Electronic Communications of the EASST
Volume 60 (2013)

Proceedings of the Seventh International Workshop on Software Quality and Maintainability

-

Bridging the gap between end user expectations, vendors' business prospects, and software engineers' requirements on the ground (SQM 2013)

A Meta Model for Software Architecture Conformance and Quality Assessment

Andreas Goeb

20 pages

Guest Editors: Eric Bouwers, Yijun Yu
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

A Meta Model for Software Architecture Conformance and Quality Assessment

Andreas Goeb*

andreas.goeb@sap.com
SAP AG, Darmstadt, Germany

Abstract: Software architecture and design suffer from a lack of documented knowledge on how different architectural styles influence software quality. Existing software quality models do not allow engineers to evaluate whether a given software system adequately implements the basic principles of the chosen architectural style, and which architectural properties and best practices beyond these principles contribute to the system's quality. In this paper, I present a meta quality model for software architectures, which can be used not only as a knowledge-base to easily compare architectural styles based on their impact on software quality, but also to increase efficiency of architectural quality analysis by leveraging existing modeling concepts and tools. An experiment performing an architecture assessment using a quality model for the SOA architectural style not only showed that the approach is applicable in practice, but also indicated a reduction of manual effort compared to other architecture assessment approaches.

Keywords: Quality Model; Software Architecture; Design; Conformance

1 Introduction

1.1 Motivation

Current trends in the software market show that quality becomes a differentiating factor among software products with decreasing functional diversification. It is widely accepted that software quality problems can be handled easier and more cost efficient, the earlier they are detected during development [RV98, BWDV00]. In particular, almost all of a software system's quality attributes are influenced by its architecture [CKK01, p. 19]. Consequently, it is particularly relevant for software engineering research and practice to develop means for efficient software quality assessment on the architectural level.

1.2 Problem

According to Svahnberg and Wohlin [SW05], there is a lack of documented knowledge on how different architectural styles influence software quality. This forces software architects to base the selection of an architectural style purely on personal experience rather than objective information.

* This work has partially been supported by the German Federal Ministry of Education and Research (BMBF) in the project Quamoco (01 IS 08023D).

Moreover, software architecture evaluation generally requires a considerable amount of manual effort, because most established techniques are based on the manual analysis of scenarios. In particular with the emergence of new deployment models in the context of cloud applications, where small increments of updated functionality are delivered in very short periods of time, this approach becomes impractical in the long term. Because of the highly individual nature of these techniques, they are not designed to be applied repeatedly. Moreover, applying these techniques to more than one software project involves individual preparation effort for all of them. Within the software development process, this also implies that quality assessment approaches used in the architecture phase fundamentally differ from those used in the implementation phase, leading to media discontinuities during quality assurance. This complicates continuous quality monitoring and control and therefore negatively impacts the costs of quality assurance.

1.3 Contribution

In this paper, I present an architecture-specific extension to the Quamoco meta quality model [WLW+12a]. The proposed quality model structure explicitly separates *conformance* concepts from *design* best-practices and can be used both to derive statements on the relationships between architectural properties and product quality, and to increase efficiency of architecture quality analysis due to large automation potential. While building architecture quality models according to this approach still involves much manual work and expert knowledge, these models can be used to repeatedly evaluate software architectures. Moreover, the investment of building such models pays off when they are reused to evaluate a larger number of software systems.

1.4 Structure

The remainder of this paper is structured as follows: Section 2 summarizes related work according to software quality models and architecture evaluation approaches. In Section 3, I introduce architecture-specific additions to the Quamoco meta quality model. Section 4 explains the contents of an architecture conformance and quality model, in particular quality goals, architectural principles, general design properties, and corresponding measures, as well as the overall approach of building architecture quality models and using them for architecture evaluation. In Section 5, the approach is experimentally applied by conducting an architecture evaluation, using a quality model for service-oriented architectures as an example for a specific architectural style. Finally, Section 6 concludes the paper and outlines directions for future work.

2 Related Work

2.1 Software Quality Models

Modeling software quality has been a topic addressed by researchers for several decades. Early models for software quality date back to the late 1970s, e. g. those by Boehm [BBK+78] or McCall [MRW77], which hierarchically decompose software quality into more tangible concepts. This approach has led to international standards for software quality like ISO 9126 or its successor ISO 25010, which are reported to be used as a basis for software quality assurance in many

software companies. However, recent studies also show that these standards are too abstract to be directly used for quality assessment [WLW+12b]. Because of this shortcoming, there have been several initiatives in defining models that not only describe software quality, but can also be used to assess software systems with regard to their quality. The Squale project enhanced the ISO 9126 quality model with so-called *practices* containing information on how to measure certain quality characteristics [MBD+09]. They also provide tools to assess source code quality in different programming languages. In summary, software quality models help in developing a common understanding of software quality. Some models can be used for automatic quality assessment, lowering the effort compared to inspection-based approaches. Most of these assessment models, however, are targeted at low-level source code constructs only, not taking architectural properties into account.

The NFR-Framework [MCN92] and its application to architectural software design by Chung et al. [CNY95] provide a framework and a methodology for handling non-functional requirements during software development. NFRs are represented as goals, which can be decomposed. The software architect assesses the contribution of each design alternative to each (sub-) goal. First, this is done from a general perspective on the underlying design patterns. The resulting goal graph is then refined in the context of the specific system under construction. The approach is very similar to the one presented here in the sense that it starts with collecting and formalizing documented knowledge about the impacts of certain architectural constructs on software quality aspects. The fundamental difference is that Chung et al. take a qualitative approach in order to guide architectural decisions during the process of building a software system, while the meta-model and the approach presented here explicitly contain measures in order to also provide quantitative insight for already existing software systems.

2.2 Software Architecture Evaluation

Software architecture is crucial for software quality. This means that the decision for a particular architecture is highly dependent on the quality goals of the involved stakeholders. Clements et al. [CKK01] phrase this very concisely: “If the sponsor of a system cannot tell you what any of the quality goals are for the system, then any architecture will do.”

