Quantifying Software Architecture Quality

Report on the First International Workshop on Software Architecture Metrics

Robert L. Nord, Ipek Ozkaya
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA, USA
{rn, ozkaya}@sei.cmu.edu

Heiko Koziolek
ABB Corporate Research
Ladenburg, Germany
heiko.koziolek@de.abb.com

Paris Avgeriou
University of Groningen
Groningen, NL
paris@cs.rug.nl

DOI:10.1145/2659118.2659140 http://doi.acm.org/10.1145/2659118.2659140

ABSTRACT
Architects of complex software systems face the challenge of how best to assess the achievement of quality attributes and other key drivers, how to reveal issues and risks early, and how to make decisions about architecture improvement. Software architecture quality has a large impact on this effort, but it is usually not assessed with quantitative measures. As the pace of software delivery and technology churn increases, organizations need guidance on how to meet the business goals of their software. There is an increasing need to provide ongoing insight into the quality of the system being developed. Additionally, it is highly desirable to improve feedback between development and deployment through measurable means for intrinsic quality, value, and cost. There is increasing attention to fields such as software analytics and empirical software engineering and measurement that can provide the theory, tooling, or inspiration to develop measurement and analysis frameworks for software architecture. This paper reports on the results of the First International Workshop on Software Architecture Metrics, where participants discussed challenges and opportunities in quantifying software architecture quality.

Categories and Subject Descriptors
D.2.8 [**Metrics**], D.2.9 [**Management**], D.2.11 [**Software Architectures**].

General Terms
Design, Management, Measurement.

Keywords
Software architecture; metrics; software analytics; technical debt; software quality; software maintenance and evolution; empirical software engineering; qualitative methods.

1. INTRODUCTION
A software architecture encompasses the important design decisions of a software system and focuses on addressing run-time, design-time, and business quality attributes. Assessing a software architecture throughout the life cycle of a software system is thus crucial to assure a software system's quality. Experts use best practices for software architectures, such as modularization with low coupling and high cohesion, separation of concerns, conceptual integrity, architecture patterns, and appropriate third-party components. During architecture reviews, they assess whether the software architecture follows such best practices and achieves a good trade-off between the required quality attributes.

This expert judgment is usually expressed in qualitative terms, as most scenario-based architecture-review methods prescribe. It is difficult to communicate these detailed technical assessments to other stakeholders, such as management or customers. Furthermore, such qualitative assessments are usually specific to a system; thus they do not allow for a comparison of the architecture under analysis to the state of the art. There is no standard set of base measures to assess software architecture quality, although there are generic quality aspects. Quantitative values are sometimes reported with respect to key architecturally significant requirements; for example, timing properties of prototypes or the full implementation are measured for performance-related quality attributes, or down-time during maintenance is measured for availability-related quality attributes. But quantitative measures are rarely seen in practice.

IEEE1061 defines a software quality metric as "a function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality." The software engineering community has developed metrics to measure the quality of source code (e.g., size, complexity, coupling, stability). There are also tools for such metrics, such as development environments like Eclipse, Visual Studio, and IDEA or static code analysis tools like Understand, Klocwork, or NDepend. There is also an active software analytics community that mines code repositories (e.g., [1][2][3][4]), issue trackers, and version histories for actionable [5] information. Other artifacts, especially the software architecture, have received less attention in research and practice though they are critical to a software system's quality [6].

To this end, we organized the First International Workshop on Software Architecture Metrics (SAM) at the Working IEEE/IFIP Conference on Software Architecture (WICSA 2014). We proposed a definition for the term *software architecture metric* as follows: "a software quality metric that concerns software architecture and quantifies architecture quality, value, and cost." Different artifacts provide input for computing a software architecture metric: informal architectural documentation, architecture models and views, architecture decisions, source code, byte code, and trace links between an architecture and other artifacts.

The goal of the workshop was to establish a community dealing with software architecture metrics. Furthermore, the workshop aspires to foster discussion on the quality of existing software architecture metrics [7][8][9][10][11] and create a future research agenda. Finally, the SAM community intends to work on a joint body of knowledge that would lower the entrance barrier of practitioners to apply software architecture metrics and extend the use of other related metrics for improving the quality of software architectures.

The remainder of this paper is structured as follows. Section 2 summarizes presentations of the workshop's participants illustrating different perspectives on the topic. Section 3 provides a list of discussed research challenges, and Section 4 sketches possible solution approaches as identified by the workshop's participants.

