The Journal of Systems and Software 116 (2016) 191–205

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

10 years of software architecture knowledge management: Practice and future

Rafael Capilla [a,*], Anton Jansen [b], Antony Tang [c], Paris Avgeriou [d], Muhammad Ali Babar [e]

[a] Rey Juan Carlos University, Madrid, Spain
[b] Philips Innovation Services, Eindhoven, The Netherlands
[c] Swinburne University of Technology, Melbourne, Australia
[d] University of Groningen, Groningen, The Netherlands
[e] University of Adelaide, Adelaide, Australia

ARTICLE INFO

Article history:
Received 11 October 2014
Revised 29 May 2015
Accepted 14 August 2015
Available online 9 September 2015

Keywords:
Architectural knowledge management
Architectural design decisions
Agile development

ABSTRACT

The importance of architectural knowledge (AK) management for software development has been highlighted over the past ten years, where a significant amount of research has been done. Since the first systems using design rationale in the seventies and eighties to the more modern approaches using AK for designing software architectures, a variety of models, approaches, and research tools have leveraged the interests of researchers and practitioners in AK management (AKM). Capturing, sharing, and using AK has many benefits for software designers and maintainers, but the cost to capture this relevant knowledge hampers a widespread use by software companies. However, as the improvements made over the last decade didn't boost a wider adoption of AKM approaches, there is a need to identify the successes and shortcomings of current AK approaches and know what industry needs from AK. Therefore, as researchers and promoters of many of the AK research tools in the early stages where AK became relevant for the software architecture community, and based on our experience and observations, we provide in this research an informal retrospective analysis of what has been done and the challenges and trends for a future research agenda to promote AK use in modern software development practices.

1. Introduction

The field of Software Architecture has matured over a period of 30 years (Shaw and Clements, 2006, Clements and Shaw, 2009) from the early basic concepts from the mid-80s to the ubiquitous proliferation of the role of a software architect in contemporary industrial practice. During this time-span, there are two prevailing paradigms that represent the essence of software architecture. The initial paradigm was purely technical and examined architecture in terms of system structure and behavior with components and connectors, views, Architecture Description Languages, Architecture Design methods, patterns, reference architectures etc. The subsequent paradigm was socio-technical and considered architecture from the point of view of its stakeholders, looking at how they reason and make decisions. The difference among the two paradigms is simple: the first concerns the end result of architecting, as it culminates in the actual design, while the second (usually referred to as Architecture Knowledge Management) concerns how we essentially reached that end result. The seed ideas for the second paradigm were planted already in the early 90s (Perry and Wolf, 1992, Kruchten, 2004), but the shift essentially started a decade ago (Bosch, 2004), where it is suggested that AK is made up of design decisions and design (Kruchten et al., 2006).

At the core of Architecture Knowledge Management, lies the principle of considering the architect as a decision maker instead of someone 'drawing boxes and lines'. This development was certainly welcomed in the professional circles of software engineering and is aligned with the recognition given to the profession in recent polls (CNN Money[1] listed Software Architect as the 8th top-paying job in 2009 and the best job in 2010). However, it was soon realized that architecture decision making has been more of an art than a craft (van Heesch and Avgeriou, 2011, Tang et al., 2010). The reasoning process of software architects is rather ad-hoc and is not supported by typical software engineering processes and tools. Architects tend to base decisions on their own experiences and expertise, which in most cases

* Corresponding author. Tel.: +34607901188.
E-mail addresses: rafael.capilla@urjc.es (R. Capilla), anton.jansen@philips.com (A. Jansen), atang@swin.edu.au (A. Tang), paris@cs.rug.nl (P. Avgeriou), ali.babar@adelaide.edu.au (M.A. Babar).

[1] http://money.cnn.com/

http://dx.doi.org/10.1016/j.jss.2015.08.054
0164-1212/

are invaluable for providing optimal solutions, but is also prone to biases and fallacies (Kruchten, 2011, Stacy and MacMillan, 1995, Tang, 2011, van Heesch et al., 2013). Architects are not likely to document their decisions and rationale, despite the well-established benefits of doing so. Subsequently consuming architecture decisions and the rest of the AK is problematic as the knowledge is often incomplete and out of date.

The shift from the first to the second paradigm sparked a substantial amount of research to solve the aforementioned problems; and this has been achieved at least to some extent. We have seen a number of meta-models that aim at representing architecture knowledge mostly focused on design decisions and rationale. Several tools have been developed and validated, with the purpose of supporting stakeholders to both produce and consume AK (Tang et al., 2010, Li et al., 2013, Tofan et al., 2014). A few studies have examined the reasoning process of architects aiming at providing process support that can systematize the decision making process (van Heesch et al., 2013). Having reached its first decade of life, it is time to revisit the related ideas and technologies of AKM and examine how far we have come and what are the promising future directions. We attempt to highlight prominent results from the research community but also present an overview from the state of practice in AKM in software industry based on interviews. This paper is built upon the previous survey of AKM tools in 2010 (Tang et al., 2010). In this paper, we updated our survey of AKM methods and tools in recent years to examine new trends and developments. We interviewed industry practitioners to understand recent challenges with regards to the use of AKM. We analyzed the results of the interviews and compared that to the facilities provided by the latest AKM tools. From this analysis, we suggested the trends and development in this field.

The remainder of this paper is as follows. In Section 2, we motivate the need for capturing AK and we highlight the different areas affected by the knowledge capturing problem. Section 3 revisits the recent research over the past ten years and compiles a retrospective view of major AK centric approaches, from the conceptual models to the research tools produced along this period. In Section 4, we discuss the AK challenges and needs of industry using AK through a set of interviews where we have drawn interesting observations of AK usage and the barriers for AK adoption from several software companies. In Section 5, we provide a set of short perspectives and trends for AK use ranging from the sustainability of AK, education, and the role of AK for agile development among others. Finally, Section 6 draws our conclusions from this retrospective analysis along the past ten years using AK in software architecture approaches.

2. The needs for capturing architectural knowledge (AK)

It has been suggested that "if it [an architecture design] is not written down, it does not exist" (Clements et al., 2011), making a point for the importance of architecture documentation. Software architectures are often constructed without documented AK. However, the intricate knowledge of a system, especially in a large and complex system easily evaporates if AK is non-documented. The consequences would be incurring design and implementation issues (Bosch, 2004). If a simple system is built by only one person, and the system is only maintained and inspected by the same person, and that person has perfect memory, the need for capturing AK is probably not there. However, these assumptions do not hold for most of the non-trivial modern-day systems. Many potential issues can arise without AK.

Rittel and Weber observed that developing software is a process of negotiation and deliberation between many stakeholders. Many unknowns arise during this process and therefore they suggested that it is a "wicked problem" (Rittel and Webber, 1973). Shaw and Clement described the coming of age of software architecture (Shaw and Clements, 2006). They outlined the maturation process which includes the development of research tools, internal and external enhancements, the development of usable tools and processes, and the popularization and adaptation of processes and technologies. These developments may have helped but they have not solved issues that require developing and managing AK.

The fundamental elements of AK were described by Perry and Wolf. They stated that software architecture comprises elements, form, rationale (Perry and Wolf, 1992). They emphasized connections between elements as "glue". These glues, as represented by views, allow designers to connect the architecture elements together. A software architecture can be neatly described by common architectural patterns or styles (Harrison et al., 2007, Garlan and Shaw, 1993, Cloutier et al., 2010), and the application of those patterns constitutes some of the most important design decisions (Harrison et al., 2007). Whilst these approaches outlined the essence of architectural knowledge, there was a movement on justifying an architectural design. Approaches like gIBIS (Conklin and Begeman, 1988), DRL (Lee and Lai, 1996), QOC (Maclean et al., 1996) were developed for capturing reasoning in a design. The common elements in these models were design issues, design alternatives, and some form of argumentation. This movement emphasized that designers need to know the end results of a design as well as to know the *intents* and the *rationale* of how a designer arrives at a design. Unfortunately, these models were not widely accepted by practitioners. Conklin and Burgess-Yakemovic explained that as architecture models and their explanations were separate, design rationale can grow into unwieldy of loosely organized textual information that is difficult to use (Conklin and Burgess-Yakemovic, 1996).

To prevent knowledge vaporization and architectural drift, the modeling of decision-centric AK re-emerged with Bosch's suggestion that software architecture design decisions lack a first class representation, and lacking a cross-cutting view (Bosch, 2004). Around the same time, Burge modeled design rationale with SEURAT (Burge, 2005) and Tyree and Akerman showed the advantages of capturing design rationale in their practice (Tyree and Akerman, 2005). A plethora of works on modeling design rationale and different types of software AK had emerged. Many of these works argued the various needs for capturing AK. The main uses for AK are nicely summarized by (de Boer et al., 2007), where the four broad categories of uses of AK are sharing, compliance, discovery and traceability. We now have ample experience of sharing, compliance, discovery and traceability of AK and are able to apply lessons learned in practice, but also to pursue open research problems (Vliet et al., 2009).