If architectural decisions are so important for the quality of a software system, architecture assessment appears to be a feasible means to provide statements about its quality. To accomplish this, architecture evaluation methods have been developed, which follow clear rules and provide a certain degree of repeatability. Clements et al. categorize these methods according to the tools they use: *Questioning* methods are based on scenarios, questionnaires, and checklists for system analysis. These are often used in conjunction with system documentation or specification, thereby not requiring the system to be already completely implemented. In contrast, *measuring* methods directly analyze the respective system by means of automatic analysis tools, e. g. calculating software metrics or simulating system behavior. In any case this second group of methods requires the presence of software artifacts and can therefore not be applied as early as scenario-based methods.

Clements et al. [CKK01] propose three scenario-based methods, namely *SAAM*, *ARID*, and *ATAM*. They all start with the elicitation of scenarios in workshops. A scenario might be: “In addition to local deployment and operation, you should also be able to operate the system on a

cloud platform". Based on a prioritized list of such scenarios, different architecture alternatives are then evaluated regarding their ability to facilitate these scenarios. Depending on the method and the particular situation, different techniques can be used, e. g. sensitivity and tradeoff analysis in ATAM, scenario walk-throughs in SAAM, or Active Design Reviews [PW85] in ARID. A comparison of these methods is shown in [CKK01, p. 256]. The authors state that a mid-size architecture evaluation using ATAM would take around 70 person days, assuming a reasonable team size.

Vogel [VAC+09] presents a general overview on architecture evaluation methods. Moreover, Zhao [Zha99] provides links to further literature. It is generally observed that according to the classification above, the overwhelming majority of architecture evaluation methods belong to the group of questioning methods, thus requiring large amounts of manual effort. This might be due to the fact that architecture analysis is generally performed in a project-specific context with individual requirements. For domain and project-independent parts of the analysis, tool supported approaches are available, e. g. ConQAT[1] can automatically compare dependencies between components of a software system with the architecture specification and visualize the results.

Losavio et al. [LCM+04, LCLR03] present an architecture measuring approach for evaluating software quality according to ISO 9126. They consecutively walk through all quality characteristics and sub-characteristics contained in the ISO standard and present measurement specifications in order to quantify these on an architectural level. In total, they present 16 *attributes* and associated *metrics*. Out of these, nine are defined in a binary fashion and require identifying whether there is a mechanism in the architecture to support the respective sub-characteristic, e. g. *co-existence* is determined by "the presence of a mechanism facilitating the co-existence" [LCM+04]. Three of the remaining metrics are defined to aggregate the respective values from the individual components and connectors. In particular, no further adjustments to these individual scores are made based on the architecture, e. g. *maturity* is defined as the sum of the maturities of all components and connectors the architecture consists of [LCM+04]. In conclusion, the proposed approach provides a unified process framework for architecture quality assessment. Since over half of the metrics are Boolean and require thorough expert assessment, the approach has to be considered mainly checklist-based. Most of the measures are defined on a high granularity that makes it difficult to automate measurement by implementing the proposed metrics in a tool.

In summary, most approaches for software architecture evaluation either do not provide a way to automate assessment steps or require executable software artifacts in order to do so. Although some scenario-based approaches offer sophisticated methodology to support project-specific architectural decisions, none of the existing approaches provides a way to quickly obtain a general statement of the overall conformance and quality of a software architecture.

3 Basic Modeling Concepts

This section briefly describes the meta quality model that we developed in the Quamoco project [WLH+12, WLW+12a]. It addresses the shortcomings of related approaches in software qual-

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

ity modeling presented in Subsection 2.1. Because this meta model provides the basis for the architecture extensions proposed in Section 4, its elements are introduced in the following.

Entities provide a decomposition of the software product. Starting from the complete Product, entities can refine other entities along an *is-a* or a *part-of* relation, e. g. both entities Source Code and Documentation refine Product. Decomposition is usually performed as required, until a depth is reached that is sufficiently detailed to describe the desired concepts. The entity Return Parameter (of a method) would refer to Parameter using the is-a relation. In turn, the parameter would specify that it is part of a Method, which is in turn part of an Interface. Note that entities describe things on a conceptual level, not individual instances within an assessed system (i. e. the return parameter of a certain operation).

These entities are characterized by *attributes* (e. g. ATOMICITY, PRECISE NAMING, COHESION) in order to define *product factors*. Product factors describe observable properties of these entities that may or may not be present in a particular system to a certain degree. The degree is expressed in the factor's *value range*, which includes all real numbers from 0 to 1. The factor [Service | PRECISE NAMING] is completely fulfilled (thus evaluating to 1.0) for a system, whose services are all named precisely.

The Quamoco meta quality model allows for several kinds of *quality aspects* in order to cover a wide range of established ways of decomposing product quality. Wagner et al. [WLH+12] structure product quality using the quality characteristics defined in ISO 25010. Other possible quality aspect forms include activity-based quality goals (c. f. [BDP06]). These have been proposed in order to provide more natural and meaningful decomposition semantics, and are therefore used in the following. Activity-based quality goals are comprised of an *activity*, which is performed on or with the system, and an *attribute* characterizing this activity. A typical quality goal from a service consumer's perspective is the efficient analysis of the functionality provided by a service: [Analysis | EFFICIENCY]. The fact that the presence of a product factor in a software system affects the fulfillment of a quality goal is represented by an *impact*. Since the effect can be positive or negative, the impact is annotated with $+$ or $-$, respectively. For example, the idea that precise naming helps a user to analyze whether a service provides the functionality he needs is represented as: [Service | PRECISE NAMING] $\xrightarrow{+}$ [Analysis | EFFICIENCY].

In summary, product factors bridge the gap between abstract categories of quality and observable properties of software artifacts. In order to assess to which degree a factor is fulfilled for a particular system, the quality model contains *measures*. They provide means to actually obtain data about the system. Depending on the particular technology or programming language used in the software product, these measures make use of different kinds of *instruments*, either tool-based ones using static analysis tools and metrics, or manual ones by defining steps for an inspection of the respective entities. For aggregation purposes, *evaluations* translate the values of measures assigned with a factor to a degree of fulfillment between 0 and 1. Figure 1 depicts the Quamoco meta quality model.

