Quality Metrics in Software Architecture

Samira Silva
Gran Sasso Science Institute (GSSI)
L'Aquila, Italy
samira.silva@gssi.it

Adiel Tuyishime
Gran Sasso Science Institute (GSSI)
L'Aquila, Italy
adiel.tuyishime@gssi.it

Tiziano Santilli
Gran Sasso Science Institute (GSSI)
L'Aquila, Italy
tiziano.santilli@gssi.it

Patrizio Pelliccione
Gran Sasso Science Institute (GSSI)
L'Aquila, Italy
patrizio.pelliccione@gssi.it

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

2023 IEEE 20th International Conference on Software Architecture (ICSA) | 979-8-3503-9749-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICSA56044.2023.00014

***Abstract*—The importance of software architecture is largely recognized also in iterative and agile development settings. However, it is quite complex to provide evidence that an architecture is of good quality and that the architectural decisions are appropriate, correct, or optimal. Architecture evaluation aims at showing and providing confidence that design decisions contribute to fulfilling the stakeholder concerns. Some architecture evaluation methods are scenario-based and aim at balancing many potentially conflicting quality attributes. Other works focus on a specific quality attribute and provide metrics to measure it.**

In this paper we survey the state of the art in metrics for evaluating quality attributes of architectures. The elicited metrics are organized into a catalog, which associates them with the specific quality attributes they aim to measure. We contribute also an MDE framework that generates web views facilitating the analysis of architectures. In this way, researchers and practitioners can easily retrieve the metrics that are appropriate to their specific needs. The catalog of metrics and quality attributes is released to the research community and open to contributions from experts and practitioners.

***Index Terms*—Quality Attributes, Quality Metrics, Software Architecture.**

I. Introduction

In stage-gate software development processes, software architecture is a high-level abstraction of the system, which (i) provides a blueprint and guides the implementation, (ii) balances different and potentially conflicting stakeholder concerns, and (iii) is informed by high-level product requirements and business goals [1]. In iterative and agile development processes, software architecture should be a (continuous) stream of decisions and can even facilitate and enable agility, when developing large and complex systems [2].

Because of its role, the architecture must be of "good quality". The work in [1] discusses various possible meanings of good quality, including: (i) the alignment of the architecture with stakeholder concerns and business goals [3], (ii) alignment, over time, of architecture descriptions with the implementation [4], and (iii) completeness of the architecture description with respect to stakeholder concerns [5].

Various approaches exist to evaluate software architectures. Scenario-based evaluation methods focus on evaluating scenarios that express some quality attributes and make them concrete [1], [6], [7]. These methods require to express quality attributes, e.g., maintainability or availability, as scenarios, and to evaluate the response of the system under evaluation to stimuli triggered by the formulated scenarios. One important aspect of scenario-based methods is the tradeoff analysis among various quality attributes, which can easily be in conflict among them (e.g. Architecture Tradeoff Analysis Method (ATAM) [6]). Besides scenario-based methods, there have been various approaches proposing precise metrics to evaluate specific quality attributes [8]–[11]. A metric or a measure aims at assigning a value related to the exhibition of a determined quality attribute in a software architecture [12].

In this paper, we focus on approaches that propose the evaluation of software architectures based on metrics. The work in [9] inspired this work even though it proposes a set of metrics to evaluate only object-oriented designs of software architectures. The work in [8] performed a systematic review and collected 40 metrics. It considers also metrics that take as input the system implementation. Instead, we focus on metrics that only require an architecture description, architecture-only metrics, to make the evaluation. Moreover, differently from us, the work in [8] focuses on the sustainability of software rather than on quality in general. The work in [10] concentrates on architecture-only metrics. 25 collected metrics are mapped to the literature and their maturity is computed according to the frequency of appearance. Finally, the work in [11] focuses on metrics to evaluate the architecture of large systems for industrial applications. Their metrics are categorized into architecture measures, design stability, and technical debt.

In this work, we aim to collect and offer to the reader an organized catalog of quality attributes and metrics to evaluate software architectures; researchers and practitioners can then easily retrieve and use the metrics that are more appropriate to their specific needs. Specifically, we aim to answer the following research questions:

RQ1: *Which metrics and quality attributes have been used to evaluate software architecture?*
RQ2: *Are the collected metrics generic or specific?*
RQ3: *Which of the collected metrics are internal and which ones are external?*

RQ4: *How the collected quality attributes and metrics can be used in practice?*

RQ1 (answered in Sections III-A and III-B) aims at identifying metrics and attributes defined to evaluate software architectures. Then, we include in the study only papers presenting quality attributes and/or well-formulated metrics. It is important to highlight that we focus on metrics that only need an architecture description as input. We then exclude from the current survey works that propose metrics that, e.g. require as input an architecture description and the system implementation. According to [13], architecture description is a "work product used to express an architecture".

RQ2 (answered in Section III-C) aims at investigating whether the collected metrics are generic or specific. Generic metrics can be used for evaluating any architecture since they focus on general-purpose issues. Specific metrics concentrate on issues of a particular domain, e.g. automotive or technology.

RQ3 (answered in Section III-D) aims at categorizing the collected metrics as internal or external [14]. Internal quality metrics measure qualities of the software architecture concerning how the system has been constructed, like maintainability, testability, and reusability. Whereas, external quality metrics measure quality attributes of software architecture as perceived from outside, like correctness, usability, and efficiency.

RQ4 (answered in Sections IV and V) aims at showing how the catalog of quality attributes and metrics can be used in practice by various stakeholders. To answer this research question, we developed a Model-Driven Engineering (MDE) [15] framework that generates aggregated views of the metrics, quality attributes, and their relationships. The framework is based on a textual Domain Specific Language (DSL) [16] that we used to implement the catalog. We show through some scenarios how various stakeholders can use it and how they can contribute in populating and refining the catalogue.

The research methodology (see Section II) we use in this work to collect metrics and quality attributes is a Systematic Literature Review (SLR) [17]. We focus on papers published from 2011 to 2022. The collected metrics and quality attributes have been categorized according to the product quality model of the ISO/IEC 25010:2011 standard [18].

II. Research Methodology

To conduct this SLR, we followed the guidelines proposed by [17]. As shown in Figure 1, our methodology is divided into the following six phases: (i) research questions, (ii) search strategy, (iii) screening, (iv) selection criteria, (v) final list selection, and (vi) findings reporting. Considering the research questions, we defined the following search string:

```
[("software architecture") AND
(evaluat* OR assess* OR analy*) AND
(quality) AND (attribute* OR metric*)]
```

The databases in which the query above mentioned is applied are: IEEExplore, ACM Digital Library, Scopus, Springer Link, and ScienceDirect. The search query has been adapted to each database employed. We considered only papers published from 2011 to 2022 and the search was performed on February

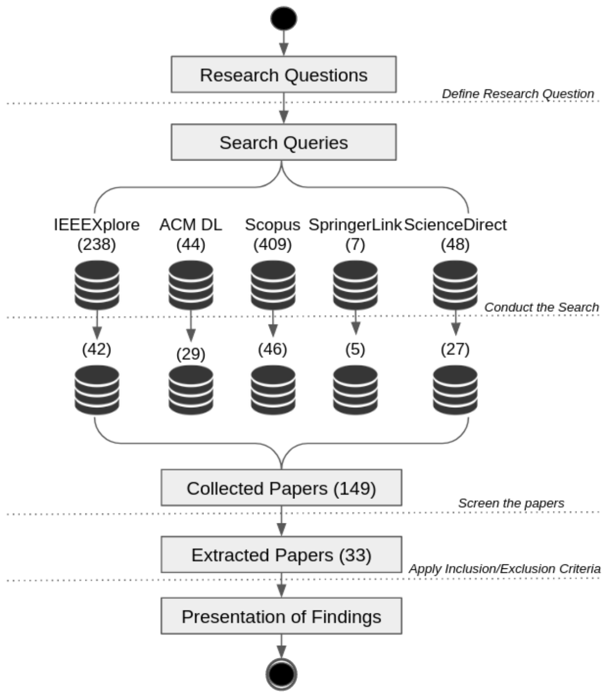


Fig. 1. Summary of the selection protocol of this SLR.