2. WORKSHOP CONTRIBUTIONS
The workshop brought together 27 attendees and featured four presentations that are available for download [12]. The presentations described how to derive appropriate metrics and apply them to architecture models and source code:

Metrics for Sustainable Software Architectures: An Industry Perspective, authored by Eric Harper, Aldo Dagnino, and Will Snipes, ABB Corporate Research. This presentation covered experiences in the development of measurement objectives using the Goal–Question–Metric approach. Industry practitioners grapple with poor and degraded architectures that slow the progress of software evolution and reduce profits. Practitioners require architecture metrics that support sustainable software architectures reflecting quality attributes such as maintainability, extensibility, component reliability, and structural integrity. Supporting architecture metrics in these domains will provide industrial practitioners with the tools they need to manage software architecture in a business-case-driven corporate environment.

Empirical Evaluation of the Understandability of Architectural Component Diagrams, authored by Srdjan Stevanetic, Muhammad Atif Javed, and Uwe Zdun, University of Vienna. This presentation reported the results from a study carried out to examine how well software architecture can be conveyed through architectural component diagrams. The statistical evaluation of the results showed that metrics such as the number of components, number of connectors, number of elements, and number of symbols used in the diagrams can decrease architectural understandability when they are above and below a certain threshold that can be roughly predicted. The results indicate that architectural understandability is linearly correlated with the perceived precision and general understandability of the diagrams.

Toward Quantitative Metrics for Architecture Models, authored by Stephan Sehestedt, Chih-Hong Cheng, and Eric Bouwers, ABB Corporate Research and Software Improvement Group. 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 important 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. This presentation addressed this challenge by proposing seven metrics that characterize the completeness, consistency, correctness, and clarity of the architecture documentation.

On the Challenges in Extracting Metrics from Java Bytecode, authored by Jean-Guy Schneider, Swinburne University of Technology. Extracting metrics from real-world software systems is generally considered to be a nontrivial task due to challenges in processing large amounts of source code and factors determining what artifacts belong to a system's build. For Java-based systems, the Bytecode generated by compilers is often considered as good as source code for analysis, but without the associated challenges of identifying how the system is to be built. Unless appropriate care is taken, metrics extracted from Java Bytecode can be imprecise or even wrong, especially when common heuristics are used. This presentation elaborated recent experiences in extracting metrics for Java-based systems, the challenges faced, and the approaches taken to use Bytecode as a source for system metrics.

3. RESEARCH CHALLENGES

During the presentations and a subsequent discussion session, the workshop participants discussed desirable properties of software architecture metrics. They should be simple to measure, use, and understand. Ideally they should be computable from the available software development artifacts without manual overhead and provide stable, repeatable results. Each metric should provide actionable information. Finally, they should be measurable in an objective, system-independent manner so that their values can be compared across systems.

In the discussions, the workshop participants identified a number of research challenges for the SAM community:

How to codify informal best practices to derive quantitative metrics: While experienced architects can make a qualitative judgment on the quality of a software architecture, it is difficult to formalize the best practices this judgment is based on. Experienced architects acquired this judgment from executing many projects, but there are only limited efforts to document this knowledge and share it across the community. There are also formalized descriptions of best practices in literature, such as architecture design pattern catalogues, architecture tactics, or architecture styles. Is it possible to express the adherence to such best practices with appropriate metrics that are simple to compute?

How to measure architecture complexity and ease of change: There are different notions of how to measure software architecture complexity from descriptions such as the entropy of the component diagrams or the amount and usage of architecturally relevant interfaces. Which measures of architecture complexity are most useful so that architects and developers can take actions on appropriate refactorings? Assessing the ease of change of an architecture requires the identification of likely modification scenarios to the system. Can this measure of technical risk management be captured in an architecture metric comparable across systems?

How to quantitatively assess the goodness of architecture decisions: Architecture design decisions are usually captured in textual form without any quantitative quality indicator. The confidence level of the decision maker regarding each decision and the amount of documented rationale could provide input to a metric for architecture decision quality. Likewise, the coverage of architecturally relevant requirements under the time and budget constraints could be helpful. The challenge is to condense the available information into an objective metric.

How to incorporate informal artifacts into metrics computations: In practice, appropriate software architecture documentation is often not available or is outdated, leaving the source code as the main source of information. Raw data mined from code repositories may be hard to understand and process for architecture metrics. Sometimes, the implementation is not yet available when an assessment of the architecture's quality is desired. However, for many systems, informal artifacts exist in the form of diagrams, whiteboard sketches, tutorials, or architecture guidelines for developers. Can this information be used to derive architecture metrics (e.g., using information retrieval techniques or by appropriately formalizing these artifacts in a short time frame)?

