Cluster Computing (2024) 27:4077–4093
https://doi.org/10.1007/s10586-024-04343-4

Formulating a quality model for cloud-native software architectures: conceptual and methodological considerations

Robin Lichtenthäler[1] · **Guido Wirtz**[1]

Received: 13 September 2023 / Revised: 23 January 2024 / Accepted: 4 February 2024 / Published online: 25 March 2024

Abstract
Interest in cloud computing is steadily increasing and the range of offerings is evolving due to continuous technological innovation. Hence, cloud-native has been established as a term for building applications in a way that maximally exploits benefits of modern cloud computing concepts. However, cloud-native as a topic is broad and the variety in cloud computing technologies is large. Thus, we identify a need in supporting developers and software architects who want to benefit from cloud-native concepts. We provide this support in the form of a quality model for cloud-native software architectures that explains how architectural characteristics impact different quality aspects. Our focus is on the design time and the aim is that architectural models of applications can be evaluated according to cloud-native characteristics and corresponding quality aspects. In this work we present our approach for formulating and validating the quality model for cloud-native software architectures as well as its current state. This presentation is based on previous work, especially a recently conducted validation survey that focused on the impacts of architectural characteristics on quality aspects. The new contribution of this work is the integrated presentation of our approach in a larger context of conceptual and methodological considerations. Further, revision of the quality model based on a repeated literature search for architectural measures is presented. We provide a more detailed look on the quality model, explaining exemplary product factors and their relevance within the topic of cloud-native. Our results provide a qualitative overview of characteristics associated with cloud native software architectures and lay the foundation for quantitative quality evaluations based on architectural models of applications.

Keywords Cloud-native · Quality model · Quality model formulation · Architectural evaluation

1 Introduction

The term cloud-native describes a concept of developing and operating applications in a way that aims to maximally exploit the benefits of modern cloud computing environments [1]. The type of applications we refer to in this context are web applications implemented by organizations specifically to serve their customers or to satisfy internal needs via either internal networks or the internet. This covers mainly, but not exclusively, so called *enterprise applications* as discussed by Cerny et al. [2] who rely on a description by Fowler [3]. With the term application we refer to such web applications in this work. We differentiate them from other software described as being cloud-native, like the software listed in the Cloud Native Computing Foundation (CNCF) Landscape.[1] That software may enable the development of or is used in cloud-native applications, but these are not cloud-native applications themselves in the sense of this work.

The implementation of such cloud-native applications covers a broad range of aspects: the choice of cloud service offerings and technologies, the functional decomposition of business logic over components, setting up the communication between components, or managing the deployment and upgrade process of applications [4, 5]. When applied correctly, many benefits are associated with cloud-native applications, such as increased performance and scalability, cost effectiveness, reliability or improved maintainability [6–8]. Therefore, when aiming to implement an application in a cloud-native way or evaluating how an existing applica-

✉ Robin Lichtenthäler
robin.lichtenthaeler@uni-bamberg.de

Guido Wirtz
guido.wirtz@uni-bamberg.de

[1] Distributed Systems Group, University of Bamberg, An der Weberei 5, 96047 Bamberg, Bavaria, Germany

[1] https://landscape.cncf.io/.

tion could benefit from cloud-native concepts, the question arises how this can be done in an informed and structured way. Because of the thematic breadth of cloud-native and the multiple, potentially conflicting, desirable benefits, this presents a challenge.

To take up this challenge from a conceptual perspective, the essential question is how certain architectural characteristics (those associated with cloud-native concepts) impact the observable behavior of an application. As architectural characteristics we consider design time properties of an application that are the result of the decisions made by its developers. This covers all implementation aspects, such as technological options or software design. As observable behavior we consider runtime properties of an application that are measurable when the application is running and being operated. Runtime properties therefore indicate how well an application fulfills its intended functionalities. This is in general also summarized as the quality of an application. The relationships between architectural characteristics and observable behavior, however, can be complex and different architectural characteristics can also have interfering effects. But if these relationships can be described in a structured and comprehensive way, it would represent the necessary knowledge to evaluate application architectures according to quality aspects. This knowledge would thus satisfy the conceptual perspective.

But to actually support developers and architects in designing and implementing applications in a cloud-native way, also the practical perspective needs to be considered. From a practical perspective, the additional question would be how this knowledge can be integrated and made useful within suitable tooling.

Based on these considerations, we see a hierarchical quality model for software architectures of cloud-native applications as a suitable approach. It can both describe the relationships between architectural characteristics and quality aspects and be integrated into practical tooling for quality evaluations.

This hierarchical structure is one of the factors that differentiates our approach from other existing work. Apel at al. [9] for example present an approach with a similar goal, but directly linked metrics to quality aspects. In addition, they focus on microservices, instead of the broader cloud-native scope. Also focusing on microservices, Soldani et al. [10] present an approach to evaluate and improve microservices architectures, but orient their approach towards best practices instead of quality aspects. Another differentiating factor is the consideration of multiple quality aspects in combination. Also focusing on microservices, for example Camilli et al. [11] present an approach to specifically evaluate the quality aspects performance and reliability. Thus they link runtime metrics directly to quality aspects, but focus less on architectural characteristics that lead to the gained results. Although we can differentiate our work from the mentioned approaches, showing its novel contribution, they are related. Therefore, we aim to also integrate their results and recommendations into our approach.

For a solid foundation to build a hierarchical quality model, we rely on the structure as described by the Quamoco meta-model [12]. In this meta-model, quality aspects differentiate distinct dimensions of quality and are structured in a hierarchy together with product factors. Product factors represent characteristics of a software. The hierarchy is built through impact relations which describe how the presence of product factors has an effect on quality aspects. Such a quality model is therefore suitable for describing how architectural characteristics of an application impact its quality aspects conceptually in a structured way. Product factors can then be linked to entities, that means distinguishable components of a software architecture, to enable their measurement. And thus, a practical evaluation of a software architecture according to the relationships of a quality model becomes possible.

Our overall aim therefore is to formulate a quality model for software architectures of cloud-native applications. This essentially means defining all the different elements required for a complete quality model: quality aspects, product factors, impact relations, entities, measures, and evaluations.

In this work we focus on the formulation and validation of all mentioned elements, except evaluations. Thus, we present our approach from an integrated, holistic perspective. This includes content from previous work. On the one hand, our initial formulation of quality aspects and product factors [8]. And on the other hand, our work on validating this initially formulated quality model [1] to which this work specifically represents an enhanced version. Validation in this context means the assurance that the relations described in the quality model reflect reality as closely as possible. Furthermore, we expand on both previous works by further defining the elements of the quality model and outline how software architectures can be suitably represented.

The guiding research questions which summarize these aims and considerations are:

RQ1: How are characteristics of cloud-native applications related to quality aspects of distinct dimensions?

RQ2: How can a quality model for software architectures of cloud-native applications be formulated and validated?

To answer these research questions, we start with foundations of quality models in Sect. 2, based on which we present our methodology for this work in Sect. 3. The results of applying this methodology, especially the current state of the quality model, is presented in Sect. 4 and further discussed in Sect. 5. The relation to other work is described in Sect. 6, before a final conclusion in Sect. 7.

2 Foundations

In this section, relevant foundations are presented to more clearly specify the type of quality model we consider (hierarchical). And we explain how quality models can be formulated and validated and how these activities are related to each other.

2.1 Hierarchical quality models

The history of quality models is tightly connected to the history of software engineering itself [13], Their core idea is to provide a conceptual foundation for linking measurable characteristics of a software with quality aspects. If the quality of software can be measured and assessed, then it is also possible to show how software quality can be improved by changing the characteristics of software. In earlier days individual metrics have been used to measure for example the complexity of source code [13]. But it has become apparent that relations between characteristics of software and its quality are more complex. Therefore, quality models such as those of McCall [14] or Boehm [15] were structured as hierarchical quality models [13]. In these higher-level quality aspects can be differentiated into quality sub-aspects for which in turn it is easier to formulate relations with characteristics of software. Hierarchical quality models are therefore essentially representable through graphs. The higher level quality aspects are independent nodes in such a graph and the other elements have effects on these quality aspects. These effects are the edges of the graph. Through this a hierarchical quality model is able to capture the complexity of how software characteristics influence quality and it is also possible to consider several aspects in relation to each other.

Work on software quality models has been consolidated in standards with the ISO 25000 series [16] being the latest. Because it considers software quality on a rather abstract level, it has since been used as a basis for formulating new quality models [13]. And it can be used to compare and differentiate quality models. A main differentiator for quality models is the scope that they cover. The scope of a quality model is defined by the number and heterogeneity of its elements. Heterogeneity can be considered regarding the number of different quality aspects included, often referring to the number of different high-level quality aspects from the ISO 25000 standard [16]. And heterogeneity can also be observed for the aspects covered by the other elements of the quality model, also referred to as the dimensions [17, 18]. Different dimensions can for example be static characteristics of an application at different layers like the infrastructure, application logic, or supported business functionality, runtime behavior, development process characteristics, or also organizational aspects.

