Software Architecture Evaluation in Practice

Retrospective on more than 50 Architecture Evaluations in Industry

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 sub-discipline in architecting with high-quality practical and scientific literature available. However, publications on industrial applications and concrete experiences are rare. We want to fill this gap and share our experiences - having performed more than 50 architecture evaluations for industrial customers in the last decade. We compiled facts and consolidated our findings about architecture evaluations in industry. In this paper, we provide a critical retrospective on more than 50 projects and share our lessons learned. This industrial and practical perspective allows practitioners to benefit from our experience in their daily architecture work and scientific community to focus their research work on the generalizability of our findings.**

Index Terms—**software architecture, 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 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 for answering the questions at hand. While many publications are available on methods for architecture evaluation, only very little is available on real and practical examples of architecture evaluations (see Section I.A). In this paper, we want to fill this gap and share our experiences from more than 50 architecture evaluation projects with other practitioners and the research community (see Section I.B). Please note that we published an extended version of this paper with a strong focus on software change as an invited keynote at CSMR / WCRE 2014 [1].

A. State of the Art and State of the Practice

Architecture evaluation is, just like software architecture itself, a well-established discipline in research and industry. In research, several methods and many refinements were proposed for architecture evaluation over the last 20 years. The foundations have been published as the first method, SAAM (Software Architecture Analysis Method) [2]. This method introduced a key idea, which is underlying most architecture evaluation methods: requirements, in particular quality attributes, are made concrete and quantifiable with so-called architecture scenarios, which are collected from stakeholders. These scenarios are then discussed with architects in order to identify risks, sensitivity points, and tradeoffs. SAAM has been refined into ATAM (Architecture Tradeoff and Analysis Method) [3], which is the de-facto standard for architecture evaluations, published by the SEI (Software Engineering Institute). While these methods are general-purpose methods applicable to all types of systems and requirements, more recent architecture evaluation methods have focused on certain quality attributes. For example, ALMA (Architecture Level Modifiability Analysis) [4] focuses on modifiability / maintainability. With this restriction, more guidance is possible in the method. Methods like Palladio [5] take this idea one step further and provide both the requirements and the respective aspects of the architecture as formal models and thus allow formal predictions and simulations. More comprehensive overviews of architecture evaluation methods can be found in [6] and [7].

While the previous methods focused on the question of how adequate an architecture is for a certain set of requirements, architecture evaluation can also cover further aspects. Only if an architecture is also implemented consistently, it allows to achieve the requirements as intended. Thus, reverse engineering [8], architecture reconstruction, and architecture compliance checking approaches models [9], [10] are other important aspects of architecture evaluation. A comprehensive overview of architecture reconstruction can be found in [11].

In industry, architecture evaluation is used in many domains to mitigate risks and make well-informed decisions about software systems. However, it is not (yet) common practice. A study on architecture evaluation in practice is reported in [12]. Our experiences confirm many of the aspects reported in this article. Architecture evaluations in industry are conducted by external entities (e.g., consulting companies, research institutes, universities) or by other internal divisions within the company (e.g., quality assurance groups).

B. Goals of this Paper and Contribution

From 2004 to 2013, we conducted more than 50 architecture evaluations for industry customers at Fraunhofer IESE (an applied research institute for software engineering located in Kaiserslautern, Germany), where at least one of the authors has been directly or indirectly involved. These projects covered a large number of different types of systems, partially evaluating more than one architecture, of industries involved, of 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. This is how we intent to complement the methodical publications on architecture evaluation. Of course, the companies and systems under evaluation have been anonymized. The 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 lowering the hurdles to making first attempts on their own. On the other hand, we aim at giving researchers insight into industrial architecture evaluations, which can serve as a basis to guide research in this area.

In Section II we start with a brief overview of our approach to architecture evaluation. Then, we sketch in Section III the context of the architecture evaluations we conducted and outline in Section IV how the evaluation projects were initiated and set up. In Section V, we present an overview of the results and of the follow-up actions. Finally, we conclude with lessons learned in Section VI and an overall discussion in Section VII.

II. Our Architecture Evaluation Approach

Our architecture evaluation approach RATE (Rapid ArchiTecture Evaluation) has been developed, refined, and, of course, applied for more than 10 years now. It is a compilation and calibration of existing approaches. This is in line with the philosophy of Fraunhofer to enhance, scale, and tailor existing methods for industrial application. We only briefly sketch the method here as this paper is mainly about our experiences about architecture evaluations (refer to [13] for more details).

The big picture and main building blocks of RATE are sketched in Figure 1. Architecture evaluations always originate in evaluation goals and questions. According to these goals and questions, stakeholder concerns are identified with stakeholders in interviews and workshops and formulated as architecture scenarios (as in ATAM). RATE consists of several evaluation aspects, which are briefly explained in the following.

Solution Adequacy Assessment is the first key evaluation aspect, which checks how adequate an architecture is with respect to the requirements stated as scenarios. Solution Adequacy Assessment is done similarly as proposed by ATAM in discussions with architects and other stakeholders to identify risks, sensitivity points, tradeoffs, strengths, and weaknesses. The key approach is to discuss architectural solutions for a certain scenario to a level that gives the evaluators enough confidence that the scenario is adequately addressed or the relevant open questions and risks are identified. Sometimes, discussion is not enough to obtain this confidence: For example, if there are strict performance requirements and architectural solutions including technologies that are not thoroughly known, another way of gaining confidence is necessary. This could be collecting data and building more formal models for simulation or building architectural prototypes to collect the necessary experiences.