15th, 2022. To reduce the number of irrelevant search results, we have applied our search over the *abstract*, *title*, and *keywords* of the papers under investigation. For the databases that do not provide the flexibility to search the string only in the title, abstract, and keywords, i.e. ACM and ScienceDirect, we used the tool Zotero[1] to guarantee that the query is being applied only in these three fields. At this first stage, the number of records retrieved for IEEExplore, ACM Digital Library, Scopus, Springer Link, and ScienceDirect was 238, 44, 409, 7, and 48, respectively.

Since many papers matched the query but were not relevant, we screened the papers by only reading the title and abstract and removed those which were clearly irrelevant. Then, merging the results obtained from screening into a single set, and removing the duplicates, 149 papers have been collected.

The search scope was restricted to the architectural metrics that take as input an architectural model and a required quality specification, expressed by attributes or composition of metrics. In our inclusion criteria, we only include journal articles, conference papers, and book chapters. We have excluded other sorts of papers, such as books, reports, and case studies. Also, papers that conduct secondary studies or were written in a language other than English were excluded. Finally, papers that were unable to download were also removed. Table I summarizes the selection criteria.

The 149 records identified by merging the results of the screening phase for the 5 different databases have been analyzed independently by at least 3 co-authors to decide whether they should be included or not. We found 5 papers with

[1]https://www.zotero.org

contradictory opinions. After applying selection criteria and discussing the conflicts, 116 records were identified to be irrelevant to our topic and were excluded. Then, we reached a final consensus and included 33 papers in the final list. Figure 1 shows the complete protocol employed in this SLR. We make available a replication package of this study online [19].

Threats to validity. Internal validity threats focus on the design of the study and specifically on whether claims are supported by the obtained data [20]. We mitigated these threats to validity by defining a research protocol to conduct this study employing well-established guidelines for systematic studies in software engineering [17]. All authors reached a consensus while developing and validating the protocol.

Construct validity threats concern the relation between the theory and the observation [20]. The protocol we followed gives us the confidence that the selected primary studies are representative of the population defined by the research questions. Moreover, we allow an independent replication of our study through a replication package that documents all steps of our research. For what concerns data extraction, three of the authors repeated the process of extracting data from the studies individually. Other two authors were supervising the process and included in the case of doubts. In this event, authors discussed upon reaching a consensus.

External validity threats focus on the generalisability of the final observed results and outcomes of the study [20]. We defined a framework for exploring quality attributes and metrics that is general and extensible. Our aim is in fact to involve the community and add new metrics or quality attributes, or refine the existing ones according to feedback from the community. On the website, we provide a feature to enable the contribution from external researchers.

Conclusion validity threats concern the credibility of the conclusions drawn from the extracted data [20]. To show the relationship between the extracted data and the obtained findings, we rigorously followed recognized guidelines. The replication package we provide documents every step of our research, thus promoting transparency and replicability.

TABLE I
INCLUSION AND EXCLUSION SELECTION CRITERIA.

Inclusion criteria	*Exclusion criteria*
I1 - Papers that describe quality attributes or/and that provide a precise formulation for quality metrics to evaluate software architecture.	**E1 -** Studies in languages other than English.
I2 - Papers containing quality metrics (if presenting quality metrics) that only need an architecture description as input.	**E2 -** Duplicate and out-of-scope papers (i.e. not fulfilling I1 or I2)
I3 - Papers published from 2011 till 2022.	**E3 -** Papers published before 2011.
I4 - Conference papers, articles, book chapters.	**E4 -** Papers published other than on conferences, journals, or books.
	E5 - Secondary studies papers.
	E6 - Papers that cannot be downloaded.

III. Results and Discussion

In this section, we report and analyze the results obtained in this systematic literature review. We discuss the results by providing an answer to RQ1 (see Sections III-A and III-B), RQ2 (see Section III-C), and RQ3 (see Section III-D).

A. Attributes and metrics for evaluating software architectures

In this section, we aim at answering the research question RQ1: *Which metrics and quality attributes have been used to evaluate software architecture?* To answer this question, we identified in the literature the quality attributes and metrics that have been used to evaluate architectures. We clustered and organized the attributes and metrics according to the ISO/IEC 25010:2011 standard [18]. This standard defines a product quality model that contains eight *quality characteristics* to take into account when evaluating the properties of a software product: Functional Suitability, Performance Efficiency, Compatibility, Usability, Reliability, Security, Maintainability and Portability. In the following, we associate quality attributes and metrics with quality characteristics. We also discuss in depth some of the quality attributes and related metrics while in Section III-B we provide a complete overview of them. Finally, we provide a larger description of each quality attribute and metric on the website[2] by reporting the various definitions that are provided in the surveyed papers.

① **Functional Suitability**: the degree to which a system provides functions that meet stated and implied needs when used under specific conditions [18]. The quality attributes related to this characteristic and gathered from the papers are: *Completeness*, *Correctness*, *Consistency*, *Functionality*, and *Suitability*. Several metrics have been collected for these attributes and a complete description of them is given on the already mentioned website. Here we present a brief description of the metrics related to *Suitability*. It is defined as the "proportion of components in the software architecture that is well fitted for the required domain" in [21], [22]. On the other hand, the work in [23] describes this attribute according to the definition of Functional Suitability of ISO/IEC 25010:2011 [18] (see above). Among the collected papers, we found three metrics to evaluate Suitability: *Distance* and *Suitability*.
  - *Distance.* It is defined as the balance between instability and abstraction. Distance values close to 0 indicate good balance, whereas values close to 1 show skewness between abstraction and dependency [24]. Besides Suitability, *Distance* is also described as a metric to evaluate Security and Stability.
  - *Suitability.* It is defined in [21], [22] as $X = A/B$, where A is the number of components in the software architecture that is well fitted for the required domain

[2]https://tiziasan.github.io/QAandMetricsForArch/

and B is the total number of components in the software architecture. *Suitability* is also seen as a metric to describe Simplicity.