2.1. Sharing

Software development is largely a group activity where many people/stakeholders work together and a shared understanding between them is essential (Fischer and Ostwald, 2001). Stakeholders need to have a shared understanding of the goals, the requirements, problems to be solved, the system behavior, how to construct them (i.e. design and implementation), and contexts such as assumptions, constraints, risks, tradeoffs etc. The communication of this knowledge to achieve a common understanding is difficult, especially involving multidisciplinary design (Bonnema, 2014). So capturing this knowledge is necessary (Babar et al., 2007). Creating such shared understanding also alleviates miscommunication and information overload (Fischer and Ostwald, 2001). Perry and Wolf observed that knowledge evaporates over time (Perry and Wolf, 1992). Nonaka and Takeuchi observed that much knowledge is tacit (Nonaka and Takeuchi, 1995). The retention and communication of knowledge are therefore essential when large software systems are maintained over long period of time by many developers. The need for capturing knowledge is important also when software is produced by developers across different geographic areas and communicating is difficult (Dutoit et al., 2001). Due to large number of stakeholders and the dispersion of knowledge, many challenges exist, such as how to share AK and how to reuse AK

effectively (Miksovic and Zimmermann, 2011). Clements et al. summarized this situation clearly: "*It [documentation] speaks for the architect today, when the architect should be doing other things besides answering a hundred questions … It speaks for the architect tomorrow, when he or she has left the project …* " (Clements et al., 2011).

2.2. Compliance

AK is a main source to enable evaluation of architecture compliance and suit for purpose. Without captured AK, it is difficult for reviewers to perform dependency and consistency analysis (Perry and Wolf, 1992). AK enables architectural issues to be found. In an empirical study, eight issue scenarios such as missing requirements, conflicting requirements, conflicts between requirements and design, missing context with profound influence on design were found through analyzing AK (Tang and Lau, 2014). Some systems have long life-spans and AK is required for evaluating changes throughout its operational lifetime (de Boer et al., 2007). Architects may prescribe architectural constraints to a desired level to indicate the desired restrictiveness of an architecture (Perry and Wolf, 1992). Additionally, architecture principles are AK that provides basic guidelines for architecture design under different circumstances. For instance, loose coupling, modularity and information hiding are some of the principles that can be used in a design (Vogel et al., 2011). Such specification draws boundaries for a design and dictates its compliance. AK can help architects identify constraints to evaluate the viability of implementation (Berg et al., 2009). AK supports architecture impact analysis. It allows architects to identify architectural elements that are affected by a change scenario (Li et al., 2013). A software architecture has many invisible qualities and these are not reflected in a single artifact. The compliance of an architecture design to achieving some quality requirements can be exhibited in AK through evaluations and reviews (de Boer et al., 2007).

2.3. Discovery

Architectural design is a continuous discovery process. It requires a designer to discover requirements, contexts, design decisions and project elements concerns (Kruchten et al., 2005). With this information, a design model can be formed by progressing through small design coalition, from individual models to coalition formation, and consensus team model (Curtis et al., 1988). An architecture model is formed through examining different aspects of the architecture views, allowing discovery of new behavior and scenarios through different viewpoints (Perry and Wolf, 1992). Design reasoning plays an important role in software architecture design. It is a discovery process through structuring and representing requirements specifications. Requirements specification consists of rigorous description of different views, e.g., use case view, context view with underlying domain model, which are helpful for understanding requirements (Ding et al., 2014). An understanding of these AK then allows reasoning to happen to uncover new design questions and design alternatives.

2.4. Traceability

A designer needs to know the contexts and the forces that shape a design. This activity requires relationships that are usually not expressly captured, e.g. the relationships between requirements, decisions and implementation. Many have argued that they should be explicitly captured (Ramesh and Jarke, 2001, Han, 2001, IEEE, 1996). There are many advantages for tracing software development artifacts. Traceability supports software understandability, it helps locating relevant AK, enables change impact analysis and facilitates design assessment (Jansen et al., 2009). Traceability can be performed in a forward direction, i.e. tracing the requirements to its implementation or in a backward direction, i.e. tracing the requirements and design that are affected by a software component (Tang et al., 2007). Software artifacts can change over time, the tracing of the evolution of software changes allows designers to understand what have changed overtime, and help them to review and maintain (Haumer et al., 1999).

3. Decision-centric AKM approaches: a retrospective analysis

Many AKM approaches have been developed over the last decade that aim at supporting one or more of the aforementioned general usages of AK (i.e. sharing, compliance, discovery, traceability) in a practical manner. An ongoing debate has been *what* exactly AK entails and what type of AK is valuable to which purpose. Each AKM approach typically uses its own set of knowledge elements and relationships, which are expressed in a meta-model. The knowledge codified in models complying with these meta-models is typically used by tooling associated with the AKM approach to support one or more usages of AK. In the remainder of this section, we investigate how AK concepts and usages of AK have evolved over the last decade. In Section 3.1, we describe the evolution of the AKM concepts and identify three distinct generations. In Section 3.2, we evaluate the associated tooling for each generation.

3.1. An overview of main AK models

Every AKM tool proposed so far (see (Tang et al., 2010) for an overview of most widely known AKM tools) has an associated repository to store and manage AK. In fact, one of the key differentiating factors among the AKM tools is the data model of the corresponding repository. Such data models define the architectural constructs and their relationships with each other, and thus represent architecture design knowledge that is used or generated during software architecture design and analysis. A data model can help organizations to define and obtain data on various aspects of their architectural assets and design rationale during the software architecture process. Moreover, a data model is one of the earliest artefacts produced and assessed in any effort of developing an appropriate AKM tool support. In this section, we provide an overview of a reference AK Model, ISO/IEC/IEEE 42010 (ISO/IEC/IEEE, 2010), for describing software architecture, as this standard also provides a data model characterizing the constructs that need to be captured for documenting software architecture design decisions and their rationale. Then we discuss some of the main data models that have been developed for AKM tools reviewed in (Tang et al., 2010).

One of the major developments in the area of building and validating meta-models for characterizing AK came in the shape of the standards for describing system and software engineering architecture (ISO/IEC/IEEE 42010 (ISO/IEC/IEEE, 2010)). The standard provided a meta-model comprised of the entities and their relationship for describing system and software architecture. The meta-model provided by ISO/IEC/IEEE 42010 purports to provide guidance for the process of describing system and software architectures. However, many researchers started using the meta-model for designing and implementing repositories for AKM tools; some researchers also used that meta-model for validating their existing or new meta-models for AKM. The standards' meta-model consists of stakeholders, system concerns, environment and architecture. Architecture is described in terms of architecture description, which consists of system elements, relationship between system elements, principles of the system design, and principles that guide evolution of system over its life cycle. The model comprehensively covers different aspects associated with architecture description which can also be considered as AK. Since the 42010 standard was developed after several series of discussions with researchers and practitioners and extensive consultations of literature, the meta-model of ISO/IEC/IEEE 42010 captures most of the key concepts and relations provided by several data models designed

for developing tools for AKM. Recently, researchers have started customizing and using the meta-model provided by (ISO/IEC/IEEE 42010) as the data model for designing and developing new sets of AKM tools or evaluating the existing ones. There is a growing advocacy against the need of developing new meta-models that can act as reference or core models for characterizing AK in general. The diversity in the currently available meta-models of AK has shown that a uniform approach cannot cover all AK variability; instead custom solutions need to be devised in order to cater for the specialized needs of either researchers or practitioners. Therefore instead of looking for the 'one and only' AK metamodel, we advise researchers to extensively and empirically validate the existing AK meta-models based on the organizational practices of AKM.

Aligned with the ISO/IEC/IEEE 42010 standard a number of meta-models have emerged over the past ten years to describe the connection between design decisions and other software artifacts and also, how decisions can be represented. Among these meta or data models we can find PAKME (Babar and Gorton, 2007), which has been built upon Data Model for Software Architecture Knowledge (DAMSAK) (Babar et al., 2006) that is a customizable model to characterize the data required to capture architecture knowledge in order to improve the quality of the software architecture process and products. The architecture knowledge constructs are specified at two levels: organizational level, where the constructs characterize generic architecture knowledge, and project level, to describe specific architecture knowledge. Some of the key constructs that characterize AK according to DAMSAK are Architecturally Significant Requirements, Scenarios, Concrete Scenarios, Analysis Model, Patterns, Design tactics, Architecture decisions, Design options, and Design rationale.

Another approach that proposed data model for characterizing design decisions was AREL (Tang et al., 2007) in which Architecture Elements (AE) considered as a part of architecture design. An AREL meta-model comprises elements of the ISO/IEC/IEEE 42010 model elements such as architecture model, system concern, architecture rationale and some viewpoint information. Its focus is on explaining and tracing an architecture design. A plug-in was developed using a commercial architectural management tool called Enterprise Architect. This approach was mainly focused on rationale-driven approach to design.