Documentation Assessment checks how well the architecture documentation suits the needs in the development process. This does, for example, take into account which stakeholders are intended consumers of the architecture documentation, how well it can serve its main purposes, how readable it is, and how up-to-date it is. Thus, the architecture scenarios are input to the documentation assessment, too.

Compliance Assessment checks the compliance of an implementation (with its so-called implemented architecture) with the intended architecture. Only if architectural concepts are implemented compliantly, the architecture has value as a predictive and descriptive instrument in the development process. Compliance Assessment typically requires reverse engineering activities to collect facts about the system, mainly from the source code but sometimes also from the running system. Then the implemented and the intended architecture have to be mapped to each other, and the compliance can finally be evaluated. This is a task that strongly benefits from tool support due to the large amount of facts typically extracted from the source code. We use SAVE (Software Architecture Visualization and Evaluation, see [16]) as our main tool for compliance assessment. It is important to note that not all architectural aspects can be checked for compliance with standard tool support. Typically, aspects regarding the development time structure that can be statically extracted from the source code are more suited for automated checks than runtime structures or data handling. If necessary and economically feasible, we construct additional checks.

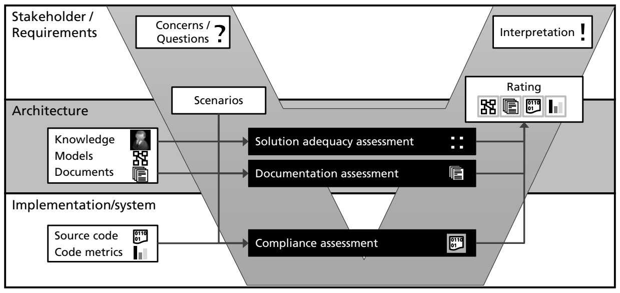

Figure 1: Architecture evaluation approach RATE

As Figure 1 shows, the individual aspects of our architecture evaluation approach are strongly connected in order to complement each other. First, it is made sure that architectural requirements are available and well described. Then, it is assessed if the architecture is adequate for the requirements and if it is documented appropriately. Then it is assessed how compliantly the architecture is realized in the code. This gives us full traceability from requirements to the code, all connected via the architecture. Sometimes, we also complement our architecture evaluations also with assessments of code quality in order to make sure, for example, sure that unreadable code does not corrupt architectural concepts for maintainability.

Of course, not every architecture evaluation contains all these evaluation aspects. Thus, each evaluation project starts with a tailoring phase, whose most important inputs are the evaluation goals and questions. Additionally, the current system state (e.g. just under development, no implementation exists / in operation for decades and needs migration) plays a major role. Further, the required confidence of the evaluation results is necessary to determine which concrete evaluation methods have to be chosen. Based on these inputs, the relevant evaluation aspects and the respective methods of RATE are selected and the evaluation project is planned. To minimize the effort needed for the evaluation, it is always conducted along the evaluation questions formalized as architecture scenarios. All subsequent activities like re-documentation or compliance checking are not conducted in full breadth but only for parts relevant to the current scenario at hand.

The interpretation of the results is crucial for benefiting from the evaluation results. This interpretation is typically not easy and requires a lot of experience. Additionally, the presentation of results and recommendations to senior management has to be done carefully to move development activities into the right direction. In order to be able to give understandable presentations, we decided to depict the outcome using traffic light colors. This is done for all types of assessments and different levels of detail: e.g. to show the adequacy of the architecture for single scenarios but also aggregated for complete attributes. Abstractly spoken, the interpretation is as follows: Green: everything alright, maybe some minor concerns. Yellow: some concerns or risks, but we expect that they can be removed with modest effort. Red: major concerns that will cause serious effort for repair. Of course, there is always a very detailed and differentiated explanation in the rating, too.

RATE is a flexible approach that can cope with all of the project situations we have experienced so far. A key ingredient is not to dogmatically require certain documents or artifacts. If we had only done those evaluation projects where the architecture documentation provided by the customer was up-to-date and adequate for assessing the architecture, we would have rarely done any project at all. Instead, we always try to compensate for missing documents and artifacts by interviewing stakeholders and architects and / or by using reverse engineering techniques. Tool support for architecture evaluations always becomes highly desirable if the implementation is involved, as the source code is typically huge and not suited for manual analysis. While tools that process the implementation can mainly deliver development time related facts, sometimes facts about the runtime are needed in order to have a sound basis for decisions. Then, instrumentation of the source code and the collection of runtime traces are necessary. Afterwards, processing of the collected data is necessary; in particular the needed architectural abstractions must be built. Further tool support is needed for simulations: Simulations can be used to get insights about specific quality attributes, such as performance with different usage profiles.

III. Context of Architecture Evaluations

Our evaluation projects have taken place in a large variety of contexts. Thus, we briefly characterize the spectrum here.

System Types: Different industries with different types of software-intensive systems were involved in the architecture evaluations. Examples are: airlines, agriculture, finance and insurance, automotive, online media and media production, plant engineering, energy management, and 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 resulted in a wide range of quality challenges, typical architectural solutions, and technologies.

Locations: Our customers and their development units are located in different countries: Finland, France, Germany (with half of the projects), Hungary, India Japan, South Korea, and United States.

