Towards Quantitative Metrics for Architecture Models

Stephan Sehestedt
ABB Corporate Research Germany
Industrial Software Systems program
68526 Ladenburg, Germany
Tel.: +49 6203 716261
stephan.sehestedt@de.abb.com

Chih-Hong Cheng
ABB Corporate Research Germany
Industrial Software Systems program
68526 Ladenburg, Germany
Tel.: +49 6203 716424
chih-hong.cheng@de.abb.com

Eric Bouwers
Software Improvement Group
Amstelplein 1
1096 HA Amsterdam
Tel.: +31 20 314 09 50
e.bouwers@sig.eu

ABSTRACT
Software architectures and their representations in models are instrumental in achieving sustainability and the fulfillment of requirements. In this context, sustainability encompasses cost efficient maintainability and evolvability, which are central concerns for long living software systems. Hence, it is of great importance to support an architect in addressing these concerns when designing and evolving architectures. However, there is no framework available in which a designed architecture can be evaluated against these important quality attributes. In this paper, we address this challenge by proposing seven metrics which characterize the completeness, consistency, correctness and clarity of the documentation. These seven metrics should enable an architect to efficiently identify issues in an architecture model.

General Terms
Architect, Metrics, Software Architecture, Documentation, Performance, Design

Keywords
Sustainability, Evolvability, Quality attributes, Completeness, Consistency, Correctness, Clarity

1. INTRODUCTION
The quality of a software system is largely attributed to its software architecture [1]. Thus, evaluation of this software architecture should be done on a regular basis. Such repeated evaluations ensure that the system remains sustainable and evolvable over a longer period of time.

Software architecture models (i.e., the designed architecture) help to achieve sustainability of software through suitable documentation. Artifacts of the modeled architecture can be linked to requirements, architectural decisions, and alternatives. Overall, we believe that the quality of a designed architecture can be observed by the evaluating the following criteria, referenced to as the "4-C criteria":

1) Completeness of the architecture model
2) Consistency of the architecture model
3) Correctness of the architecture model
4) Clarity of the architecture model

Conference'10, Month 1–2, 2010, City, State, Country.

Completeness refers to the coverage of quality aspects and requirements in the architecture model. *Consistency* is concerned with conflicting choices of, e.g., technologies or inconsistencies between architecture model and decisions. *Correctness* captures whether every decision follows from a requirement. Finally, *clarity* deals with the documentation being suitable for the target audience, e.g., can a developer successfully implement the system based on the architecture model.

To the best of our knowledge, there is currently no framework available in which the 4-C criteria of a designed architecture can be evaluated in a consistent and repeatable manner. Based on our experience, this leads to architecture models that are inconsistent and hard to implement, which results in major rework later on in software projects.

In this paper, we address this omission of system independent, metrics to evaluate the quality of an architecture model with respect to the 4-C criteria. To do this, we present a set of metrics that help an architect to identify issues in an architecture model. We envision that an architect can use these metrics as a validation mechanism before the architecture models are further discussed amongst the stakeholders.

Note that we do not propose to quantify the quality of a software architecture directly. The proposed metrics rather support an architect in identifying issues within the architecture models (and its documentation) based on the information in the model, and possibly data from a benchmark database.

The challenge in designing such metrics is that architecture models are generally not formal models and that expert knowledge is required for computing the metrics. Thus, a balance needs to be found between effort for the evaluation and the quality of the output.

In the remainder of this paper we outline the metrics and their rationale. A complete validation of the metrics using a set of benchmark architectures is left as future work.

2. RELATED WORK
The importance of software architectures is reflected in the efforts to develop tools and methods for evaluation. Such evaluation can be exercised on designed and implemented architectures with low or high efforts, e.g., several days and experts in ATAM. Below we briefly discuss the most closely related works with a focus on designed architectures.

