A portfolio of internal quality metrics for software architects

Miroslaw Staron[1] and Wilhelm Meding[2]

[1] Computer Science and Engineering, University of Gothenburg, Sweden
miroslaw.staron@gu.se,
[2] Ericsson AB, Sweden
wilhelm.meding@ericsson.com

Abstract. Evolving the architecture of the software together with the evolution of the design is one of the key areas in maintaining the high quality. In this paper we present a portfolio of indicators addressing a set of three areas of information needs for large software development companies of embedded software. The portfolio is a result of our studies of literature and at Software Center (nine companies and five universities) with the goal to identify the main information needs and quality metrics for the role of software architects. As a result of our studies we could elicit such information needs as architecture measures, design stability, and technical debt/risk. Nine information needs with one corresponding indicator each fulfill these information needs were identified in literature and through the interviews and workshops with the practitioners.

Key words: metrics, software architecture

1 Introduction

Software architecting as an area has gained increasing visibility in the last two decades as the software industry recognized the role of software architectures in maintaining high quality and ensuring longevity and sustainability of the software products [21]. Even though this recognition is not new, there is still no consensus how to measure various aspects of software architectures beyond the basic structural properties of the software architecture as a design artifact. In the literature we can encounter studies applying base measures for object-oriented designs to software architectures [9] and studies designing low level software architecture measures such as number of interfaces [18]. However, an architect is often faced with the problem which high-level measures (indicators) should he/she use when monitoring the architecture during one software development project, addressing such information needs like – when is the architecture stable? or when is the architecture mature enough to start system testing?

In this paper we set off to identify which measures can be used for the above purpose by reviewing some of the most common measures of internal quality of software architectures and evaluate their applicability in industrial contexts in the Software Center research program which consists of five universities and nine companies. The goal of our study was to address the following research question:

Which software architecture measures fulfill the information needs of software architects of large software products?

The goal is to suggest a new measurement set and the result from our studies is a portfolio of measures and indicators addressing three elicited areas of information needs – architectural measures, design stability, and technical debt/risk. Each of these areas have three information needs with one indicator linked to each information need such as architecture changes, internal and external coupling, system complexity. The portfolio is accompanied with the visualization methods for each of these indicators which helps the architects to maintain a consistent overview of the indicators across projects and products. The portfolio can be extended by the architects to add new measures and indicators which address more specific information needs (e.g. at a specific point of time in the project).

This paper is structured as follows. Section 2 describes the most relevant related work to our study. Next, section 3 outlines the research process in this study. Section 4 present the architecture measures identified in our literature study, which is followed by the description of the subset of the measures identified as important for the software architects in the studied companies in Section 5. Finally we present the summary and conclusions in Section 6.

2 Related work

One of the most popular methods for evaluating software architectures in general is to use qualitative methods like ATAM [7] where the architecture is analyzed based on scenarios or perspectives. These methods are used for final assessments of the quality of the architectures, but as they are manual they need effort and therefore cannot be conducted in a continuous manner. However, as many of contemporary projects are conducted using Agile methodologies, Lean software development [16] or using the minimum viable product approach [17], these methods are not feasible in practice. Therefore the architects are willing to trade-off the quality of the evaluation to the speed of the feedback on their architecture, which leads to more extensive use of measure-based evaluation of software architectures.

Wagner et al. [29] presented a method for aligning quality models with the measurements and their goals where the gap between the abstract level of the goals and the concrete level of the measures is bridged. Our approach is a similar attempt but based on a specific scope (software architecture) and including the visualization of the results, thus our approach can be seen as an instantiation of Wagner et al.'s approach in a specific context.

One of the tools and methods supporting the architects' work with measures is the MetricViewer [27] which augments software architecture diagrams expressed in UML with such measures as coupling, cohesion or depth of inheritance tree. This augmentation is important to reason about the designs, but they are not linked to the information needs of the stakeholders to monitor attain-

ment of their goals, which otherwise require them to conduct the same analyses manually.

Similarly to Tameer et al. Vasconcelos et al [28] propose a set of metrics for measuring architectures based on low level properties of software architectures, such as number of possible operating systems or the number of secure components. Our work complements their study by focusing on internal quality properties related to the design and not quality in use.