Jansen et al. developed an AKM tool suite, entitled Knowledge Architect, which involved the development of several data models to characterize AK (Jansen et al., 2009). The tool suite is based on an essential data model focusing on the concepts of Knowledge Entities and Artifacts. This essential data model is extended to represent different types of AK, used for different purposes within different plug-ins of the tool suite: a Word plugin specializes the specialized data model for AK in documentation focusing on Decisions, Concerns, Alternatives (Jansen et al., 2009), exactly as 42010 defines these concepts; an Excel plug-in specializes the data model for AK in quantitative analysis focusing on Quality Attributes, System Parameters and Analysis Functions (Jansen et al., 2008). Both plugins and corresponding data models were developed as part of an industrial project on analysis and documentation in the radio astronomy domain[2]. Based partially on the experience on developing and using the Knowledge Architect in practice, Manteuffel et al. developed the Decision Architect[3], a tool supporting a documentation framework of architecture decisions (Manteuffel et al., 2014). The data model of the Decision Architect is based on the documentation framework that is comprised of five different viewpoints on documenting decisions (van Heesch et al., 2012a, 2012b), according to the guidelines of ISO/IEC/IEEE 42010 (ISO/IEC/IEEE, 2010). In contrast with other data models of AKM tools, the Decision Architect defines a different data model for each viewpoint, catering for the needs of each specific set of stakeholder concerns; for example the Decision Relationship view focuses on the types of relationships between decisions, as well as the states of decisions and how decisions are grouped. The Decision Architect reuses many of the concepts of ISO/IEC/IEEE 42010, while the meta-model of each view extends the ISO/IEC/IEEE 42010 meta-model with few new concepts; for example the decision relationship view is based on the ISO/IEC/IEEE 42010 concept of Architecture Decision and adds the concepts of Relationship and Relationship Type.

Another meta-model characterizing AK for developing tool support was provided by Capilla et al. 2006 (Capilla et al., 2006). Their architecture design decision meta-model consisted of several entities that were also part of the IEEE1471 standard for describing systems and software architectures as well as the data models underpinning several contemporary tools at that time. The architecture design decisions meta-model included entities like Architecture, Decision (which are composed of Style, Pattern, Variation point which are themselves entities) They also had entities like Functional and Non-functional Requirements, Stakeholder, and decision model. They demonstrated the use of the meta-model for developing an AKM tool called, Architecture Design Decision Support System (ADDSS), which was a web-based tool and there have been several updates and releases of ADDSS based on the same meta-model. Coincidently that meta-model is also quite aligned with new standards (ISO/IEC/IEEE42010) for describing software systems.

The meta-models and models for characterizing AK appeared (except ISO/IEC/IEEE42010) in the reports purported to present the AKM tools that were built using those meta-model and models. For example, the models underpinning ADDSS appeared in 2006, while the AK model used for PAKME was reported in 2007. Around the same time, the AREL had been reported. These three models were developed from 2004 to 2006 and were not subsequently extended. The work on the AK model underpinning the Knowledge Architect was carried out from 2005 to 2007. The Knowledge Architect has several metamodels, each for a specific third-party tool like MSWord and MSExcel; all such meta-models are unified in a central meta-model modeled with Protégé. During that time (2004–2007), the meta-model in the standard for describing software architecture IEEE 1471 was evolving to culminate in ISO/IEC/IEEE42010 around 2010–2011.

Since all these models characterizing AKM were derived by the close-knitted community that had been actively working on AKM between 2004 and 2013, there are several similarities among these models despite being developed independently. For example, there are several common concepts, albeit syntactically named differently, like decisions, design options, rationale, concerns, views, and stakeholders. Of course, each of the abovementioned models also has several distinct entities to characterize the diversity and focus of the tools developed based on each of the AK models. The Decision Architect is a Plug-in of a commercial modeling tool, Enterprise Architect. It has been implemented using the meta-model described in (van Heesch et al., 2012a, 2012b), which is in turn based on the metamodel of ISO/IEC/IEEE42010 (Manteuffel et al., 2014). This is one of the newest AKM tools that appeared in 2014, and made use of the commonly accepted (by that time) meta-model of ISO/IEC/IEEE42010.. Whilst it is clear that there are (and will always be) different AKM models that serve different needs and purposes for AKM in different contexts, there is an increasing need of empirical validation of the benefits and effectiveness of different AKM models so that the needs for unique AKM models can be established and empirically justified. Such validation will also provide commercial tools builders more confidence for leveraging the AKM models for building new tools or features for support AKM.

[2] In collaboration with ASTRON, http://www.astron.nl

[3] http://decisions.codeplex.com

3.2. Generations of AK management tools

In addition to the design rationale approaches discussed in Section 2 , over the past ten years there has been a significant body of research in the software architecture field to incorporate architectural design decisions as first-class elements. A detailed comparison and discussion of the first tools for AK management can be found in our earlier work (Tang et al., 2010). We suggested a framework to compare the tools around a set of processes aimed to capture, reason, share, and use architectural knowledge from the producer and consumer perspective and mainly focused on the previous reference model IEEE 1471–2000 (IEEE, 2000). As the research on design reasoning and architectural knowledge has improved significantly in terms of new tools, meta-models and approaches using AK, it is time now to provide a retrospective view of the period 2004–2014. We will discuss the AKM tools around three generations using the AK producer and consumer model and the processes that suit for both categories of stakeholders.

First generation (2004–2006): In this period the first AKM tools were developed independently of each other, and the creators of each tool didn't know in some cases about each other's works. In the beginning, most of the tools focused on the representation and knowledge capturing problem using templates list of attributes like those based on Tyree and Akerman's seminal work (Tyree and Akerman, 2005). During the period, the following five tools (RAT, Archium, AREL, PAKME and ADDSS) were developed.

One of the earliest tools incorporating design rationale was the **RAT** (A Rational-based Analysis Tool) from (Wolf and Dutoit, 2004), which integrates requirements with rationale information to capture the alternatives, assessments, arguments and justifications as model negotiations. RAT supports collaborative aspects for multiple stakeholders during the analysis activity and it provides traceability during the creation of different models where the rationale is captured. RAT relies on an adaptation of the QOC (Question-Option-Criteria) model for handling the reasoning and knowledge behind the creation of the aforementioned analysis models. The **Archium** tool (Jansen and Bosch, 2005) focuses on capturing and maintaining requirements, decisions and architecture descriptions using a domain-specific language, that enable, for instance, detecting incompatible decisions. Similar to RAT, Archium provides some basic reasoning mechanism aimed to detect incompatible decisions. **ADDSS** (Architecture Design Decision Support System, 2006) (Capilla et al., 2006) and **PAKME** (Process-based Architecture Knowledge Management Environment, 2006–2007) (Babar and Gorton, 2007) are similar tools for managing, capturing and documenting design decisions using a Web platform. While ADDSS uses a Web interface to capture requirements, decisions, and coarse-grained architectural elements and their links between them and the decisions themselves on the basis of a decision view for software architecture (Dueñas and Capilla, 2005), PAKME relies on an open source groupware platform (i.e., Hipergate) which captures requirements, decisions, and architectures and establish trace links like ADDSS but with special focus on quality attributes and scenarios. The last tool of this period is **AREL** (Architectural Rationale and Element Linkage, 2007), developed by Tang et al. (Tang et al., 2007) as a plug-in on top of the Enterprise Architect tool and captures architecture descriptions linked to their design decisions and design rationale at the same time the software architect depicts the design using UML descriptions. Functional and non-functional requirements, as relevant design concerns, are also captured to motivate the decisions. AREL uses Enterprise Architect, so the collaboration is supported by the EA model sharing. As a summary of the first generation, all these tools have put the basis for the second generation tools more focused on the sharing aspects between the relevant stakeholders.

Second generation (2007–2010): This second period focuses more on the sharing aspects of AKM and some basic personalization mechanisms rather than in capturing and representing AK. As knowledge sharing has been considered relevant by some authors, we can highlight the following research tools (i.e., Eagle, ADkwik, SEURAT, The Knowledge Architect and ADDM).