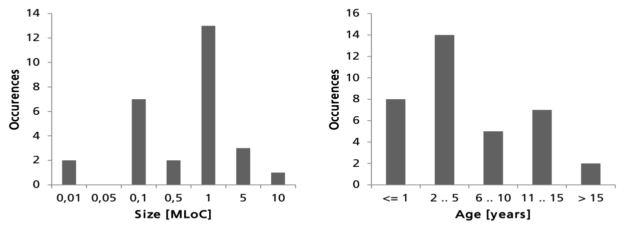

Figure 2: Size and age of evaluated systems

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 ranges from around 10 KLoC to around 10 MLoC; the distribution is depicted in Figure 2. We cannot provide system size for all systems under evaluation. This limitation is due to for several reasons: In part the systems had not been implemented and thus size and implementation language had not been clear. In other cases, we did not analyze the source code and thus had no access to its size figures.

System Age: The age of the systems under evaluation also covers a large spectrum and the evaluations took place at different points in the lifecycle of the software systems. Several systems were evaluated very early in their lifecycle during development, partly before they had been implemented. Other systems were evaluated in their early years because problems were detected after delivery of the systems. Another important point in time is after about 10 to 15 years, when major rework, like modernization of technologies or substantial reduction of technical debt, becomes necessary to keep a system alive and successful. Finally, we also had systems under evaluation with far more than 15 years, sometimes 20 and even 30 years old, which are still operated and undergo continuous maintenance. Often, the question comes up how long such a system can be operated into the future and how retirement and migration strategies could look like. A distribution of the age of the systems under evaluation is depicted in Figure 2.

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

IV. Setup of Architecture Evaluations

Due to the large diversity, we provide an overview of who initiated the evaluations, in which situations they took place, what the evaluation goals were, and how the evaluation projects were set up.

A. *Architecture Evaluation Owners*

Our architecture evaluations were initiated and owned by very distinct stakeholder groups with different purposes in mind. A key distinction is whether the owner is in the same company that develops the system under evaluation or in another company. In the same company, different groups can have an interest in architecture evaluation, as depicted in Figure 3. In another company, typically the customers of a system initiate an architecture evaluation. Either they are already customers and want to assess certain risks or identify reasons for problems, or they are potential customers and want to avoid risks before investment.

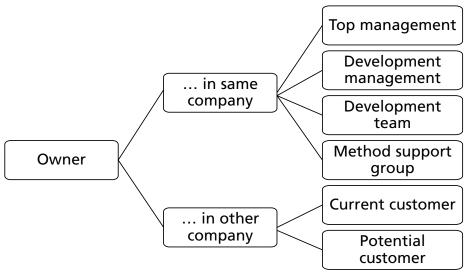

Figure 3: Architecture evaluation owners

B. *Initial Situations*

The situations of the systems in which the architecture evaluation took place were very diverse. We classified the evaluation projects along two dimensions, as depicted in Figure 4: "How critical was the situation?" and "Was the goal only an evaluation or also a direct improvement?" Each of the resulting areas has a name that indicates the project type. *Clash* and *Emergency* are distinguished: Clash denotes a situation where several companies are stakeholders of the evaluation, typically a customer and a provider of the system, with an unclear situation about the system's quality and a high potential of latent or current conflicts. In a similar way, emergency characterizes critical situations, but only within an organization. Figure 4 further depicts the rough number of projects conducted in each of the categories.

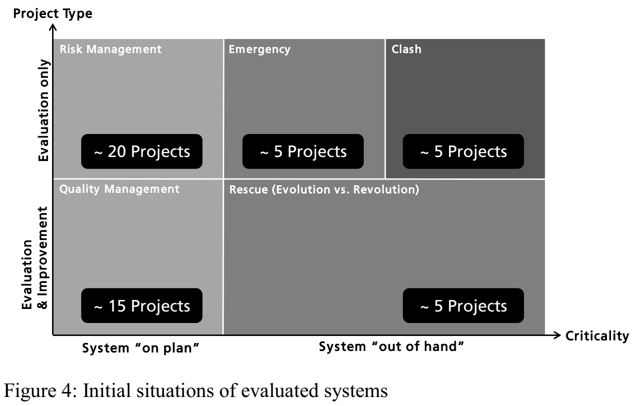

Figure 4: Initial situations of evaluated systems

C. *Evaluation Goals and Questions*

Similar to the diversity in the initial situations, there was also a strong diversity in the evaluation goals and questions. The following list gives an excerpt of typical and recurring evaluation goals and questions:

- How adequate is our architecture as a basis for our future product portfolio?
- Which framework / technology fits our needs best?
- How can we improve performance / maintainability / …?
- How can our system be modularized to meet new business goals like separate selling?
- What is the overall quality of our system and should we maintain it or develop from scratch?
- How adequate is the architecture just designed to meet our key requirements?
- How can we modernize our system to meet new requirements and use modern technologies?

D. *Evaluation Project Settings*

Depending on the diversity of the initial situations and the evaluation goals, the resulting evaluation projects also exhibit strong diversity. We want to give some basic figures to allow better understanding of what these projects looked like.

People involved: Fraunhofer IESE typically conducts architecture evaluation projects with two people. In situations with very large projects, the number was also higher. The organization that develops the system under evaluation was involved ranged between 1 and 30, with a typical number of 8 stakeholders. If there was also a customer company involved, it was typically involved with up to 15 stakeholders, too.

Effort spent: Fraunhofer IESE spent between 4 and 200 person-days on the architecture evaluations. Most projects were conducted with 20 to 80 person-days; the others can be considered as outliers. Companies that had their system's architecture evaluated, spent between 2 and 60 person-days.

