On Bridging the Gap between Practice and Vision for Software Architecture Reconstruction and Evolution - A Toolbox Perspective

Ana Dragomir, M. Firdaus Harun, Horst Lichter
RWTH Aachen University
Research Group Software Construction
Aachen, Germany
{ana.dragomir, firdaus.harun, lichter}@swc.rwth-aachen.de

ABSTRACT

Up-to-date architecture views help to better understand and meaningfully evolve software systems. Despite their importance, the views are typically either not defined or not monitored and updated when changes to the actual systems are performed. They thus become subject of continuous degradation. To reconstruct the views, architecture monitoring and reconstruction tools have been developed and proposed. However, according to our analysis of the state of the art and state of the practice, we have identified that existing tools still lack some important ingredients needed to meaningfully monitor and reconstruct the architecture description of software systems. This paper gives an insight of these improvement potentials and proposes a vision for the development of a stronger industry-oriented software architecture monitoring, reconstruction and evolution tool.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures

General Terms

Documentation, Design

Keywords

Software Architecture Reconstruction; Software Architecture Monitoring; Software Architecture Evaluation

1. INTRODUCTION

The architecture of software systems directly influences crucial quality attributes and therefore should be considered whenever important decisions regarding their evolution must be taken. However, even though the importance of software architectures is widely acknowledged, complete and/or up-to-date architecture descriptions rarely exist ([4], [3], [5]). While methods and corresponding tool support for reconstructing the current views of a system's architecture have been developed and proposed, there are still important improvement potentials that need to be addressed by these in order to meaningfully support the needs of the architects employing them.

WICSA '14, April 07-11 2014, Sydney, NSW, Australia

2. GOALS

The primary goal of this paper is to discuss new potential ideas for improvement of software architecture reconstruction tools.

To achieve this we first give an overview of the current state of the art (Section 3). Secondly, we share our experiences and challenges that we have faced with our industry cooperation partners - two software companies operating in two very different domains, having different sizes and employing very different internal processes (Section 4). We then identify six possible improvement potentials that should be addressed by the future work in the field (Section 6) and present a vision on how these potentials should be applied in the practice (Section 7).

3. CURRENT STATE OF THE ART

The idea of extracting up-to-date software architecture descriptions from the source code of software systems is not new. E.g., a comprehensive, yet not exhaustive, list of software architecture reconstruction tools can be found in [4]. While scalable solutions for retrieving the structural view of the architecture do exist (e.g., Sonargraph-Architect [2], SAVE [5], etc.), most of the current state of the art addresses "exemplary" situations and is usually not appropriate to be applied on large-scale, heterogeneous systems. Even if some tools do offer technology bindings for various programming languages (e.g., [3], [5], [2], [7]), the considered heterogeneous systems are analyzed separately and their inter-play within a complex landscape is not recovered.

In order to allow the architects to define high-level architectural elements (e.g., layers, modules, etc) and communication rules between them, the reconstruction tools implement specific meta-models that can be instantiated accordingly. E.g., Sonargraph-Architect [2] allows the definition of layers, layer groups, vertical slices, vertical slices groups and subsystems. The LISA [3] toolkit allows the definition of modules, subsystems and layers or of components, ports and contracts - in the case of component-based architectures. To assess the quality of the reconstructed views some tools offer

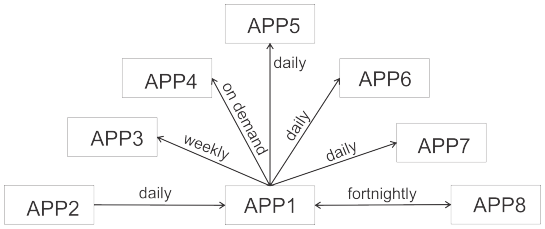

Figure 1: Information Flow Diagram - Example

metric dashboards. However, these metrics are usually hard-coded and thus inflexible (e.g., [1]). Sonargraph-Architect [2] differentiates itself from the other tools by offering a built-in dashboard containing size- and structure-related metrics. The dashboard can be easily customized to include additional metrics that were not considered previously.

Last but not least, while some solutions ([2]) do allow the simulation of evolution (e.g.: simulate moving a class in another package), to the best of our knowledge no tool offers the possibility to document alternative evolution variants and to compare them to each other.

4. CURRENT STATE OF THE PRACTICE

The state of the practice described in this section is based on our experiences with two industry cooperation partners. Because of confidentiality reasons, we will use fictive names to refer to our partners in the context of the current paper.