Eagle (Farenhorst et al., 2007) is an architectural knowledge sharing platform that uses a blackboard architecture for effective knowledge sharing. Eagle states some desired properties for knowledge sharing such as: stakeholder-specific content, easy manipulation of content, knowledge codification and personalization, and collaborative groupware aspects. In order to support these properties, Eagle provides blogs and wikis to capture and organize the architectural knowledge and provide collaborative and communication facilities among users, RSS feeds to notify the stakeholders about the decisions, text mining as a personalization mechanism, and expert finding facility to easily find colleagues based on experience, interests or projects they work. The second tool **Adkwik** (Schuster et al., 2007) is a web-centric collaborative system and uses a Wiki to capture, organize, and share software architecture design rationale. ADkwik supports the collaborative work of software architects making easy to share the knowledge about architectural design decisions across project boundaries. ADkwik can handle design issues and design decisions with their alternatives, and provides a user interface for the decision identification, decision making, and decision enforcement in a development team. The domain model of ADkwik supports the following four functional building blocks: *collaboration features, dependency management, decision workflow and content repository*. ADkwik has knowledge acquisition and presentation capabilities similar to PAKME but enhances the collaborative aspects. The Software Using RATionale (SEURAT) tool (Burge and Brown, 2008) is an Eclipse plug-is that supports rationale capture and use and integrates the rationale with the source code. SEURAT supports different ways to present the rationale and includes an Argument Ontology that contains a hierarchy of quality attributes to evaluate the pros and cons of the arguments. Requirements are also included in the rationale and arguments for and against alternatives. SEURAT provides traceability support of functional and non-functional requirements to decisions and to code. **The Knowledge Architect** (2008) is a tool suite developed by (Jansen et al., 2008) for capturing, sharing and managing AK. The tool suite is based on a client-server pattern and offers a number of different client tools: a plug-in for Microsoft Word and a plug-in for Excel that enable capturing (by annotation) and use of AK within software architecture documents and spreadsheets respectively; a tool for visualizing and analyzing the traceability relationships between Knowledge Entities; a source code analysis tool for extracting architecture knowledge from Python programs. The heart of the tool suite is the server, an AK repository which provides various interfaces for the client tools to store and retrieve AK, and interoperates with third-party tools. Each client tool uses its own AK metamodel, and all AK stored in the repository can be linked with appropriate traces thus supporting sharing and traceability of AK. Chen et al. (2010) are the creators of the **ADDM** tool (Architecture Design Decision Management, 2010), which is not tied to a particular ADD model and centered on customizable mechanisms and solutions that can be adapted for different users in order to organize and manipulate better AK. The customizable features of ADDM, such as a personalized AK reuse facility or customized storage, increases user satisfaction and provides a more flexible tool. Some extensions to the tools have made, such as for instance basic customization and reuse mechanisms incorporated into ADDSS version 2.0 (Capilla et al., 2008) as well as timers in version 2.1 to measure the capturing effort of the decisions and RSS feeds to keep users aware of the changes in the decisions and tailor AK to different stakeholders .

Third generation (2011–2014): In this latest period the available tools supporting the creation and management of AK focus now on the collaborative aspects, reuse of AK, fuzzy decision-making and assessment facilities to advice on the creation and use of architectural knowledge. Nowak and Pautasso (2013) developed the Software

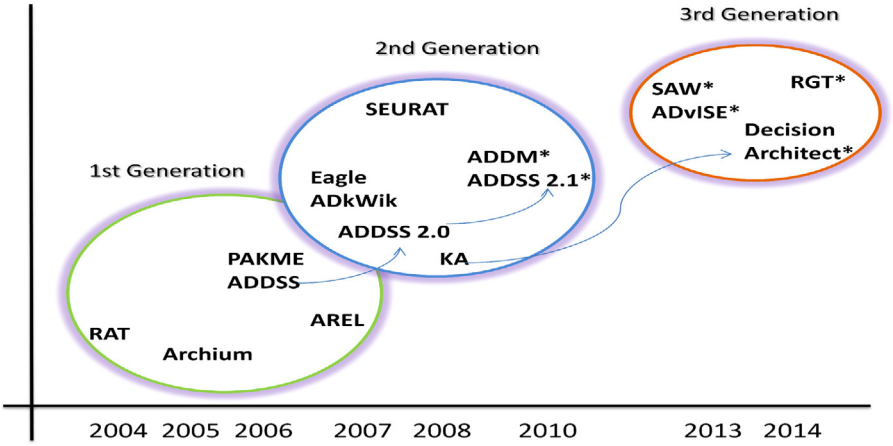

Fig. 1. Evolution of AKM research tools since 2004 (those marked with "∗" are still alive or used). The x-axis represents the time while the y-axis doesn't have any particular meaning for the tools grouped around generations.

Architecture Warehouse (**SAW**) (Nowak and Pautasso, 2013), as a web-based tool that supports distributed architecture design teams, in line with some tools of the second generation. However, SAW induces synergy effects between knowledge reuse and remote collaboration to have feedback about the implications and constraints of a design decision and improve the quality of the decisions made by reaching a collaborative consensus. SAW captures, manages, shares, analyzes reusable AK and is capable of handling multiple decision models. Knowledge sharing is achieved through a Wiki style, while a basic fuzzy decision model that rely on Positive, Negative and Open statuses is also supported. Lytra et al. (Lytra et al., 2013) built ADvISE (Architectural Design Decision Support Framework) which is designed to support the modeling of reusable architectural design decisions using the QOC method and taking into account a certain degree of uncertainty during the decision-making process. The ADvISE tool can create reusable ADD models and generate questionnaires for making actual decisions based on new questions. An additional tool, called VbMF (View-based Modeling Framework) is used in conjunction with ADvISE to provide the C&C (Component & Connector) architectural view model and the necessary trace links between design artifacts and the decisions. An AK transformation engine enacts the transformation from actual decisions to design views and using recurring pattern primitives as reusable AK. Manteuffel et al. have developed the **Decision Architect**[4] (Manteuffel et al., 2014) built on top of Sparx Systems' Enterprise Architect (EA), by using the experience and lessons learned from the Knowledge Architect tool suite (Jansen et al., 2008). The Decision Architect documents decisions in a user-friendly and efficient way as Enterprise Architect models and seamlessly trace them to other architectural design elements. The conceptual framework of Decision Architect uses five different viewpoints (van Heesch et al., 2012a, 2012b) to support all stakeholders concerns involved in the creation of the architecture using design decisions and design rationale and to evaluate the right alternatives based on requirements, qualities and other influencing forces. The tool was developed in cooperation with ABB, in order to integrate a decision documentation tool into an existing modeling platform, thus allowing architects to use the same tool that they use to create models, diagrams and views of the architecture, to document the decisions that constitute these artifacts. This narrows the gap between architecture documentation and architecture decision documentation. Decision Architect was successfully used within ABB and is still being used in different projects, while it became available as Open Source to the wider community. Tang and Lau (2014) provide a certain degree of assessment using AREL's associations feature to evaluate manually software architecture issues on behalf of the traceability facility of the tool. Our last tool is provided by Tofan and Galster (2014). It is a web-based open source online tool that uses the Repertory Grid technique. The **RGT** tool supports decision capturing, decision analysis and architectural group decision making. The tool supports prioritization of concerns using the hundred-dollar approach (i.e., assigns a number between 1 and 100 to reflect the importance of the concern). The tool supports a chronological viewpoint to describe the changes of the decisions over time and dependencies between decisions. In Fig. 1 we show the evolution of AKM tools.

Summary of AKM research tools: The majority of the effort to set the basis of AK management in support of architectural design decisions has been realized in the period 2004–2010, where the first research tools where developed and the ISO/IEC 42010 arrived. As a summary of the major capabilities supported by the aforementioned research tools we provide in Table 1, a list of the existing features according the roles assumed by the different stakeholders of the AK producer-consumer model (Feilkas et al., 2009). In this model, the role of AK producers focus on the synthesis and sharing of AK using generic and context knowledge, while AK consumers retrieve, evaluate and learn from the AK. In Table 1, if a tool poses a particular capability it is marked with "X"; while the symbol "—" means an absence of that capability. Those tools that exhibit partially a certain capability are marked with "P", which means that a particular tool partially supports such capability.

From the activities of AK producers and consumers (e.g., capture, share, reuse, assess, etc.) we evaluated the selected AK tools based on eleven (11) generic AKM features. All the tools examined support capturing AK (CAP) while the majority of them are focused on AK management (MGM) in the sense that a tool can, at least, capture, store, maintain and document the design decisions alongside with requirements and design artefacts using appropriate trace links. Tools based on Web and Wiki technologies can provide sharing capabilities (SHA) while all the tools examined produce some kind of documentation (DOC) of the AK stored. The evolution of the AK is considered a key feature in the tools of they provide some kind of versioning or decision history mechanisms (EVO) while reuse of AK is rarely seen (REU) and it often consists of some kind of retrieval facility of the design decisions or design patterns captured. The reasoning capability (REA) refers to any basic inference mechanism that automatically can, for instance, detect inconsistencies (e.g., RAT or Archium), state new questions (AdvISE) or perform some fuzzy reasoning (SAW). In addition, only one tool provides uncertainty (UCT) support, which for example can consist of the usage of fuzzy reasoning mechanism often based on positive or negative evaluation of the questions or decisions

[4] Available as Open Source at http://decisions.codeplex.com/

Table 1
A comparative list of features of AK research tools from the producer-consumer perspective.