Key factors driving effort: We identified a number of key factors that influence the effort to be spent in an architecture evaluation project.

- Number and type of evaluation questions
- Number of stakeholders to be involved
- Organizational complexity
- System size and complexity
- Criticality of the situation
- Need for fast results
- Required confidence and details of results

V. Facts about Architecture Evaluations

In our architecture evaluation projects, varying parts of our architecture evaluation method have been applied, depending on the initial situation and the evaluation questions of the customer. In this section, we summarize and characterize key results according to different aspects of architecture evaluation. Not all evaluation projects involved solution adequacy assessments and compliance assessments. The concrete numbers of applications are presented. Some of our evaluation projects are quick walkthroughs, which provided some first evidence but are not counted. Finally, we present an overview of the follow-ups of the evaluation projects.

A. Results on Architectural Requirements

Sound knowledge of architectural requirements is a fundamental prerequisite for an architecture evaluation. There was not a single project among the ones under evaluation that had a documentation of architectural requirements that could be used as the basis for a solution adequacy assessment. Thus, in all relevant projects we had to compensate for identification and documentation of architectural requirements, at least to some extent. In the context of systems that had been developed a decade and more ago, it is less astonishing that no complete and up-to-date documentation of architectural requirements exists. However, one might wonder why the architectural requirements are also not known in detail in the context of a new development, when a new architecture is currently being designed.

Architectural scenarios as an established means for documenting architectural requirements were rarely used by the industry customers before we came in to do the evaluation project. The result is that architectural requirements are often not detailed and elaborated enough. In about 10% of the evaluations, we found documentations of architectural requirements that read more or less like this: "Our system has to be fast, maintainable, secure, highly availably and has to provide great user experience". While this is, of course, an extreme case, only rarely can a good level of detail be found that could serve as a profound basis for discussion about architectural solutions. In particular, aspects like the concrete stimulus, environment, and a concrete system response and its measure are often missing.

In stakeholder workshops and interviews, we identified and prioritized the relevant architectural requirements as a basis for the architecture evaluations. In typical projects, 10 to 20 stakeholders contributed architectural requirements. This approach typically works well and with some guidance, stakeholders can contribute architectural requirements at an adequate level of detail. We always document architectural requirements as architectural scenarios in evaluation projects. We can observe some differences in the knowledge about different types of quality attributes.

Runtime quality attributes like performance or availability are typically known best. As they are observable in the running system, stakeholders have the best understanding of these quality attributes. However, differences can also be observed between systems that have been in operation for a long time and systems that are c under development. For systems already being operated, stakeholders can typically name the requirements that are critical or not fulfilled very well. Requirements that are fulfilled well become hygiene factors and are not recognized in that much detail. For systems that are currently under development, the known level of detail is typically lower but rather homogeneous.

Development time quality attributes are often not known so well. Only technical stakeholders have a clear understanding: management people sometimes have an abstract understanding that the maintainability or testability has to be high. In general, it can be observed that development time quality attributes are much more difficult to quantify.

Operation time quality attributes are another type we consider explicitly here. These are quality attributes that deal with a system being operated. These quality attributes (e.g., how a system is monitored or updated) are often neglected and also not known in detail. We experienced that in mature organizations operation departments are recognized as stakeholders, but they are often neglected and would not even be invited to the workshops for requirements collection.

B. Results of Solution Adequacy Assessment

In 34 architecture evaluation projects, we conducted a solution adequacy assessment. In two other projects, a solution adequacy assessment was ordered but there was so little information and input that no solution adequacy assessment could be performed. 19 of the projects covered the evaluation of a broad range of quality attributes, which were expressed as architecture scenarios. The average number of elicited architecture scenarios was 33, with a minimum of 7 scenarios and a maximum of 82 scenarios. Out of these elicited scenarios, an average of 18 scenarios were evaluated in detail; they were selected according to the prioritization of the stakeholders. In terms of the overall rating of solution adequacy, we found the following distribution across the projects:

Rating	**# eval.**
Green Everything alright with only some minor concerns → for the elicited architecture scenarios, architects can explain with confidence how the architecture addresses the requirements with design decisions. No major risks are identified and no important questions are left open.	17
Yellow Some concerns or risks that can be removed with modest effort → for the elicited architecture scenarios, architects can explain with confidence how the architecture addresses the requirements with design decisions. Some risks or open questions have been identified, but no show stoppers. With modest rework, the problems can be solved.	11
Red Major concerns or risks that cause serious effort for repair → there are architecture scenarios that are not or not sufficiently addressed by architecture decisions. Overall, the architecture is in a state that requires major rework.	6

In the category of projects with **green** solution adequacy, we see many systems (13 out of 17) aged between 0 and 5 years, that were thoroughly designed. The reason for the evaluation was typically *Risk Management* and *Quality Management* (see categories in Figure 4). Additionally, there are 4 projects aged 8 to 20 years, which were well maintained with continuous effort spent on architectural improvements.

In the category of projects with **yellow** solution adequacy, we see 4 out of 11 projects aged between 10 and 15 years that have undergone some serious change in architectural requirements, which are not fully reflected in the systems' current architecture. Typically, this situation was recognized by the architecture evaluation owner and the evaluation was intended as the start of improvement. Additionally, there are 7 projects between 2 and 5 years that show some deficiencies in the initial architecture definition.