The ISO/IEC 25000 Software Quality Requirements and Evaluation (SQuaRE) standard provides a set of reference measures for software designs and architectures. As per time of writing of this book the standard is not fully adopted but the main part are already approved and the work is fully ongoing regarding the measures, their definitions and usage. The standard presents the following set of measures related to product, design and architecture in one or its chapters – ISO/IEC 25023 - Software and Software Product Quality Measures [5]. The measures related to the execution of the product and do not focus on the internal quality of the product with such example measures as the size (e.g. number of components) or the complexity (e.g. control flow complexity). Therefore we need to turn to scientific literature to understand the measures and indicators related to software architectures. There we can find measures which are of interest for software architects.

Finally, the software engineering standard ISO/IEC 15939:2007 [14] provides a normative specification for the processes used to define, collect, and analyze quantitative data in software projects or organizations. The central role in the standard is played by the information product which is a set of one or more indicators with their associated interpretations that address the information need – an insight necessary for a stakeholder to manage objectives, goals, risks, and problems observed in the measured objects . These measured objects can be entities like projects, organizations, software products, etc. characterized by a set of attributes. We use the following definitions: (i) base measure measure defined in terms of an attribute and the method for quantifying it; (ii) derived measure measure that is defined as a function of two or more values of base measures; (iii) indicator measure that provides an estimate or evaluation of specified attributes derived from a model with respect to defined information needs and (iv) information product one or more indicators and their associated interpretations that address an information need. The view on measures presented in ISO/IEC 15939 is consistent with other engineering disciplines, the standard states at many places that it is based on such standards as ISO/IEC 15288:2007 (System lifecycle processes), ISO/IEC 14598-1:1999 (Information technology – Software product evaluation), ISO/IEC 9126-x, ISO/IEC 25000 series of standards, or International vocabulary of basic and general terms in metrology (VIM) [12].

3 Research process

The research process in our study is a mix of a literature study, interview and workshop. The goal of the study was to identify the most important information

needs of software architects and to provide the reference measurement system to fulfill these needs. We combined the research methods as follows:

- We conducted a literature review using snowballing and following the principles of systematic mapping of Petersen et al. [15]. It resulted in identifying 54 measures that can be applied to software architectures and three areas of information needs were elicited.
- We organized the measures according to the ISO/IEC 15939 standard's measurement information model [14] into base measures, derived measures and indicators.
- After that, we grouped them into three areas based on the information needs of software architects elicited from the literature study. We evaluated the applicability of the measures through an interview with an architect from one of the automotive companies. The architect was pointed out as an expert by the company representatives and worked in the area for a number of car projects before.
- We finally designed the portfolio of indicators which fulfill these information needs (presented in this paper).
- We presented to the architects and project managers at the defense company where we obtained feedback on the feasibility of this portfolio. In total two architects, two project managers and two quality managers took part.

The results from the evaluations were that the portfolio is promising and we are currently working on its full fledged implementation as a measurement system for the companies in Software Center.

4 Architecture measures in literature

We use the ISO/IEC 15939 measurement information model to organize the measures used for quantifying properties of software architectures. Conceptually we can also consider the fact the the higher in the model the measure is, the more advanced information need it fulfills. In figure 1 we can see a number of measures divided into three levels – the more basic ones at the bottom and the more complex one at the top.

The more advanced information needs are related to the work on the architects whereas the more basic ones are more related to the architecture as an artifact in software development. So, now that we have the model, let's look into one of the standards where the software measures are defined – ISO/IEC 25000.

Let's start with the base measures which quantify the architecture as shown in table 1 – we can quickly notice that the list of these measures correspond to the entity they measure. Their measurement method (the algorithm how to calculate the base measure) is very similar and is based on counting entities of a specific type. The list in table 1 shows a set of example of such base measures.

Collecting the measures presented in the table provides the architects with the understanding of the properties of the architecture, but the architects still