Quality model elements can be grouped into *modules* to facilitate reuse and maintenance of quality models. As an example, source code quality models can be split into modules according to programming paradigms or languages, so that general source-code related concepts can be reused within an object orientation module, which is then further operationalized by modules containing measures and instruments for C# or Java. This way, technology-independent information can be reused, while technology-dependent modules add automation by linking general

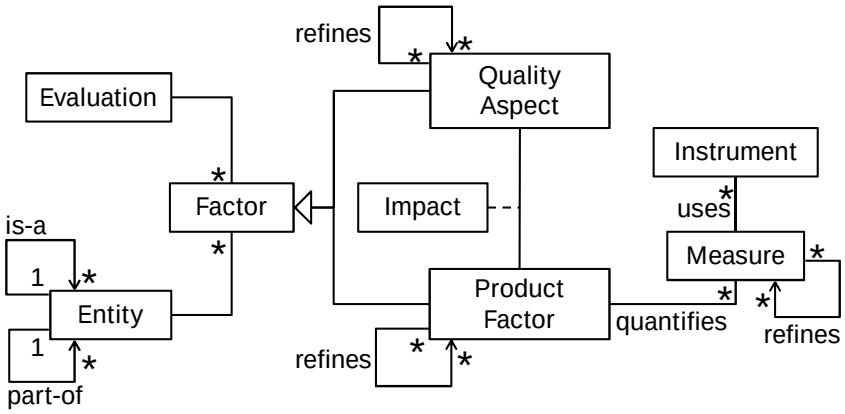

Figure 1: Quamoco meta quality model (source: [WLH+12])

measures to analysis tools. In addition, this modularization concept can be used to extend or adapt quality models. Project-specific quality requirements can be added as an individual module, and evaluation formulas can be overridden in order to adapt priorities

according to the project goals. Another possible scenario for using modules is splitting up a quality model into a light variant, which only contains measures that can quickly be obtained by tools, and a full variant, which adds more elaborate analyses to the assessment. The light variant could then be included into a continuous integration environment, whereas the full variant could be executed manually before certain milestones or quality gates.

More details on the modeling concepts, the elements that constitute the quality model, as well as the relations between them, can be found in [WLH+12]. This paper proposes an extension to this meta model to specifically address software architecture quality so that architectural styles can be compared based on their impact on software quality and existing software architectures can be evaluated with respect to both architecture conformance and quality.

4 Architecture Model Extension

To specifically address software architecture quality in the context of a given architectural style, I propose an extension to the meta model presented in the previous section. In order to retain compatibility with the existing tools for editing, maintaining, visualizing, and executing quality models, this extension is based on conventions, so that e. g. instead of formally adding a new model element type, I propose adding certain semantics to existing element types. Technically, already the Quamoco meta quality model does so by using the *factor* concept for both quality aspects and product factors.

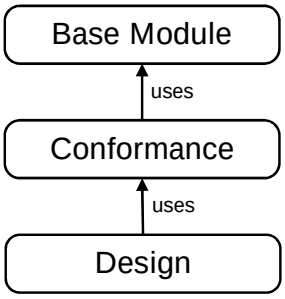

Figure 2: Modular structure

4.1 Modules

Quality goals like [Adaptation | EFFICIENCY] are usually independent from architectural styles. In particular this is the case for quality standards that do not make any assumptions on the architecture of the software to assess (e. g. ISO 25010). Therefore, quality goals should be usable across different quality models and are hence defined within an independent *Base* module.

According to the overall question, which quality goals are directly influenced by the underlying principles of a certain architectural style, all quality model elements directly related to these principles are subsumed in a module named *Conformance*. Other quality-related concepts beyond these principles constitute the *Design* module.

Within the *conformance* module, the main elements are the *Architecture Principles*, which are modeled as a special case of product factors. Their degree of fulfillment states how well the system under evaluation implements the respective principle. Since these principles are often defined on an abstract level, they are refined by product factors describing directly observable properties of system entities, which in turn are quantified by measures. The details of the conformance module are described in Subsection 4.3.

Further design aspects contributing to software quality are subsumed in the *design* module, which contains product factors that cannot directly be deferred from principles (e. g. parameter granularity of operations). The design module is described in Subsection 4.4. The modular structure of the architecture quality model can be found in Figure 2. The *uses* relation between the *design* and the *conformance* module indicates that basic measures defined in the latter could be referenced from the former in order to avoid a duplication of modeling effort.

In addition to the modules described here, the modularization approach can be utilized to extend the quality model with domain or project-specific modules. Moreover, it is also possible to include modules for quality assessment on a different level of detail, e. g. source code, in order to aggregate all assessment results into a single hierarchy of quality goals.

4.2 Quality Goals

Classical quality attribute hierarchies have been criticized, because their decomposition appears to reflect different, implicit criteria and lacks clear semantics [JKC04]. To tackle this concern, activity-based quality models (ABQM) have been proposed in order to provide more natural

and meaningful decomposition semantics [DWP+07]. In my approach, I support this view and propose to describe quality goals using activities and attributes.

A set of relevant activities can be obtained from existing ABQMs as well as literature on the software life-cycle, e. g. IEEE 1074 [IEE06]. Definitions of traditional quality attributes like ISO 25010's quality characteristics often refer to activities and sometimes even mention corresponding attributes. The relation between these quality attributes and activity-based quality goals has been discussed in more detail by Lochmann and Goeb [LG11]. There, ISO 25010's quality characteristics ("-ilities") are explicitly part of the quality model, represented as high-level product factors that have an influence on activity-based quality goals, e. g. the property of a software product's UI not to use color as the only means of encoding important information has an influence on the factor [Product | ACCESSIBILITY], which in turn positively impacts the quality goal [Perceive | EFFECTIVENESS][2].