Tool	CAP	MGM	SHA	DOC	EVO	REU	REA	UCT	COL	PER	ASS
1st generation											
RAT	X	P	–	P	–	–	X	–	X		X
Archium	X	P	–	X	–	–	P	–	–	–	–
PAKME	X	X	P	X	X	–	–	–	P	–	–
ADDSS	X	X	P	X	X	–	–	–	P	–	–
AREL	X	P	–	X	X	–	–	–	X	–	P
2nd generation											
Eagle	X	X	X	X	P	–	–	–	X	X	–
ADkwik	X	X	X	X	P	X	–	–	X	–	–
SEURAT	X	P	–	P	–	–	P	–	–	–	X
KA	X	X	–	X	–	–	–	–	X	P	P
ADDM	X	X	X	X	P	X	–	–	X	X	–
ADDSS 2.0/2.1	X	X	P	X	X	P	–	–	P	P	–
3rd generation											
SAW	X	X	X	X	–	–	P	X	X	–	P
ADvISE	X	X	X	X	–	X	P	–	X	–	X
Decision Architect	X	X	X	X	X	–	–	–	X	P	P
RGT	X	P	X	X	X	–	–	–	X	–	X

Legend: *CAP (Capture), MGM(Management), SHR(Share), DOC(Document), EVO(Evolution), REU(Reuse), REA(Reasoning), UCT(Uncertainty), COL(Collaborative), PER(Personalization), ASS(Assessment), X (Capability supported), P(Capability partially supported), –(Capability not supported).*

stated. In addition to the sharing mechanisms, we found collaborative (COL) and personalization (PER) mechanisms that extend basic knowledge sharing features to specialized group of users. Collaborative features enable several users to work concurrently with the tool repository, at least to a certain extent. Finally, the assessment (ASS) capability refers to those cases where quality attributes can be used to assess the most suitable decisions or when the decisions evaluated may influence a particular non-functional requirement. The trend exhibited by some of the most recent AKM tools in favor of advanced collaborative and reasoning capabilities reflects the need of software architects to stimulate new questions and discussions around the decisions. New collaborative aspects are useful for distributed teams when a consensus is required and for group decision making (GDM). As Table 1 shows, part of the discussion and assessment capabilities can be based using uncertainty as a reasoning mechanism, as it has been proven helpful in those cases of incomplete information. The difficulty to reuse (REU) AK is one of the reasons why only few tools provide limited support. While reusing codified AK in the form of well-known design patterns and corporate knowledge is achievable, the reuse of recurring decisions cross similar projects becomes harder and requires an initial investment and a guidance model, such as suggested in (Zimmermann et al., 2008).

4. AK in industry: challenges and needs

There are a great many varieties of ways in which software can be developed. The organization of a software project can range from a single person to thousands of developers spread across geographic locations over long periods of time, or somewhere in between. There are also many development life-cycle methodologies. Some developers and development teams choose to follow one methodology strictly, some choose a hybrid method, or some may just-do-it without adopting any methodology at all. The capturing and sharing of AK under these different circumstances vary greatly. In this section, we analyze the current state of the AK modeling practice by conducting interviews with practitioners from different companies. We analyze the barriers to the adoption of AK, and the complexity of implementing AK in different industrial environments and contexts. We finally describe some variations of AK use in different software development contexts, such as global software development practices, agile and lean approaches, and AK in open source projects among others. From the various research works (Tang et al., 2010, Malavolta et al., 2013), we see that there is a demand to capture AK in certain circumstance but we do not see an emerging approach. Therefore, we highlight the particularities using AK in other development approaches compared to the more traditional architecting practice.

4.1. AKM practice in industry

To understand the state of the AKM practice, we conducted six in-depth interviews with architects from industry, some of them belonging to a large worldwide corporation. All architects had over five years of experience, work in sizeable development environments, and have worked on at least two different systems in different business contexts. The interviews aimed at getting a general understanding of how architects use AK and AK tools in their environments, whether they capture and use AK systematically or not, and how they view the values of AK. Each architect was asked nine questions, which came out of an iterative brainstorm from the authors. The interviews were voice recorded. Based on these recordings a textual summary was formulated as answer to each question in a spreadsheet, which was used to identify common patterns. In short, we asked the following nine questions:

- Q1: What type of architectural knowledge (AK) do you document and keep? (AK examples are requirements, design, decisions, diagrams, drawings and design notes, design patterns etc.)
- Q2: Do you know of any ISO standard or any models to capture AK? What, if any, do you use?
- Q3: What is the most useful architectural documentation? (AK examples are requirements, design, decisions, diagrams, drawings and design notes, design patterns etc.) How often do you use each of them?
- Q4: What AK did you capture in your last project? What format was the knowledge stored in? How did stakeholders find them?
- Q5: Do you capture AK consistently in all projects, or do they differ for each project and why?
- Q6: Is it worthwhile to capture AK for the organization? Why and why not?
- Q7: What would have been worthwhile to have captured, but you did not have it when you needed it?
- Q8: Are you aware of any tools to share and manage the AK?

Q9: What features do you expect from a tool to capture and maintain AK?

In general, respondents capture requirements and functional/logical views of the system as AK (**Q1**). Requirements, if captured, are documented in Word files, whereas the architecture views are typically made in PowerPoint/Visio, which are complemented with a describing text in a document or wiki. The views are typically based on loose interpretations of UML semantics, as exemplified by one respondent; "*We use UML but not adhere to its standards. As long as we can communicate with each other, we are fine*." This confirms earlier research findings on the rather loose use of UML semantics in practice (Lange et al., 2006, Petre, 2013).

Most of the respondents are aware of the existence of ISO/IEC 42010, but are not making use of it nor any other systematic architecture documentation approach (**Q2**). This might explain why we get a rather diverse picture when asking what the most useful architectural documentation is (**Q3**). One respondent stated: "*The main documentation we do is API*", where as another respondent took a more distant perspective: "*The pictures that show how the different parts fit together,* i.e. *the components and how they fit together, a functional breakdown*". A third respondent went even further stating; "*structural modeling information, design decisions, requirements, quality attributes, and the traces between them*". Hence, what is regarded useful is rather different for the respondents. One reason might be the lack of use of standards, the other the different needs in different contexts.

Unsurprisingly, the AK captured by the respondents is not uniform either (**Q4**). One respondent said: "*We capture AK in the form of diagrams. We capture data and event flow. We use Visio and high-level description. We publish on Wiki pages*", another responded with: "*The last one we created functional descriptions in a word files, which was really useful*", and a third said: "*design decisions models, quality attributes, structural architecture models (components and decompositions). The knowledge was stored in UML models and text in a (Word) document".* The only observation we can make from the data is that the correlation between what AK is found useful and what is actually captured seems rather strong.

Most respondents adjust their AK capture based on the context of their development work (**Q5**). As one respondent describes: "*I would say that each project is different. It has to be a decision of the team and the stakeholders. What is necessary and what is important. On smaller projects the overhead if it is only being used by one customer/project. But if you do something global that should be repeatable then it is important*." Hence, the AKM is rather ad-hoc and experience based, and scaling the AK capture (and the associated overhead) is a concern.

AK is regarded as important by all the respondents for their organization (**Q6**). All stress the importance of knowledge transfer of AK and the need to capture it somehow. As one respondent puts it: "*Architecture design is the blueprint of the system. People get the definitions of the system. Otherwise it is by word of mouth and can disappear*". However, the AK capture in practice is far from complete or perfect, as the same respondent remarks: "*Even then, we often have to talk to the original designers to get clarifications.*"

The knowledge, people find worthwhile to have captured, but didn't have (**Q7**) relates to two types of knowledge; design rationale and the architecture design. For the latter, one respondent said: "*The history of the previous generation, which is not documented at all, as it started 25 years ago*". Another respondent reacted with: "*Sometimes we don't have the high-level design diagrams to explain how the components work together. It was not obvious by reading the code*". For the former, the design rationale, one respondent stated: "*For some design, there was no explanation of why it was designed in a certain way*", another said: "*Most of the times what was missing are two things: traces between requirements and models and linked to that the decisions that belong to that. For most projects this is missing*". Hence, it seems you first need to have an architecture design description, before people care about having the rationale of the design.

Some respondents have tried various AKM tools (**Q8**), but faced some challenges in maturity and sharing/integration capabilities preventing usage. For example, one respondent said: "*We did not use AK tools because once we started, they are not compatible with other media, and we cannot easily transfer data from one tool to another*". One respondent on what AKM tool he has used; "*The big one thing is the decision architect, although it is not ready for primetime*".

Features respondents expect from a tool to capture and maintain AK (**Q9**) mostly touches upon the clarity/familiarity of the underlying concepts to all receivers and the ease of use. For example, one respondent formulated his wishes for AK tooling as; "*easy to use, and easy to understand for someone who doesn't use it every day. At least that is my problem with models*". Another respondent stated: "*It should be simple. It should be as simple as possible and reliable.*" Hence, simplicity is regarded an important property of an AK tool to succeed.