The **first industry partner, LC,** is a large software company that has more than 1000 IT employees in Europe and is CMMI Level 3 certified.

LC is developing and maintaining very *heterogeneous software systems* that are written in programming languages such as Cobol, Java and even Smalltalk.

Being CMMI Level 3 certified, LC implements the "Decision Analysis and Resolution" process area and documents the wide majority of decisions taken within various projects - i.e. including the most important architectural decisions. In addition to this, all the software development projects in the company are required to create an architecture description document. According to the established development process in LC, the architecture documentation is created very early in the development lifecycle. On the other hand, architecture decisions can be taken at any time throughout the entire process. While decisions do change the actual architecture of the developed systems, the actual architecture documentation and the architecture decisions are kept separately. Since *traceability links between the two are currently not maintained* (e.g.: Which components/part of the architecture are affected by the current decision? What later decisions have affected a given part of the architecture description?), the architecture documentation will eventually become outdated and can only be understood if analyzing all the decisions taken after its definition.

Furthermore, LC has been investing in the recent years efforts into documenting their enterprise architecture. One of the key documented aspects is the inter-systems information flow. An obfuscated example of an information flow diagram can be seen in Figure 1. Here, we can observe all the information flows that the application depicted in the middle has with other applications in LC's application landscape. Unfortunately, this data is inputted only manually. *Architecture reconstruction and/or monitoring techniques are not employed.* This induces a huge overhead that needs to be invested into (1) initially describing all the information flows and (2) maintaining these consistent with the actual software landscape.

The **second industry partner, MC,** is a small to medium company employing more than 200 developers, distributed geographically in Europe, Asia and Australia. They do not rely on certified processes, but rather work according to established internal process fragments. With few exceptions that are employing C++ and .Net technologies, most of the systems developed at MC are Java-based and use the OSGi technology.

To help developers understand the detailed design of the systems, a reference book called the developer handbook (DHB) is provided. The DHB contains very low-level descriptions of the applications and modules. Also, this *documentation is not linked or traceable to any existing software artifacts and source code.* Even more, because the DHB contains the descriptions of all the systems and all their versions and because all developers are entitled to edit it, its size increases rapidly while the quality of the content degrades fast, even leading to the presence of contradictory information and thus to misunderstandings and confusions.

Furthermore, MC is lacking an abstract view of all their systems. Since the complexity of the systems is constantly increasing, maintenance and refactoring activities are increasingly more complicated. In a joint project with MC, we have started to analyze their system structures in order to define an up-to-date architectural description that can sustain their further refactoring and evolution activities. We have used functional decomposition techniques for defining the most important architectural elements (modules, composite modules, subsystems, etc.). Next, we have explored the corresponding source code to determine what the possible candidates for the aforementioned architecture elements are. To ease our work, we have employed the Sonargraph-Architect reconstruction tool and created an architecture description based on layers - the central architectural element defined in the Sonargraph-Architect's meta-model. However, the layered presentation led to misunderstandings: the MC developers and architects do not consider that their system is a layered one. This led to confusions due to *terminology differences.* The efforts for defining a clear architecture description are still ongoing.

We can conclude that the current state of the practice analyzed by us does not employ or employs very late architecture reconstruction tools. Terminology differences between the employed tool and the concepts used by the architects to express their architecture can lead to misunderstandings. Furthermore, the inter-systems information flows and their attributes are modelled manually causing important disadvantages. Last but not least, the architecture- and architecture decisions documentation are done manually and therefore important traceability relations are typically missing.

5. STATE OF THE ART VS. STATE OF THE PRACTICE

As depicted in Section 4, the problems faced by the industry are very complex. No single reconstruction tool solves all these problems, but typically addresses a subset thereof. However, there exist also problems (e.g., reconstructing the

architecture of heterogeneous systems) that are not addressed by any of the existing tools. We propose the creation of a modular reconstruction toolbox that facilitates the plug-in-based integration of reconstruction tools as well as the addition of new ones. We thus aim to facilitate the tools' interoperability towards addressing the specific goals and needs of the architects. A brief description of our vision will be given in Section 7.

6. IMPROVE POTENTIALS

In this section we identify six improvement potentials that, if adopted, could help closing or shrinking the gap between the actual software architecture practice and a visionary situation where software organizations are applying architecture reconstruction and monitoring tools consistently.