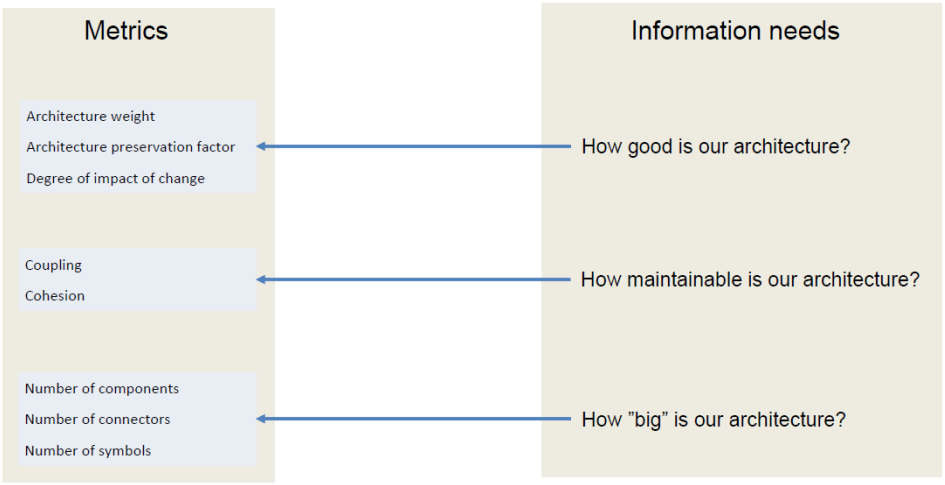

Fig. 1. Higher level measures correspond to more advanced information needs – an example.

Table 1. Base measures for software architectures

Measure	Description
Number of components [26]	The basic measure quantifying the size of the architecture in terms of its basic building block – components.
Number of connectors [26]	The basic measure quantifying the internal connectivity of the architecture in terms of its basic connectors.
Number of processing units [10]	The basic measure quantifying the size of the physical architecture in terms of the processing units.
Number of data repositories [10]	The complementary measure quantifying the size in terms of data repositories.
Number of persistent components [10]	Quantifies the size in terms of the needs for persistency.
Number of links [10]	Quantifies the complexity of the architecture, similarly to the McCabe cyclomatic complexity measure. It is sometimes broken down per type of link (e.g asynchronous – synchronous, data – control).
Number of types of communication mechanisms [10]	Quantifies the complexity of the architecture in terms of the need to implement multiple communication mechanisms.
Number of external interfaces [6]	Quantifies the coupling between the architectural components and the external systems.
Number of internal interfaces [6]	Quantifies the coupling among the architectural components.
Number of services [6]	Quantifies the cohesion of the architecture in terms of how many services it provides/fulfills.
Number of concurrent components [6]	The measure counts the components which have concurrent calculations as part of their behavior.
Number of changes in the architecture [2]	The measure quantifies the number of changes (e.g. changed classes, attributes) in the architecture
Fanout from the simplest structure [3]	The measure quantifies the degree of the lowest complexity of the coupling of the architecture

need to provide the context to these numbers in order to reason about the architectures. For example the number of components by itself does not provide

much insight, however, if put together with a timeline and plotted as a trend allows to extrapolate the information and therefore allow the architects to assess if the architecture is overly large and should be refactored.

In addition to the measures for the architecture we can also find many measures which are related to software design in general – e.g. object-oriented measures or complexity measures. Examples of these are presented in table 2.

Table 2. Base measures for software design

Measure	Description
Weighted methods per class [1]	The number of methods weighed by their complexity.
Depth of inheritance tree [1]	The longest path from the current class to its first predecessor in the inheritance hierarchy.
Cyclomatic complexity [11]	Quantifying the control path complexity in terms of the number of independent execution paths of a program. Used often as part of safety assessment in ISO/IEC 26262
Dependencies between blocks/modules/classes [25]	Quantifying the dependencies between classes or components in the system.
Abstractness of a Simulink block [13]	Quantifies the ration of the contained abstract blocks to the total number of contained blocks.

Once again these examples show that the measures are related to the design the quantification of its properties. Such measures as the *abstractness of a Simulink block*, however, are composed of multiple other measures and therefore are classified as derived measures and as such are closer to the information need of architects. In the literature we can find a large number of measures for designs and their combinations and therefore when choosing measures it is crucial to start from the information needs of the architects [19] since these information needs can effectively filter out measures which are possible to collect, but not relevant for the company (and as such could be considered as waste).

In the next section we identify which measures from the above two groups are to be included in the portfolio and which areas they belong to.

5 Metrics portfolio for the architects

The measures presented so far can be collected but as the measurement standards prescribe – they need to be useful for the stakeholders in their decision processes [20], [14]. Therefore we organize these measures into three areas corresponding to the information needs of software architects. As architecting is a process which involves the software architecture artifacts we recognize the need of grouping these indicators into areas related both to the product and the process.