2. **Performance Efficiency**: the performance relative to the amount of resources used under stated conditions [18]. The quality attribute related to this characteristic and gathered from the papers is *Performance*. *Performance* is defined as the degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage [25]. The work in [26] states that "it demonstrates the capability of the software product to provide appropriate performance, related to the amount of resources used in certain circumstances". This QA is mentioned in various other works [23], [26]–[32]. Two metrics have been collected for this attribute: *Response Time* and *Throughput*.
   - *Response Time.* The works in [28], [31], [32] describe the software architecture through an Architectural Description Language (ADL). They compute the response time for a component output port as a combination of the response time of its input ports.
   - *Throughput.* The works in [28], [31], [32] describe the software architecture through an ADL. The throughput for a component to process the data is calculated by a combination of the throughput of its input ports. In [29] throughput is defined as the number of documents successfully displayed for users in a given interval, considering a software architecture simulated with Stochastic Activity Networks. This metric is also described to measure Sustainability.
3. **Compatibility**: the degree to which a system or component can exchange information with other systems or components, and/or perform its required functions while sharing the same hardware or software environment [18]. The quality attributes related to this characteristic and gathered from the papers are *Compatibility* and *Interoperability*. Several metrics have been collected for these attributes and a complete description of them is given on the already mentioned website. We here describe *Interoperability*. *Interoperability* is defined as the "proportion of components in the software architecture that is easy to integrate with other systems" [21], [22]. One metric has been collected for this attribute: *Interoperability*.
   - *Interoperability.* It is the proportion of components in the software architecture that is easy to integrate with other systems [21], [22].
4. **Usability**: the degree to which a system can be used by specific users in a specific context to achieve specific goals with effectiveness, efficiency, and satisfaction [18]. The quality attributes related to this characteristic and gathered from the papers are: *Discoverability*, *Flexibility*, *Simplicity*, *Understandability*, and *Usability*. Several metrics have been collected for these attributes and a complete description of them is given on the already mentioned website. Here we present a brief description of the metrics related to *Understandability*. *Understandability*, when considering the software architecture context, concerns whether the SA is understandable to average architects [33]. The work in [34] still states that "there is no general agreement on what 'software understanding' means exactly". Among the collected papers, five metrics to evaluate this attribute have been found: *Coupling/Decoupling*, *No.of Runnables and Inter-runnable per Software*, and *Understandability*.
   - *Coupling/Decoupling.* Coupling Factor (COF) measures how tightly the system's classes are coupled. It is computed by dividing the total number of references from classes to other classes divided by $n^2 - n$, where n is the number of classes in the system [23]. Coupling is also seen as a metric to describe other quality attributes. They are: Accountability, Analyzibility, Coupling, Complexity, Confidentiality, Flexibility, Functionality, Integrity, Maintainability, Modifiability, Modularity, Reliability, Reusability, Robustness, and Testability.
   - *No.of Runnables and Inter-runnable per Software.* It is the total number of runnable and inter-runnable variables in the software components, respectively [35]. This metric also measures Complexity and Security.
   - *Understandability.* It can be calculated as the number of connections between components divided by $(N^2 - N)$, where N is the number of components in the software architecture [33].
5. **Reliability**: the degree to which a system or component performs specific functions under specific conditions for a specific period of time [18]. The quality attributes related to this characteristic and gathered from the papers are: *Availability*, *Maturity*, *Reliability*, *Responsibility*, *Robustness*, and *Stability*. Several metrics have been collected for these attributes and a complete description of them is given on the already mentioned website. Here we present a brief description of the metrics related to *Availability*. *Availability* is defined as the proportion of components that is well suitable to handle a large amount of data in the software architecture [21], [22]. Also, the work in [25] describes availability as the limit of the probability that the system is functioning correctly at time t, as t approaches infinity. Finally, Availability is seen as a measure of readiness for correct service in [36]. This attribute is also mentioned in [30], [31]. Among the collected papers, three metrics to evaluate this attribute have been found: *Availability*, *MTBF*, and *Reliability*.
   - *Availability.* It is defined in [21], [22] as $X = A/B$, where A is the number of components in the software architecture that is well suitable to handle a large amount of data B is the total number of components. In [25], Availability is measured as the ratio between the Mean Time To Failure (MTTF) over the sum of Mean Time To Repair (MTTR) and MTTF. Finally, the

work in [36] states that it is the "proportion of time the system is able to provide its intended level of service, i.e., operates without failure".

- *MTBF.* It computes the Mean Time Between Failures (MTBF) for a component output as a combination of the MTBF of its input ports [31].
- *Reliability.* Reliability fails when at least one of the components of the system fails, that is, when at least a component is in a state of failure [29]. This metric also measures Maintainability and Reliability.

6) **Security**: the degree to which a system protects information and data so that persons or other systems have the degree of data access appropriate to their types and levels of authorization [18]. The quality attributes related to this characteristic and gathered from the papers are: *Accountability*, *Authenticity*, *Confidentiality*, *Integrity*, *Non-repudiation*, and *Security*. Several metrics have been collected for these attributes and a complete description of them is given on the already mentioned website. Here we present a brief description of the metrics related to *Authenticity*. *Authenticity* is a "degree to which the identity of a subject or resource can be proved to be the one claimed" [37]. Four metrics have been collected for this attribute: *Cohesion*, *Complexity*, *Inheritance*, and *Polymorphism*.
   - *Cohesion.* It evaluates how closely linked the methods and attributes in a class are. Strong cohesiveness is indicated by a high degree of overlap between method parameters and attribute types, according to [37]. Cohesion is also related to the attributes: Accountability, Analyzability, Cohesion, Integrity, Maintainability, Modifiability, Modularity, Non-repudiation, Reusability, and Testability.
   - *Complexity.* A measurement of how difficult it is to comprehend the relationship between classes and their internal and exterior structures, according to [37]. Complexity is also related to the attributes: Analyzability, Complexity, Maintainability, Modifiability, Modularity, Non-repudiation, Reliability, Reusability, Robustness, and Testability.
   - *Inheritance.* It measures the "is-a" relationship between classes [37]. Inheritance is also related to the attributes: Analyzability, Integrity, Maintainability, Modifiability, Modularity, and Testability.
   - *Polymorphism.* It measures the ability to substitute objects with identical interfaces at runtime [37]. Polymorphism is also related to other attributes, such as Accountability, Complexity, Maintainability, Reusability, Security, and Testability.

7) **Maintainability**: the degree of effectiveness and efficiency with which a system can be modified for improvement, correction, or adaptation to changes in the operational environment and/or requirements [18]. The quality attributes related to this characteristic and gathered from the papers are: *Analyzability*, *Clearness*, *Cohesion*, *Coupling*, *Complexity*, *Evolvability*, *Extendibility*, *Maintainability*, *Modifiability*, *Modularity*, *Reusability*, *Sustainability*, and *Testability*. Several metrics have been collected for these attributes and a complete description of them is given on the already mentioned website. Here we present a brief description of the metrics related to *Clearness*. *Clearness* is defined as the extent to which the documentation is appropriate for the intended audience, e.g., "can a developer successfully implement the system based on the architecture model?" [38]. Among the collected papers, three metrics to evaluate this attribute have been found: *Best Practices Adherence*, *Decision Traceability*, and *Decomposition*.
   - *Best Practices Adherence.* It measures the use of patterns or tactics appropriately in software architecture design. It is computed as the number of concerns appropriately addressed divided by the total number of concerns [38].
   - *Decision Traceability.* It characterizes how well architecture design decisions are connected to requirements by computing the proportion of architecture design decisions(ADD) that are linked to requirements [38].
   - *Decomposition.* It indicates whether or not modules are decomposed into sub-modules such that all modules keep a similar degree of decomposition at the lowest level [38].

8) **Portability**: the degree of effectiveness and efficiency with which a system or component can be transferred to another hardware, software, or operational environment [18]. The quality attributes related to this characteristic and gathered from the papers are: *Adaptability*, *Portability*, and *Scalability*. Various metrics have been collected for these attributes and a complete description of them is given on the website. Here we focus on *Portability*, which is defined in [39] as the degree to which a system or component may be transferred effectively and efficiently from a hardware, software, or operational and/or usage environment to another one. The work in [26] defines *Portability* as the ability to run the system under various computing environments. The work in [25] highlights that the environment may include organizational, hardware, or software environment. *Portability* is also mentioned in [23]. *Dependency* is the metric we found to evaluate this attribute.
   - *Dependency.* It is defined as the amount of dependency of interfaces on basic software [39]. It is also computed as the total edges including the weights of the edges divided by the number of vertices in a dependency graph [24].