Since the overview paper on quality models by Ferenc et al. [13], further quality models have been proposed and in turn been reviewed by researchers. It is thus worthwhile to analyze such reviews for insights on important aspects of quality models. Reviews exist for quality models in general [17, 19, 20], but also for quality models in specific areas, for example web service [21]. One finding is that from the high-level quality aspects, mostly maintainability, reliability, and performance efficiency have been in focus of quality models [19]. Another finding by Galli et al. [20] is that although many quality models have been proposed in literature, their relevance, also regarding the practical use in the industry, differs significantly. The quality models with the highest relevance, as reported by Galli et al. [20], are the ISO 25000 standard [16] with its predecessor ISO 9126, the SQALE model [22] and the Quamoco model [12]. The respective reasons, however, are different. The ISO 25000 standards [16] popularity is because of its simplicity and the possibility to adapt it to more specific contexts [20]. The popularity of SQALE [22] is based on its widespread tooling support, e.g. within Sonar,[2] which facilitates its integration into the development process [20]. And the importance of Quamoco [12] lies within the extensive research it is based on and the formulation of a comprehensive meta model for hierarchical quality models that new quality models can build on.

The importance of effective tooling support for the adoption of quality models as in the case of SQALE is also stated by Yan et al. [19]. Related to the aspect of tooling, an additional success factor for the adoption of quality models is the completeness of definitions for its elements. To analyze this, Nistala et al. [17] differentiate between different phases of when quality models can be used (planning, realization, documentation, and assessment) and then map the type of elements of quality models to the phases in which they are used. Through this, they identify significant differences for the definition completeness of quality model elements. Furthermore, they observe a gap regarding the mapping of certain software properties and processes to the quality aspects in focus. These are, however, important for the practical application of a quality model. The aspect that definitions of quality model elements are not always complete is also stated for web service quality models by Oriol et al. [21].

Major aspects are therefore the elements of a quality model themselves and a suitable tooling support. Considering the elements of a quality model, the mentioned Quamoco meta model [12] from our perspective provides the most comprehensive foundation to build on. The main elements of the Quamoco meta model are shown in Fig. 1 and explained in the following.

[2] https://www.sonarsource.com/blog/sqale-the-ultimate-quality-model-to-assess-technical-debt/.

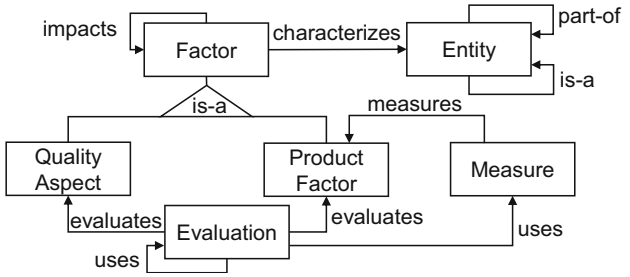

Fig. 1 Elements of the Quamoco meta model [12]

The core elements of the Quamoco meta model are *factors* which are differentiated into *quality aspects* and *product factors*. Quality aspects represent the rather abstract quality characteristics for example from the ISO 25000 standard. Product factors capture characteristics of a software which should be measurable for a specific software based on *measures*. Through the *impact* relationship, it is possible to define a hierarchy of factors where quality aspects define the top-level elements and product factors are structured below. The impact relationship can be defined coarsely, for example, as being positive or negative, describing how a factor impacts another factor, if that factor is present in a software. More detailed statements about how certain product factors impact quality aspects, also using mathematical approaches, can be made using *evaluations*. Evaluations can be considered as a kind of rules that configure the exact way in which the quality of a software system can be evaluated using measures, product factors and quality aspects. Evaluations can also be set in relation to each other, for example to enable aggregations of evaluations. Finally, to connect the quality model with the actual software under consideration, *entities* are used that represent different components of a software system. The extent to which product factors are present in a certain software system is characterized through the specific set of entities which describes this software system. Based on how these entities are structured and which properties they have, product factors can be measured, in the end enabling a quality evaluation of the software system.

In the rest of this article we use the terms introduced by the Quamoco model to describe our approach for formulating and validating a quality model for cloud-native applications.

2.2 Quality model formulation

With *Formulation* we refer to the process of creating a new quality model. In essence, this means that elements corresponding to the types of the Quamoco meta model [12] need to be formulated and set into relation. There are several ways to do this and some constraints that need to be considered.

First, the scope of the quality model should be decided on. That means for example what kind of software is considered, at which layer it is considered and also which quality aspects are of interest. Only if the scope is defined, it is possible to decide which elements to include in the quality model and which not. The quality aspects and product factors should be defined first, because they represent the core of the quality model. Entities, measures and evaluations should be defined in a second step, suitable to the factors. In the end, completeness of a quality model needs to be ensured so that all factors are defined in a clear and understandable way, measures are available to quantify factors, and evaluations enable the quality assessment of a system. Considering the structure of the quality model, AL-Badareen et al. [23] have investigated rules and constraints to consider when formulating quality models. One exemplary rule is that there should be only one path from an element to a certain top-level quality aspect [23] in order to keep evaluations regarding this quality aspect unambiguous. The most important aspect, however, for a quality model to be useful and of good quality itself, is that the conditions and relations it describes reflect reality as closely as possible. Although an exact reflection of reality is impossible, because a model per its definition has to abstract from the real world, it should be ensured that software quality relations are modeled realistically. This is also stated by Wagner et al. who emphasize the importance of a *Rationale* [12] for each stated impact relation between factors of the quality model. The respective rationale included in a quality model is a direct result of the approach used for formulating a quality model. Different approaches have been listed by Moody [24], including for example: theory-based (deductive), experience-based (codification), observation-based (inductive), or consensus-based (social). Based on the available possibilities, a suitable choice needs to be made on the approach used for formulating a quality model. In any case, the chosen approach should be documented and explained so that it can be reproduced and evaluated by others.

2.3 Quality model validation

As stated in the previous section, a quality model necessarily abstracts from reality, but should nevertheless reflect reality as closely as possible for meaningful assessments of software. Ensuring this reflection of actually observable quality impacts is therefore an important property of a useful quality model. We refer to this assurance as *validation* [19, 25] in the sense that the theory underlying the formulation of quality model elements is validated using suitable approaches. There are several aspects of a quality model that can be validated and validations can in turn be also done in several ways. In previous work [25] we have investigated validation approaches and scopes for quality models in more detail. An important aspect that emerged is that not all types of validation are possible during all phases of the creation process of a quality model. Especially in an early phase, when only

quality aspects, product factors, and their respective impact relations are formulated, a complete validation of the quality model based on an assessment of software is difficult because of the missing measures and evaluations. Nevertheless it is possible to validate these formulated elements, but a suitable approach needs to be chosen. To decide which validation approach is a suitable one, it is necessary to check whether an approach can accomplish the goal of a successful validation. And that goal is the confirmation of relations and statements defined during the initial formulation of a quality model based on a different type of rationale than initially used. For example, if a quality model has been formulated based on experience by experts, it could be validated using an experimental approach relying on measures of an actual software system. Or, if a quality model has been formulated based on theory, e.g., relying on literature, it could be validated through an empirical survey (consensus-based) which is also the scenario that is considered in this work. Approaches that do not require the complete formulation of all elements of a quality model are for example consensus-based empirical approaches, such as interviews or surveys. These can be used to validate factors by asking for the clearness of their descriptions or their applicability to software systems. Or they can be used to validate the formulated impact relationships between factors by asking for the type and strength of impact certain factors have on other factors. Although using a somewhat different categorization of validation approaches, Yan et al. [19] found this to be the most used validation approach, called *expert-opinion* by them.

An iterative approach of formulation, validation, and refinement through new formulations can therefore be taken and validation can be applied repeatedly and from different perspectives. As a general rule it can be stated that the more validations from different perspectives are performed, the more confidence exists that the quality model is appropriate and usable. Validations during the creation process should thus also be complemented by validations once all elements of a quality model are defined. This explicitly includes experimental approaches during which quantitative measures should be taken to evaluate the factors for a specific software system and compare them with additional measures, for example taken at runtime. Validation in this regard has also been named as the "*representation condition*" by Galli et al. [20] which they define as the assertion that "*properties of real-world entities measured are mapped in numeric representations in such a way that the numeric representations are equivalent to the reality*" [20].

3 Methodology

Building on the presented foundations regarding quality models, we explain in this section our methodology for formulating and validating a quality model for software architectures of cloud-native applications. This methodology integrates the previous works [8] and especially [1] to which this work represents an extended version. In addition, we have developed the quality model further by repeating and therefore updating the literature search for suitable measures from [8]. Furthermore, we provide a more detailed view on the refinement of the quality model done after the validation survey presented in [1]. To provide a better overview, we have summarized our methodology in Fig. 2 and explain the steps which can be seen there in the following.