5.1 Areas

In our portfolio we group the indicators into three areas related to the basic properties of the design, the stability of it and the quality of it:

Area: architecture measures Architecture measures – this area groups the product-related indicators that address the information need about *how to monitor the basic properties of the architecture like its component coupling.*

Area: design stability Design stability – this area groups the process-related indicators that address the information need about *how to ascertain controlled evolution of the architectural design.*

Area: technical debt/risk Technical debt/technical risk – this area groups the product-related indicators that address the information need about *how to ascertain the correct implementation of the architecture.*

In the following subsections we present the measures and the suggested way to present them. One of the criteria for each of these areas in our study was the upper limit of the number of indicators to be four. The limitations are based on the empirical studies of the cognitive aspects of measurement such as the ability to take in information by the stakeholders [24].

5.2 Area: architecture measures

In our portfolio we could identify 14 measures as applicable to measure the basic properties of the architecture. However, when discussing these measures with the architects the majority of the measures seemed to quantify the basic properties of the designs. However, the indicators found in the study in this area are:

Software architecture changes To monitor and control changes over time the architects should be able to monitor the trends in changes of software architecture at the highest level [2]. Based on our literature studies and discussions with practitioners we identified the following measure to be a good indicator of the changes – *number of changes in the architecture per time unit (e.g. week)* [4, 3, 8].

Complexity To manage module complexity the architects need to understand the degree of coupling between components as the coupling is perceived as cost-consuming and error-prone in the long-term evolution of the architecture. The identified indicator is *Average squared deviation of actual fanout from the simplest structure.*

External interfaces To control the degree of coupling on the interface level (i.e. a subset of all types of couplings) the architects need to observe the number of internal interfaces –*number of interfaces.*

Internal interfaces To have control of the external dependencies of the product the architects need to monitor the coupling of the product to external software products – *number of interfaces.*

The suggested presentation of these measures is presented in Figure 2.

5.3 Area: design stability

The next area which is of importance for the architects is related to the need for monitoring the large code base for stability. Generally, in this area we used the

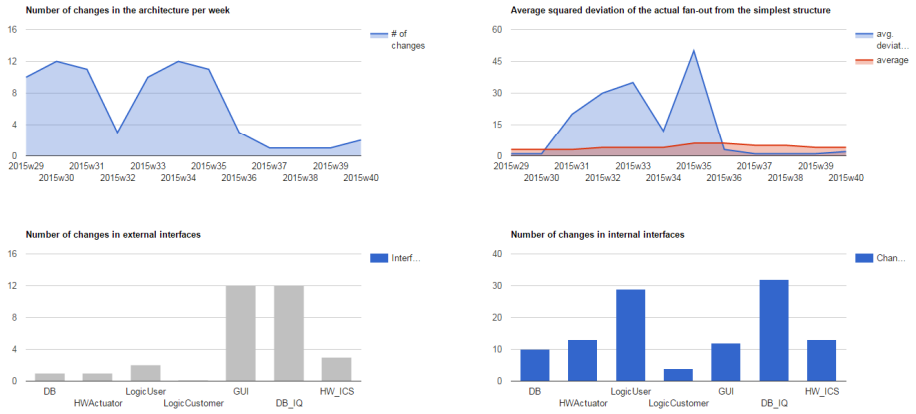

Fig. 2. Visualization of the measures in the architecture property area

visualizations from our previous research into code stability [22, 23]. We identified the following three indicators to be efficient in monitoring and visualizing the stability:

Code stability To monitor the code maturity over the time the architects need to see how much code has been changed over time as it allows them to identify code areas where more testing is needed due to recent changes. The measure used for this purpose is *number of changes per module per time unit.*

Defects per modules To monitor the aging of the code the architects need to monitor defect-proneness per component per time, using a similar measure as the code stability – *number of defects per module per time unit (e.g. week).*

Interface stability To control the stability of the architecture over its interfaces the architects measure the stability of the interfaces – *number of changes to the interfaces per time unit.*

We have found that in this view it is important to be able to visualize the entire code/product base in one view and therefore the dashboards which visualizes the stability is based on the notion of heatmaps [22]. In Figure 3 we present such a visualization with three heatmaps corresponding to these three stability indicators. Each of the figures is a heatmap which visualizes different aspects, but each of them is organized in the same way – columns designate the weeks, rows designate the single code modules or interfaces and the intensity of the color of each cell designates the number of changes to the module or interface during the particular week.

5.4 Area: technical debt/risk

The last area in our portfolio is related to the quality of the architecture over a longer period of time. In this area we identified the following two indicators:

Fig. 3. Visualization of the measures in the architecture stability area

Coupling To have manageable design complexity the architects need to have a possibility to quickly overview the coupling between the components in the architecture – measured by *number of explicit architectural dependencies*, where the explicit dependencies are links between the components which are introduced by the architects.