Quality goals can be refined using general techniques from the requirement engineering field (e. g. [LL00, vL01, DSG12]). In the quality model, this refinement can either be done along the attributes (e. g. [Use | QUALITY] is refined to [Use | EFFICIENCY], [Use | EFFECTIVENESS], and [Use | CONTINUITY]), or along the activities (e. g. [Use | EFFICIENCY] is refined to [Perceive | EFFICIENCY], [Understand | EFFICIENCY], and [Act | EFFICIENCY]). For *maintainability*, e. g., Deissenboeck et al. [DWP+07] provide a thorough decomposition of maintenance activities.

In the following, I use SOA as an example for an architectural style and derive some activity-based quality goals from typical scenarios. In order for a potential service consumer to decide whether a service offers the desired functionality and is therefore feasible for a given usage scenario, he first has to understand it. The respective quality goal is [Analysis | EFFICIENCY], since this analysis should be as efficient as possible. Similarly, other activities imply the goal of being conducted efficiently, in particular [Composition | EFFICIENCY], [Adaptation | EFFICIENCY], and [Test | EFFICIENCY], which are self-explanatory. The degree to which a service satisfies consumers' needs in terms of functionality and therefore enables effective service consumption can be expressed as [Consumption | EFFECTIVENESS]. Interaction between services is crucial for an SOA to be effective. Since service interoperation is achieved by composing services, this quality goal of effective interaction between services can be represented as [Composition | EFFECTIVENESS].

4.3 Conformance—Architectural Principles

The *conformance* module of an architecture quality model contains all essential principles that constitute a particular architectural style. As shown in Figure 3, the conformance module consists of two kinds of factors, namely *principle factors* and *conformance factors*. The former provide a general definition and explanation of an architectural principle and its impacts on quality goals, which can be either positive or negative. The latter refine these principle factors into properties that can directly be observed from the system artifacts. In order to quantify the degree of such a property's presence or absence in a software architecture, each conformance factor is assigned with one or several measures. An evaluation function assigned to each factor puts these measurement results into relation and maps them on a scale representing the factors' degree of fulfillment.

[2] Although the syntax has been adapted to be consistent with the model presented here, the semantics of the original paper have been preserved.

An example from the SOA domain would be the principle of [SERVICE COMPOSITION], expressing that services can be composed in order to create new services. This principle is refined into conformance factors: A factor [Service | OUTGOING DEPENDENCIES] could describe the fact that services that depend on other services do make use of composition and hence support the composition principle. A second conformance factor, [Service | CONSUMPTION RATIO], could describe to which degree services wihtin the system consume other services. [SERVICE COMPOSITION] itself has a positive impact on [Consumption | EFFECTIVENESS], because a system that makes use of service composition allows for fine-grained reuse of services and therefore facilitates effective service consumption. To quantify [Service | CONSUMPTION RATIO], e. g. the measure *Consumer Provider Ratio* is defined, which describes the ratio between provider and consumer services within the system. Provider services are services that are consumed by other services within the system, whereas consumer services consume other services. Of course, services can be both providers and consumers at the same time. This way, the rather intangible architectural principle of composition can be refined with the help of conformance factors into observable properties that are quantified by a set of defined measures. At the same time, the effects of adhering to this principle are captured in terms of impacts on quality goals.

4.4 Design—Architectural Best-Practices

Adhering to a certain architectural style is not sufficient to ensure good quality. Usually, architectural principles are accompanied by guidelines and best-practices. The *design* module contains factors and measures describing these additional properties that are not covered by the basic principles of an architectural style. While the general Quamoco approach does not restrict the type of product factors contained in a model, this module explicitly separates architectural best-practices from other kinds of factors in order to provide a more concise view on the overall product quality.

In contrast to conformance factors, design factors directly define impacts on quality goals. They can, however, be organized hierarchically in order to group similar low-level properties and make navigating the model easier. Typical topics to be covered in the design module are *dependencies*, *documentation and naming*, *granularity*, or *size and complexity*. Each of these topics can be addressed by several product factors, describing respective architectural properties. Concerning granularity, e. g., one of these factors could be [Operation | FUNCTIONAL GRANULARITY], which expresses the property that a service operation should perform exactly one function, e. g. searching a customer database for entries matching a provided search pattern. This factor is quantified using the measure *Operations performing multiple functions*, which provides guidance for system experts to assess service operations and report those, which perform more than one function.

These factors and measures typically resemble a collection of design guidelines and best-practices that are known to have an influence on certain quality goals. In order to obtain a comprehensive set of factors and measures, a thorough analysis is required, followed by a validation in order to be sure that all typical aspects of the particular architectural style are appropriately covered by the model. A validation method for quality models has been proposed and applied in the context of a quality model for embedded systems by Mayr et al. [MPK+12]. Architectural design factors and measures for SOA have been published by Goeb and Lochmann [GL11].

4.5 Instantiation and Usage

The meta quality model defined above can be instantiated to build a quality model for an architectural style by combining various sources of knowledge like personal experience or documented research studies. Usually these sources vary depending on the type of model elements. In order to create the set of principles in the conformance module, literature on that particular architectural style is probably most appropriate. The refinement into factors can be performed based on personal experience as well as existing models or frameworks. Likewise, there is a large amount of well-evaluated research studies on the impacts of particular design properties on different aspects of software quality. The advantage of a formally defined model compared to these textual representations is that the consolidated model can be visualized, navigated and analyzed more easily using appropriate tools. In addition, contradictions or missing information become more evident in a formal model. Literature might not provide a consistent view on how different measures should influence a factor's degree of fulfillment.

As part of a larger research effort I created a corresponding quality model for SOA, containing SOA principles as well as further design factors. This model has been created over the course of the recent years and will be published separately, including an expert-based evaluation of its overall structure as well as its contents. In total, the SOA quality model consists of 111 elements and therefore cannot be presented here in detail. An overview is, however, depicted in Figure 4.

