our approach in assessing the quality attributes of microservices included in our minimum set, therefore providing food-for-thought for researchers investigating a validation framework.

VI. Related Work

A number of studies [28], [30], [31] have discussed the validation process involved in migrations from monoliths to microservices. However, our approach differs by focusing on identifying the minimum comprehensive set, general and commonly applicable in the industry for validating microservice migration cuts. Figure 3 shows the occurrence of attributes from our minimum set across the considered studies.

A list of criteria was assembled [27] with a focus on maintainability, while analyzing the criteria with respect to their applicability to the microservices architecture and, separately, their measurability [29].

A similar study [30] identifies a set of constraints and validation criteria prone to automation of assessing the conformance to microservice architecture. However, their focus consists of two aspects: independence and shared dependencies between the components, while our study refines an exhaustive set of quality attributes based on two meta-criteria: meaningfulness regarding the decomposition context and the feasibility of implementation.

Other research validates the authors' approach with respect to: (a) a limited set of "relevant principles of microservices": granularity, cohesion, coupling, and technology heterogeneity [16]; (b) a self-made expected architecture [4]; (c) independence, deployment, standardized interfaces, and granularity of a microservice [15].

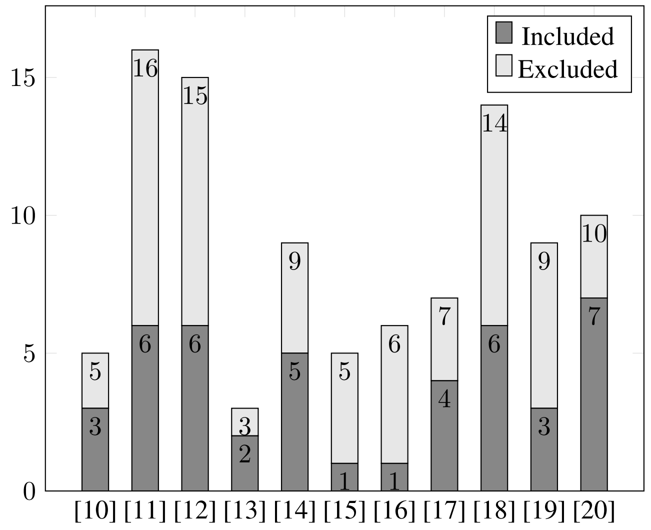

Fig. 3: Minimum set of attributes occurrence in literature.

A similar study that considered the comprehensive set of quality criteria for microservices lacks the analysis of feasibility and required workloads for testing [17].

Research conducted on a broad range of quality attributes of SOA [19] also encompasses the entire lifecycle of a microservice, while our focus is on identifying the minimum set of quality attributes applicable to microservice resulted from semi-automatic decompositions.

VII. Conclusions and Future Work

In this study we aimed to provide a sufficient and necessary set of quality attributes applicable to microservices resulted from semi-automatic decomposition tools or techniques. We identified an exhaustive set of quality criteria used in the literature considering both MSA and SOA.

Next, we discussed the inclusion of each identified quality attribute into our minimum recommended set of quality attributes based on two meta-criteria: meaningfulness in the decomposition context and feasibility of implementation and augmented the discussion with a case study based on an interview of an industry practitioner.

Although work still has to be done, this area is in full maturation process. Identifying a subset of quality criteria for microservices that are prone to automatic validation paved the way to implementing an automatic validation tool. The future tool should focus on assessing the quality of microservices resulted from semi-automatic use of the above mentioned decomposition techniques, creating a complete solution for migrating monolithic applications to MSA.

References

[1] M. Fowler and J. Lewis, Microservices, 2014. [Online]. Available: https://martinfowler.com/articles/microservices.html
[2] M. Richards. "Microservices vs. Service-Oriented Architecture". O'Reilly Media, 2016.

[3] P. D. Francesco, I. Malavolta and P. Lago, ”Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption”, in International Conference on Software Architecture (ICSA), Gothenburg, 2017, pp. 21-28.

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

[5] Conway, M.: Conways Law, [Online]. Available: http://melconway.com/Home/Conways_Law.html, last accessed 19/11/2018.

[6] E. van Eyk, L. Toader, S. Talluri, L. Versluis, A. Uta and A. Iosup, ”Serverless is More: From PaaS to Present Cloud Computing,” in IEEE Internet Computing, 2018 pp. 8-17.