How to define domain-specific software architecture metrics: For certain business domains (e.g., banking, insurance, avionics, industrial) or technical domains (e.g., embedded, distributed, desktop), there might be an opportunity to compare the architecture quality among systems with domain-specific measures. For example, industrial control systems follow certain architectural patterns, whose adherence could be encoded into architectural metrics. Real-time systems have specific architectural features, for which specific metrics could be defined. Domains with more mature architectural approaches, such as reference architectures or architecture frameworks, are good candidates to start an investigation for domain-specific architecture metrics.

When quantitative metrics are more beneficial than qualitative assessments:
Metrics condense certain facts into a single number and may not be the best representation in many cases. They are tuned for a specific audience and present a quick overview and identification of gaps before drilling down into more details. More detailed qualitative assessments can provide more insight when addressing a low metric score. For best practice adherence, checklists are often used in practice instead of metrics. Therefore, the use cases of software architecture metrics need to be better defined, to evaluate where they are best used in addition to qualitative assessment.

How to avoid abuse of software architecture metrics: Abuse of metrics is not limited to software architecture metrics. Systems are often optimized for certain metrics while disregarding other important issues that are not captured by the applied metrics system. For example, if the quality of a layering structure is captured by metrics, but not the quality of third party components, it is easy to achieve good values for the metric but still implement a poor system. A way to deal with this situation is to use contrasting metrics in order to avoid "blind" optimization of certain metrics. For example, checking homogeneous module sizes could provide a false sense of security if, for example, we have only two similar-sized modules in the system with poor internal structure. Therefore, it is advisable to check for maximum module sizes.

Misusing metrics cannot be prevented, but measures could be taken to limit the amount of abuse. We anticipate that more research challenges will arise once researchers and practitioners gain more experience with software architecture metrics.

4. SOLUTION APPROACHES

In addition to the research challenges, the workshop participants proposed a number of solution approaches that could help tackle the challenges. Some of these solution approaches are already geared toward the implications of using software architecture metrics in practice.

Create an architecture metrics catalog: Numerous software architecture metrics have already been defined in literature but are scattered across research papers. There is no central catalog of existing metrics that could help practitioners assess the appropriate coverage of software architecture quality attributes. Similarly to a pattern catalog, the metrics should be described using a common template (e.g., context, goals, underlying principles, inputs needed, computation, output, implications, improvement actions, empirical experience from existing systems). A metrics catalog could also state thresholds or target values for software architecture metrics. Having a comprehensive overview would allow practitioners to select metrics for their systems based on a Goal–Question–Metrics approach and would help researchers to identify areas for which there are currently no metrics available.

Derive architecture metrics from patterns and styles: There may be potential to mine existing architecture patterns with the goal of deriving architecture metrics from them. For example, violations of patterns could be quantified, or the appropriate use of a certain pattern language could be checked. There is a large body of literature on patterns that can serve as a starting point for such an investigation.

Establish a common test bed for architecture metrics: Benchmarks are used in performance engineering to test systems according to a common number of test cases. To evaluate the usefulness of existing software architecture metrics and to develop new ones, having a standard repository of systems to test as a “metrics benchmark” could improve research work. Such a repository would ideally contain architecture documentation, requirements, and source code of software systems representative of certain domains (e.g., open source software). These artifacts would serve as input for calculating software architecture metrics and would help researchers to quickly carry out experiments.

Develop good metrics computation tools: Despite a significant commercial market for software metrics tools, there is still potential to develop better software tools to compute, visualize, and improve on software architecture metrics. In the future, there should be configurable dashboards to visualize metrics, which would also report trends if metrics are recomputed based on changed artifacts. Integration into popular integrated development environments and architecture modeling tools is desirable. Any tools should be accompanied by training material and explanations for interpreting and addressing the computed metrics.