6.1 Traceability Links

Problem Description. "The architecture as a whole does not exist in any artifact that we actually implement" [5]. As a consequence, even if an up-to-date architecture description exists, this is prone to getting out of sync with the underlying system, if no effort is invested to prevent this. We consider that the following information should be made traceable and monitored continuously: (1) The static and dynamic views of the architecture description should be linked with the actual underlying systems (currently the state of the art mostly addresses the static views alone [4], [3]); (2) If documented separately, the architecture description should be linked with further taken decisions

Targeted benefits. If the description is consistent with the actual architecture, one could rely on it to systematically evolve a system, while preserving the architecture restrictions and properties. Furthermore, traceability links between the architecture description and the architecture decisions would ensure a better understanding of the way the architecture has evolved up to the present situation, the considered tradeoffs and the rationale behind the made choices.

State of the art status. Many software architecture reconstruction ([4]) as well as decisions documentation approaches ([6]) have been proposed. However, a definitive answer to the traceability problem has not been given.

6.2 Common Terminology

Problem Description. As exemplified in Section 4, architects are reluctant to accept the reconstruction results, if these are described using different concepts than the ones they are using when referring to their system's architecture (e.g.: "layers" instead of "modules"). The architects should thus be able to first express the architecture meta-model that they are familiar with and then instantiate it accordingly in the reconstruction tool of their choice.

Targeted benefits. The creation of a common language basis between the architects and the reconstruction tool will lead to the reduction of misunderstandings and terminology misuses when interpreting the reconstructed results. This will naturally also increase the acceptance of the reconstruction tools in industry contexts.

State of the art status. "If all you have is a hammer, everything looks like a nail". As pointed out in Section 3, when applying reconstruction techniques, architects usually need to express their top-level architecture using the constructs offered by the reconstruction tools they are applying.

6.3 Metrics

Problem Description. Architectural metrics are typically not available to facilitate the architect's understanding of the quality of the reconstructed architecture and the most important or urgent improvement directions that should be considered.

Targeted benefits. If similar solutions to the Sonargraph-Architect's dashboard were employed on a more frequent basis allowing the architects to define structural and behavioural metrics of relevance for their systems, more insightful information regarding the current status of the analyzed architecture that could better sustain the further evolution activities could be gained. E.g., consider two architectural violations V1 and V2. While V1 and V2 might be considered just as severe from a logical point of view, V1 might occur more often and additionally might cause a more severe performance problem than V2. Thus, on a priority list of restructurings V1 should be considered first. However, this would have possibly not been noticed, if a careful analysis of the behavior were not employed.

State of the art status. As depicted in Section 3, solutions for flexible architectural metric dashboards have already been employed - however, just for the static architecture view. To the best of our knowledge, metrics for the assessment of software architecture behavior are not employed by any reconstruction tool.

6.4 Variants Building

Problem Description. Once an up-to-date architecture description has been recovered, understood and evaluated, the problem of meaningfully evolving the system still persists.

By analyzing the repository of architecture decisions documented by two large-scale projects (the first one comprises 45000 IT person days while the second 9000 IT person days) undergone in LC, we have acknowledged that, when facing this issue, architects have typically constructed software architecture evolution variants or alternatives. These were then manually assessed in order to identify the variant that best suits their evolution goals while respecting the predefined established architectural rules and guidelines.

To better sustain the software architecture evolution, reconstruction tools should support the definition and assessment of evolution variants. In particular the assessment should play a key role, in order to support the architects to identify out of the defined evolution variants exactly those that are better aligned with their goals.

Targeted benefits. By allowing the definition of evolution variants on top of the extracted architecture views, one can better ensure that the evolution direction really addresses the architectural problems currently existing in the system. Furthermore, the metrics (or a subset thereof) used to evaluate the reconstructed views (see Section 6.3) could then be also applicable to compare the defined evolution variants to each other. Consequently, hints or recommendations could be offered regarding which variants are more likely to bring more benefits if implemented, thus better sustaining important architecture decisions.

State of the art status. Some efforts have already been done (e.g. [2]) to allow the architects specify some modifications of the recovered architecture and assess their effect. However, their capabilities are still limited.

6.5 Scalability and Heterogeneity

Problem Description. When considering industry contexts, the employed systems are typically large-scale and heterogeneous ones. Furthermore, these systems are exchanging data with each other and therefore their analysis should not be done in isolation alone. Rather, these large-scale, inter-communicating systems should be analyzed as part of a software landscape. Currently, as described in Section 4, this is typically done manually, leading to serious disadvantages.