A comprehensive survey covering scenario and metrics based architecture evaluation methods with a focus on sustainability is given by Koziolek [2]. ATAM is a prominent example of a scenario based evaluation method in which trade-offs in an architecture can be identified. The effort required is high,

especially as the outcome depends on the experience of the participants. Metrics based methods on the other hand are lower in effort. Most regarded approaches are targeting implemented architectures while our focus is on the designed architecture. Babar et.al. [3] surveyed architecture evaluation methods and organized them in a framework to support selecting the best method for a particular purpose. One driving factor in differentiating the regarded methods is which quality attributes they cover, noting that it is necessary to cover several quality attributes as they may interact.

Sethi et al. [4] deal with evaluating the modularity of an architecture design. A UML component diagram is transformed into an augmented constraint network and design structure matrix. Based on this design volatility metrics, a concern scope metric, concern overlap metric and a change impact metric are proposed. The assumption in this work is that formalized component diagrams exist and that environmental conditions and design rules are modeled. In most cases this will require significant refinement of models trading high effort with precision. Sant'Anna et al [5] proposed a framework in which architectural concerns are used to determine whether a designed architecture is well modularized. This is done by checking if concerns are addressed in one or many components, interfaces, and architectural operations. However, the metrics are simply based on counting, e.g., how many components address the same concern, and no definition of goodness is given. Hence, the proposed metrics do not result in actionable items and require additional analysis.

Nakamura et al. [6] proposes to represent the software structure as a graph that represents the architecture. When a change to the architecture occurs, its magnitude can be determined by the distance between graphs. It is proposed to use this as an indicator of the cost of changes. However, for a designed architecture it is not always appropriate to spend efforts in modeling changes explicitly. Instead, it may be useful to have an expert's assessment together with some automatic computations. Paakki et al. [7] mine architecture documentation for patterns, e.g., architectural patterns, to predict the quality of the architecture and the resulting system. Patterns are identified using a set of known patterns to match against. The method relies on the quality of the pattern mining algorithm and a library of patterns. It is unclear, though, what it means when a pattern is found or what actions can be derived from the findings.

Tang and Han [8] propose a methodology to analyze an architecture during its design phase to assess risks, costs and benefits of an architecture. To do this, detailed data is required about, amongst others the development costs and potential legal costs arising from a design. Thus, this method requires significant effort in acquiring and updating the relevant data during an architecture's design. In contrast, we envision a more lightweight approach as suggested by Bouwers et al. [9] for implemented architectures.

Our approach is different to past work in our focus on metrics that do not directly assess the quality or fitness of an architecture. Instead we focus on evaluating an architecture's documentation with the goal to support an architect in evaluating the architecture on a regular basis or in, e.g., preparing ATAM sessions. Reasons for this differing approach are that architecture models and related documentation are not usually in formal models as for example required by Sethi et al. [4]. Hence, not all information is readily usable or available for computing related metrics.

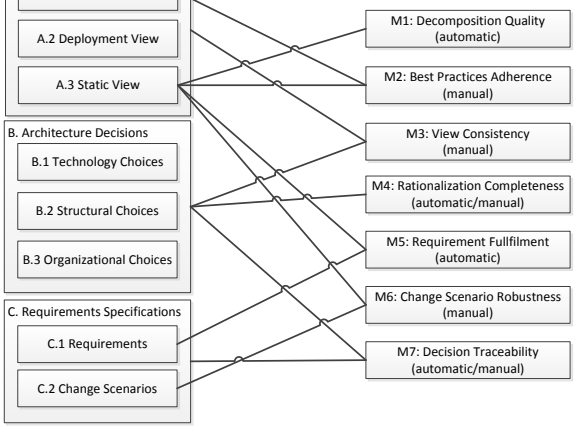

Figure 1: The proposed Metrics and their required inputs.

3. ARCHITECTURE MODEL METRICS

The proposed metrics address quality attributes within views of architecture models and architectural decisions, while considering their relations to requirements and changes.

Figure 1 lists those metrics and illustrates the type of required inputs. As can be seen, the computed metrics may depend on a category (e.g., M4 uses category B) or on individual item within a category (e.g., metric M6 uses A.3 and C.2). Metrics M3, M5, and M6 link the architecture model to decisions and specifications. M1 and M2 are concerned with the quality of the architecture model in itself regardless of the specifications and decisions. Lastly, M4 deals with the architecture decisions and M7 handles the interaction between decisions and requirements.