Step A: First, we have formulated the quality aspects and product factors relevant to cloud-native applications together with their connecting impact relationships. To do so, we relied on literature, namely the ISO 25000 standard [16], existing definitions for the term cloud-native, and practitioner books on the topic. This has been done and presented in previous work [8].

Step B: In addition to the factors of the quality model, we have also performed a literature search to identify suitable measures that can be used to evaluate the formulated factors. This has also been done in the mentioned previous work [8]. We found measures, especially for the factors grouped under the maintainability quality aspect, but for many factors suitable measures still need to be defined.

Step C: Because we wanted to validate the elements formulated up until this point before proceeding further with the work on the quality model, we performed a survey to validate the existing elements. The focus was specifically on the impact relationships between product factors and quality aspects. The survey and its result have been presented in [1] and it led to a refinement of the quality model elements. This refinement is described in Sect. 3.1.

Repeat Step B: To further develop the quality model in this work, we have repeated and revised the literature search for measures to identify additional measures from newly published work. How we repeated and revised this search is described in Sect. 3.2.

Step D: In parallel to the validation of the existing entities, we prepared the actual evaluation of software architectures by formulating suitable entities to which factors can be linked. In addition, we reviewed modeling options for describing software architectures of cloud-native applications. This review and initial tooling support has been presented in [26].

Step E: (*Planned in future work*) Based on the found measures and the development of tooling support, we plan to validate the quality model from additional perspectives. In specific, we plan for an experimental approach with software architectures of applications that serve as use cases.

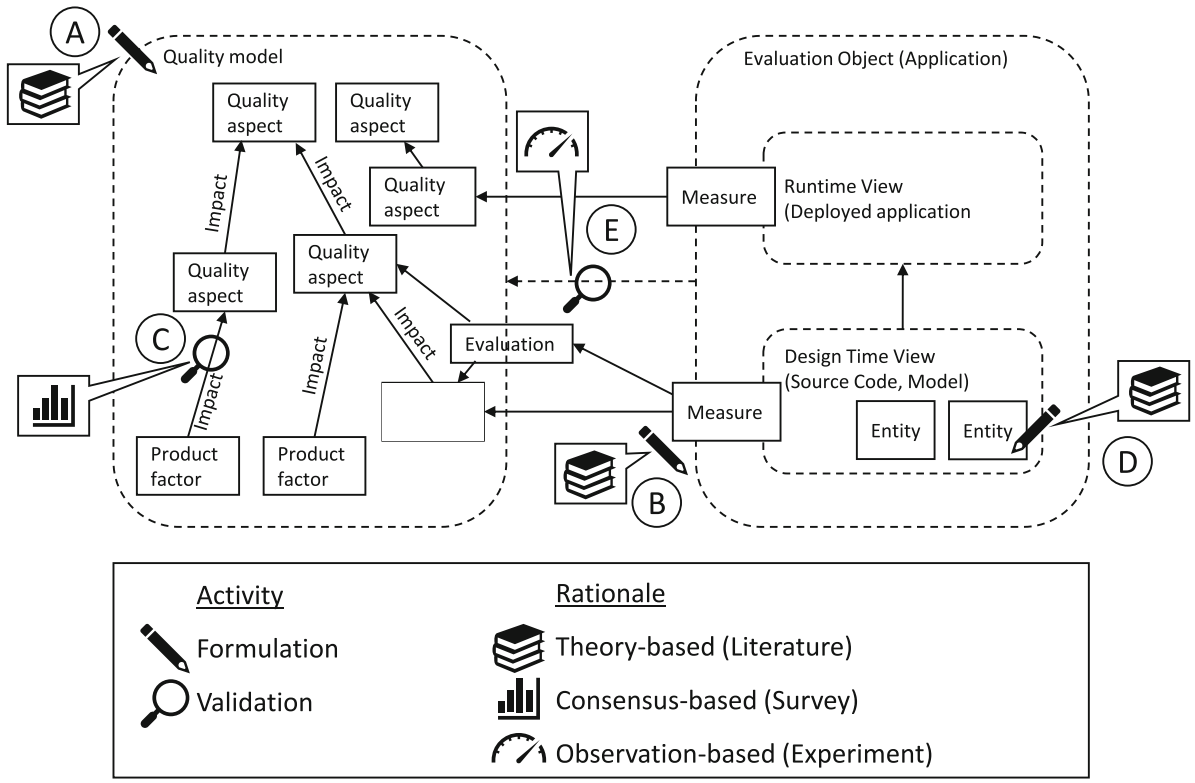

Fig. 2 The overall methodology of our approach

3.1 Validation survey (Step C)

Because the initial quality model was formulated based on literature [8] and only to the extent of quality aspects and product factors, we chose to use an empirical survey-based approach for the validation of the formulated elements. A detailed description of how the survey was prepared and conducted can be found in [1]. In this work, we only repeat the main aspects and focus on how the survey results were used to refine the quality model.

As survey participants we targeted IT professionals who have practical experience with implementing and deploying web applications on cloud infrastructures. We contacted potential participants based on relevant publications or talks they had given, social media groups and communities with a fitting topic, as well as personal contacts. The basic approach of the survey was to build pairs of product factors (i.e., architectural characteristics of a cloud-native application) and quality aspects (i.e., the sub-characteristics from the ISO 25000 standard [16] such as Availability, Fault Tolerance, or Modifiability). In total, 45 product factors and 24 quality aspects were considered. For each product factor the survey participants were then tasked to freely rate potential impacts on any of the quality aspects. The impacts stated in the initial quality model were intentionally not shown to the participants to avoid bias in their answers. Impacts could be rated on a 5-point scale of:

- factor has **positive impact (++)** on quality aspect
- factor has **slightly positive impact (+)** on quality aspect
- factor has **no impact (0)** on quality aspect
- factor has **slightly negative impact (−)** on quality aspect
- factor has **negative impact (−−)** on quality aspect

Thus, the survey resulted in a set of impact ratings stated for each pair of a product factor and a quality aspect. For each set we calculated a mean value using numeric values of + 2, + 1, 0, − 1, and − 2 for the respective impact ratings and a p value as an indicator for the significance of a result. The p value was calculated using the *Exact multinomial test of goodness-of-fit* [27]. We chose this test, because it is suitable when there are multiple values of one nominal variable (the different types of impact) and the sample size is small. For the test we assumed equal probabilities for each impact rating, except for the rating of 0 (no impact). Because "no impact" was selected as a default if a participant did not explicitly state any impact, we assumed it to be twice as likely as the other rating options. Based on the mean values and the p-Values, we refined the impact relationships of the quality model from two points of view: First, we iterated over the impact relations already stated in the initially

formulated quality model and checked whether they could be confirmed by the survey results or had to be rejected. And second, guided by the significance indicator, we iterated over the impact relations not yet considered, but found through the survey, and decided whether to include them as new impact relations in the model or not. Orthogonal to these two points of view, we also considered each product factor individually with its rated impacts on quality aspects to identify factors which might need to be refined as such. In fact, a few factors showed to be ambiguous, because either significant impacts on several different quality aspects were found or because conflicting types of impact (that means positive and negative) were found for the same quality aspect. In these cases, we revised the factors as such and included the found impacts as new impact relations in the quality model. Revision could also mean splitting a factor into new separate factors which cover distinguishable aspects of the factor that showed ambiguous results. All refinements were based on the interpretations by us as authors, but in each case we also reconsidered the initially used literature to substantiate our decisions. To summarize, the following refinement actions were applied to the quality model based on the validation survey results:

- Removal of an initially stated impact relation (17 times)
- Addition of an newly found impact relation (16 times)
- Reformulation of a product factor and selection of suitable impact relations (2 times)
- Splitting a product factor in two or more new product factors which cover distinguishable sub-aspects of the original product factor (1 time)

Which refinements regarding specific impacts and factors were applied is also listed online.[3]

3.2 Repeated literature search (Repeat Step B)

As mentioned, the overall goal of our work is to have a formulated and validated quality model for software architectures of cloud-native applications. Measures, in this regard, are one type of model elements needed to make factors measurable. Once suitable measures are formulated, additional types of validation can be applied including quantitative experiments. Because we identified a lack of suitable measures in previous work [8] and some time has passed since the initial literature search, we decided to repeat and revise this literature search. Through this, we aim to include literature published in the meantime and identify potentially missed literature. In a second search done on 2023-07-13 with exactly the same search terms as in the initial search, we included literature that had been published since the first search. However, based on a closer review of the search terms and results, we realized that papers may have been missed due to the usage of either singular or plural forms in literature. For example, in the initial search terms, we included the term “microservices”, but not “microservice”. Through this, the paper by Zdun et al. [28] was not in our results, although relevant metrics are presented in it. Thus, we revised the search terms and performed a third search on 2024-01-16. The revised search strings include one focusing microservices architectures as an architectural style commonly associated with cloud-native applications:

```
(Abstract:(architecture)) AND (Abstract:(measure)
OR Abstract:(measures) OR Abstract:(metric) OR
Abstract:(metrics)) AND
(Abstract:(service-oriented) OR
Abstract:(microservices) OR
Abstract:(microservice))
```