For the weighing of model elements against each other in order to allow quality assessment using the model, I propose an iterative approach: First, initial evaluation functions should be manually defined for each factor based on personal experience. Once a quality model is completely defined and operationalized, a benchmarking technique should be employed to calibrate these functions. This is achieved by assessing a certain amount of software systems using the quality model and thus observing typical value ranges in real-world systems. More information on how to use benchmarking approaches for the calibration of software quality models can be found in [Loc12].

In order to perform a model-based architecture quality analysis, all measures defined in the model have to be provided with measurement values. Using the Quamoco tool chain, this can either be achieved by implementing according adapters for automatically obtainable values or by generating a spreadsheet template from the model, which can be filled with the respective values by an inspector. In a second phase, these measurement values are aggregated along the hierarchy of refinement and impact relations defined in the model, evaluating the formulas provided for each model element. Because this is a core functionality of the Quamoco approach, it is not elaborated here in more detail. An exemplary quality assessment can be found in Section 5.

4.6 Summary

The proposed meta quality model for software architecture evaluation is comprised of three modules. The *base* module contains definitions of quality goals and relations between them. Usually, these quality goals are structured hierarchically. To represent quality goals, I propose the activity-based notion, so that each quality goal is expressed as a pair of an activity and an attribute. Hierarchical refinement of quality goals can be done along both activities and attributes.

The *conformance* module contains information regarding the core principles of a certain archi-

tectural style. These principles are represented as *principle factors*. In order to provide means for architecture conformance assessment, these principles are refined by *conformance factors*, which resemble observable properties of the software architecture to reflect these principles. These factors are quantified by *measures*, which can either be obtained by measurement tools and metrics, or manually during architecture inspections. Besides architecture conformance assessment, the conformance module can provide valuable insight regarding the effect of an architectural style on software product quality. In order to achieve this, principle factors describe *impacts* on the quality goals defined in the base module.

The *design* module covers quality-relevant architectural properties originating from guidelines and best-practices, which are not directly related to the principles of an architectural style. Usually, conformance to architectural principles helps achieving high quality, but is not sufficient. An analysis of the quality model can easily identify quality goals that are not sufficiently covered by architectural principles and hence lead to the definition of further design guidelines in order to achieve this coverage. These guidelines are represented by *design factors*, which again resemble observable properties of a software architecture. Design factors also define *impacts* on quality goals, so that architecture quality assessment can be done on the combination of conformance and design factors. The resulting meta quality model for software architecture conformance and quality assessment is depicted in Figure 3.

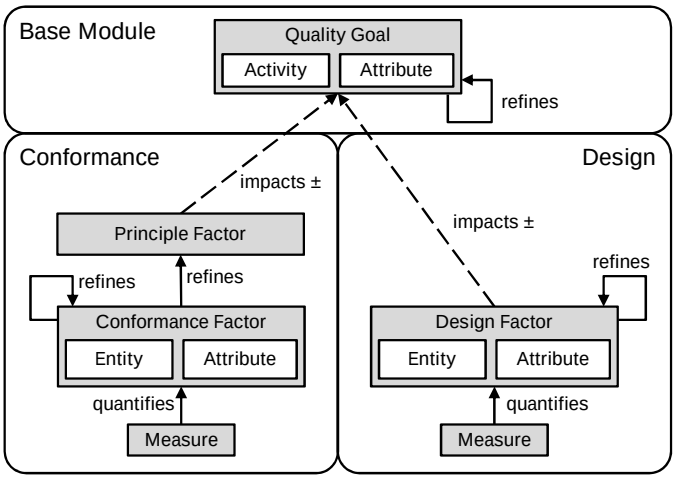

Figure 3: Software architecture meta quality model

In summary, the proposed structure allows the separation of quality-related effects of an architectural style from general best-practices that improve software quality. In addition, architectural principles become tangible by refining them to observable properties of architecture artifacts. In early project phases, this transparency ensures a common understanding throughout the project. Once according models are available for different architectural styles, they can also serve as a decision basis in order to decide for a particular one.

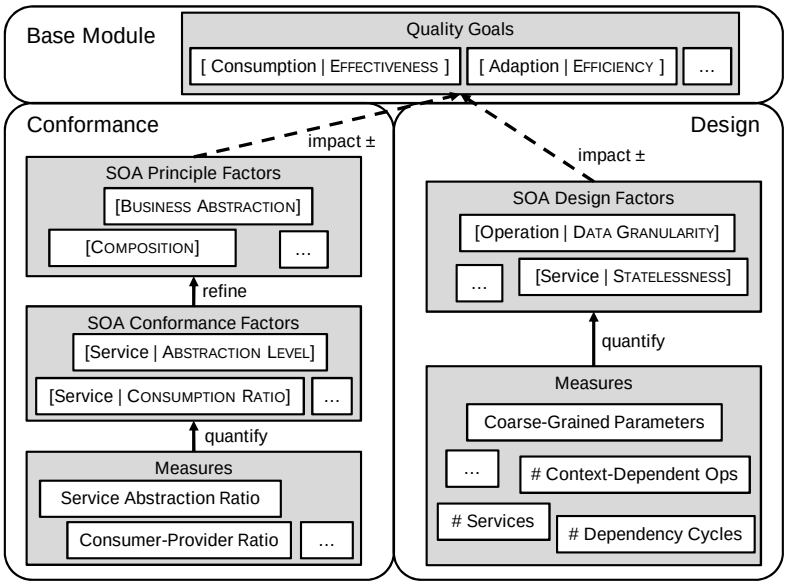

Figure 4: SOA conformance and quality model (excerpt)

5 Validation

In order to validate that an architecture quality model built in accordance with the proposed meta model can be used for architecture quality assessment in practice, I designed the following experiment: I used the architecture conformance and quality model for SOA described in Subsection 4.5 to assess an actual SOA-based software system, which was developed as part of the European SmartProducts research project[3]. This project was selected for the experiment because the number of services was small enough to manually trace the evaluation results from the values on the *quality goal* level down to values for individual measures. This was considered an essential requirement for a first applicability test. Figure 4 shows the SOA conformance and quality model's structure as well as *some* of the contained quality goals, factors and measures.