In the following subsections, we discuss the types of inputs and methods how metrics are computed.

3.1 Input

As the input of the proposed metrics, three types of data are considered (see left side of Fig. 1):

- (TYPE A - Architecture model) Includes one or more views of the designed system, such as module structures (static; A.1), component-and-connectors (dynamic; A.2), and HW-SW allocation structures (deployment; A.3).
- (TYPE B- Architectural decisions) An architecture model is the result of multiple design decisions, including structural choices (B.1), organizational choices (B.2), or technology choices (B.3).
- (TYPE C - Requirement specifications) This includes requirements (C.1) and potential change scenarios (C.2).

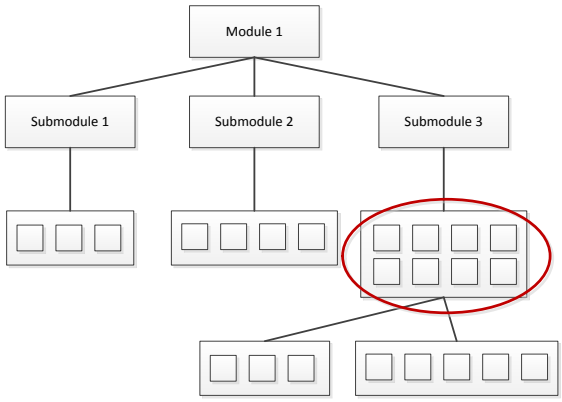

Figure 2: The decomposition of submodule 3 is probably not sufficient on Level 3 (red ellipse). Thus further decomposition is recommended.

3.2 Metrics

[M1] Decomposition Quality (from A.3): The decomposition quality metric gives an indication of whether or not modules are decomposed into submodules such that at the lowest level a similar degree of decomposition is achieved across all modules. This can be seen as an inequality measure, where the situation of having many modules in one submodule and fewer in others is not desirable. This situation is illustrated in Figure 2, where the decomposition of submodule 3 (red circle in the figure) results in more submodules than for the others on level 3 of the hierarchy. Smaller differences are likely no problem; larger inequalities should get the attention of the architect. Similar to the component size uniformity [10], this could be computed using the Gini coefficient:

$$M1 = 1 - Gini(submodules)$$

The result is a value in the range of [0,1], where 1 means all submodules being decomposed to a similar degree on an architectural level. Naturally, without inspecting the implementation of the software architecture the sizes of the modules and submodules cannot be taken into account. However, when the architecture is implemented the metric may evolve into a measure for component balance as proposed in [10].

[M2] Best Practices Adherence (from A.1 and A.3): Adherence to best practices is concerned with the architect using, e.g., patterns or tactics appropriately in the design of the software architecture. The challenge is that one can mine for patterns, but there is currently no objective way to determine whether the patterns are applied according to best practices. Hence, we propose a semi-automatic approach. Inputs for this metric are the number of concerns and the number of concerns which have been appropriately addressed. The appropriateness, however, needs to be determined by an expert. Thus, the metric can be calculated as:

M2 = #concerns appropriately addressed/#concerns

Where higher values are considered to be better. In addition, measurements related to Pattern Conformity as discussed by Lilienthal [11] can be used to get even more insight into this metric.

[M3] View Consistency (from A and B): Architectural models may comprise different views (e.g., static/dynamic/deployment views) and include technological or organizational choices. It is possible that all these views, when fused together or viewed individually, create inconsistencies or contradictions that make an implementation potentially impossible. To derive a metric whose value is between 0 and 1, our strategy is to first normalize the number of inconsistencies by a factor considering the number of components, requirements and decisions. Then the normalized value is compared with a pool of collected architectures (benchmarks) to derive its "goodness" in terms of percentage. In summary, M3 is computed by the following metric:

$M3 =$ *The relative performance for the architecture over the set of all benchmarked architectures, in terms of the following computed value:* $normalized(\#\ of\ contradictions)$.