B. Summary of the identified quality attributes and metrics

Figure 2 provides an overall view of the identified quality attributes and metrics. We report in gray the 8 main quality characteristics, in red the quality attributes, and in cyan the metrics. Dark arrows represent connections between a quality

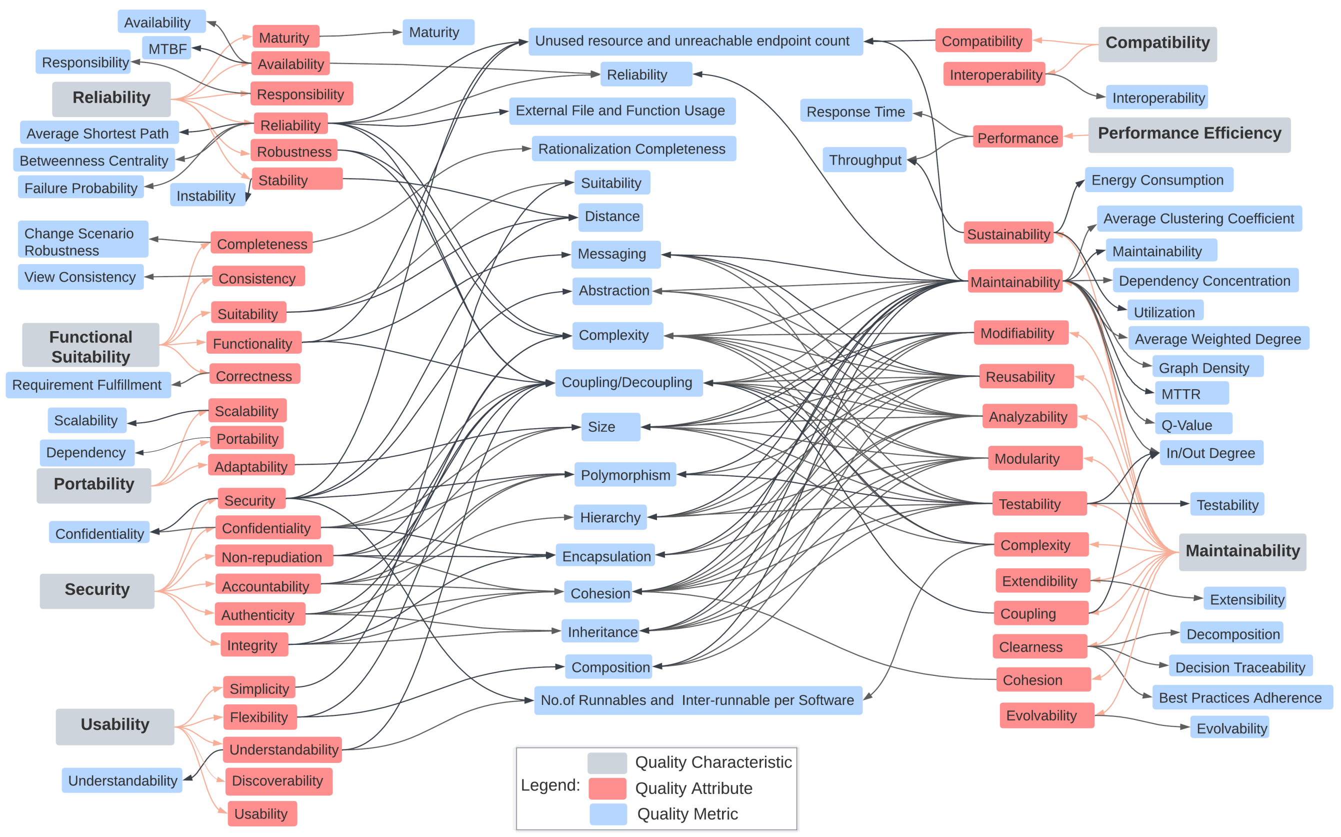


Fig. 2. Clustering of metrics and quality attributes.

attribute and the respective metrics. Light red edges show how the quality characteristics are divided into quality attributes. It is interesting to notice that there are metrics that can define, alone or together with others, a number of different quality attributes. For example, if we take the Cohesion metric, we can see how it is linked to various quality attributes of maintainability, but also to various quality attributes related to security. Cohesion is a measure of the degree to which the elements of the components are functionally related. Indeed, a system easy to maintain usually has high cohesion and low coupling; this confirms that cohesion influences the maintainability of the system [40]. Another metric that is used in different quality attributes and quality characteristics is Size, as we find it connected with various quality attributes related to maintainability, but also with quality attributes related to portability and security. In fact, a system with a large number of components, sub-components, and connectors may result difficult to maintain.

C. *Generic vs specific metrics*

We answer RQ2: "*Are the collected metrics generic or specific?*" by considering the attributes and metrics already explained in the previous section. Table II reports the classification of the collected occurrences of metrics into generic and specific. We classify as *specific* metrics that are defined for a specific domain, such as metrics to evaluate and improve the quality of the system in the automotive domain (i.e., structural complexity, coupling measures, etc) [45], or technology-specific, e.g., micro-services-based [51] or object-oriented design [41]. On the other hand, *generic* metrics are the metrics that are general and may be applied in different domains. For instance, the works in [28], [31], [32] propose metrics (i.e., Failure Probability, Response Time, Throughput) that can be applied to different types of domains.

Some metrics are defined as specific in some papers and as generic in others. This is the case, for instance, of the *complexity* metric. In [37], [41], it is designed for architectures with underlying object-oriented design. In [45], it is defined for the automotive domain. On the other hand, in [42] it is defined as a generic metric, i.e., for the evaluation of a generic architecture, without focusing on a specific technology or domain.

Overall, we reported 42 occurrences of specific metrics and 76 occurrences of generic metrics that are applicable to domain-independent software architectures.

D. *Internal vs external metrics*

To answer RQ3: "*Which of the collected metrics are internal and which ones are external?*", we categorized the collected metrics into internal and external. External quality characteristics and metrics are important to stakeholders since they are used to measure how well the design meets their demands. On

TABLE II
METRICS

Metric	Specific	Generic
Abstraction	[37], [24], [41]	[42] , [43]
Availability	[36]	[22], [21], [25]
Average Clustering Coefficient		[44]
Average Shortest Path		[44]
Average Weighted Degree		[44]
Best Practices Adherence		[38]
Betweenness Centrality		[44]
Change Scenario Robustness		[38]
Cohesion	[37], [39], [41], [42]	[22], [21]
Complexity	[37], [41], [45]	[42]
Composition	[37], [41]	[42]
Confidentiality		[29]
Coupling/Decoupling	[23], [24], [37], [41], [45]	[21], [22], [42], [43], [46]–[48]
Decision Traceability		[38]
Decomposition		[38]
Dependency	[39], [24]	
Dependency Concentration	[24]	
Distance	[24]	[43]
Encapsulation	[37] , [41]	[42]
Energy Consumption		[49]
Extensibility		[22], [21], [33]
External File and Function Usage	[24]	
Evolvability		[33]
Failure Probaility		[28], [31], [32]
Graph Density	[24]	
Hierarchy	[37] , [41]	[42]
In/out Degree	[24]	[44]
Inheritance	[37] , [41]	[42]
Instability	[24]	[43]
Interoperability		[22], [21]
Maintainability		[33]
Maturity		[22], [21]
Messaging	[37]	[42]
MTBF		[31]
MTTR		[31]
No.of Runnables and Inter-runnable per Software		[35]
Polymorphism	[37], [41]	[42]
Q-value	[24]	
Rationalization Completeness		[38]
Reliability	[36]	[29], [50]
Requirement fulfillment		[38]
Response Time		[28], [31], [32]
Responsibility		[22], [21]
Scalability		[22], [21]
Size	[37], [39], [41], [51]	[22], [21], [42] , [43], [52]
Suitability		[21], [22]
Testability		[33]
Throughput		[28], [29], [31], [32], [49]
Utilization		[49]
Understandability		[33]
Unused Resource and Unreachable Endpoint Count	[51]	
View Consistency		[38]