5.1 Goal

I conducted this experiment to validate the practical applicability of the presented approach for software architecture conformance and quality assessment. Practical applicability is measured by the success of conducting an evaluation based on the set of measures defined in the model. Furthermore, the experiment should demonstrate the compatibility of the underlying concepts with existing tools for quality modeling and assessment.

[3] http://www.smartproducts-project.eu/

5.2 Setup and Procedure

Due to the diversity of analyzed artifacts (specifications, models, source code), tool-based measurement would have required implementing respective measurement tools for each of these artifact types and has therefore been discarded. Instead, I adapted the quality model, converting all measures to be manually assessed. The Quamoco tool chain [DHH+11] requires manual measures to be numerical. Following this rule, it was possible to export an Excel template containing all measurement instruments along with their descriptions and a column to enter the manually obtained measurement results. From the SmartProducts project team, five researchers familiar with the system's architecture used this template and manually conducted all the measurements contained therein. All of the experts hold a university degree in computer science, two of them also hold a Ph. D. In addition, they all have several years of experience in developing software. Because the only tool the experts used was the Excel template, they did not have to be introduced to the whole quality model in advance, but could focus on collecting measurement data without being biased by knowing how the evaluation would be impacted by either of the measurement values.

Table 1 shows the measure names as well as the assessed measurement values. The values represent the mean values of the results obtained by the five experts. Since the validation goal was to show the applicability of the approach rather than the actual quality assessment result values, no special emphasis was put on analyzing individual data points with regard to error margins. The filled template was imported into the assessment tool chain as the basis for further processing according to the SOA model.

Due to the capabilities of the Quamoco tooling in handling manually obtained measurement results, the remaining steps of the quality assessment did not involve any manual effort. The measurement data was processed and transformed into an HTML dashboard presenting scores for each quality goal contained in the quality model based on the evaluation formulas defined there. These results were then presented to the project team, followed by a discussion on their opinion regarding the approach of model-based, semi-automatic architecture quality assessment in general, and the applicability of the SOA quality model in particular.

5.3 Results

In general, the experts highly appreciated the effort savings due to the automation provided by the quality model and the associated tools. Compared to other architecture assessment techniques that had been used in the project before (e. g. *Active Design Reviews* [PW85]), they reported that especially preparation and data analysis were far less time-consuming, so that they were able to assess the system within one working day instead of several days for preparing review questionnaires and analyzing the results. At the same time the experts reported that the actual measurement required a profound knowledge of the system and could not easily be conducted by external people. This appears reasonable, taking into account that many of the measures were initially meant to be conducted automatically using appropriate tools, but had been adapted to be conducted manually due to the different development stages of the involved artifacts.

Obviously the time savings for the analysis of a single project's architecture do not compensate for the initial effort of building the SOA model. However, since the model is available and

Table 1: Raw measurement results

Measure Name	Value
Contract Implementation Coupling	0.00
Inconsistent Interface Standards	0.50
Number of Utility Services	5.00
Average Service Fan-In	2.00
Average Service Fan-Out	2.00
Number of Domain Services	3.00
Number of Process Services	4.00
Total Number of Services	12.00
Consistent and Unambiguous Documentation	50.00
Number of Dependency Cycles	0.00
Average Dependency Depth	2.00
Efficiently Groupable Parameters	6.00
Inconcisely Named Operations	28.00
Inconcisely Named Parameters	35.00
Inconcisely Named Services	5.00
Number of Context-Dependent Operations	5.00
Number of Exposed Operations	0.00
Total Number of Operations	55.00
Total Number of Parameters	100.00
Operations with External Dependencies	10.00
Operations Performing Multiple Functions	10.00
Average Dependencies per Service	2.00
Semantically Equivalent Services	0.00
Interface Data Cohesion	0.40
Interface Sequential Cohesion	0.15
Interface Usage Cohesion	0.50
Functional Domain Coverage	11.00

reusable, other projects can make use of it and (optionally) invest some time in project-specific tailoring.

The analysis tool provides the user with different visualizations (see e. g. [WLH+12]) and a hierarchical HTML-based table. Figure 5 shows an excerpt from this table. The first column contains the hierarchical decomposition of quality goals, including the impacting factors and the respective measures. The right column shows the analysis results. In the case of measures, these results correspond to the numbers read from the template file. For factors and quality goals, the value range is the interval $[0;1]$, which represents the respective degree of fulfillment.

Concerning the interpretation of these values it is important to mention that they are not meant to be absolute statements. Experience with model-based quality assessment has shown that quality models only produce objective results if they are properly calibrated using a large number of reference products. The experiment presented here only aimed at showing the general applicability of the approach for automatic quality assessment and the compatibility of the adapted meta quality model with the Quamoco tool chain. The calibration procedure for the Quamoco base quality model for source code is outlined in [WLH+12] and can be applied for architecture conformance and quality models as well.

Despite this fact, the results for some of the factors can already provide hints regarding architecture quality. One interesting finding was that the system scored comparably low on the [BUSINESS ABSTRACTION] SOA principle. Discussions revealed that the particular system was indeed not completely following the SOA principle of providing mainly business process-relevant functionality as services, but also offered "low-level APIs". Likewise, other assessment results on a general level matched the overall perception of the involved experts. Hence, model-based architecture conformance and quality analysis can help software architects to increase transparency regarding the conformance to architectural principles as well as potential quality problems with comparably low effort, even before the software product is completely implemented.

Because the main goal of this experiment was to show the approach's applicability for quality assessment, a more detailed interpretation of assessment results was not performed. Such an analysis would have required a calibration of the model's evaluation formulas in order to provide sound results. A more detailed discussion on this topic can be found in the following section.