Implicit architectural dependencies To monitor where the code deviates from the architecture the architects need to observe whether there are no additional dependencies introduced during the detailed design of the software – this is measured by *number of implicit architectural dependencies*, where the implicit dependencies are such links between the components which are part of the code, but not introduced in the architecture documentation diagrams [25].

The visualization of the architectural dependencies shows the degree of coupling and is based on the circular diagrams as presented in Figure 4 where each area on the border of the circle represents a component and a line shows the dependencies between the components.

6 Conclusions

In this paper we set off to investigate which measures of software architectures could fulfill information needs of software architects in large software development companies in the embedded software development. We have reviewed a

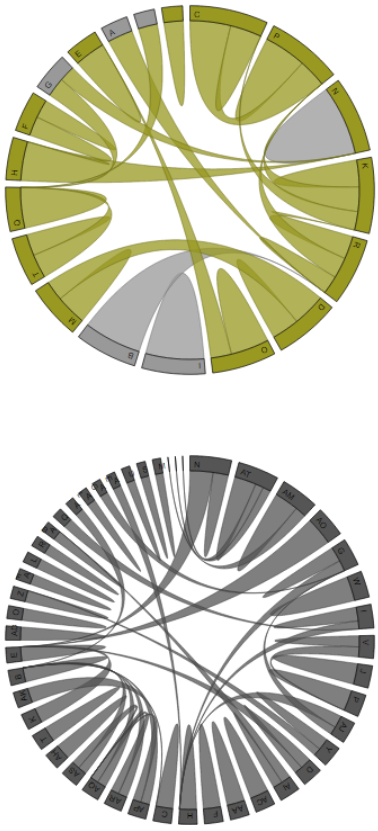

Fig. 4. Visualization of the measures in the architecture technical debt/risk

number of measures available in the standard (like ISO/IEC 25023) and available in the literature, shortlisting 54 measures clearly applicable for the role of software architect. The proposed portfolio of measures and indicators is applicable for the context where they were studied. However, as our studies were based on literature we can see that these results can also be applicable to other domains, which is one of the directions of our study.

In our future work we plan to conduct a full fledged evaluation of the portfolio by applying it to complex software development projects in the area of Internet of Things in our partner companies at Software Center.

Acknowledgment

This research has been partially carried out in the Software Centre, University of Gothenburg, Ericsson AB, and Volvo Car Group.

References

1. Shyam R Chidamber and Chris F Kemerer. A metrics suite for object oriented design. *Software Engineering, IEEE Transactions on*, 20(6):476–493, 1994.
2. Darko Durisic, Martin Nilsson, Miroslaw Staron, and Jörgen Hansson. Measuring the impact of changes to the complexity and coupling properties of automotive software systems. *Journal of Systems and Software*, 86(5):1275–1293, 2013.
3. Darko Durisic, Miroslaw Staron, and Martin Nilsson. Measuring the size of changes in automotive software systems and their impact on product quality. In *Proceedings of the 12th International Conference on Product Focused Software Development and Process Improvement*, pages 10–13. ACM, 2011.
4. Darko Durisic, Miroslaw Staron, Milan Tichy, and Jorgen Hansson. Quantifying long-term evolution of industrial meta-models-a case study. In *Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2014 Joint Conference of the International Workshop on*, pages 104–113. IEEE, 2014.
5. ISO/IEC. ISO/IEC 25023 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Measurement of system and software product quality. Technical report, 2016.
6. S Kalyanasundaram, K Ponnambalam, A Singh, BJ Stacey, and R Munikoti. Metrics for software architecture: a case study in the telecommunication domain. In *Electrical and Computer Engineering, 1998. IEEE Canadian Conference on*, volume 2, pages 715–718. IEEE, 1998.
7. Rick Kazman, Mark Klein, and Paul Clements. Atam: Method for architecture evaluation. Technical report, DTIC Document, 2000.
8. Ludwik Kuzniarz and Miroslaw Staron. Inconsistencies in student designs. In *the Proceedings of The 2nd Workshop on Consistency Problems in UML-based Software Development, San Francisco, CA*, pages 9–18, 2003.
9. Mikael Lindvall, Roseanne Tesoriero Tvedt, and Patricia Costa. An empirically-based process for software architecture evaluation. *Empirical Software Engineering*, 8(1):83–108, 2003.
10. Chung-Horng Lung and Kalai Kalaichelvan. An approach to quantitative software architecture sensitivity analysis. *International Journal of Software Engineering and Knowledge Engineering*, 10(01):97–114, 2000.
11. Thomas J McCabe. A complexity measure. *Software Engineering, IEEE Transactions on*, (4):308–320, 1976.
12. International Bureau of Weights and Measures. *International vocabulary of basic and general terms in metrology*. International Organization for Standardization, Genve, Switzerland, 2nd edition, 1993.
13. Marta Olszewska. Simulink-specific design quality metrics. *Turku Centre for Computer Science*, 2011.
14. International Standard Organization and International Electrotechnical Commission. Software and systems engineering, software measurement process. Technical report, ISO/IEC, 2007.
15. Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic mapping studies in software engineering. In *12th international conference on evaluation and assessment in software engineering*, volume 17, pages 1–10. sn, 2008.
16. Mary Poppendieck. Lean software development. In *Companion to the proceedings of the 29th International Conference on Software Engineering*, pages 165–166. IEEE Computer Society, 2007.