And a second search string focuses on cloud applications, especially their quality:

```
(Abstract:(cloud-native) OR Abstract:(“cloud
computing”)) AND (Abstract:(measure) OR
Abstract:(measures) OR Abstract:(metric) OR
Abstract:(metrics)) AND (Abstract:(quality))
```

These search strings were adapted to search the ACM Digital Library,[4] IEEE Xplore,[5] and Springer Link.[6]

The steps and numbers of papers considered from these searches are combined in Fig. 3. The number left of the + operator describes the results from the second search and the number right of it describes the results from the third search. The overall process is also described in more detail online.[7]

As relevant results, we considered only literature in which:

- a perspective is taken corresponding to that of application developers (in contrast to the perspective of cloud providers),
- software architectures are considered on a level suitable to the quality model (in contrast to lower levels such as internal component design or higher levels such as the end-user perspective),
- measures are presented that can be evaluated at design time (in contrast to runtime specific measures),
- calculations for measures are specified clearly so that they can be adopted

[3] https://r0light.github.io/cna-quality-model/survey.

[4] https://dl.acm.org/.

[5] https://ieeexplore.ieee.org/Xplore/home.jsp.

[6] https://link.springer.com.

[7] https://r0light.github.io/cna-quality-model/search-process.

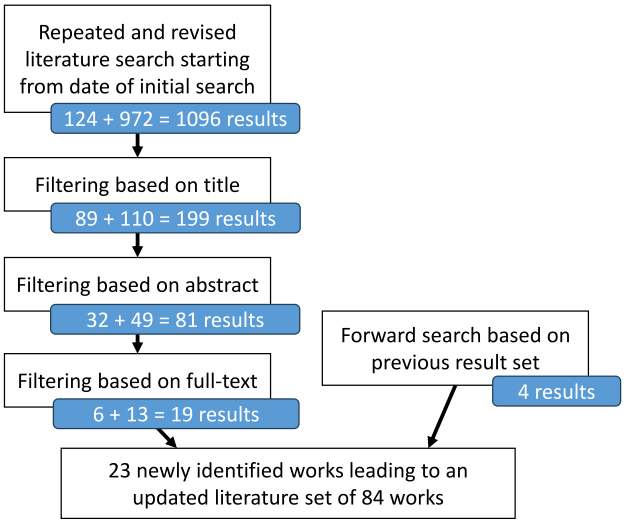

Fig. 3 Repeated and revised literature search results

As it can bee seen in Fig. 3, we could thereby extend the literature set describing suitable measures. 23 additional works that present measures suitable for the factors of the quality model were found so that in total 84 works can be considered for the quality model. It was encouraging that also measures could be found for factors for which previously no measures were found in the literature. We included these additional measures in the quality model which therefore represents an improved basis for quantitative validations relying on such measures. The successful identification of additional measures also shows the possibility of using an iterative approach for the formulation of a quality model in the sense that such a literature search can be done repeatedly to continuously update and refine a quality model.

The results of applying this methodology are presented in the following Sect. 4. It has to be noted that this approach is one possible option for the development of a quality model which we found suitable in the context of our goal. Other approaches for the formulation and validation of a quality model are also possible. Our description of foundations for the formulation and validation of quality models in Sect. 2 as well as the documentation of our methodology aim to make different approaches comparable.

4 The cloud-native quality model

In this section we present the current state of the quality model. Therefore the different types of elements are explained and then exemplary factors from the model are described in more detail. Using the examples, we also show how the formulation and validation steps lead to their current state.

4.1 Elements of the quality model

The quality model for software architectures of cloud-native applications is based on the Quamoco Metamodel [12] and in its current status, quality aspects, product factors, impacts, entities and partly measures are defined. This means the completion of providing measures to make product factors measurable as well as the definition of evaluations that enable meaningful analyses are still needed for quantitative evaluations of software architectures. In contrast, a qualitative evaluation based on a manual assessment of a software architecture regarding the product factors of the quality model would already be possible. We therefore present the quality model in its current form in this section and focus specifically on the different product factors of the quality model We describe how they should be understood, which rationale was used for their formulation and how a potential evaluation may use them. The quality model in visual form is shown in Fig. 4. It includes product factors in white boxes and quality aspects in gray boxes. The connecting arrows represent the impact relations with □ indicating a positive impact and □ indicating a negative impact. It is important to state that the focus of the quality model is on the design time of an application. Thus, the included product factors are intended to be evaluable at design time based on an architectural representation of an application. Additional aspects, such as characteristics of the deployment process, are also important within the topic of cloud-native, but out of the scope of this quality model.

However, a decision needed to be made, about what a suitable architectural representation of an application would be. While in general, applying the quality model directly to source code and deployment descriptions would be imaginable, it makes the implementation increasingly complex, because then a multitude of technologies would have to be supported. We therefore chose to rely on a model of a software architecture and formulated potential entities of such a model in [8]. Additionally, to rely on existing approaches, we reviewed currently available modeling options [26] for software architectures of cloud-native applications, as described in Sect. 3: Step D. The result are the entities shown in Fig. 5.

These entities together with further properties describing them in more detail enable the modeling of software architectures of cloud-native applications so that these architectures can be evaluated. The formulation of the specific properties added to each type of entity strongly depends on the respective measures associated with the product factors. These properties together with the component structure provide the data basis for measuring. The definition of properties is therefore intended to be more flexible so that they can be adapted as required by the used measures.