TABLE III
INTERNAL AND EXTERNAL METRICS

Metric	Internal	External
Abstraction	✓	
Availability		✓
Average Clustering Coefficient	✓	
Average Shortest Path	✓	
Average Weighted Degree	✓	
Best Practices Adherence	✓	
Betweenness Centrality	✓	
Change Scenario Robustness	✓	
Cohesion	✓	
Complexity	✓	
Composition	✓	
Confidentiality		✓
Coupling/Decoupling	✓	
Decision Traceability		✓
Decomposition	✓	
Dependency	✓	
Dependency Concentration	✓	
Distance	✓	
Encapsulation	✓	
Energy Consumption		✓
Evolvability	✓	
Extensibility	✓	
External File and Function Usage	✓	
Failure Probability		✓
Graph Density	✓	
Hierarchy	✓	
In/Out Degree	✓	
Inheritance	✓	
Instability	✓	
Interoperability	✓	
Maintainability	✓	
Maturity	✓	
Messaging	✓	
MTBF		✓
MTTR		✓
No.of Runnables and Inter-runnable per Software	✓	
Polymorphism	✓	
Q-Value	✓	
Rationalization Completeness		✓
Reliability		✓
Requirement Fulfillment		✓
Response Time		✓
Responsibility	✓	
Scalability	✓	
Size	✓	
Suitability		✓
Testability	✓	
Throughput		✓
Understandability	✓	
Unused resource and unreachable endpoint count	✓	
Utilization		✓
View Consistency	✓	

the other hand, developers and software architects concentrate on internal quality characteristics since they are the ones that impact external qualities. For instance, a software architecture with good internal quality, such as low coupling and high cohesion, is easier to maintain and scale. Table III reports the categorization of collected metrics into internal and external based on their definitions. It is noticed that most of the metrics are categorized as internal. We deduce that software architects that provided these metrics concentrate more on internal attributes. This can give an indication to the research community that additional external metrics might be needed.

IV. FRAMEWORK

To put into practice the results we found during our research and keep the results up-to-date we propose a Framework [19], which generates informative and aggregated web views of the metrics, quality attributes, quality characteristics, and their relationships. This section together with Section V answer

RQ4: "*How the collected quality attributes and metrics can be used in practice?*"

The proposed framework has been implemented with MDE techniques and it is based on a metamodel[3] which is shown in Figure 3. An *ArchitecturalQualityModel* is composed of

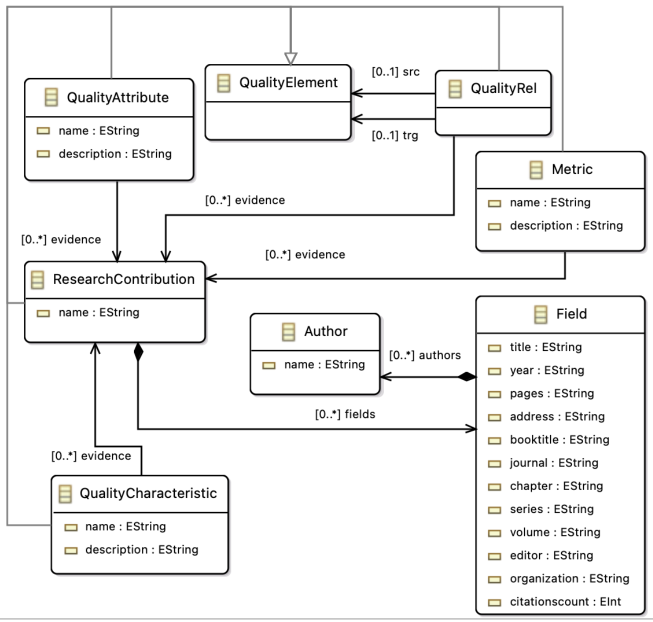


Fig. 3. Metamodel for defining the Quality Catalog

*QualityElement*s, which could be a *Metric*, a *QualityCharacteristic* or a *QualityAttribute*. Also *QualityRel*, which stands for quality relationship, can be instantiated to define relationships between quality elements, e.g., a metric used to measure a quality attribute. Each quality element can be defined with a corresponding description, which could be an excerpt of the description given in the literature. Moreover, *ResearchContribution*s can be defined in order to link the source literature contribution in which the quality element or a relationship has been defined or proposed. It is worth noting that a quality element or relationship could be linked to more than one research evidence. This is an important aspect giving the possibility to extract important information by querying the model defined on top of this metamodel. For instance, a quality relation linked to multiple evidences shows trustworthiness with respect to a single work of evidence. We have also included *citationscount* as field of the research contribution, for the same reason.

Based on this framework we have implemented a DSL [16] with xText [53] allowing the textual definition for our quality repository. Indeed, this DSL is based on a grammar (derived from the metamodel in Figure 3), allowing software architects, practitioners, and researchers to define quality aspects and their relationships. Moreover, we have integrated rules inspired by the Bibtex grammar[4] to allow contributors to directly work with bibtex entries in the DSL, to define research contributions. By using this DSL, contributors can define their own quality attributes and metrics and/or refine those provided by us (we call to the file containing these definitions as quality definition file) [19]. The process of usage and contribution has been depicted in Figure 4. By downloading and installing the XText plugin we developed from [19], contributors can play with the definitions that conform to the grammar we described before. The quality definition file is associated with the `.archquality` extension, which will make sure that the Eclipse editor will recognize the content of the model instantiation. By using the provided editor, all the developed features will be available, e.g., syntax highlighting, code completion, error quick fixes, etc.

Following the process, a contributor who wants to define new quality attributes and/or metrics or new relationships discovered in her studies can create a branch of the repository, even if actually the only file needed is the `.archquality` definition (up-to-date in the *catalog* folder of [19]). Contributors can then add or edit existing definitions by exploiting the textual syntax, as can be seen in Fig. 4 where a screenshot of the editor is reported. After that, contributors can emit a pull request that will be evaluated by the repository maintainer and in case the provided changes in definitions are legit and supported by the correct linked research contributions, a merge of the `.archquality` model will be executed. Every time the `.archquality` is updated, a code generation step is also executed. This step can be also executed locally to get a visual preview of the results since an xTend code generator is also included in the plugin. This code generator is in charge of generating multiple web views, which we describe in the following and that are browsable in the already mentioned online version of the generated repository available on the website[5] under the *Interactive Clustering of metrics and quality attributes*.