Overseeing the results of our interviews, we have a glimpse that systematic AKM has not gained full traction in the industry yet. This result is somewhat expected with the many other software industry practices that we observed, and various studies on the use of AK modeling that indicate practitioners do not systematically document AK (Malavolta et al., 2013, Petre, 2013).

From our results, we see that a certain architectural documentation maturity is required before rationale capture makes sense. Not surprising, is the need to tailor the AK captured to the particular context in which the architect operates. However, *how* to perform this tailoring is a key question for the adoption and practice in the industry. Another challenge is the need for a highly integrated, easy-to-use, and shared AKM tool that can be tailored to a particular system/project context.

4.2. Barriers and adoption of AKM technologies

The previous findings are not surprising. Recent studies within open source software (OSS) and other development shops indicate that people do not capture AK as much as we think (Ding et al., 2014). There are good reasons by which many organizations do not perceive valuable capturing architectural knowledge for their business goals. Today, there are many socio-organizational and technical levels discouraging its use as the adoption of an AKM strategy is rarely seen or hidden by companies because none of the existing commercial software modeling tools can provide effective AKM support. In the software development lifecycle (SDLC), the key design decisions should be captured in the early stages of the design process alongside with requirements and design artifacts. Hence, software architects become the key actors for capturing and using the relevant AK as they are considered the primary decision makers of the software architecting activity. There are many important reasons by which software architects and developers should capture AK as a first-class entity and capital knowledge for any organization. From our experience and observations in the development and use of five of the research tools discussed in Section 3.2, we summarize below the main reasons by which, we believe AK should be captured and used.

- *Avoid knowledge vaporization*: The problem of knowledge getting lost because experts leave an organization or are reassigned to a different department is a major concern for any organization (Bosch, 2004). Hence, the challenge to convert the tacit AK into explicit and documented AK (Tofan et al., 2011) is a good reason by which this knowledge should be captured alongside other software artifacts.
- *Understand the ripple effect of decisions*: Capturing the relevant design decisions and establishing the trace links between them help to estimate better the impact on other decisions when one or several decisions change.

- *Identify and track the root causes of changes and estimate better the impact analysis*: The definition of appropriate trace links between requirements, decisions, and design artifacts bridge the gap between analysis and design and help to identify the root causes of a change in code or estimate better the consequences in design and code from changes in (volatile) requirements.
- *Share decisions among the relevant stakeholders*: Not only the software architects but other relevant stakeholders such as customers, business managers, end-users, and developers can benefit of knowledge sharing practices when key design decisions are made along the software development process. This is crucial in GSD (Global Software Development) where the distributed teams worldwide need to be aware about what other decision makers do.
- *Understand the evolution of the system*: Capturing the decisions and their alternatives as a decision history (Capilla et al., 2011) or decision chronology (van Heesch et al., 2012a, 2012b) can help revert to the original considerations in case of bad decisions and to understand the evolution of the system and the decisions as well.
- *Understand the underpinning reasons of decisions and build on experience*: In team decision makings, novice software engineers can benefit from senior architects if the rationale and the underpinning reasons of the decisions made are stored. Hence, new decisions can be made based on past (good or bad) experiences.
- *Identify critical decisions and runtime concerns*: Some decisions are more important than others and in many critical and real-time systems it should be possible to identify those critical modules in the architecture when a module fails, and hence identify the decisions that led to the creation of such design artifact or piece of code. Tracking runtime decisions (Capilla et al., 2011) is a new challenging feature not yet supported by current AKM tools.

Although the reasons described before for the adoption of AK are important for any software organization, it seems that a broader adoption beyond the current research on AK is still in its infancy. Nowadays, organizations are still not using AK in their current software development practice (with some exceptions, e.g. (Manteuffel et al., 2014)) while only few case studies and experiences report data on the field. Consequently, we discuss the barriers (numbered as Bn below) that inhibit capturing and use of AK and we relate these barriers to the responses to the questions provided from the interviews (see Section 4.1).

- **B1.** *Lack of motivation or incentive*: The relevant stakeholders involved in the development of a system do not perceive useful (Lee and Kruchten, 2007) or valuable for the organization capturing AK **(WHY** capture AK) for several reasons: lack of appropriate tools to manage and (re)use AK, experts are afraid to store their own expertise, or lack of time and budget (Tang et al., 2006). For instance, this barrier reflects responses to questions Q6, Q7 and partially Q8 as the stakeholders need to perceive values for capturing the relevant AK. For instance, some of participants said that AK tools were not compatible with other tools and that deterred them (answer to Q8).
- **B2.** *Lack of adequate tools*: Often AK is not captured because the lack of adequate tools. In many cases the extra efforts are required during design not a single tools can do modeling and storing AK. Sometimes software architects do not know **HOW** to capture knowledge, and which is the best approach to represent and retain the key design decisions and organize the knowledge, (e.g., list of items using Templates, a Wiki, and Ontology). This barrier is reflected by responses to questions Q8 and Q9 regarding adequacy of tools to systematize and make easy AK use and management.
- **B3.** ***WHAT*** *to capture:* Software architects often do not know what knowledge is relevant and valuable to be captured, as well as how to capture it (Harrison et al., 2007). Research tools provide interfaces to capture a long list of items which discourage designers to spend time to store all the items for the design decisions, in particular for large projects. In addition, because trace links, requirements, design artifacts and the decisions are worth storing, but not all the research tools provide such capabilities. Therefore, there is a need to achieve a consensus on what is relevant to capture and what is not. The need to define what to capture is reflected by the responses to questions Q1–Q4, as there are different ways to represent and capture AK and each organization must define how much AK is needed and how to document it.
- **B4.** *Effort in capturing AK:* Managers and designers do not often perceive valuable spent effort and resources capturing the relevant design decisions, producing additional documentation and maintaining the decisions captured with the links to other software artifacts. The efforts or costs necessary for capturing the decisions and AK are immediately realized and they occur during the different phase of the software development lifecycle (Capilla et al., 2008). However, the benefits of capturing AK are not something that is immediately recognized and easily justifiable. The effort spent during the knowledge capturing process is also related to Q4, as each project and stakeholders may need different AK types. Some of the participants recognized the value for capturing UML models and other capturing relevant knowledge in Word documents.
- **B5.** *Disrupting the design flow:* Architects are immersed into the creative flow of designing and are not willing to disrupt it for simply documenting their decisions and rationale (Harrison et al., 2007). Thus they defer documentation until later in the project timeline, at which point they may be pressed on time or don't remember anymore the details of their decision making. Question Q5 is related to barrier B5 as the timeline and project development approach may drive the amount of AK captured for each type of project. Some participants highlighted that different projects may require different effort levels for capturing AK such as capturing decisions systematically.
- **B6.** *Lack of stakeholder understanding*: The relevant stakeholders not only need to know why AK is of value (see first bullet point). They need to understand the implications of AK and their stakes in the system. Often, this requires the architect to transform the AK into a suitable form for someone with a (software) engineering domain background. The last barrier relates to questions Q1–Q3, and different AK types and ways to represent these will make the captured AK useful and manageable for a particular organization for maintenance or training.

As a remedy for the barriers that hamper the use of AK, we recommend a set of factors to reduce the lack of motivation that causes inhibits stakeholders and organizations from capturing the key design decisions. Some of the remedies encompass a set of smaller remedies or concrete tasks that can be done to overcome the aforementioned barriers. Among all the possible remedies (numbered as Rn below) we highlight the followings:

- **R1**. *Encourage and convince the stakeholders*: Not only software architects but also software maintainers could benefit from storing key design decisions for the following reasons: revert from wrong decisions, understand the evolution of the system, estimate change impact analysis, control the ripple effect of decisions that change, understand the rationale of the decisions made, the alternatives considered, and in summary how the system was built when experts are not available. This remedy addresses barriers B1 and B6.
- **R2.** *Systematize the knowledge capturing process:* If managers and software architects are convinced to capture AK, they can provide a systematic process and mentality in the organization to help stakeholders to capture, share, and use the key design decisions in all development phases. The systematization of the AK to be

Table 2
Internal and external factors that influence positively or negatively the use of AK.

Stakeholders	Internal factors		External factors		Solutions
	Positive	Negative	Positive	Negative	
End-users business managers	Share decisions	Don't trust the utility of AK	Reduce the budget for software maintenance	Lack of budget; No time for AK	Increase budget; Convince managers that maintenance will be more efficient and less costly overall
Software architects	Share decisions; Understand evolution; Understand reasons; Identify critical decisions; Quality decisions; Understanding what to capture; Prevent knowledge vaporization	Lack of motivation; Afraid to share own expertise; Afraid to be challenged on decisions Effort to create extra trace links	Learn from other experts; Reuse AK from other projects	Lack of tools; Lack of time; Effort required	Convince software architects on the utility to document and use AK; Provide adequate tools; Adjust project schedule; Systematize the AK process; Provide adequate tools; Establish right design culture Convince software architects that additional trace links using design decisions will ease software maintenance and reduce maintenance effort
Software maintainers	Understand the ripple effect; Identify toot causes of changes; Understand evolution and impact analysis	Effort to maintain extra trace links	Reduce the burden of maintenance	Lack of time; Effort required	Provide adequate tools Keep the size of the decisions network manageable

captured may depend also of the software development approach taken. This remedy addresses barriers B1 and B5.