Where higher values are considered to be better. Note that apart from this benchmarked value, this measurement can also be viewed as a violation. This would mean that no contradiction is allowed within the entire design. We envision that the benchmarked values are not only used in earlier stages of the design to track progress but also in final stages where the number of contradictions must be zero.

[M4] Rationalization Completeness (from B): For rationalization completeness we propose to quantify rationales and alternatives being associated with architecture design decisions (ADD). This alone is not sufficient as the appropriateness of rationales and alternatives need to be assessed, too. Hence, the metric consists of the number of ADDs without rationale/alternative determined by expert judgment in:

M4 = #ADDs with rationale or alternative/#ADDs

Where higher values are considered to be better.

[M5] Requirement Fulfillment (from A.3 and C.1): Requirements fulfillment quality can be determined by inspecting the associations between requirements and artifacts in the model. Intuitively, all requirements should be reflected in architectural artifacts and given full traceability the metric can be defined as:

M5 = #requirements associated with artifacts in the model / #requirements in RS

Where higher values are considered to be better.

[M6] Change Scenario Robustness (from A.3 and C.2): Change scenario robustness tries to capture how well suited an architecture is for defined change scenarios (CS). This can be calculated by estimating the impact of a change by the number of modules affected. For this the change scenarios should be weighted, e.g., by the probability of occurrence P(CS), and the weights should be normalized. The metric could thus be:

$M6 = 1 - \sum_{cs \in CS}(n * P(cs) * \#modules\ affected)/\#modules$

Where higher values are considered to be better, with *n* being a normalization factor $^{1}/_{\sum_{cs \in CS} P(cs)}$. The normalization factor ensures that that all *P(cs)* sum up to *1*.

[M7] Decision Traceability (from B and C): Decision traceability aims to characterize to what extent architecture design decisions (ADD) are properly linked to requirements. In theory, each ADD should stem from a requirement to ensure that the designed architecture does not impose unnecessary restrictions. In addition, inspecting the decisions that do not have a requirement might uncover previously undocumented requirements.

Similar to M3, we propose to quantify this traceability as:

M7: = #ADDs with requirement / #ADD

Where higher values are considered to be better. This metric is similar to M4. The difference is that M4 focuses on the rationale of the ADDs (i.e. is the decision valid), while the focus of this metric is on the requirements that underlie an ADD (i.e. the cause of an ADD).

As with M3, we envision this metric to be used as a progress metric in the early stage of the design process, while in the later stages it can be considered a violation if the value is larger than 0.

3.3 Discussion

Relating Metrics and Quality Attributes: Here we summarize how above metrics are related to the 4-C criteria:

- Completeness is covered by metric M4 and M6.
- Correctness is covered by metric M5.
- Consistency is covered by metric M3.
- Clearness is covered by metrics M1 M2, and M7.

M6 is used as an additional metric that offers an estimate on the temporal behavior of the architecture, where M1 to M5 and M7 are targeting a snapshot on the currently documented one.

Independence: Within the set of proposed metrics, only the computation of M3 requires a comparison to a benchmark suite. Still, the design intention of M3 targets platform independency due to the use of normalization factors. For other metrics, all of them can be derived by analyzing the investigated architecture. Among the proposed metrics, we envision that 3 out of 7 metrics (M2, M3, M5) need to be computed manually with expert support.

Suggested Usage: We believe that these metrics are relatively easy compute compared to existing architectural evaluation methods. Therefore, the proposed architecture metrics should be applicable in supporting architecture reviews (e.g., weekly reviews or ATAM). These metrics are easily understandable and provide instant diagnosis concerning the health of the architecture.

Succinctness: By using a limited set of metrics we ensure that the architect is not overwhelmed by all statistics. Still, the proposed list of metrics is by no means exhaustive. It is rather a starting point to find more metrics or new definitions of metrics which allow for, e.g., more automation.