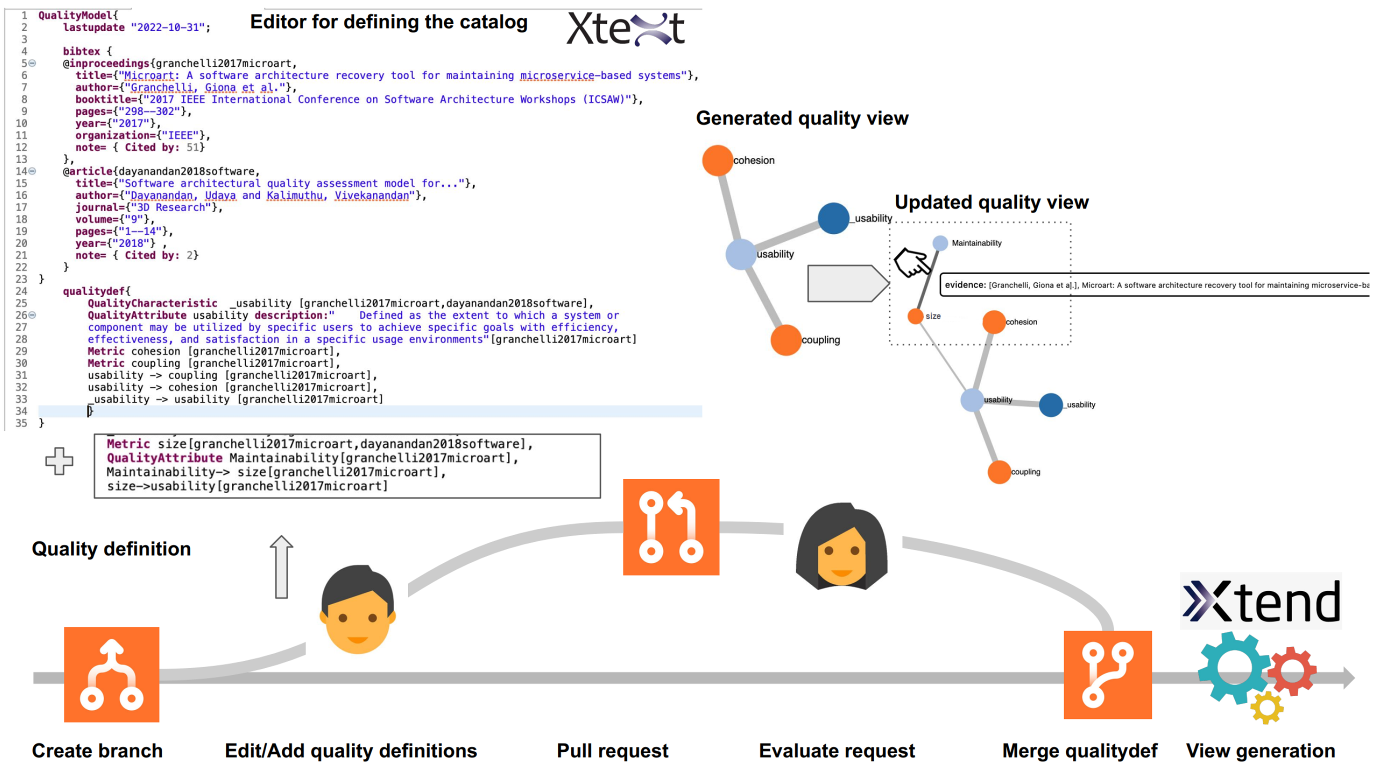
The first section called "Metrics" contains the generated list of all the metrics applicable to the architectures we found during our study of the associated literature. In addition to the name we also have a short description and a link to the articles that defined them. The `.archquality` model contains 52 metrics and the web view contains indeed the same number in the list. It is worth noting that this view is generated by interpreting the declared textual model, for instance the declaration at line 26 of Figure 4 in the editor screenshot.

The second section is called "Quality Attributes" and, as in the previous section, allows you to have an overview of all the quality attributes that we discovered during the analysis of the literature, i.e. 41 quality attributes, complete with descriptions and links to the articles that describe them. The code generation is similar to the previous one and for sake of space, we do not report the templates. All these views are generated by an xTend code generator processing the `.archquality` model and are updated every time a merge of the quality definition is executed.

[3]Some metamodel elements have been hidden for sake of readability

[4]https://slebok.github.io/zoo/markup/textual/bibtex/bibtex/hillairet/extracted/index.html

[5]https://tiziasan.github.io/QAandMetricsForArch/



```
QualityModel{
    lastupdate "2022-10-31";

    bibtex {
    @inproceedings{granchelli2017microart,
      title={"Microart: A software architecture recovery tool for maintaining microservice-based systems"},
      author={"Granchelli, Giona et al."},
      booktitle={"2017 IEEE International Conference on Software Architecture Workshops (ICSAW)"},
      pages={"298--302"},
      year={"2017"},
      organization={"IEEE"},
      note= { Cited by: 51}
    },
    @article{dayanandan2018software,
      title={"Software architectural quality assessment model for..."},
      author={"Dayanandan, Udaya and Kalimuthu, Vivekanandan"},
      journal={"3D Research"},
      volume={"9"},
      pages={"1--14"},
      year={"2018"} ,
      note= { Cited by: 2}
    }
}
    qualitydef{
        QualityCharacteristic  _usability [granchelli2017microart,dayanandan2018software],
        QualityAttribute usability description:"    Defined as the extent to which a system or
        component may be utilized by specific users to achieve specific goals with efficiency,
        effectiveness, and satisfaction in a specific usage environments"[granchelli2017microart]
        Metric cohesion [granchelli2017microart],
        Metric coupling [granchelli2017microart],
        usability -> coupling [granchelli2017microart],
        usability -> cohesion [granchelli2017microart],
        _usability -> usability [granchelli2017microart]
        }
}
```

```
Metric size[granchelli2017microart,dayanandan2018software],
QualityAttribute Maintainability[granchelli2017microart],
Maintainability-> size[granchelli2017microart],
size->usability[granchelli2017microart]
```

Fig. 4. Usage and contribution process of the proposed framework

The third section, "Interactive clustering of metrics and quality attributes", shows an interactive graph that allows us to see how the quality attributes, metrics, and quality characteristics are related. Also in this case a code generator has been employed to get the web view. This code generator integrates the generation of a JSON containing data defined in the `.archquality` model and an HTML file that statically creates the graph-based web view. This HTML file uses the d3js[6] library to animate and generate the view. The code generator uses the portion of metamodel allowing the declaration of `QualityRel`, where by navigating `src` and `trg` we can obtain the edges between two nodes. The nodes generated are distinguished into multiple types, i.e. metrics, quality attributes, and characteristics by color. Moreover, by using the fields attached to a research contribution we can animate the size of the nodes and the thickness of the edges, based on the trustworthiness of a research contribution. For instance, the example that we used is based on the number of citations of the papers.

In Figure 4, the new subgraph added by the contributor is made of elements smaller than the existing ones. This is due to the number of citations of the work attached to the other nodes, i.e. 51 vs. 2[7]. Finally, a tooltip is generated when hovering on nodes or edges declared with attached researched contributions in the DSL model. This allows quick discovery of the research contribution citing the quality aspects represented in this view.

[6]https://d3js.org/

[7]The number of citations and data in the screenshot are not corresponding to the reality.