6 Conclusions and Outlook

In this paper, I presented a variant of the Quamoco quality modeling approach specifically addressing software architecture conformance and quality. While relying on Quamoco's meta quality model, several conventions have been applied in order to describe both the inherent properties of an architectural style and additional properties the architecture should possess in order to achieve high software quality. These two flavors of architecture quality are represented in the model's *conformance* and *design* modules, respectively. The model relies on the activity-based quality modeling approach, which means that quality goals are expressed via activities conducted with the system and attributes of these activities. This way the multifaceted concept of quality is structured along intuitive decomposition semantics by splitting activities into sub-activities.

The building blocks of a software architecture are represented by entities, which are characterized by attributes in order to describe factors that can be observed in a software architecture.

Quality Assessment (Main)

Quality Assessment

Element	Evaluation Result
Property @Product []	
Quality @UseCase []	0,613
Quality @Adaptation [refined]	0,652
Quality @Operation [refined]	0,512
Quality @Ext. Analysis [refined]	0,687
Quality @Int. Analysis [refined]	0,499
Efficiency @Int. Analysis [refined]	0,499
Cohesion @ServiceInterface [impacted]	0,417
Complexity @ServiceInterface [impacted]	0,841
Design Size @SOA System [impacted]	0,240
Interface Complexity @SOA System [impacted]	0,000
Meaningful Names @ServiceInterface [impacted]	0,573
Proper Documentation @ServiceInterface [impacted]	1,000
Quality @Composition [refined]	0,705
Quality @Consumption [refined]	0,498
Effectiveness @Consumption [refined]	0,490
Efficiency @Consumption [refined]	0,523
Low Fragmentation @SOA System [impacted]	0,091
Short Dependency Paths @Service [impacted]	0,956
DDT	threshold=[0;3] value=0,944
DDT [measures]	2,000
SDT	threshold=[0;5] value=0,967
Continuity @Consumption [refined]	0,479
Quality @Test [refined]	0,736

Figure 5: Assessment output view

These factors have impacts on quality goals, making them the intermediary between general quality goals and actual measurements and metrics.

An experiment applying an architecture quality model for SOA to an actual software system showed that architecture quality models built according to our proposed structure can be used for architecture evaluation. Five experts out of the team who built the system that was used for evaluation stated that the greatest benefit of the approach is the strong reduction of manual analysis and processing effort compared to other architecture evaluation approaches applied in the project before. Given a limited overall time frame, this allows for architecture assessment more often, leading to increased transparency regarding architectural properties and hence to more responsive quality control. In addition, the most significant assessment results matched the perception of the project team. In order to show the value of the SOA quality model beyond these tendencies, further empirical studies are needed.

The architecture meta quality model provides a framework to consistently collect and conserve quality knowledge. Architecture quality models can be used to investigate for a given quality goal, how it is affected by the principles of an architectural style, and which additional design patterns should be implemented in order to reach this quality goal. This helps software architects to argue to which degree a decision for a certain architectural style already has positive impacts on software quality, and whether these might be affected by ignoring further design properties.

Moreover, architecture quality models build the basis for standardized and reproducible architecture quality assessment by formalizing relationships between measures, factors and quality goals. Back in 1992, Grady described his vision regarding the importance of metrics in software engineering in the year 2000 as follows: “First, tools will automatically measure size and complexity for all the work products that engineers develop. Besides warnings and error messages, the tools will predict *potential* problem areas based on metric data thresholds” [Gra92, p. 220]. For source code, this goal might have actually been reached, taking the large amount of dashboards and related tools into account. With respect to software architecture, however, even twelve years after the mentioned date, this vision has not become reality yet. The proposed architecture conformance and quality model might be a step into this direction, since it provides clear relationships between important concepts and allows for automated assessment. Hence, it contributes to the reproducibility of quality analysis, making sure that incremental assessments after changes in the software are based on the same evaluation rules. This also lowers the required effort, since aggregation, processing and visualization of results are automatically conducted using quality analysis tools like ConQAT. Further automation potential is given by the possibility to also use tools to obtain the measurement values.

Further areas of future work include applying the modeling approach to a number of architectural styles. Doing so, the resulting quality models can help to compare architectural styles based on their impact on quality and provide decision support for software architects. For a particular software development project, they are provided with a means to choose the architectural style that most adequately covers the quality requirements defined for that project. In the long term, a framework could emerge to describe architectural styles from a quality perspective, providing a solid basis for decisions in early software development phases.

Acknowledgements: Many thanks go to the SmartProducts project team at TU Darmstadt for taking part in the applicability experiment.

Bibliography

[BBK+78] B. W. Boehm, J. R. Brown, H. Kaspar, M. Lipow, G. J. Macleod, M. J. Merrit. *Characteristics of Software Quality*. North-Holland, 1978.

[BDP06] M. Broy, F. Deissenboeck, M. Pizka. Demystifying Maintainability. In *Proc. 4th Workshop on Software Quality (WoSQ)*. Pp. 21–26. 2006.
doi:10.1145/1137702.1137708

[BWDV00] L. Briand, J. Wüst, J. Daly, D. Victor Porter. Exploring the Relationships Between Design Measures and Software Quality in Object-oriented Systems. *Journal of Systems and Software* 51(3):245–273, 2000.
doi:10.1016/S0164-1212(99)00102-8

[CKK01] P. Clements, R. Kazman, M. Klein. *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley Professional, 2001.

[CNY95] L. Chung, B. A. Nixon, E. Yu. Using Non-Functional Requirements to Systematically Select Among Alternatives in Architectural Design. In *1st Int. Workshop on Architectures for Software Systems*. Pp. 31–43. 1995.