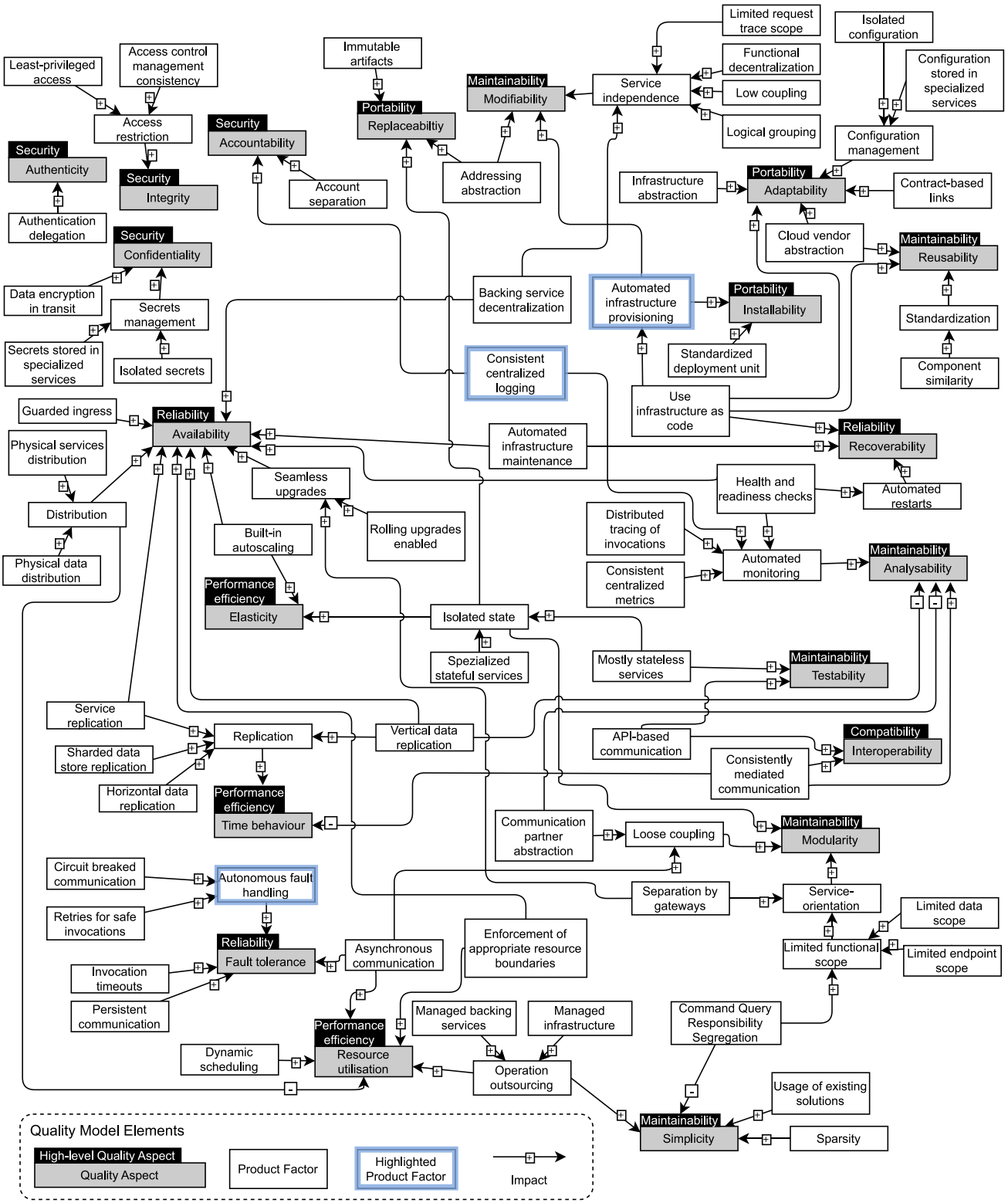

Fig. 4 The quality model for software architectures of cloud-native applications

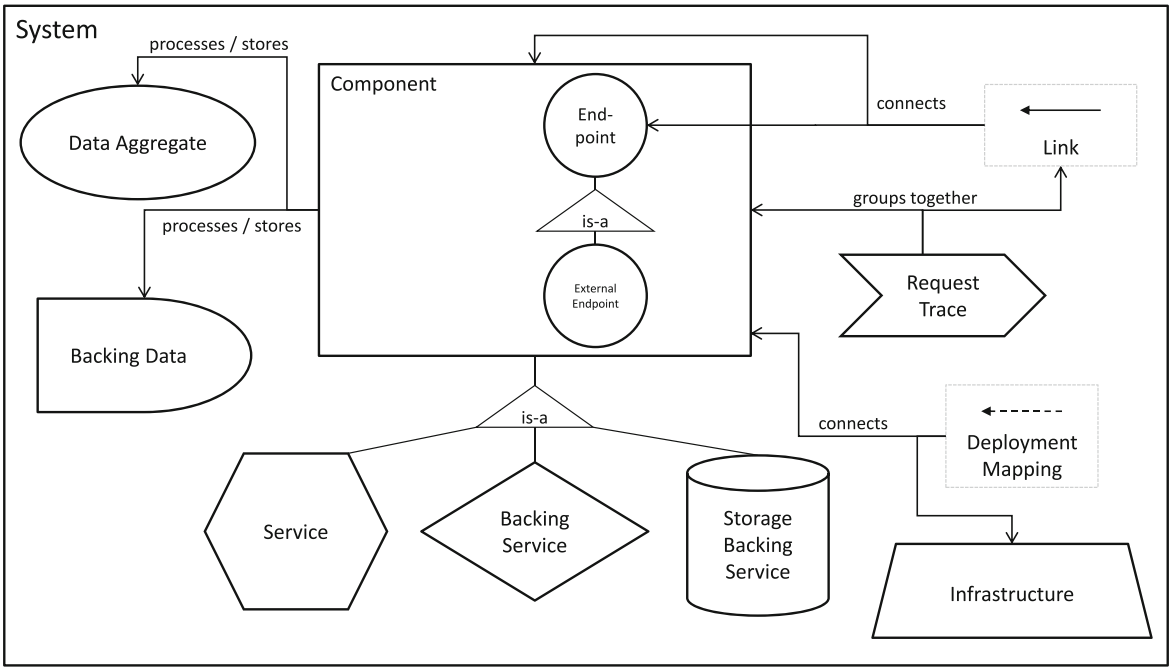

Fig. 5 Entities proposed for the architectural description of cloud-native applications

4.2 Exemplary product factors

Based on the general description of the elements in Sect. 4.1, we now present three exemplary factors in more detail. These factors are also highlighted in blue in Fig. 4. We specifically describe how they were formulated, validated, and how they can be evaluated based on a cloud-native architecture considering required entities and measures:

Automated infrastructure provisioning This product factor is described as "*Infrastructure provisioning should be automated based on component requirements which are either stated explicitly or inferred from the component which should be deployed. The infrastructure and tools used should require only minimal manual effort. Ideally it should be combined with continuous delivery processes so that no further interaction is needed for a component deployment.*" and is considered to have positive impacts on both, *Modifiability* and *Installability*. The initial formulation is for example based on literature by Reznik et al. [29] who write that "*Any manual work that is required in between the changes committed by the developer and the delivery to production will significantly reduce the speed of delivery*" [29, Pattern: Automated Infrastructure (Chapter 10)] and therefore argue that through automated provisioning of required infrastructure, modifications are simplified and can be done more quickly. In the originally formulated quality model this aspect was subsumed in a product factor called *Automated infrastructure*, but the results from the validation survey [1] showed an ambiguity for this factor, because several impacts on different quality aspects were reported. This can be seen in Table 1 where selected results from the validation survey are listed. On the basis of the survey results we therefore refined the factor and split it up into two separate factors *Automated infrastructure provisioning* and *Automated infrastructure maintenance* which impact the respective quality aspects. What has to be noted is that we kept the positive impact from *Automated infrastructure provisioning* on *Modifiability*, although the survey results did not show a clear confirmation. The reason is that we used the survey results as a basis for refining the quality model and not as definitive results, because the sample was not representative and participants were also not able to provide a reasoning for their ratings. To evaluate the product factor *Automated infrastructure provisioning*, the *infrastructure* entities need to be considered and it has to be evaluated whether they can be provisioned automatically, that means with little or no manual actions. This could, for example, be the case when the infrastructure is managed by a cloud provider and can be provisioned once needed by a component. Another case would be the usage of Infrastructure as Code (IaC) approaches for which an additional product factor *Use infrastructure as code* exists. A suitable measure could then, for example, be the ratio of infrastructure entities which can be provisioned automatically as a basic indicator.

Autonomous Fault Handling This factor is described as "*In cloud-native applications services should expect faults at different levels and either handle them or minimize their impact by relying on the capabilities of cloud environments.*".

Table 1 Exemplary results from the validation survey [1]

Factor	Quality Aspect	Hypothesized	Mean	Impact	Validation	p value
Automated infrastructure	Modifiability	Positive	0.45	+	✗	0.0224
	Modularity	n/a	0.60	+	n/a	0.0031
	Reusability	n/a	0.80	+	n/a	0.0028
	Testability	n/a	0.65	+	n/a	0.0997
	Capability	n/a	0.55	+	n/a	0.0008
	Elasticity	n/a	0.80	+	n/a	0.0028
	Resource utilization	n/a	0.60	+	n/a	0.0031
	Time-behaviour	n/a	0.60	+	n/a	0.0031
	Installability	n/a	0.95	+	n/a	0.0050
	Recoverability	Positive	1.15	+	✓	0.0006
	Availability	n/a	1.15	+	n/a	0.0006
	Fault tolerance	n/a	0.85	+	n/a	0.0006
Circuit breaked communication	Fault tolerance	Positive	1.12	+	(✓)	0.1455
	Recoverability	n/a	0.62	(+)	n/a	0.3182
Consistent centralized logging	Analyzability	Positive	1.33	+	(✓)	0.1185
	Testability	n/a	0.67	(+)	n/a	0.3158
	Recoverability	Positive	0.33	+	(✗)	0.1718
	Accountability	n/a	0.67	(+)	n/a	0.6399
Retries for safe invocations	Fault tolerance	Positive	1.20	+	(✓)	0.2695
	Availability	n/a	1.00	(+)	n/a	0.7222
	Recoverability	n/a	0.80	(+)	n/a	0.3519

"n/a" means that an impact was not included in the initial quality model, but was stated by survey participants

It is a mediating factor for the product factors *Invocation timeouts*, *Retries for safe invocations*, and *Circuit breaked communication*. *Circuit breaked communication* has been mentioned regularly in the literature [30, Chapter 10.1], [31, Use Circuit Breakers for Nontransient Failures (Chapter 6)], [32, 3.2.3] and it can be used "to prevent components from doing operations that will likely fail and are not transient" [31] so that an alternative action can be used until the affected component is healthy again. *Invocation timeouts* [33, Chapter 3], [32, 3.2.3] and *Retries for safe invocations* [30, 9.1], [31, Chapter 6], [33, Chapter 3] are also based on literature. In the validation survey, however, we only considered *Retries for safe invocations* and *Circuit breaked communication*, because we assumed *Invocation timeouts* to be less specific to cloud-native applications. For both it can be seen in Table 1 that their positive impact on *Fault Tolerance* could be confirmed, although we had to mark the results as potentially valid due to a low number of responses. To evaluate these product factors, the *link* entities need to be analyzed to check whether these mechanisms are used and in addition for the product factor *Retries for safe invocations*, the *endpoint* to which a *link* is connected, needs to be analyzed for checking whether it is safe to be invoked multiple times. Suitable measures have already been proposed in the literature, such as *Number of Links with retry logic* or *Number of Links with Complex Failover* by Apel et al. [9].