In the category of projects with **red** solution adequacy, we see 3 out of 6 projects that are in an early stage of the product lifecycle, 2 even before system delivery. It was recognized or suspected by the owner of the architecture evaluation that there could be serious problems. However, the development team did not agree and the results had to be delivered by the architecture evaluation project. 2 other projects had come into a difficult development situation after years of opportunistic development without much care about the architecture.

Another aspect was remarkable in the solution adequacy assessments: We found that, on average, there is a much stronger focus on the technical and infrastructure aspects of an architecture than on the business aspects. That is, we found projects that spent nearly their whole effort on the specification of technical styles in the architecture and on the profound selection of technologies like Enterprise Service Buses (ESB). On the other hand, they neglected the definition of concrete business logic components or at least rules for defining them. Additionally, they neglected how the business logic should be mapped to the technologies they selected. In the solution adequacy assessment workshops several times this led to the situation that architects explained: "That is all covered by the Enterprise Service Bus for us". The 3 projects rated red in their early lifecycle stage all suffered from the problem that selecting a technology was the only key architecting effort.

C. Results of Documentation Assessment

Similar to the architectural requirements, there was no single project, in which we found an architecture documentation that could have served as a basis for the architecture evaluation. This does not mean that there is generally no architecture documentation, but it typically does not cover enough or the right type of information to explain how architectural requirements are supposed to be addressed.

Consequently, we had to recover the current architecture with the help of the customers' architects in the course of the evaluation projects. This typically works well and we met many very knowledgeable architects. As architecture re-documentation was necessary to a larger or smaller extent in all architecture evaluation projects, our customers got, besides the benefits of the evaluation insights, another advantage: They got a starting point of explicit documentation with a lot of relevant information that had only been known implicitly before.

In nearly all evaluation projects we found that architecture documentation needs strong improvement. In about half of the projects, no architecture documentation was available at all. In the other projects, the documentation was mostly not up-to-date, not adequate for specific usages, and did not contain enough information.

D. Results of Compliance Assessment

In 26 architecture evaluation projects, we conducted a compliance assessment. This number is smaller than that of the solution adequacy assessments as not all customers ordered a compliance assessment and some projects were so early in the lifecycle that no implementation existed yet.

In terms of the overall rating of compliance between intended and implemented architecture, we found the following distribution across the projects:

Rating	# eval.
Green Everything alright with only some minor concerns → no or only minor deviations between intended and implemented architecture are found.	10
Yellow Some concerns or risks that can be removed with modest effort → overall, the implemented architecture is compliant to the intended architecture. However, some deviations have been identified that should be removed and will cause modest effort.	7
Red Major concerns or risks that cause serious effort for repair → overall, there are major deficiencies in the compliance between intended and implemented architecture. There is a large number of architecture violations or those found are very critical, or both. It is expected that repairing the implementation will entail significant effort.	9

For architecture violations identified with tool support, a thorough analysis, categorization, and interpretation is necessary. Sometimes, the architecture violations are very systematic and can even be an indication that the intended architecture should be adapted. Sometimes, the architecture violations have only minimal impact and can even be repaired with automated refactoring. However, there are as well architecture violations that are highly critical in terms of their negative impact on software quality and in terms of their removal costs.

In the category of projects with **green** compliance, we had projects that typically came with good or even very good overall quality. In these companies, there was high awareness for quality. Nevertheless, architecture compliance was mostly not enforced with manual inspections or even tool support.

In the category of projects with **yellow** compliance, the projects are very diverse and no clear characteristics can be observed.

In the category of projects with **red** compliance, we see 5 out of 9 systems that are aged between 7 and 15 years. It is clearly visible that during maintenance and evolution, more and more architecture violations are introduced. The results in these cases were several ten thousands of architecture violations, which can only be repaired with extremely high effort. Additionally, there were systems of younger age, which also suffered from bad compliance. These cases mainly resulted from ill-defined or badly understood architectural concepts so that even during initial development, a high number of architecture violations were introduced. When there was such low architecture compliance, the overall quality of the software was adversely influenced. In most cases, even the users perceived problems, e.g. in the form of insufficient availability or performance. This was because the architecture violations corrupted key architectural concepts.

E. Follow-ups on Evaluation Projects

Architecture evaluations are performed to increase confidence regarding decision-making about the system under evaluation. After presenting the outcome in a final meeting, our customers had to decide what to do next. We did a post-mortem analysis to find out what actions were taken afterwards. We found the following distribution of action item categories performed by the customers (please note that multiple actions were possible):

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

In the category COACH, an initiative for training and improvement of architecture capabilities in the organization was started. Coaching was never done in isolation, in one case it was performed together with an improvement of the existing architecture, and in 2 cases it was performed together with the design of the new next generation architecture and a dedicated project for removing architecture violations.

In the category SELECT, one of the candidate systems / technologies being evaluated was actually selected. In 5 cases the architecture evaluation provided valuable input to decision-making and management found itself confident to select one winner out of the alternatives offered.

In the category REMOVE, we observed the definition of an explicit project for removing architecture violations. This happened in 3 cases where significant amounts of time and effort (e.g., 1 team working for 6 months) were spent on the removal of architecture violations (i.e., changing the code to remove violations by refactoring or re-implementing).

In the category IMPROVE, we dedicated effort was spent on improving the existing architecture. This was in particularly true for projects in an early stage (without an implementation or at the beginning of development). At this point in time, corrective decisions regarding the design could be integrated in the development process. In one case, the improvement took place on both levels, the architecture and the implementation. On the one hand, the architecture was changed to accommodate the findings of the architecture evaluation, and on the other hand, significant numbers of architectural violations in the implementation were removed at the same time.