17. Eric Ries. *The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses*. Random House LLC, 2011.
18. Cláudio SantAnna, Eduardo Figueiredo, Alessandro Garcia, and Carlos JP Lucena. On the modularity of software architectures: A concern-driven measurement framework. In *Software Architecture*, pages 207–224. Springer, 2007.
19. M. Staron, W. Meding, G. Karlsson, and C. Nilsson. Developing measurement systems: an industrial case study. *Journal of Software Maintenance and Evolution: Research and Practice*, pages n/a–n/a, 2010.
20. Miroslaw Staron. Critical role of measures in decision processes: Managerial and technical measures in the context of large software development organizations. *Information and Software Technology*, 54(8):887–899, 2012.
21. Miroslaw Staron. Software engineering in low-to middle-income countries. *Knowledge for a Sustainable World: A Southern African-Nordic contribution*, page 139, 2015.
22. Miroslaw Staron, Jorgen Hansson, Robert Feldt, Anders Henriksson, Wilhelm Meding, Sven Nilsson, and Christoffer Hoglund. Measuring and visualizing code stability–a case study at three companies. In *Software Measurement and the 2013 Eighth International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2013 Joint Conference of the 23rd International Workshop on*, pages 191–200. IEEE, 2013.
23. Miroslaw Staron, Ludwik Kuzniarz, and Ludwik Wallin. Case study on a process of industrial mda realization: determinants of effectiveness. *Nordic Journal of Computing*, 11(3):254–278, 2004.
24. Miroslaw Staron, Wilhelm Meding, Jörgen Hansson, Christoffer Höglund, Kent Niesel, and Vilhelm Bergmann. Dashboards for continuous monitoring of quality for software product under development. *System Qualities and Software Architecture (SQSA)*, 2013.
25. Miroslaw Staron, Wilhelm Meding, Christoffer Hoglund, and Jorgen Hansson. Identifying implicit architectural dependencies using measures of source code change waves. In *Software Engineering and Advanced Applications (SEAA), 2013 39th EUROMICRO Conference on*, pages 325–332. IEEE, 2013.
26. Srdjan Stevanetic, Muhammad Atif Javed, and Uwe Zdun. Empirical evaluation of the understandability of architectural component diagrams. In *Proceedings of the WICSA 2014 Companion Volume*, page 4. ACM, 2014.
27. Maurice Termeer, Christian FJ Lange, Alexandru Telea, and Michel RV Chaudron. Visual exploration of combined architectural and metric information. In *Visualizing Software for Understanding and Analysis, 2005. VISSOFT 2005. 3rd IEEE International Workshop on*, pages 1–6. IEEE, 2005.
28. André Vasconcelos, Pedro Sousa, and José Tribolet. Information system architecture metrics: an enterprise engineering evaluation approach. *The Electronic Journal Information Systems Evaluation*, 10(1):91–122, 2007.
29. Stefan Wagner, Klaus Lochmann, Lars Heinemann, Michael Kläs, Adam Trendowicz, Reinhold Plösch, Andreas Seidl, Andreas Goeb, and Jonathan Streit. The quamoco product quality modelling and assessment approach. In *Proceedings of the 34th international conference on software engineering*, pages 1133–1142. IEEE Press, 2012.