5. ACKNOWLEDGMENTS

Putting together SAM 2014 was a team effort. We thank the authors of all submitted papers and invited speakers for providing the content of the program. We would like to express our gratitude to the program committee, who contributed their expertise, provided valuable feedback to the authors, and helped improve the quality of the accepted papers:

- Pierre America, Philips Research, NL
- Steffen Becker, University of Paderborn, DE
- Eric Bouwers, Technical University Delft, NL
- Yuangfang Cai, Drexel University, US
- Jane Cleland-Huang, DePaul University, US
- Neil Ernst, Software Engineering Institute, US
- Rich Hilliard, Consulting Software Systems Architect, US
- Oliver Hummel, Karlsruhe Institute of Technology, DE
- Anton Jansen, ABB, SE
- Rainer Koschke, University of Bremen, DE
- Philippe Kruchten, University of British Columbia, CA
- Tim Menzies, West Virginia University, US
- Matthias Naab, Fraunhofer IESE, DE
- Oscar Pastor, Valencia University of Technology, ES
- Neeraj Sangal, Lattix, US
- Jean-Guy Schneider, Swinburne University of Technology, AU
- Carolyn Seaman, University of Maryland Baltimore County, US
- Bran Selic, Malina Software Corp., CA
- Will Snipes, ABB, US
- Michael Stal, Siemens, DE
- Robert Stoddard, Software Engineering Institute, US
- Uwe Zdun, University of Vienna, AT
- Liming Zhu, National ICT Australia, AU
- Olaf Zimmermann, University of Applied Sciences, CH

6. CONCLUSIONS

This was the first edition of the SAM workshop and acted as a kickoff for the discussion on software architecture metrics as well as a first meeting and formulation of the SAM community. Given the broad participation by both academics and industrial practitioners and the intense discussions that were held, we believe there is interest and momentum to establish a substantial stream of work in this area. We already consider that the research challenges and the solution approaches identified are a great starting point for the research community, hopefully in close collaboration with the industry. Finally, we believe in the cross-fertilization between the SAM community and the other communities related to metrics, software analytics, and mining software repositories. We are amenable to organizing future editions of the workshop and supporting the SAM community to grow and flourish.

7. DISCLAIMER

8. REFERENCES

[1] The Working Conference on Mining Software Repositories. http://www.msrconf.org/.

[2] Menzies, T. and Zimmermann, T. 2013. Software analytics: so what? *IEEE Software* 30, 4 (Jul./Aug. 2013), 31-37.

[3] Xie, T., Zimmermann, T., and van Deursen, A. 2013. Introduction to the special issue on mining software repositories. *Empir. Softw. Eng.* 18, 6 (Dec. 2013), 1043-1046.

[4] Workshop on Emerging Trends in Software Metrics (WeTSOM). http://www.rcost.unisannio.it/wetsom2014/#previouseditions.

[5] Zimmermann, T., Weißgerber, P., Diehl, S., and Zeller, A. 2004. Mining version histories to guide software changes. In *Proceedings of the 26th International Conference on Software Engineering* (Edinburgh, UK, May 23-28, 2004). IEEE Computer Society, Washington, DC, 563-572.

[6] Avgeriou, P., Stal, M., and Hilliard, R. 2013. Architecture sustainability. *IEEE Software* 30, 6 (Nov./Dec. 2013), 40-44.

[7] Koziolek, H. 2011. Sustainability evaluation of software architectures: a systematic review. In *Proceedings of the Joint ACM SIGSOFT Conference – QoSA and ACM SIGSOFT Symposium – ISARCS on Quality of Software Architectures – QoSA and Architecting Critical Systems* (Boulder, CO, Jun. 20-24, 2011). ISARCS, New York, NY, 3-12.

[8] Kruchten, P., Nord, R., and Ozkaya, I., Eds. 2012. *IEEE Software*, Special Issue on Technical Debt, 29, 6 (Nov./Dec. 2012).

[9] Callo Arias, T., van der Spek, P., and Avgeriou, P. 2011. A practice-driven systematic review of dependency analysis solutions. *Empir. Softw. Eng.* 16, 5 (Oct. 2011), 1-43.

[10] Macia, I., Garcia, J., Popescu, D., Garcia, A., Medvidovic, N., and von Staa, V. 2012. Are automatically-detected code anomalies relevant to architectural modularity? An exploratory analysis of evolving systems. In *Proceedings of the 11th Annual International Conference on Aspect-Oriented Software Development* (Potsdam, Germany, Mar. 25-30, 2012). ACM, New York, NY, 167–178.

[11] Bouwers, E., van Deursen, A., and Visser, J. 2013. Evaluating usefulness of software metrics: an industrial experience report. In *Proceedings of the 2013 International Conference on Software Engineering* (San Francisco, CA, May 18-26, 2013). IEEE Press, Piscataway, NJ, 921-930.

[12] First International Workshop on Software Architecture Metrics. http://www.sei.cmu.edu/community/sam2014/.