In the category NEW, we list projects where the need for a complete redesign of the architecture has been accepted and decided. In 5 cases, instead of improving the existing architecture, the findings and insights on the existing systems served as input to the design of a new architecture (followed by a re-implementation from scratch). Although conceptual reuse and very limited code reuse took place, fundamental design decisions were made in the light of the findings from the architecture evaluation.

In the category STOP, we had 3 cases where all engineering activities were canceled and the product development (or the next release) was stopped completely. Management was convinced that achieving adequate quality was no longer possible with reasonable effort in reasonable time. In these cases, the implementation exhibited such severe flaws in the architecture and a high number of violations in the implementation that it was decided, it would be better to stop altogether.

In the category NONE OK, we listed the 11 cases where nothing was done, because the architecture evaluation basically did not reveal any severe findings. There were only minor points (if at all), which could be addressed as part of regular development

In the last category NOTHING, we have 8 cases where nothing happened after the evaluations – although the results revealed many action items, either in the architecture or in the implementation. We have explicit confirmation by the customers that – in fact – nothing happened afterwards (although it might be that we were not informed on purpose). The need was recognized and acknowledged, but no budget or time was allocated to actually doing something about it.

VI. LESSONS LEARNED

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 of the limitations that our lessons learned might not be valid in projects settings with other context factors. And of course, we do not claim generalizability. Nevertheless, we perceive each single lesson learned as a valuable piece of experience that might help other researchers and practitioners to avoid pitfalls and facilitate their own evaluations. We explicitly encourage practitioners to complement our experiences with their experiences. Further, we aim at inspiring other researcher in investigating the generalizability of our lessons learned.

A. What We Learned about Architectures in Practice

Early and essential architecture design decisions are indeed fundamental. No matter how long the system has 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 the common belief that architectures stick to their initial ideas for a long time in the system lifecycle, at least for the 13 systems aged ten or more years. In these cases, the fundamental decisions were really fundamental. In some cases, it was even attempted to change these core design concepts of the system, but more often than not these projects failed or got canceled because they exceeded the allocated time and/or efforts significantly.

Architectural diagrams are helpful, but require explanations. Diagrams often lack textual descriptions. They show the manifestation of design decisions, and almost never document the decisions and their rationales leading to the manifestation. This is something we experienced quite often in our projects. Even after spending hours questioning the team of architects about the content of a diagram, nobody bothered with updating the descriptions of the diagrams. Although the need became obvious during the discussion, the diagrams remained without explanations.

Standard templates are frequently used but often misunderstood. Architecture documentation templates exist in many industrial companies, either standard templates published in literature or custom templates defined internally. Many architects stick to the templates for compliance reasons but misunderstand or misuse the rules and policies of the template. This undermines the purpose of sharing information among different projects by using templates.

B. What We Learned about Architecting in Practice

Architecting is a first-class role during development, but not during maintenance. Over the years, architecting has become 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 change requests or design solutions. Over time, this, leads to a drift between architecture and implementation and confirms the fact of architecture erosion.

Development becomes agile, but architecting in sprints only is not enough. Reviewing the past decade of architecture evaluation projects, we can see that more and more of our customers have adopted agile development processes. Architecting has to be "re-defined" or "re-invented" in the context of more and more companies "going agile". The main point we observed is that if architecting is performed in the scope of the current sprint only, it does not help to solve problems that arise across individual sprints, across teams, and in distributed development. This holds especially true for quality requirements, which cannot be solved in one sprint only.

C. What We Learned about Evaluations

Some architecture problems can be fixed easily. In our experience, problems like missing documentation or missing support for several new scenarios can be fixed as long as the basic knowledge about the system is up-to-date and implicitly known by the architects. The same holds for minor incompliance in the code, which typically can be addressed in the next iteration of development.

Some architecture problems can't be fixed (easily). Problems like major incompliance in the code or a strong degree of degeneration of the 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 a missing thoroughness in definition of the 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.

Better an end with terror than terror without an end. Some architecture problems could be fixed, but they were not addressed. Missing commitment of management in the fixing phase leads to the 8 cases where the need of fighting the architecture problems has been acknowledged but nonetheless no concrete actions were performed. In our opinion, it is better to make a painful break (by investing effort to get the architecture right instead of delivering the next set of features) than to remain in the status quo.

Having one's own tool is beneficial for fast customization of analyses. Having full control over the tool (in our case SAVE) enabled us to tweak the extractors, analyzers, and the visualization of the tool to the evaluation at hand. In many cases, the standard features and capabilities were not sufficient and required case-specific adaptations. Being able to adapt the tooling empowered us to deliver fast results in many projects.

D. What We Learned about Interpretations

No standard interpretation of evaluation results is possible | Interpretation has to consider evaluation questions and context factors. Even when there are quantitative results (e.g., the number of architecture violations), the interpretation of the results remains a difficult but crucial step in architecture evaluations. Due to the nature of software architecture and software architecture evaluation methods, the evaluation results often cannot be fully objective and quantifiable. It is very important for evaluators to manage the expectations of evaluation owners and stakeholders and to clearly communicate this. For instance, it is not possible to establish standard thresholds for the number of acceptable architecture violations. Rather, it is always necessary to keep the goals and the context of the customer in mind to answer the question in a way that is most beneficial for him. Over time, we learned that interpretation comprises the preparation of potential follow-up decisions, and that this was the customer's expectation towards us. Just providing facts and findings is not enough. Today, we consider an architecture evaluation to be successful when at least one of the action items proposed has been implemented or one of the recommendations made leads to a decision.