Scalable solutions for the analysis of heterogeneous systems should be offered to enhance the understanding of large systems and landscapes. To deal with the complexity, we think that up-to-date architecture descriptions on different levels of abstraction (inter-systems, inter-components, inter-layers, inter-classes, etc) and from different perspectives (structural or behavioral) should be provided.

Targeted benefits. If descriptions of the architectural structure and behavior were available on more abstraction levels, the architect could adjust the level of complexity he currently wants to deal with according to his current needs. From an understandability point of view, once the architect comprehends the higher abstraction levels, he will then be able to drill down and require more details that further complete his picture of the system. The reconstruction of the "interplay" of heterogeneous systems would also bring important advantages, unburdening architects from manually documenting this and keeping the documentation synchronized with the real systems.

State of the art status. We are not the first to notice that scalability and heterogeneity are two crucial aspects that need to be dealt with when employing software architecture reconstruction (e.g., [4]). However, this problem still persists. Reconstruction tools still focus on homogenous systems and descriptions on different levels of abstraction are usually available for the static view alone.

7. VISION

We consider that expressing and using a company-specific common terminology is central for enabling the use of architecture reconstruction approaches in the industry. Thus, the architects should be able to adjust the output of the reconstruction tools (expressed in an "intermediate terminology" after having been extracted via technology-specific adapters that address the problem of **heterogeneity**) and to make it correspond to the architecture meta-model of their interest. Having created a common terminology basis, we foresee the shift from monolithic reconstruction tools to more modular **toolbox**-like approaches. Modern techniques should be employed to allow the easy, on-demand addition and construction of plugins that perform various services based on the common terminology while ensuring the preservation of **traceability** links between the created artifacts. E.g., based on the reconstructed views, metric plugins can be added to compute **structural and behavioral metrics** and/or to **allow the definition and assessment of variants**. Last but not least, to deal with the **scalability** issue, abstraction levels can be defined according to the meta-model underlying the intermediate or common terminology and then use these to better structure the output of the other employed plugins.

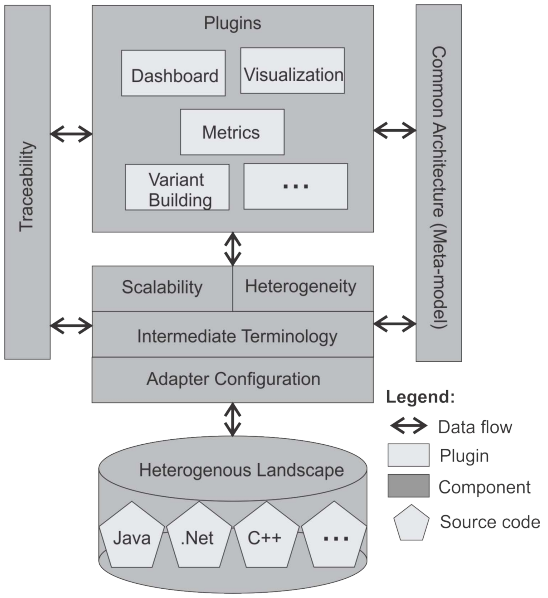

Figure 2: Vision Overview

8. CONCLUSION

In this paper we have presented six important improvement potentials that should be addressed by software architecture reconstruction approaches in order to answer the real needs of the industry. We have identified these potentials based on our experience with two real-world software companies. We have then sketched a vision regarding how these improvement potentials should be adopted to create a reasonable, industry-oriented software architecture reconstruction toolbox.

9. REFERENCES

[1] The STAN Reconstruction Tool. `http://stan4j.com`.

[2] Sonargraph Architect. `https://www.hello2morrow.com/products/sonargraph/architect`, 2013.

[3] G. Buchgeher and R. Weinreich. Connecting architecture and implementation. In *Proceedings of OTM Workshops*, volume 5872, pages 316–326, November 2009.

[4] S. Ducasse and D. Pollet. Software architecture reconstruction: A process-oriented taxonomy. In *Proceedings of IEEE Transactions on Software Engineering*, volume 35, pages 573–591, July 2009.

[5] M. Lindvall and D. Muthig. Bridging the software architecture gap. In *Proceedings of Journal of IEEE Computer*, volume 41, pages 98–101, June 2008.

[6] M. Shahin, P. Liang, and M. Khayyambashi. Architectural design decision: existing models and tools. In *Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, pages 293–296, September 2009.

[7] A. van Hoorn, J. Waller, and W. Hasselbring. Kieker: A framework for application performance monitoring and dynamic software analysis. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE 2012)*, pages 247–248, April 2012.