Consistent centralized logging This factor is described as "*Logging functionality should be concentrated in a centralized component which combines and stores logs from the components of a system. The logs should also be consistent regarding their format and level of granularity so that a correlation and analyzability of logs is facilitated.*". It has been formulated based on literature and covers aspects such as that logs are stored in a central component [31, Use a Unified Logging System (Chapter 6)], Logs are collected in a consistent format [31, Common and Structured Logging Format (Chapter 6)], or that functionalities for log aggregation and analysis are provided [30, Chapter 11.1], [32, Applying the Log aggregation pattern (11.3.2)]. The validation survey confirmed the positive impact on *Analyzability*, although the result had to be marked as potentially valid, because the p-Value was above the chosen significance level of 0.1. Furthermore, a positive impact on *Accountability* was added to the quality model based on the survey results. To evaluate this factor, the *backing service* is relevant, since a potential logging service would be modeled as such and then it could be checked which *components* provide logs to this *backing service*, for example by sending logs to a certain *endpoint*. As a suitable measure, the *Ratio of Components or Infrastructure nodes that export logs to a central service* by Ntentos et al. [34] could be used.

4.3 Support through measures

The support of the product factors through measures is a key aspect for enabling quantitative evaluations of software architectures. As described in Sect. 3.2, we have repeated the initial literature search for suitable measures and could identify additional measures for product factors of the quality model. This included also product factors for which we previously had not found any metrics, such as *Guarded Ingress* for which Ntentos et al. [34] proposed a metric called *Ingress Traffic Control utilization metric (ING)* which we slightly adapted to our model as *Ratio of endpoints whose ingress is guarded*. This adaptation was necessary because of the differing approaches for representing software architectures. Ntentos et al. [34] have developed their own modeling approach and for our model we have proposed the entities shown in Fig. 5.

As a general observation regarding the architectural measures proposed in the literature, we can state that a main focus has been on maintainability aspects, especially regarding coupling and cohesion [35, 36]. But more recently also measures in other areas such as security have been proposed [34].

Regarding measure support for the product factors of the quality model, there are two aspects which need to be addressed. On the one hand there are product factors for which no or only a few suitable measures have been proposed in the literature yet. For these, it is necessary to propose and validate new measures. And on the other hand, there are product factors for which numerous measures have been proposed, partly being very similar in terms of what they measure. For these, it is necessary to make a selection of meaningful measures that reflect the product factors in a suitable and understandable way. It has to be noted that so far only measures that have been already presented in the literature have been included in the quality model. As a next step further measures need to be formulated and validated in a structured way.

5 Discussion

In this section we want to reflect on the current state of the quality model, provide answers to the initially posed research questions, and outline future work regarding the formulation and validation of the quality model.

An interesting aspect that came up during our work on the quality model is how product factors and design patterns are related. Many of the practitioner books [30, 32, 33, 37] present patterns to achieve cloud-nativeness and also in the scientific literature the evaluation of the quality of an architecture is often based on patterns [36, 38–41]. From our point of view, product factors and patterns are highly interrelated and in some cases a product factor directly maps to a pattern (for example *Circuit breaked communication* [30]). Ideally, however, patterns are reflected by certain formations of entities and their respective properties within an architecture. When an architecture is evaluated, the fact that a certain pattern was used, should be measurable through corresponding architectural measures which measure a related product factor. The quality aspect that this product factor impacts should then be the same as the one also targeted by the original pattern, leading essentially to the same outcome that if a certain pattern was used, it should have an impact on a certain quality aspect.

Another interesting aspect to discuss is that a majority of impact relations in the quality model describes positive impacts on quality aspects. This is to a large extent due to our chosen methodology which considered practitioner books on how to implement cloud-native applications for the initial formulation of the quality model. This literature highlighted the potential benefits of using cloud-native concepts. Through the validation survey we also aimed at potential negative impacts which could indeed be identified, but only to some extent, for example the negative impact of *Consistently mediated communication* on *Time behaviour*. Because there are, however, typically trade-offs between quality aspects [42], this should in the end also be visible in the quality model. Further work on the quality model, especially considering use cases, should therefore also put a focus on potential negative impacts of application characteristics.

While keeping these aspects in mind, we are nevertheless confident that our quality model shown in Fig. 4 adequately describes how characteristics of cloud-native applications impact multiple quality aspects, therefore providing an answer to **RQ1**. Because of the reliance on the Quamoco meta model [12], the quality model is formulated on a sound theoretical foundation. It provides the basis for being further developed to enable quality evaluations of software architectures. The methodology that we used for formulating the quality model aims to show on the one hand how the model was developed in this specific case. But on the other hand, it also shows what needs to be considered more generally when developing a quality model and which approaches can be used. The presentation and demonstrated application of our methodology therefore represents the answer to **RQ2** of how a quality model can be formulated and validated. We want to make clear that it is one of several ways of formulating and validating a quality model and that different approaches can be taken, also in an iterative way. In fact, the more a quality model can be validated from different perspectives and based on different types of rationale, the more confidence can be put in its applicability.

Thus, a major aspect for future work is to use additional validation approaches for the quality model, namely an experimental approach as already mentioned in Sect. 3

Step: E. To facilitate this, corresponding tooling support is crucial. Specifically, the tooling support should enable the intuitive modeling of software architectures using the entities shown in Fig. 5, the specification of measures based on modeled architectures, and their automated calculation so that they can be used for evaluations of the quality model. A prototype focusing on the modeling functionality has been presented in [26] and is the basis for further implementation of tooling support.[8] One aspect to discuss in this regard is how architectural models are created. Generally, it is possible to either manually model software architectures based on the understanding of the software or to automatically generate models based on artifacts such as source code or deployment descriptors. While it would significantly facilitate the usage of tooling if models could be generated automatically, the problem is the heterogeneity of technologies available in the context of cloud-native applications to which tooling in turn would need to be adapted. Ntentos et al. [34] have taken this approach of generating models automatically from IaC artifacts, but had to start with one specific technology to then continuously include further technologies. We decided to preliminary provide only manual modeling of software architectures, because we wanted to focus on the relations between architectural characteristics and quality aspects in the quality model. Therefore our aim for the tooling support is to make the manual modeling nevertheless as intuitive as possible. In addition, we based the modeling approach on TOSCA [43] as a standard. This provides the foundation for transforming deployment descriptors of various technologies to a TOSCA template, if adapters are available, and therefore improves the compatibility of our tooling.

6 Related work

In a broader context, our work can be seen as a part of research on Software Architecture Optimization methods which has been reviewed by Aleti et al. [44]. By using the taxonomy presented by Aleti et al. [44], our work can be characterized so that it can be better related to and compared with other work. We have done this in Table 2 which shows the different categories of the taxonomy by Aleti et al. [44] and a selection of related work to which our approach is compared to. The common aim of these approaches is to evaluate the quality of an architecture and, based on this evaluation, suggest potential options for quality improvements. Differences, however, can be seen when considering the categorizations in more detail.

The scope is set by the *problem* category which characterizes the problem tackled by an approach. It can be differentiated based on the *domain*, the *phase* of the software development lifecycle it is used in, and the covered quality aspects. Regarding quality aspects, Aleti et al. differentiate between *quality attributes* which are to be optimized, and *constraints* which are specified upfront and limit the potential solutions. Furthermore, with the *dimensionality* it can be expressed whether multiple quality dimensions are considered simultaneously, potentially conflicting with each other, or if there is a single objective. In contrast to other approaches, when using a hierarchical quality model, typically multiple quality attributes are considered and a quality evaluation and potential optimization aims to weigh multiple objectives against each other. This is a commonality between our approach and the approaches in Table 2. On which basis the quality attributes are defined, however, differs. While our approach is based on the quality aspects of the ISO 25000 standard [16] combined with cloud-native characteristics, Mayr et al. [45] focus on requirements derived from expert knowledge. Achilleos et al. [46] formulate Service Level Objectives (SLO), for example regarding the response time of a system. And Soldani et al. [10] rely on best practices for microservices-based systems obtained from literature. With quality models, constraints are formulated implicitly within the model, while Achilleos et al. [46] allow for a custom specification of constraints. For the approach of Soldani et al. [10] constraints may come from external business-specific considerations that might justify violations of certain best practices. Regarding the *solution* characteristics, it can be seen that the other approaches are similarly based on models for the *quality evaluation*, but the types of models used for the *Architecture representation* differ significantly. They range from directly using source code [45] or using deployment-focused descriptions such as CAMEL [46] or μTOSCA [10]. The *degrees of freedom* describe which aspects of an architecture are considered to be changeable in order to improve the quality of a system. These degrees are highly related to the used representation of a system and while, for example, Achilleos et al. [46] have a more narrow focus on deployment configurations to also enable runtime adaptations, our scope is broader considering both, how components are structured and interact with each other and the deployment perspective. For the *strategy* used to enable optimizations, multi-objective problems are often approximate and problem-specific options need to be evaluated. The approach of Mayr et al. [45] can be categorized as being more exact, because of the focus on requirements which allows for an optimization in terms of either fulfilling requirements or not. And regarding *constraint handling*, different approaches are used related to how constraints are expressed. For example, in the case of Achilleos et al. [46] deployment options that violate specified constraints are not considered at all, while in our approach constraints occur implicitly within the model and therefore potential violations might be acceptable if other quality aspects are considered more important. Finally, a comparison is possible regarding how

[8] https://github.com/r0light/cna-quality-tool.