Source code measurement provides data and 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, most of the time, the interpretation of the measurement results was not connected to the architecture (and thus could not be used as input). Thus, the measurement results were more or less useless in the 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. Thus, our evaluation approach has a strong focus on guiding the evaluation with evaluation questions and architectural requirements.

Representation of evaluation results for management people and non-technical decision makers is challenging. Often, the sponsors of an architecture evaluation are people from senior management. Architectures – even though abstracting from the system under evaluation –are still technical constructs. Presenting the evaluation results to such stakeholders, who typically do not have much of technological background, is very challenging. On the one hand, the results have to be very condensed and easily understandable. Recommendations and alternative ways should be shown and supported with quantitative data. On the other hand, evaluators have to be careful to present subtle differences in an understandable way as these can have a substantial impact on far-reaching decisions. This holds especially true for qualities at development time (e.g., maintainability, reusability). Our traffic-light colored rating was motivated by the need to give clear presentations.

E. What We Learned from More Than 50 Other Evaluations that We Offered but which did Not Take Place[1]

Patient died on the way to the hospital. In some cases, the project was even stopped by management before the architecture evaluation (already under discussion) could take place and potential improvements could have been identified.

Daily workload wins over architecture evaluations. There was often the willingness to do an architecture evaluation project, but time and effort could not be spent (by the way, this is a fact that corresponds to typical architecture practice in these companies as well). Instead of analyzing the root cause, firefighting work was performed at the symptoms.

Rather refactor in the small than challenge your own decisions made in the past. In some cases architects did not dare to question their own decisions from the past. They ignored the possibility that decisions that once were correct in the past might not be correct any more in the present, as the context and the system has evolved.

Plain numbers are preferred over statements and interpretations provided by architecture evaluations. Many customers decided to buy a code metric tool instead of performing an architecture evaluation. We value the capabilities of metrics and continuous measurement, but we doubt their use in deriving answers to any of the typical architecture evaluation questions (as discussed in Section 4.3).

F. What We Learned about Industry

Evaluation results are expected to be delivered immediately. Despite feeling and communicating the pressing need for having an architectural evaluation (or rather having important questions or doubts in decision making), ordering an architecture evaluation projects for some reasons can take up to several months. Once ordered, expectations regarding the delivery results do not accommodate the long waiting time for being allowed to start the evaluation. Industry customers expect evaluation results to be delivered promptly, which is contrary to other architecture-related projects we did in the past (e.g., supporting the design of an architecture or coaching of architecture capabilities).

[1] Note in this section we discuss situations where customers had an interest in an architecture evaluation, but, in the end, the project did not take place.

Stakeholders sometimes try to influence the interpretation to achieve their own goals. Such attempts are not very frequent but they do occur (as the result of architecture evaluations can have significant impact on stakeholders). Being neutral is a key prerequisite for evaluators and, external evaluators are often hired exactly for this reason.

"It depends" is not a good final answer. Although true in many cases, it is important to clearly distinguish and delineate alternatives among follow-up actions. Companies want to know what they can do next. Findings and recommendations ideally should be translated into business terms (gain or loss of money, meeting or failing deadlines, etc.).

Industry likes Word, PowerPoint and UML modeling tools for documenting architectures. In none of our projects were formal architecture description languages used. Most of our customers applied a mix of Word, PowerPoint and UML diagrams to document their architecture (if it was documented at all). This gives evidence that architecture description languages (ADLs) have not (yet) been adopted in industry.

New features kill architecture work. As architecture does not deliver any direct end customer value, it is at risk of being put off or getting dropped. However, the insight that architecting delivers business value to the developing company (if it is going to maintain the software) by preparing the future or making development more efficient is often neglected. Even if customers were made aware of critical issues by us, we had 8 cases where nothing was done afterwards.

Architecting lacks a clear mission in software projects. Our experiences show that architecting typically goes slowly because it lacks a goal-oriented focus. Architects in industry often spend a lot of time on designing templates, evaluating several technologies, modeling and pimping diagrams, but forget the point of architecting: delivering solutions for current and future design problems. As there are always a lot of problems, we think it is important to explicitly focus, plan and track architecture work. For this reason, we proposed the construct of architecture engagement purposes (see [14]), an auxiliary construct to align architecting with other engineering activities in the product lifecycle.

G. What We Learned about Architecting in Research

Academia likes toy examples, but industry requires solutions with practical scalability. Toy examples are great for explaining new ideas and approaches, but do not serve nor scale when adopting approaches in industry. Many major issues of an approach only arise when dealing with problems exceeding a certain scale. Academia in many cases fails to give guidance on how to scale up their approaches and techniques. Moreover, many (junior) researchers in in the field of architecture do not really know the practical challenges and thus, cannot try to solve them. Of course, we do not believe that academia should solve all architecting problems in industry, but publications in research should state more honestly to which level they really scale. More often than not, we perceive that more is claimed than is actually delivered.

Eat your own dog food. The architecture of prototypes and tools for architects (e.g., modeling, analysis, or reconstruction tools) would benefit if researchers and vendors would actually do what they tell others and publish in papers. Many widely accepted concepts (like view-based architecture) could be used for publications on these tools. We fear that if researchers do not use our own best practices, this could undermine the reputation of our field.