Executability: It has been argued that metrics should lead to actions in order to ensure that the metrics are seen as useful. Our straight-forward metrics and the definition of desirable values for those ensure that architects can plan actions to improve the designed software architecture.

4. Conclusions

In paper we introduced a list of four criteria, referred to as the 4-C criteria - completeness, consistency, correctness, and clarity of the architecture model. We addressed the issue of lacking a framework to evaluate a designed architecture with respect to these criteria. To this end, seven metrics were proposed to cover different aspects of the architecture model in order to support an architect to produce high quality documentation for a given architecture. Based on this we expect that more metrics will be presented in the future.

The proposed metrics cover all of the 4-C criteria to different degrees. Therefore, we plan to increase this coverage by providing additional metrics. Our future work also encompasses the implementation and empirical testing of the proposed metrics. Thereby, ideally an implementation would integrate these metrics into existing tools to minimize overhead. Lastly, we plan to introduce more automation by means of, e.g., machine learning, pattern matching, or logic-based reasoning, in order to reduce the efforts an expert would have to invest.

5. REFERENCES

[1] Clements, P., Kazman, R., and Klein, M. Evaluating Software Architectures: Methods and Case Studies. Addison-Wesley Professional. 2002.

[2] Koziolek, H. 2011. Sustainability evaluation of software architectures: A systematic review. Proceedings of the Joint ACM SIGSOFT Conference - QoSA and ACM SIGSOFT Symposium on Quality of Software Architectures and Architecting Critical Systems (Boulder, Colorado, USA). QoSA-ISARCS '11. DOI= http://doi.acm.org/10.1145/2000259.2000263

[3] Babar, M.A., Zhu, L., Jeffery, R. A Framework for Classifying and Comparing Software Architecture Evaluation Methods. *Proceedings of the IEEE Australian Software Engineering Conference.* ASWEC'00. DOI= 10.1109/ASWEC.2004.1290484

[4] Sethi, K. Cai,Y., Huynh, S., Garcia, R. Assessing Design Modularity and Stability using Analytical Decision Models. Technical Report DU-CS-08-03. Drexel University, 2008. USA.

[5] Sant'Anna, C., Figueiredo, E., Garcia, A., Lucena1, C. 2007. On the Modularity Assessment of Software Architectures: Do my architectural concerns count? *Proceedings of the International Workshop on Aspects in Architecture Descriptions* (Vancouver, Canada. 12 March 2007). AARCH'07.

[6] Nakamura, T., Basili, V. 2005. Metrics of Software Architecture Changes Based on Structural Distance. *Proceedings of the International Symposium on Software Metris* (Washington, DC, USA, 2005). METRICS'05. DOI= http://dx.doi.org/10.1109/METRICS.2005.35

[7] Paakki, J., Karhinen, A., Gustafsson, J., Nenonen, L., Verkamo, A. 2000. Software Metrics by Architectural Pattern Mining. *Proceedings of the IFIP International Conference on Software: Theory and Practice* (Beijing, China, 2000).

[8] Tang, A., Han.J., Architecture rationalization: a methodology for architecture verifiability, traceability and completeness. *Proceedings of the IEEE International Conference and Workshops on the Engineering of Computer-Based Systems*, pp. 135-144, 2005.

[9] Bouwers, E., van Deursen. A., A Lightweight Sanity Check for Implemented Architectures. *IEEE Software*, Volume 27, Number 4, pp. 44-50, ISSN 0740-7459, 2010.

[10] Bouwers, E., Visser, J., and van Deursen.A., 2011. Quantifying the Analyzability of Software Architectures. *Proceedings of the Working IEEE/IFIP Conference on Software Architecture* (Washington, DC, USA, 2011). WICSA'11. DOI= http://dx.doi.org/10.1109/WICSA.2011.20

[11] Lilienthal, C., "Architectural Complexity of Large-Scale Software Systems," *Proceedings of the European Conference on Software Maintenance and Reengineering* (2009). CSMR '09. vol., no., pp.17,26, 24-27 March 2009.