Table 2 Comparison of our approach to other work based on the taxonomy by Aleti et al. [44]

Category	Our approach	Mayr et al. [45]	Achilleos et al. [46]	Soldani et al. [10]
Problem				
Domain	Cloud applications	Embedded systems	Cloud deployment	Microservices
Phase	Design time	Design time	Design- and runtime	Design time
Quality attribute	Multiple	Multiple, based on requirements	SLOs focusing on performance, cost, security	Best practices
Dimensionality	Multi-objective	Multi-objective	Multi-objective	Multi-objective
Constraint	Implicit	Implicit	Custom, e.g. cost, hardware	External
Solution				
Architecture representation	TOSCA extension	Source code	CAMEL	μTOSCA
Quality evaluation	Model-based	Model-based	Model-based	Model-based
Degrees of freedom	Component types, interactions and deployment	Source code changes	Deployment configuration	Component interactions and configuration
Optimization strategy	Problem-specific approximate	Problem-specific exact	Problem-specific approximate	Problem-specific approximate
Constraint handling	General	Penalty	Prohibit	General
Validation				
Approach validation	Survey	Expert judgment	Use cases and survey	Use cases
Optimization validation	t.b.d	Expert judgment	Use cases	Use cases

an approach as such is validated and whether an optimization done through the approach has been validated. In our case, a validation has been performed through the survey and an actual optimization based on the quality model is planned for future work. For the other compared approaches, several different types of validations have been done, although applying an approach to an actual use case is a common procedure.

With this comparison, we aim to set our work in a broader context and compare it to other related work. Although only a selection of different work was used, other work can also be compared to ours based on the taxonomy by Aleti et al. [44]. But from this comparison it becomes clear already that despite the common goal of evaluating and improving architectural aspects of applications, there is a large variety of problems and corresponding solutions to consider. Especially for approaches that cover a similar domain as ours, there is a significant potential for synergies in terms of integrating knowledge from these approaches in our quality model. A closely related domain is that of microservices for which approaches have been presented that evaluate the architecture of a microservices based system. The work included in our comparison by Soldani et al. [10] belongs to this category, but also works focusing on security aspects of microservices [28, 34, 47]. Furthermore, in the context of microservices, the analysis of anti-patterns [48, 49] or patterns [41, 42] together with an application to microservices architectures has been investigated. Another closely related domain is that of cloud deployments for which the work included in the comparison by Achilleos et al. [46] represents an example. Additional related work again applies a focus on patterns used in the context of cloud deployments [40, 50, 51] and how these can be used to detect problems or compare different deployment options. As mentioned, many of these related works provide aspects that are integrable in our approach. But they differ regarding their specific domain, formulation of quality attributes, architectural representations, and degrees of freedom. To summarize, our work thus represents a novel approach which can be differentiated from other existing work specifically based on:

- the considered domain: Because we cover both, how components of cloud applications interact and how they are deployed
- the consideration of multiple quality aspects in a hierarchy: Because we aim at the capability to also express trade-offs between quality aspects
- the abstraction from specific technology: Because we take into account the technological heterogeneity of cloud applications and therefore abstract from it

7 Conclusion

With cloud-native being a recent and noteworthy topic, being able to assess how applications can benefit from cloud-native concepts and technologies is an advantageous capability for software developers and architects. Due to its breadth of

scope and the technological heterogeneity, however, this represents a challenge. With the quality model for software architectures of cloud-native applications presented in this work, we provide an approach for tackling this challenge. From a theoretical perspective, the quality model aims to provide a structured understanding about how cloud-native characteristics impact different quality aspects. This is in line with the possibility of using the quality model for a qualitative evaluation of software architectures. For a quantitative evaluation and therefore a more practical perspective, however, a further enhancement of the quality model is needed for which we have built the foundation and plan to continue in future work. Furthermore, our work represents a methodological contribution on how quality models can be formulated and validated. Through the structured and comprehensive description of the approach for formulating and validating the quality model, we aim to make approaches for developing quality models more comparable and therefore comprehensible.

Author contributions All authors contributed to the study conception and design. Material preparation, data collection and analysis were performed by Robin Lichtenthäler and Guido Wirtz. The first draft of the manuscript was written by Robin Lichtenthäler and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Funding Open Access funding enabled and organized by Projekt DEAL. The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

Data availability The qualitiy model, as well as more details on the formulation and validation processes, such as literature references and the raw result data from the survey can be found here: https://r0light.github.io/cna-quality-model/.

Declarations

Competing interests The authors have no relevant financial or non-financial interests to disclose.

Ethical approval Not applicable.

Consent to participate Not applicable.

Consent for publication Not applicable.

References

1. Lichtenthäler, R., Fritzsch, J., Wirtz, G.: Cloud-native architectural characteristics and their impacts on software quality: a validation survey. In: 2023 IEEE International Conference on Service-Oriented System Engineering (SOSE). IEEE Computer Society, Los Alamitos, CA, USA (2023). https://doi.org/10.1109/SOSE58276.2023.00008
2. Cerny, T., et al.: On code analysis opportunities and challenges for enterprise systems and microservices. IEEE Access **8**, 159449–159470 (2020). https://doi.org/10.1109/access.2020.3019985
3. Fowler, M.: Patterns of Enterprise Application Architecture, 1st edn. Pearson International, Toronto (2002)
4. Gannon, D., Barga, R., Sundaresan, N.: Cloud-native applications. IEEE Cloud Comput. **4**, 16–21 (2017). https://doi.org/10.1109/mcc.2017.4250939
5. Kratzke, N., Quint, P.-C.: Understanding cloud-native applications after 10 years of cloud computing—a systematic mapping study. J. Syst. Softw. **126**, 1–16 (2017). https://doi.org/10.1016/j.jss.2017.01.001
6. Balalaie, A., Heydarnoori, A., Jamshidi, P.: Migrating to Cloud-Native Architectures Using Microservices: An Experience Report, pp. 201–215. Springer, New York (2016). https://doi.org/10.1007/978-3-319-33313-7_15
7. Torkura, K. A., Sukmana, M. I., Meinel, C.: Integrating continuous security assessments in microservices and cloud native applications. In: Proceedings of the10th International Conference on Utility and Cloud Computing. ACM (2017). https://doi.org/10.1145/3147213.3147229
8. Lichtenthäler, R., Wirtz, G.: Towards a quality model for cloud-native applications. In: Service-Oriented and Cloud Computing, pp. 109–117. Springer, New York (2022). https://doi.org/10.1007/978-3-031-04718-3_7
9. Apel, S., Hertrampf, F., Späthe, S.: Towards a metrics-based software quality rating for a microservice architecture. In: 19th I4CS, pp. 205–220. Springer, New York (2019). https://doi.org/10.1007/978-3-030-22482-0_15
10. Soldani, J., Muntoni, G., Neri, D., Brogi, A.: The μTOSCA toolchain: mining, analyzing, and refactoring microservice-based architectures. Software (2021). https://doi.org/10.1002/spe.2974
11. Camilli, M., Guerriero, A., Janes, A., Russo, B., Russo, S.: Microservices integrated performance and reliability testing. In: Proceedings of the 3rd ACM/IEEE International Conference on Automation of Software Test, AST '22, 29–39. Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3524481.3527233
12. Wagner, S. et al.: The quamoco quality meta-model. Tech. Rep. TUM-I128, TU München, Institut für Informatik (2012). https://mediatum.ub.tum.de/attfile/1110600/hd2/incoming/2012-Jul/517198.pdf
13. Ferenc, R., Hegedűs, P., Gyimóthy, T.: Software product quality models. In: Evolving Software Systems, pp. 65–100. Springer, Berlin (2013). https://doi.org/10.1007/978-3-642-45398-4_3
14. McCall, J.A., Richards, P.K., Walters, G.F.: Factors in software quality, vol. I. Concepts and definitions of software quality. Techreport ADA049014, General Electric Co (1977). https://apps.dtic.mil/sti/pdfs/ADA049014.pdf
15. Boehm, B.W., Brown, J.R., Lipow, M.: Quantitative evaluation of software quality. In: Proceedings of the 2nd International Conference on Software Engineering, ICSE '76, pp. 592-605. IEEE Computer Society Press, Washington, DC, USA (1976). https://doi.org/10.5555/800253.807736
16. ISO/IEC: ISO/IEC 25000 Systems and software engineering—Systems and software Quality Requirements and Evaluation (SQuaRE) (2014). https://www.iso.org/standard/64764.html

