Mitigating the Risk of Software Change in Practice

Retrospective on More Than 50 Architecture Evaluations in Industry (Keynote Paper)

Jens Knodel, Matthias Naab
Fraunhofer IESE
Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany
{jens.knodel, matthias.naab}@iese.fraunhofer.de

***Abstract*—Architecture evaluation has become a mature instrument to make decisions about software systems, assess and mitigate risks, and to identify ways for improvement and migration of software systems. While scientific literature on approaches is available, publications on practical experiences are rather limited. In this paper, we share our experiences - after having performed more than 50 architecture evaluations for industrial customers in the last decade. We compiled facts and consolidate our findings about architecture evaluations in industry and especially highlight the role of reverse engineering in these projects. We share our lessons learned and provide data on common believes and provide examples for common misconceptions on the power of reverse engineering. This industrial and practical perspective allows practitioners to benefit from our experience in their daily architecture work and how to guide research of the scientific community.**

***Keywords*—architecture evaluation, empirical evidences, experience report**

I. Introduction

Software architecture evaluation is a powerful means to make decisions about software systems, assess and mitigate risks, and to identify ways for improvement and migration of software systems. Architecture evaluation achieves these goals by predicting properties of software systems before they have been built or by answering questions about existing systems. Architecture evaluation is both effective and efficient: effective as it is based on abstractions of the system under evaluation and efficient as it can always focus only on those facts that are relevant to answer the questions at hand. From 2004 to 2013, we conducted more than 50 architecture evaluations for industrial customers at Fraunhofer IESE (an applied research institute for software engineering located in Kaiserslautern, Germany). These projects covered a large number of types of systems, industries involved, evaluation questions asked, and of course a whole spectrum of different evaluation results.

The contribution of this paper is to present our experiences together with context factors, empirical data, and lessons learned. By this we aim at complementing the methodical publications on architecture evaluation. Of course, the companies and systems under evaluation have been anonymized. Target audience for this experience report are both, practitioners and researchers. On the one hand, we aim at encouraging practitioners to conduct architecture evaluations by showing their impact and at lowering the hurdles to make first attempts on their own. On the other hand, we aim at giving researchers insight in industrial architecture evaluations, which can serve as basis to guide research in this area. In particular, we focus on aspects concerning architecture reconstruction and reverse engineering.

II. Our Architecture Evaluation Approach

Our architecture evaluation approach RATE (Rapid ArchiTecture Evaluation) has been developed, refined calibrated, and, of course, applied for more than 10 years now. It is a compilation and calibration of existing approaches. This is following the philosophy of Fraunhofer to enhance, scale and tailor existing methods for industrial application. The evolution of RATE is always driven by the projects with industrial customers. We only briefly sketch the method as this paper is mainly on the experiences about architecture evaluations.

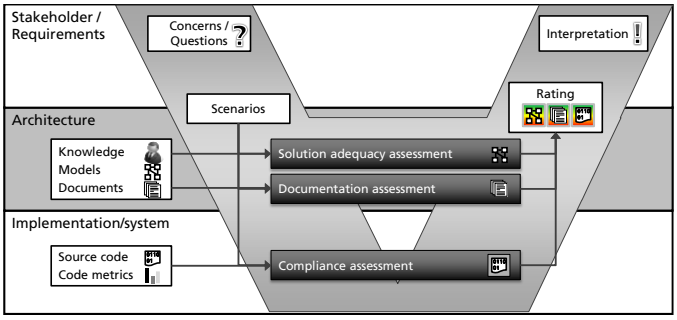

Figure 1: Architecture evaluation approach RATE

III. Context

System Types: Different industries with different types of software-intensive systems were involved in the architecture evaluations. Examples are: airline, agriculture, finance and insurance, automotive, online media and media production, plant engineering, energy management, mobile systems across different industries. This covered both, classical embedded systems and information systems, but also systems spanning both system types. The variety of industries results to a diverse bandwidth of quality challenges, typical architectural solutions, and technologies.

System Size: The size of the systems under evaluation is roughly measured in Lines of Code (LoC). Due to different implementation languages, this is not fully comparable and only indicates rough ranges of size. The size of systems under evaluation spans from around 10 KLoC to around 10 MLoC and a distribution is depicted in Figure 2. We cannot provide

system size for all systems under evaluation. This limitation is for several reasons: Partially, systems have not been implemented and thus size and implementation language had not been clear.