V. Putting the Framework in Practice

This section presents some exemplary scenarios describing how the framework can be used. We define them by referring to possible stakeholders that can benefit from the framework and describe how they can use it, e.g. (i) software architects who want to evaluate their software architecture, (ii) researchers who work in software architecture evaluation, (iii) model-driven engineers in charge of implementing frameworks for evaluating software architectures, and (iv) tool vendors.

a) Research in Architectural Quality: As widely demonstrated in the first part of this paper, there has been huge attention to software quality and in particular architectural quality in research over the last decades. The proposed framework can be used as a repository of quality aspects, metrics, and their relationships and can be maintained in order to keep it up-to-date. This means that researchers and practitioners can contribute to it by reporting new insights and in case missing relationships or concepts are highlighted, research contributions and research directions can be suggested. This could also be the case of reporting a wrong relationship or suspicious concepts that would need to be investigated. This scenario can be put into practice by applying the process exposed in the previous section. Moreover, our framework provides a possible correlation between the quality aspects and this can be used as ground truth for experiments. For instance, if our framework highlights a possible correlation between *Complexity* and *Size* of an architectural model, increasing the size of an architecture should lead to an increase in its complexity.

b) Implementing Software Quality Evaluation Tools: Tools vendors and practitioners have similar purposes, even if the business model is different: they aim at implementing tools and frameworks for evaluating software. When software architectures are modeled with dedicated ADLs [54] the modeling environment may be enriched with evaluation features, based on the implementation of quality metrics. For instance in [55], the authors propose an approach for the specification, aggregation, and evaluation of software quality attributes for the architecture of microservice-based systems. MicroQuality provides reusable quality attributes defined with language-independent software quality attributes for microservices architectures, but also other measurement approaches are available with tool-dependent metrics [56]. Having a navigable representation of quality aspects could ease the task of defining new metrics for measuring quality aspects. Indeed, by browsing our representation, practitioners can easily find measurable definitions for the quality aspects and metrics they are implementing as part of the tool. The DSL is used to generate architectural quality specifications that can be processed with model transformations or other code generators with different aims.

c) Impact Traceability of Quality Aspects: When software architects are working to incrementally build software architectures, our framework can be consulted to understand how a quality attribute can impact others. This could lead to specific design decisions. For instance, by inspecting the *Performance* quality attribute, also *Energy Consumption* will be impacted, as expected. If a software architecture needs an improvement in *Usability*, inspecting the clustering view, it can be noticed that also *Interoperability* can be impacted, but there is no impact direct/indirect on *Extensibility*, even if expected. This could lead to further analysis of the existing literature and experiments to confirm it. It is widely accepted that high-quality software has low coupling and complexity and high cohesion. The greater the cohesiveness, the less work is required for *Maintainability*, the higher coupling the more difficult for the overall system to maintain. By exploring the view, it is possible to access and discover the quality features and associated evaluation metrics that affect *Maintainability*. If a user wants to assess the *Maintainability* of software architectures, some of the quality metrics that can be used are *Cohesion*, *Coupling*, and *Complexity*.

d) Reusability in Quality models: A quality model [57] is an analytical model that provides a quantitative assessment of selected quality characteristics based on measurement data. When an assessment framework is implemented, even in software architectures, an implementation of the measurement process is a fundamental step that is always re-implemented. The model, view and data we provide can be used as a prescription model for implementing architectural metrics with a reusable or general-purpose language. In this way, software plugins could be implemented and reused in multiple projects without the need to re-implement. This could also lead to a SaaS [58] implementation which would be independent of the architectural language used.

VI. Conclusion and Future work

This work reports quality metrics that have been proposed to evaluate quality attributes for software architectures. In particular, we focus on metrics that only require as input an architecture description. We then exclude from this study metrics that require other artifacts or system implementation. The collection of such metrics has been performed through an SLR browsing papers published from 2011 and available in IEEExplore, ACM Digital Library, Scopus, Springer Link, and ScienceDirect. After the selection process, we identified 33 papers, which constitute our primary studies. From these papers, we identified 52 metrics, which are related to 41 quality attributes. The most common metrics are Cohesion, Coupling/Decoupling, and Size. Maintainability, Reliability, and Testability are the quality attributes with the largest occurrence in the selected papers. The quality metrics are categorized according to the eight quality characteristics of the ISO/IEC 25010:2011 standard [18].

We also contributed with a classification of metrics into generic (76 occurrences) or specific (42 occurrences), and internal (38 metrics) or external (14 metrics). Finally, we also provided a framework based on a textual DSL that is used to implement the catalog output of our study, and we showed through some scenarios how various stakeholders can use it and how they can contribute with the tool. In future works, we plan to perform a strong evaluation of the DSL and the catalog with possible stakeholders. We intend to conduct a user study evaluation involving practitioners and experts in the field.

The catalog of quality attributes and metrics we provide in this work is meant to be a living object that will grow over time with new metrics, attributes, and/or categorizations. It is important to highlight that the catalog does not belong to us, but to the community of experts and practitioners who contribute to and use it. For this reason, we provided also a website to give access to the details of the study and to involve the research community in populating and refining the catalog.

References

[1] S. M. Ågren, E. Knauss, R. Heldal, P. Pelliccione, A. Alminger, M. Antonsson, T. Karlkvist, and A. Lindeborg, "Architecture evaluation in continuous development," *Journal of Systems and Software*, vol. 184, 2022.

[2] P. Pelliccione, E. Knauss, R. Heldal, S. Magnus Ågren, P. Mallozzi, A. Alminger, and D. Borgentun, "Automotive architecture framework: The experience of volvo cars," *Journal of Systems Architecture*, vol. 77, 2017.

[3] A. Bucaioni, P. Pelliccione, and R. Wohlrab, "Aligning architecture with business goals in the automotive domain," in *2021 IEEE 18th International Conference on Software Architecture (ICSA)*, 2021.

[4] R. Wohlrab, U. Eliasson, P. Pelliccione, and R. Heldal, "Improving the consistency and usefulness of architecture descriptions: Guidelines for architects," in *2019 IEEE International Conference on Software Architecture (ICSA)*, 2019.

[5] I. 42010:2011, "Systems and software engineering — architecture description," p. 37, 2011.