[DHH+11] F. Deissenboeck, L. Heinemann, M. Herrmannsdoerfer, K. Lochmann, S. Wagner. The Quamoco Tool Chain for Quality Modeling and Assessment. In *Proc. 33rd Int. Conf. on Software engineering (ICSE)*. Pp. 1007–1009. 2011.
doi:10.1145/1985793.1985977

[DSG12] S. Doeweling, B. Schmidt, A. Goeb. A Model for the Design of Interactive Systems based on Activity Theory. In *Proc. ACM Conf. on Computer Supported Cooperative Work (CSCW)*. Pp. 539–548. 2012.
doi:10.1145/2145204.2145287

[DWP+07] F. Deissenboeck, S. Wagner, M. Pizka, S. Teuchert, J.-F. Girard. An Activity-Based Quality Model for Maintainability. In *Proc. IEEE Int. Conf. on Software Maintenance (ICSM)*. Pp. 184–193. 2007.
doi:10.1109/ICSM.2007.4362631

[GL11] A. Goeb, K. Lochmann. A Software Quality Model for SOA. In *Proc. 8th Int. Workshop on Software Quality (WoSQ)*. Pp. 18–25. 2011.
doi:10.1145/2024587.2024593

[Gra92] R. B. Grady. *Practical Software Metrics for Project Management and Process Improvement*. Prentice Hall, 1992.

[IEE06] IEEE. Std 1074-2006 – IEEE Standard for Developing a Software Project Life Cycle Process. 2006.
doi:10.1109/IEEESTD.2006.219190

[JKC04] H.-W. Jung, S.-G. Kim, C.-S. Chung. Measuring Software Product Quality: A Survey of ISO/IEC 9126. *IEEE Software* 21(5):88–92, 2004.
doi:10.1109/MS.2004.1331309

[vL01] A. van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. In *Proc. Int. Symposium on Requirements Engineering*. 2001.
doi:10.1109/ISRE.2001.948567

[LCLR03] F. Losavio, L. Chirinos, N. Lévy, A. Ramdane-Cherif. Quality Characteristics for Software Architecture. *The Journal of Object Technology* 2(2):133–150, 2003.
doi:10.5381/jot.2003.2.2.a2

[LCM+04] F. Losavio, L. Chirinos, A. Matteo, N. Levy, A. Ramdane-Cherif. ISO Quality Standards for Measuring Architectures. *Journal of systems and software* 72(2):209–223, 2004.
doi:10.1016/S0164-1212(03)00114-6

[LG11] K. Lochmann, A. Goeb. A Unifying Model for Software Quality. In *Proc. 8th Int. Workshop on Software Quality (WoSQ)*. Pp. 3–10. 2011.
doi:10.1145/2024587.2024591

[LL00] A. van Lamsweerde, E. Letier. Handling Obstacles in Goal-Oriented Requirements Engineering. *IEEE Transactions on Software Engineering* 26(10):978–1005, 2000.
doi:10.1109/32.879820

[Loc12] K. Lochmann. A Benchmarking-inspired Approach to Determine Threshold Values for Metrics. *SIGSOFT Softw. Eng. Notes* 37(6):1–8, Nov. 2012.
doi:10.1145/2382756.2382782

[MBD+09] K. Mordal-Manet, F. Balmas, S. Denier, S. Ducasse, H. Wertz, J. Laval, F. Bellingard, P. Vaillergues. The Squale Model – A Practice-based Industrial Quality Model. In *Proc. IEEE Int. Conf. on Software Maintenance (ICSM)*. Pp. 531–534. 2009.
doi:10.1109/ICSM.2009.5306381

[MCN92] J. Mylopoulos, L. Chung, B. Nixon. Representing and Using Nonfunctional Requirements: A Process-Oriented Approach. *IEEE Transactions on Software Engineering* 18(6):483–497, 1992.
doi:10.1109/32.142871

[MPK+12] A. Mayr, R. Plösch, M. Kläs, C. Lampasona, M. Saft. A Comprehensive Code-based Quality Model for Embedded Systems. In *Proc. 23rd Int. Symposium on Software Reliability Engineering (ISSRE)*. 2012.
doi:10.1109/ISSRE.2012.4

[MRW77] J. A. McCall, P. K. Richards, G. F. Walters. Factors in Software Quality. Technical report RADC-TR-77-369, Rome Air Development Center, 1977.

[PW85] D. L. Parnas, D. M. Weiss. Active Design Reviews: Principles and Practices. In *Proc. 8th Int. Conf. on Software Engineering (ICSE)*. Pp. 132–136. 1985.

[RV98] A. Rivers, M. Vouk. Resource-Constrained Non-Operational Testing of Software. In *Proc. 9th Software Int. Symposium on Reliability Engineering*. Pp. 154–163. 1998. doi:10.1109/ISSRE.1998.730874

[SW05] M. Svahnberg, C. Wohlin. An Investigation of a Method for Identifying a Software Architecture Candidate with Respect to Quality Attributes. *Empirical Software Engineering* 10(2):149–181, 2005. doi:10.1007/s10664-004-6190-y

[VAC+09] O. Vogel, I. Arnold, A. Chughtai, E. Ihler, T. Kehrer, U. Mehlig, U. Zdun. *Software-Architektur: Grundlagen – Konzepte – Praxis*. Spektrum Akademischer Verlag Heidelberg, 2nd edition, 2009.

[WLH+12] S. Wagner, K. Lochmann, L. Heinemann, M. Klaes, A. Seidl, A. Goeb, J. Streit, A. Trendowicz, R. Ploesch. The Quamoco Product Quality Modelling and Assessment Approach. In *Proc. 34th Int. Conf. on Software Engineering (ICSE)*. 2012. doi:10.1109/ICSE.2012.6227106

[WLW+12a] S. Wagner, K. Lochmann, S. Winter, F. Deissenboeck, E. Juergens, M. Herrmannsdoerfer, L. Heinemann, M. Klaes, A. Trendowicz, J. Heidrich, R. Ploesch, A. Goeb, C. Koerner, K. Schoder, C. Schubert. The Quamoco Quality Meta-Model. Technical report TUM-I128, Technische Universität München, 2012.

[WLW+12b] S. Wagner, K. Lochmann, S. Winter, A. Goeb, M. Klaes, S. Nunnenmacher. Software Quality Models in Practice. Technical report TUM-I129, Technische Universität München, 2012.

[Zha99] J. Zhao. Bibliography of Software Architecture Analysis. *Software Engineering Notes* 24(4):61–62, 1999.