- **R3.** *Lean approaches for capturing AK*: Not all AK needs to be captured. Hence, some proposals suggest capturing only a minimal subset of items to describe the design decisions and hence, reduce the burden of capturing effort during the design process (Falessi et al., 2013). Others suggest reusing generic Architecture Knowledge such as Architecture Patterns that already provide a rich documentation of the context, the problem, the solution, the consequences of the solution, the relations to other patterns etc. (Harrison et al., 2007). Patterns concern some of the most important design decisions, so reusing the pattern documentation to capture the knowledge of the corresponding decisions is inexpensive yet quite effective. Capturing only the key design decisions and reducing the number of alternatives for each design decision is another way to spend less capturing effort and making knowledge capture cost-effective. This remedy addresses barriers B3 and B4.
- **R4.** *Embed the design rationale with current modeling approaches and tools:* The duality for having two different tools is not affordable for users who in many cases must duplicate effort. Therefore, new tools that are able to integrate all aspects of the design process including the capturing of AK must be provided in one single tool (Capilla, 2009, Manteuffel et al., 2014). Barrier B2 is addressed by this remedy.

Whittom and Roy (Whittom and Roy, 2009) describe a set of intrinsic and extrinsic factors as motivators for Knowledge Management practices. We summarize in Table 2 those internal and external factors that motivate and discourage the use of AKM activities of interest for different stakeholders and the solutions that can be adopted to mitigate the barriers. An internal factor refers to some activity or influencing factor directly related to the adoption of an AK-centric approach while external factors are not directly related to AKM activities but affect indirectly the design of a software system or software project.

4.3. Variations in software development contexts

Traditionally, AKM research tools have been tested in academic and some industrial settings that can be best characterized as a traditional development context. However, there are some variations according to other software development practices that may influence the capture and usage of this AK. From our perspective and observations in research and industry we have seen a diversity of the contexts where AK can be used and managed differently. Among the different development strategies currently used we identify the following contexts:

(a) **AKM in Global Software Development and outsourcing practices**: Some of the tools discussed provide knowledge sharing capabilities in the form as a Wiki (Eagle, ADkwik) or collaborative features for multiple stakeholders (e.g., RAT, ADDSS, PAKME) where users with different roles can discuss around the decisions made and make groupware decisions (e.g., SAW, RGT), e.g. for distributed architecture teams (please refer to Table 1 for the capabilities of AKM research tools).

(b) **Decision-making using agile and lean development approaches**: Compared to more traditional software development approaches, agile decision making for agile projects can contribute to making decisions faster. The adoption of lean models aimed at storing a reduced amount of items for each design decision as a way to decrease the capturing effort (Falessi et al., 2013) can help to speed up some AKM activities according to the timeframe typically used by agile development approaches like Scrum. Nevertheless, the effectiveness of decision making in agile development is sometimes poorly understood (Drury et al., 2011), as decisions made in agile periods, often group decisions, are limited by the Scrum's sprints. Therefore some obstacles may arise during an agile development approach such as: unwillingness to commit a decision, conflicting priorities when multiple customers provide competing requirements, lack of ownership of the decisions made or lack of empowerment to make decisions. Although there is plethora of software development methods ranging from plan-driven to pure agile approaches, it is difficult to adapt the AKM activities to each development approach equally. For instance, a lightweight agile method like the Feature-driven Development (FDD) approach has well defined milestones that benefits the quality of the product, compared to Scrum, and hence the quality of the decisions using *design inspections*. The variety of agile methods makes difficult to align equally all AKM activities with agile software development, but the iterative, incremental, and evolutionary nature of agile approaches, the face to face conversations, and the continuous attention for a good design, as relevant factors stated in the Agile Manifesto,

influence the way and time in which the design decisions can be and are made compared to the more traditional software development approaches. Consequently, the variability observed between agile and non-agile development approaches during the decision-making phase introduces diversity in the reasoning activity (e.g., periods where decisions are made, less emphasis on strategic decisions, disagreement to plan decisions upfront).

(c) **AK in Open source projects:** Not all software projects follow strict organizational rules or formal project management strategies. Today, many successful software projects and incubators adopt an open-source software (OSS) development strategy. For instance, van der Ven and Bosch (2013) analyze the role of design decisions stored in distributed OSS repositories like GitHub and the different levels of realization of decisions that can be used and stored (from high level decisions to realization decisions like patterns). One visible consequence of decisions stored in OSS repositories is that decisions can be mined and reused across different projects. Another aspect refer to the legal aspects of decisions belonging to OSS projects (Hamouda et al., 2010), where multiple stakeholders with different roles can access open-source patterns versus proprietary patterns developed under and for software companies. Regarding the technology used, some tools like ADvISE use open source software (e.g., an open source fuzzy library) to support the AKM features. Sharing knowledge in OSS projects is another capability highlighted by some authors (Babar et al., 2009) that contributes to the acquisition and dissemination of the key design decisions among the stakeholders. Other experiences (Stamelos and Kakarontzas, 2009) examine ways where knowledge is shared in Free Libre Open Source Software (FLOSS) and highlight the differences between OSS and non-OSS communities, as FLOSS projects follow a pattern-centric approach to manage and share the AK among developers in different locations. Therefore, the organizational aspects of OSS projects contribute to expand the discussion space where the interaction between multiple stakeholders can complicate the synchronization of the decisions and the identification of the key decision makers. Finally, Zimmermann et al. (2012) point out the role of AKM solutions for strategic outsourcing in the context of IT services and how we can ease the creation of architecture decision knowledge conducted on two SOA projects. In their approach, the authors state that the combination of reference architecture, a meta-model, and twelve modeling practices and principles can help AKM activities using a solution decision advisor (SDA) tool to support knowledge engineering and provide guidance for decision-makers.

(d) **Application-specific AK:** There are few experiences reporting the use of AK in industrial settings, so it is difficult to evaluate the real impacts of the use of AKM research tools or how knowledge is captured and used in industry projects. The role of AK in enterprise architecture frameworks like TOGAF is widely recognized by several authors (Proper, 2011), as the solution architecture provides models of actual designs and the underlying design decisions as well. Architectural knowledge can be used strategically by entrepreneurial firms (Baldwin, 2010) to change firm's scope and boundaries when resources are poor and when innovation is needed to solve problems better and identify bottlenecks in existing systems. Other experiences (Feilkas et al., 2009) describe the possible loss of the implicit knowledge during the evolution of three industrial projects, and try to answer why the architecture decay or why the documentations doesn't reflect the intended architecture hen developers do not understand the complete architecture of the system and knowledge is hidden. In (Borches and Bonnema, 2010), the authors present an approach for capturing and share architectural knowledge to support decision making during system evolution, and they applied the proposed model the Philips MRI scanners product family.

One consequence of these different software development contexts is the way AK is consumed and produced. In our previous work (Tang et al., 2010), we described the scope of AK by describing the general activities and stakeholders of AK in a producer-consumer model. Here, we describe how the development context affects these activities and stakeholders. The original consumer-producer model (Feilkas et al., 2009) has at the heart the concept of *Reasoning Knowledge*, which includes design decisions, rationale, alternatives considered, tradeoffs made and forces identified. Related to this knowledge are *General Knowledge, Context Knowledge,* and *Design knowledge*, which could be *traced* to the *Reasoning Knowledge.* General *Knowledge* includes knowledge that is usable in any system design (e.g. architectural styles, patterns, and tactics), whereas *Context Knowledge* contains the knowledge about the problem space, which is specific to a system, e.g. architectural significant requirements and project context. *Design Knowledge* includes the knowledge about the design of the system, e.g. architectural models or views. Together these four types of knowledge make up the concept of *Architectural Knowledge.*

A *Consumer* can *Learn* and *Search/Retrieve* this AK. In addition, a Consumer can *evaluate* the *Reasoning/Design Knowledge*. A *Producer* produces *Reasoning Knowledge* by *architecting* and *Design Knowledge* by *synthesizing.* The *Producer* can also *distill* and *apply General Knowledge* while *architecting/synthesizing* and *integrate Context Knowledge*. Last, but not least, the *Producer* can *share* the produced *Architectural Knowledge.*

Table 3 describes some old and new AK categories and activities derived from the variations in software development contexts and from the capabilities exhibited by latest AKM tools. For instance, regarding new knowledge categories, Context AK is extended by domain-specific AK belonging to specialized knowledge frameworks or knowledge coming from application domains. The collaborative capabilities of some tools and the distributed nature of OSS AK are widely recognized as a new way for knowledge sharing, and where decisions are disseminated across the development team. In such cases, the evaluation is often done by design teams (group and distributed decision-making teams). In other cases, the knowledge is

Table 3
Existing and new categories and AK activities using the producer-consumer model.