[6] P. Clements, R. Kazman, and M. Klein, *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley Professional, 2001.
[7] R. Kazman, M. Gagliardi, and W. Wood, "Scaling up software architecture analysis," *Journal of Systems and Software*, vol. 85, no. 7, 2012.
[8] H. Koziolek, "Sustainability evaluation of software architectures: a systematic review," *QoSA-ISARCS*, 2011.
[9] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on software engineering*, vol. 20, no. 6, 1994.
[10] U. Z. Srdjan Stevanetic, "Software metrics for measuring the understandability of architectural structures: a systematic mapping study," *EASE '15*, 2015.
[11] M. Staron and W. Meding, "A portfolio of internal quality metrics for software architects," in *International Conference on Software Quality*, 2017.
[12] F. Losavio, L. Chirinos, N. Lévy, and A. Ramdane-Cherif, "Quality characteristics for software architecture," *Journal of object Technology*, vol. 2, no. 2, 2003.
[13] ISO/IEC, "ISO/IEC/IEEE 42010:2011 Systems and software engineering – Architecture description," 2011.
[14] N. Fenton and J. Bieman, *Software Metrics: A Rigorous and Practical Approach, Third Edition*, 3rd ed. USA: CRC Press, Inc., 2014.
[15] S. Kent, "Model driven engineering," in *International conference on integrated formal methods*. Springer, 2002, pp. 286–298.
[16] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," *ACM computing surveys (CSUR)*, vol. 37, no. 4, pp. 316–344, 2005.
[17] B. A. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Keele University and Durham University Joint Report, Tech. Rep., 2007.
[18] I. 25010:2011, "Systems and software engineering — systems and software quality requirements and evaluation (square) — system and software quality models," 2011.
[19] "Github repository of the proposed framework," https://github.com/tiziasan/QAandMetricsForArch, accessed: 2022-11-05.
[20] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*, ser. Computer Science. Springer, 2012.
[21] S. Orlov and A. Vishnyakov, "Decision making for the software architecture structure based on the criteria importance theory," *Procedia Computer Science*, vol. 104, 2017.
[22] ——, "Hierarchical criterion approach for architecture structure selection of transportation software," *Procedia Engineering*, vol. 178, 2017.
[23] C. Etzel, F. Hofhammer, and B. Bauer, "Towards metrics for analyzing system architectures modeled with east-adl," 2020.
[24] D. Tiwari, H. Washizaki, Y. Fukazawa, T. Fukuoka, J. Tamaki, N. Hosotani, and M. Kohama, "Metrics driven architectural analysis using dependency graphs for c language projects," in *COMPSAC*, 2019.
[25] N. Eftekhari, M. P. Rad, and H. Alinejad-Rokny, "Evaluation and classifying software architecture styles due to quality attributes," 2011.
[26] S. Moaven and J. Habibi, "A fuzzy-AHP-based approach to select software architecture based on quality attributes (FASSA)," *Knowledge and Inf. Systems*, vol. 62, no. 12, 2020.
[27] D. G. M. Lindvall, "Adam: External dependency-driven architecture discovery and analysis of quality attributes," *ACM Transactions on Software Engineering and Methodology*, vol. 23, no. 2, pp. 1–51, 2014.
[28] I. Derbel, L. L. Jilani, and A. Mili, "Computing attributes of software architectures a static method and its validation," in *2015 International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*. IEEE, 2015, pp. 55–66.
[29] A. Sedaghatbaf and M. A. Azgomi, "Software architecture modeling and evaluation based on stochastic activity networks," in *International Conference on Fundamentals of Software Engineering*. Springer, 2015.
[30] S. Kadri, S. Aouag, and D. Hedjazi, "MS-QuAAF: A generic evaluation framework for monitoring software architecture quality," *Information and Software Technology*, vol. 140, 2021.
[31] I. Derbel, L. L. Jilani, and A. Mili, "Automated quantitative attributes prediction from architectural description language," in *2014 9th International Conference on Software Paradigm Trends (ICSOFT-PT)*. IEEE, 2014, pp. 87–94.
[32] ——, "ACME+: An ADL for Quantitative Analysis of Quality Attributes," in *International Conference on Evaluation of Novel Approaches to Software Engineering*. Springer, 2013, pp. 16–32.
[33] T. Wang and B. Li, "Analyzing software architecture evolvability based on multiple architectural attributes measurements," in *2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2019, pp. 204–215.
[34] P. Dugerdil and M. Niculescu, "Visualizing software structure understandability," in *23rd Australian Software Engineering Conference*, 2014.
[35] H. Venkitachalam, K. A. Powale, C. Granrath, and J. Richenhagen, "Automated continuous evaluation of autosar software architecture for complex powertrain systems," *INFORMATIK 2017*, 2017.
[36] A. Sedaghatbaf and A. M. Abdollahi, "A method for dependability evaluation of software architectures," *Computing*, vol. 100, no. 2, 2018.
[37] U. Dayanandan and V. Kalimuthu, "Software architectural quality assessment model for security analysis using fuzzy analytical hierarchy process (fahp) method," *3D Research*, vol. 9, no. 3, 2018.
[38] S. Sehestedt, C.-H. Cheng, and E. Bouwers, "Towards quantitative metrics for architecture models," in *WICSA 2014*, 2014.
[39] H. Venkitachalam, J. Richenhagen, A. Schlosser, and T. Tasky, "Metrics for verification and validation of architecture in powertrain software development," in *1st International Workshop on Automotive Software Architecture*, 2015.
[40] M. Perepletchikov, C. Ryan, and K. Frampton, "Cohesion metrics for predicting maintainability of service-oriented software," in *Seventh International Conference on Quality Software (QSIC 2007)*. IEEE, 2007, pp. 328–335.
[41] U. Dayanandan and K. Vivekanandan, "An empirical evaluation model for software architecture maintainability for object oriented design," in *Proceedings of the International Conference on Informatics and Analytics*, 2016, pp. 1–4.
[42] U. Dayanandan and V. Kalimuthu, "A fuzzy analytical hierarchy process (fahp) based software quality assessment model: Maintainability analysis," *International Journal of Intelligent Engineering and Systems*, vol. 11, no. 4, 2018.
[43] K. M. Hansen, K. Jonasson, and H. Neukirchen, "An empirical study of software architectures' effect on product quality," *Journal of Systems and Software*, vol. 84, no. 7, pp. 1233–1243, 2011.
[44] M. Vimaladevi and G. Zayaraz, "Design level quality analysis using fuzzy uml models and weighted complex networks," in *2017 International Conference on Communication and Signal Processing (ICCSP)*, 2017.
[45] D. Durisic, M. Nilsson, M. Staron, and J. Hansson, "Measuring the impact of changes to the complexity and coupling properties of automotive software systems," *Journal of Systems and Software*, vol. 86, no. 5, 2013.
[46] R. Mo, Y. Cai, R. Kazman, L. Xiao, and Q. F. Q, "Decoupling level: A new metric for architectural maintenance complexity," *ICSE 2016*, vol. 14, 2016.
[47] R. Mo, W. Snipes, Y. Cai, S. Ramaswamy, R. Kazman, and M. Naedele, "Experiences applying automated architecture analysis tool suites," in *ASE*, 2018.
[48] S. da Silva Amorim, J. D. McGregor, E. S. de Almeida, and C. von Flach G. Chavez, "Flexibility in ecosystem architectures," in *ECSA Workshops*, 2014.
[49] E. Jagroep, J. M. van der Werf, S. Brinkkemper, L. Blom, and R. van Vliet, "Extending software architecture views with an energy consumption perspective," *Computing*, vol. 99, no. 6, 2017.
[50] J. M. Franco, R. Barbosa, and M. Zenha-Rela, "Reliability analysis of software architecture evolution," in *2013 Sixth Latin-American Symposium on Dependable Computing*, 2013.
[51] T. Asik and Y. E. Selcuk, "Policy enforcement upon software based on microservice architecture," in *SERA*, 2017.
[52] D. Perez-Palacin, R. Mirandola, and J. Merseguer, "On the relationships between qos and software adaptability at the architectural level," *Journal of Systems and Software*, vol. 87, 2014.
[53] L. Bettini, *Implementing domain-specific languages with Xtext and Xtend*. Packt Publishing Ltd, 2016.
[54] I. Malavolta, P. Lago, H. Muccini, P. Pelliccione, and A. Tang, "What industry needs from architectural languages: A survey," *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 869–891, 2012.
[55] M. Cardarelli, L. Iovino, P. Di Francesco, A. Di Salle, I. Malavolta, and P. Lago, "An extensible data-driven approach for evaluating the quality of microservice architectures," ser. SAC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1225–1234. [Online]. Available: https://doi.org/10.1145/3297280.3297400

[56] R. Lincke, J. Lundberg, and W. Löwe, “Comparing software metrics tools,” in *Proceedings of the 2008 International Symposium on Software Testing and Analysis*, ser. ISSTA ’08. New York, NY, USA: Association for Computing Machinery, 2008, p. 131–142. [Online]. Available: https://doi.org/10.1145/1390630.1390648

[57] D. Samadhiya, S.-H. Wang, and D. Chen, “Quality models: Role and value in software engineering,” in *2010 2nd International Conference on Software Technology and Engineering*, vol. 1. IEEE, 2010, pp. V1–320.

[58] W. Sun, K. Zhang, S.-K. Chen, X. Zhang, and H. Liang, “Software as a service: An integration perspective,” in *International Conference on Service-Oriented Computing*. Springer, 2007, pp. 558–569.