[7] Dragoni, Nicola, Saverio Giallorenzo, Alberto Lluch-Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin and Larisa Safina, Microservices: Yesterday, Today, and Tomorrow, in Present and Ulterior Software Engineering (2017).

[8] G. Granchelli, M. Cardarelli, P. D. Francesco, I. Malavolta, L. Iovino and A. D. Salle, ”Towards Recovering the Software Architecture of Microservice-Based Systems”, International Conference on Software Architecture Workshops (ICSAW), Gothenburg, 2017, pp. 46-53.

[9] Sherry X. Sun and Jing Zhao, ”A decomposition-based approach for service composition with global QoS guarantees”. Inf. Sci. 199, 2012, pp. 138-153.

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

[11] Engel, Thomas, Melanie Langermeier, Bernhard Bauer and Alexander Hofmann. Evaluation of Microservice Architectures: A Metric and Tool-Based Approach, CAiSE Forum, 2018.

[12] G. Mazlami, J. Cito and P. Leitner, ”Extraction of Microservices from Monolithic Software Architectures”, in International Conference on Web Services (ICWS), Honolulu, HI, 2017, pp. 524-529.

[13] J. Gouigoux and D. Tamzalit, ”From Monolith to Microservices: Lessons Learned on an Industrial Migration to a Web Oriented Architecture”, in International Conference on Software Architecture Workshops (ICSAW), Gothenburg, 2017, pp. 62-65.

[14] S. Hassan, N. Ali and R. Bahsoon, ”Microservice Ambients: An Architectural Meta-Modelling Approach for Microservice Granularity”, in International Conference on Software Architecture (ICSA), Gothenburg, 2017, pp. 1-10.

[15] D. I. Savchenko, G. I. Radchenko and O. Taipale, ”Microservices validation: Mjolnirr platform case study”, in 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, 2015, pp. 235-240.

[16] R. Chen, S. Li and Z. Li, ”From Monolith to Microservices: A Dataflow-Driven Approach” in 24th Asia-Pacific Software Engineering Conference (APSEC), Nanjing, 2017, pp. 466-475.

[17] Shahbaz Ahmed, Khan Ghayyur, Abdul Razzaq, Saeed Ullah and Salman Ahmed, ”Matrix Clustering based Migration of System Application to Microservices Architecture”, in International Journal of Advanced Computer Science and Applications (IJACSA), 2018, vol. 9, no. 1.

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

[19] L. Bass, P. Merson and L. OBrien, ”Quality attributes for Service-Oriented Architectures”. Department of Defense, Technical Report September, 2005.

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

[21] J. Lin, L.C. Lin, S. Huang, ”Migrating web applications to clouds with microservice architectures”,in International Conference on Applied System Innovation (ICASI), Okinawa, 2016, pp. 1-4.

[22] M. Razavian and P. Lago, ”Understanding SOA migration using a conceptual framework”, in Journal of Systems Integration”, 2010, pp. 33-43.

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

[24] B. Kitchenham and S. L. Pfleeger. ”Principles of survey research: part 5: populations and samples”, in ACM SIGSOFT Software Engineering Notes, 2002, pp. 17-20.

[25] C. Richardson, ”Pattern: Microservice architecture”, 2017. [Online]. Available: http://microservices.io/patterns/microservices.html

[26] E. Evans, ”Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design”. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

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

[28] D. Rud, A. Schmietendorf, and R. R. Dumke, ”Product Metrics for Service-Oriented Infrastructures”, in IWSM/MetriKon, 2006.

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

[30] U. Zdun, E. Navarro, F. Leymann, ”Ensuring and Assessing Architecture Conformance to Microservice Decomposition Patterns”, in ICSOC, 2017, pp. 411-429.

[31] M. Fowler, ”Bounded Context” 15 January 2014. [Online]. Available: https://martinfowler.com/bliki/BoundedContext.html

[32] ”ApacheBench (ab)”. [Online]. Available: https://httpd.apache.org/docs/2.4/programs/ab.html

[33] ”Netflix: ChaosMonkey”. [Online]. Available: https://github.com/Netflix/chaosmonkey

[34] E. Evans, ”Domain-Driven Design: Tacking Complexity in the Heart of Software”, published by Addison-Wesley Longman Publishing Co., Boston, USA, 2004.