17. Nistala, P., Nori, K.V., Reddy, R.: Software quality models: a systematic mapping study. In: 2019 IEEE/ACM International Conference on Software and System Processes (ICSSP). IEEE (2019). https://doi.org/10.1109/icssp.2019.00025
18. Deissenboeck, F., Wagner, S., Pizka, M., Teuchert, S., Girard, J.-F.: An activity-based quality model for maintainability. In: 2007 IEEE International Conference on Software Maintenance, pp. 184–193. IEEE (2007). https://doi.org/10.1109/icsm.2007.4362631
19. Yan, M., Xia, X., Zhang, X., Xu, L., Yang, D.: A systematic mapping study of quality assessment models for software products. In: International Conference on Software Analysis, Testing and Evolution (SATE). IEEE (2017). https://doi.org/10.1109/sate.2017.16
20. Galli, T., Chiclana, F., Siewe, F.: Software product quality models, developments, trends, and evaluation. SN Comput. Sci. (2020). https://doi.org/10.1007/s42979-020-00140-z
21. Oriol, M., Marco, J., Franch, X.: Quality models for web services: a systematic mapping. Inf. Softw. Technol. **56**, 1167–1182 (2014). https://doi.org/10.1016/j.infsof.2014.03.012
22. Letouzey, J.-L., Coq, T.: The SQALE analysis model: an analysis model compliant with the representation condition for assessing the quality of software source code. In: 2010 Second International Conference on Advances in System Testing and Validation Lifecycle. IEEE (2010). https://doi.org/10.1109/valid.2010.31
23. AL-Badareen, A.B., Desharnais, J.-M., Abran, A.: A suite of rules for developing and evaluating software quality models. In: Software Measurement, pp. 1–13. Springer, New York (2015). https://doi.org/10.1007/978-3-319-24285-9_1
24. Moody, D.L.: Theoretical and practical issues in evaluating the quality of conceptual models: current state and future directions. Data Knowl. Eng. **55**, 243–276 (2005). https://doi.org/10.1016/j.datak.2004.12.005
25. Lichtenthäler, R., Wirtz, G.: A review of approaches for quality model validations in the context of cloud-native applications. In: 14th Central European Workshop on Services and their Composition (ZEUS), pp. 30–41. CEUR-WS (2022). https://ceur-ws.org/Vol-3113/paper6.pdf
26. Dürr, K., Lichtenthäler, R.: An evaluation of modeling options for cloud-native application architectures to enable quality investigations. In: 2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC). IEEE (2022). https://doi.org/10.1109/ucc56403.2022.00053
27. McDonald, J.H.: Handbook of Biological Statistics, 3rd edn. Sparky House Publishing, Baltimore (2014)
28. Zdun, U., et al.: Microservice security metrics for secure communication, identity management, and observability. ACM Trans. Softw. Eng. Methodol. (2023). https://doi.org/10.1145/3532183
29. Reznik, P., Dobson, J., Gienow, M.: Cloud Native Transformation. O'Reilly, Newton (2019)
30. Davis, C.: Cloud Native Patterns. Manning, Shelter Island (2019)
31. Scholl, B., Swanson, T., Jausovec, P.: Cloud Native. O'Reilly, Newton (2019)
32. Richardson, C.: Microservices Patterns, 1st edn. Manning, Shelter Island (2019)
33. Indrasiri, K., Suhothayan, S.: Design Patterns for Cloud Native Applications. O'Reilly, Newton (2021)
34. Ntentos, E., Zdun, U., Falazi, G., Breitenbucher, U., Leymann, F.: Assessing architecture conformance to security-related practices in infrastructure as code based deployments. In: 2022 IEEE International Conference on Services Computing (SCC). IEEE (2022). https://doi.org/10.1109/scc55611.2022.00029
35. Bogner, J., Wagner, S., Zimmermann, A.: Automatically measuring the maintainability of service-and microservice-based systems: a literature review. In: Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement, pp. 107–115. ACM (2017). https://doi.org/10.1145/3143434.3143443
36. Ntentos, E., Zdun, U., Plakidas, K., Meixner, S., Geiger, S.: Assessing architecture conformance to coupling-related patterns and practices in microservices. In: ECSA, pp. 3–20. Springer, New York (2020). https://doi.org/10.1007/978-3-030-58923-3_1
37. Ibryam, B., Huß, R.: Kubernetes Patterns. O'Reilly, Newton (2020)
38. Zdun, U., Navarro, E., Leymann, F.: in Ensuring and assessing architecture conformance to microservice decomposition patterns. In: ICSOC, pp. 411–429. Springer, New York (2017). https://doi.org/10.1007/978-3-319-69035-3_29
39. Ntentos, E., Zdun, U., Plakidas, K., Meixner, S., Geiger, S.: Metrics for assessing architecture conformance to microservice architecture patterns and practices. In: ICSOC, pp. 580–596. Springer, New York (2020). https://doi.org/10.1007/978-3-030-65310-1_42
40. Yussupov, V. et al.: Serverless or serverful? a pattern-based approach for exploring hosting alternatives. In: Service-Oriented Computing, pp. 45–67. Springer, New York (2022). https://doi.org/10.1007/978-3-031-18304-1_3
41. Daniel, J., Guerra, E., Rosa, T., Goldman, A.: Towards the detection of microservice patterns based on metrics. In: 2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 132–139 (2023). https://doi.org/10.1109/SEAA60479.2023.00029
42. Vale, G. et al.: Designing microservice systems using patterns: an empirical study on quality trade-offs. In: 2022 IEEE 19th International Conference on Software Architecture (ICSA). IEEE Computer Society (2022). https://doi.org/10.1109/ICSA-C54293.2022.00020. arxiv: 2201.03598
43. OASIS: TOSCA Simple Profile in YAML Version 1.3 (2020). https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/. OASIS Standard
44. Aleti, A., Buhnova, B., Grunske, L., Koziolek, A., Meedeniya, I.: Software architecture optimization methods: a systematic literature review. IEEE Trans. Softw. Eng. **39**, 658–683 (2013). https://doi.org/10.1109/tse.2012.64
45. Mayr, A., Plosch, R., Klas, M., Lampasona, C., Saft, M.: A comprehensive code-based quality model for embedded systems: systematic development and validation by industrial projects. In: 23rd International Symposium on Software Reliability Engineering. IEEE (2012). https://doi.org/10.1109/issre.2012.4
46. Achilleos, A.P., et al.: The cloud application modelling and execution language. J. Cloud Comput. (2019). https://doi.org/10.1186/s13677-019-0138-7
47. Bambhore Tukaram, A. et al.: Towards a security benchmark for the architectural design of microservice applications. In: Proceedings of the 17th International Conference on Availability, Reliability and Security, ARES 2022. ACM (2022). https://doi.org/10.1145/3538969.3543807
48. Ponce, F., Soldani, J., Astudillo, H., Brogi, A.: Smells and refactorings for microservices security: a multivocal literature review. J. Syst. Softw. **192**, 111393 (2022). https://doi.org/10.1016/j.jss.2022.111393
49. Taibi, D., Lenarduzzi, V., Pahl, C.: Microservices anti-patterns: a taxonomy. In: Microservices, pp. 111–128. Springer, New York (2019). https://doi.org/10.1007/978-3-030-31646-4_5
50. Saatkamp, K., Breitenbücher, U., Kopp, O., Leymann, F.: An approach to automatically detect problems in restructured deployment models based on formalizing architecture and design patterns. SICS Soft.-Intensive Cyber-Phys. Syst. (2019). https://doi.org/10.1007/s00450-019-00397-7
51. Sousa, T., Ferreira, H.S., Correia, F.F.: A survey on the adoption of patterns for engineering software for the cloud. IEEE Trans. Softw. Eng. (2021). https://doi.org/10.1109/tse.2021.3052177

Robin Lichtenthäler M.Sc., is a Ph.D. student at the Distributed Systems Group of the University of Bamberg. In his research, he focuses on software architectures of cloud-native applications and their defining characteristics. He is currently working on a quality model for cloudnative software architectures based on which applications can be evaluated.

Guido Wirtz is a full professor of computer science, heads the Distributed Systems Group of the University of Bamberg, and is Vice President of Technology and Innovation of the University of Bamberg. He received his Ph.D. from the University of Bonn and his habilitation from the University of Siegen. His main research areas are in the field of software development for complex, esp. distributed, systems on all levels. This includes design methods, visual languages and tools for distributed systems development as well as middleware, SOA and cloud computing. Current interests are on the seamless transition from business processes to their implementation in a SOA and cloud context as well as in new models for cloud computing like, e.g., serverless.