System Age: The age of systems under evaluation also covers a large spectrum and the evaluation took place at different points in the life-cycle of the software systems. A distribution of the age of systems under evaluation is depicted in Figure 2.

Main Implementation Language: The systems under development came with a spectrum of different implementation technologies. The main implementation language has been (in decreasing frequency): Java, C, C++, C#, Delphi, Fortran, Cobol, and also exotic languages like Gen or Camos.

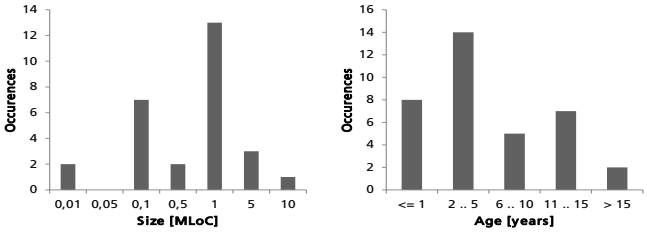

Figure 2: Size and age of evaluated systems

Follow-Up Actions: Architecture evaluations are performed to increase confidence in decision making about the system under evaluation. We did a post-analysis to find out what action items were taken afterwards:

Follow-Up Action		# eval.
COACH	Initiative for coaching architecture capabilities	3
SELECT	Selection of one of the systems / technology	5
REMOVE	Project for removing architecture violations	5
IMPROVE	Improvement of existing architecture	14
NEW	Project to design next generation architecture	5
STOP	Project stopped	3
NONE OK	None (because everything was OK)	11
NOTHING	None (although actions would be necessary)	8

IV. LESSONS LEARNED (SELECTED SAMPLE)

All of our lessons learned have been derived from the practical experiences made in the projects. At least one of the authors has been directly or indirectly involved in each of the projects. We are aware that our lessons learned might not be valid under projects settings with other context factors.

Early and essential architecture design decisions are indeed fundamental. No matter how long the system evolved, the initial description of the architecture still is valid (and used) for communicating the basic ideas and key functions of the systems. This means we can confirm that common believe that architectures stick to their initial ideas for a long time in system life-cycle, at least for the 13 systems aged ten or more years.

Architecting is a first class role during development, but not during maintenance. Over the years, architecting has been established as a first class role during initial development. In many cases, experienced developers are promoted to be responsible for architecting. However, during maintenance the situation is different: no architects are available to review the change request or design solutions. This leads to a drift between architecture and implementation over time and confirms the fact of architecture erosion.

Some architecture problems can't be fixed (easily). Problems like high degree of incompliance in code or strong degree of degeneration of architecture over time show a systemic misunderstanding of the architectural concepts among architects and developers and would require enormous effort to remove. In our evaluations, we had just one case, where such effort was actually spent without any other action (like improvement and coaching). In this case, there was an urgent need to reduce side-effects in the implementation as a change in one place in most cases resulted in a problem in another place. Another problem that is difficult to fix afterwards is the missing thoroughness in definition of initial architecture. This holds especially true for agile development organizations, where the decisions in sprint one are just made, without considering the architectural needs of upcoming sprints.

Source code measurement provides confidence, but its value is overestimated. We experienced several times that measurement programs collecting tons of metrics (e.g., lines of code, cyclomatic complexity) had been established in customer companies. Management was confident to being to control what could be measured. However, the interpretation of the measurement results was most of the time not connected to the architecture. Thus, the measurement results were more or less useless in context of architecture evaluations.

Tool-based reverse engineering often leads to impressive but useless visualizations. Reverse engineering of implementation artifacts is often used in architecture evaluations and partially also in the development process of our customers. We experienced that whenever such reverse engineering activities were not driven by clear evaluation questions, complex and threatening visualizations resulted. Such visualizations serve to increase awareness, but do not serve to guide any improvements.

V. SUMMARY

We strongly believe that architecture evaluations are a sound instrument to increase confidence in decision making and reveal points for architecture improvements. We strongly recommend all practitioners: "Evaluate your Architecture – early and regularly!". Our experiences give evidence that it can be an extremely useful instrument.

Note that this paper presents a condensed version of previous work [1].

REFERENCES

[1] J. Knodel, M. Naab, "Mitigating the Risk of Software Change in Practice - Retrospective on More Than 50 Architecture Evaluations in Industry (Keynote Paper)", IEEE CSMR-18/WCRE-21 Software Evolution Week, 2014.