H. How Our Evaluation Approach Evolved

Architecture evaluations have to evaluate implicit decisions made in the heads, explicit decisions found in documentation, and manifested decisions in system implementations. The big picture and integration of assessment techniques (as depicted in Figure 1) emerged over time. At Fraunhofer IESE, architecture evaluation research was driven from three directions: reconstruction (see [15]), tool development [16], and literature about architecture in general (e.g. [17], [3], [18], to name but a few). In our first evaluation projects, we started with coarse-grained recovery and reconstruction. Over time, we learned that customers rather require one concrete answer to one current, urgent, and pressing question (where architecture evaluation may or may not be the means to answer their question). They do not care about exploiting the power of reconstruction for other parts. This resulted in our request-driven reverse engineering approach where we always have concrete stakeholder scenarios guiding all subsequent analysis. Hence, this manifests the need to evaluate implicit decisions in architects' minds, explicit decision described in documentation, and last but not least, the source code and the running system.

All architecture evaluation is not the same. We are covering more and more initial situations in which architecture evaluation can provide benefits. Furthermore, in any new projects, we learn about something that is different than before. Every software system is unique in its characteristics, lifecycle, and context including the people and organization behind the system, and the same holds true for the evaluation of the system. In this way, architecture evaluations are always interesting, as the evaluators learn about something new in a rather short period of time.

VII. Conclusions

With this paper, we strongly recommend to all practitioners: "Evaluate your architecture – early and regularly!". Our experiences from more than 50 architecture evaluations give evidence that it can be an extremely useful instrument to support architecture decision making and to guide strategic alignment of business and technologies in a software system.

We will continue to collect further data on architecture evaluations as future projects take place. We hope this experience report complements the existing body of knowledge on architecture evaluation. From a research point of view, we see the need to extend and scale existing methods on how to evaluate systems that are part of interconnected, integrated and complex software ecosystems. This holds also for architecture analysis tools which need to keep pace with new technologies and programming paradigms. With experience reports like this we intend to pave the way towards increased industrial applications of architecture evaluations as an instrument to support decision making.

We would like to acknowledge all colleagues, researchers and practitioners who provided us with inspiration, ideas, discussions and feedback on architecture evaluation.

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.

[2] R. Kazman, L. Bass, M. Webb, G. Abowd, "SAAM: a method for analyzing the properties of software architectures", 16th International Conference on Software Engineering (ICSE), 1994.

[3] P. Clements, R. Kazman, L. Bass, "Evaluating Software Architectures", Addison Wesley, 2001.

[4] P. Bengtsson, N. Lassing, J. Bosch, H. van Vliet, "Architecture-level modifiability analysis (ALMA)", Journal of Systems and Software, vol 69(1-2), pp. 129-147, January 2004.

[5] S. Becker, H. Koziolek, R. Reussner, "The Palladio component model for model-driven performance prediction", Journal of Systems and Software, vol. 82(1), pp. 3-22, January 2009.

[6] L. Dobrica, E. Niemala, "A survey on software architecture analysis methods", IEEE Transactions on Software Engineering, vol 28(7), pp. 638-653, July 2002.

[7] M.A. Babar, I. Gorton, "Comparison of scenario.based software architecture evaluation methods, 11th Asia-Pacific Software Engineering Conference, 2004

[8] E. J. Chikofsky, J. H. Cross II, "Reverse Engineering and Design Recovery: A Taxonomy" (Vol. 7, pp. 13-17): IEEE Computer Society Press, 1990.

[9] G. C.Murphy, D. Notkin, K. J. Sullivan, "Software reflexion models: bridging the gap between design and implementation". IEEE Transactions on Software Engineering, 27(4), 364-380, 2001

[10] J. Knodel, D. Popescu, "A Comparison of Static Architecture Compliance Checking Approaches". Sixth Working IEEE/IFIP Conference on Software Architecture (WICSA), 2007.

[11] D. Pollet, S. Ducasse, et al. "Towards A Process-Oriented Software Architecture Reconstruction Taxonomy". 11th European Conference on Software Maintenance and Reengineering (CSMR) 2007.

[12] M.A. Babar, I. Gorton, "Software Architecture Reviews: The State of the Practice", IEEE Computer, 42(7): pp. 26-32, 2009

[13] J. Knodel, "Rapid ArchiTecture Evaluation (RATE)", IESE-Report; 105.11/E), 2011

[14] T. Keuler, J. Knodel, M. Naab, D. Rost, "Architecture Engagement Purposes: Towards a Framework for Planning "Just Enough"-Architecting in Software Engineering", WICSA/ECSA, 2012

[15] J. Knodel, D. Muthig, "A Decade of Reverse Engineering at Fraunhofer IESE - The Changing Role of Reverse Engineering in Applied Research", 10th Workshop Software Reengineering (WSR), 2008.

[16] J. Knodel, S. Duszynski, M. Lindvall, "SAVE: Software Architecture Visualization and Evaluation" 13th European Conference on Software Maintenance and Reengineering (CSMR 2009), Kaiserslautern, Germany, 2009.

[17] P. Clements, D.. Garlan et al., "Documenting Software Architectures: Views and Beyond". Pearson Education, 2002.

[18] P. Clements, R. Kazman, "Software Architecture in Practices". Addison-Wesley Longman Publishing Co., Inc., 2003