Knowledge categories	Activities on the producer side	Activities on the consumer side
General AK	Apply, Distill, Share	Learn
Context AK &Domain-specific AK (New)	Integrate, Specify, Clarify (New)	Search, Retrieve
Reasoning AK	Architect, Advice (New)	Evaluate, Assess
Design AK	Synthesize complete AK	Evaluate complete AK
	Synthesize incomplete AK using fuzzy reasoning and risk decision-making (New)	Evaluate incomplete AK (New)
Collaborative and OSS distributed AK (New)	Groupware decision-making (New), disseminate (New)	Groupware evaluation (New)

Table 4
Comparison of roles of AK activities in software development contexts.

AK Activity	GSD	Agile	Open Source
Architecting	Groupware decisions	Reduce decision scope/ funneled decision making	Benevolent dictator, voting, commit-sponsor
Sharing	Communication through tele-conferencing, instant messaging, e-mail, video conference	Maximize communication bandwidth among team members. Stand-up meetings, fast iterations.	Disseminate, e-mail lists, forums. Share by example.
Learn	Socialization, internationalization	Agile socialization	Open source internationalization
Evaluation	Groupware evaluation	Evaluate decisions with a limited scope, not strategic ones	Distributed evaluation

incomplete due to several reasons, and decision-makers need to synthesize fuzzy decisions upon incomplete knowledge where risk plays a central role. Advising and assessing on AK are new capabilities that some tools provide to suggest new decisions based on previous decisions made. As software architects become knowledge experts, advising and assessing novice architects, and even performing training activities will be a part of the knowledge cycle. We observed that the producer-consumer model is still valid for many of the forms and organizations where knowledge is produced and consumed, but we added new knowledge categories and activities necessary to reflect recent developments in the field.

Additionally, the development context affects the way in which the aforementioned activities on these AK categories are performed, defines the typical consumers and producers of it, and favors certain AK knowledge types above others. Table 4 explains some of the differences using AK in the aforementioned contexts. For instance, while traditional development approaches relies on a chief architect who is the responsible for making the key technical decisions, in GSD the role is assumed by a group, often distributed, of decision makers, and where knowledge sharing covers a bigger importance. In OSS projects the responsibilities for making decisions are even much more disseminated and it often happens that some decisions are never committed. Informal meetings or forums are typical ways to create and share knowledge. Finally, agile approaches with stringent need to make quick decisions for narrowed problems, and performing fast iterations is the main characteristic of this development approach. By contrary, strategic decisions still need to be made in many cases outside agile sprints as they involve more people and need more time to be decided. The table does not provide a complete list of AK activities for each development context but gives an idea on how the traditional AK processes differ from one context to another.

5. Trends and development

The literature discussed in the previous sections seems to suggest that the optimal strategy to manage AK needs to include both codification and personalization and not limit to only one of them. Codification alone may be too heavyweight and bureaucratic for most organizations, while personalization is too elusive for others (thus bearing the risk of knowledge vaporization). The exact mix of codification and personalization is to be determined according to the context (e.g. team size, organization culture, distribution). Once concrete success story from the industry is the formation of Communities of Practice, where personalization leads to efficient knowledge sharing: *groups of practitioners use a simple medium (*e.g. *mailing list) to ask others to share their AK*. Another key learning from managing AK in practice is that organizations need to realize that the amount of explicit AK is only the 'tip of the iceberg', compared to the amount of tacit AK: most of the knowledge in an organization will never be documented. This realization is not easy for large organizations that aim at safeguarding their corporate assets, but it is a perquisite for making an explicit and informed decision about the extent to which they aim at codifying knowledge.

The meta-models and corresponding tools that have been proposed so far cover a wide range of use cases, domains and organizational contexts. The experience of the community so far indicates that a one-size-fits-all approach would not work: meta-models and tools need to be organization-, domain-, or project-specific. However any such custom tool should be lightweight, otherwise it will never be adopted by architects, and it should be descriptive instead of prescriptive (i.e. give advice to architects rather than make the decision for them). There still needs to be substantial improvement in the usability of the tools, their visualization features, as well as their 'intelligence', i.e. their ability to (semi-) automate some of the use cases for AK management. Finally tools should interoperate with other tools from the different phases of the lifecycle in order to establish traces (e.g. between decision decisions and requirements or code).

Another key learning of the community is that architects are not likely to document their decisions during the course of architecting. Any documentation of AK is most likely to happen after the fact, as a post-hoc recording of what decisions were made and why. This can realistically happen during an architecture review (van Heesch et al., 2014), where the AK documentation is a by-product, not an activity where the AK document is the goal in itself. Another efficient way of documenting AK is by focusing on the application of architecture patterns, which constitute some of the most important design decisions; by reusing the rich documentation of patterns, most of the information related to Design Decisions can be inexpensively recorded. Finally there is enough evidence to suggest that a viewpoint-based approach to document decision is fruitful, as it aligns well ISO 42010 and can support the reasoning process of architects during analysis, synthesis and evaluation (van Heesch et al., 2013).

5.1. Sustainability and longevity of AK

As stated at the beginning, knowledge vaporization is an important problem for organizations where non-expert architects must learn from more experienced people and from good and bad decisions. Therefore, and in order to prevent a loss of the AK captured, it is important for the organization to count with a significant body of stable knowledge where decisions can survive as much as possible.

The lack of long-term thinking in systems and software aging are problems software engineers must face every day, and the sustainability and longevity of the software are two sides that affect software quality (Becker, 2014) in various forms. It is argued that technical and design sustainability must be defined and measured using a composite of several quality attributes (Venters et al., 2014) as a way to prevent architecture erosion and keep the relevant design decisions sustainable too.

In this light, approaches like the one described by Zdun et al. (Zdun et al., 2013) promote the sustainability of the AK in the long-term through the SMART (i.e., **S**trategic, **M**easurable and manageable, **A**chievable and **R**ealistic, **T**imeles) approach for capturing the right set of trace links that help to increase decision's sustainability and those timeless decisions in favor of a reduction of the decision documentation effort.

As systems must be maintained over time and bad decisions made (that may incur in the Design Debt problem (Cunningham, 1992, McConnell, 2008) during architecting and coding practices must be minimized, it is key to apply sustainable criteria in order to achieve the maximum stability of the system and architecture when decisions change. Such goals are important in terms of cost, maintenance effort, and changes in the development approach for each particular organization or architecting teams. Some recent work addresses the sustainability in architecture and code and how it impacts in the design decisions as well. For instance, Koziolek et al. (2013) provide a taxonomy of metrics to measure architecture sustainability. However, it is difficult to measure the sustainability of the design using one single metric, as the decisions (e.g., a pattern or an anti-pattern) made can be spread across different software artifacts.

Because the stability of the architecture is crucial for the evolution of the system and the decisions as well, measuring how sustainable the AK is during architectural changes can give us an estimation of the maintenance and documentation effort needed when new requirements trigger new decisions.

5.2. Education and training

The research and academic community has been promoting the use of AK through research and education. Practitioners generally agree that AK is useful. However, their practice of AK is ad-hoc. From the interviews, we learned that some architects recognized that there are international standards (Q2) and yet they do not document AK systematically (Q1, Q3 and Q4). This shows that the knowledge and practice of managing AK has room to improve. In order to improve the situation, we have to consider four concerns from the education and training perspective.

First, an understanding of the costs and the benefits of managing AK is required by project managers and architects. The costs of producing AK are immediately recognized by project managers, especially when a project is under time and cost pressure. The benefits of AK or the future costs of not having AK, however, are not easily quantifiable. As such, it is easy for architects to postpone this cost to the future. In order to sensibly manage and balance the costs and the benefits of AKM, architects and other decision makers must understand the balance between costs spent on capturing AK now and the savings of future costs.

Second, architects document what they think are important. There is little guidance to help them select and structure the knowledge to suit their needs. KM needs differ between organizations, software team structures, level of expertise required, and so on. These parameters influence the cost-benefit structure of AKM. It has been found that effectiveness of AK retrieval depends on the needs of the AK users (de Graaf et al., 2014). The lack of guidance means that architects do not have sufficient information to judge how much AK is needed. In the meantime, each architect or architecture team would do what they think is right, and AK capture is just guesswork of what AK would be used for.

Third, commercial AKM tools are essential to aid architect. However, research findings have yet to impress commercial AKM tool makers what facilities are required. This situation is also reflected in our interviews. There are great variations of what tools are used and the manner in which the AK is captured. UML, Archimate and Visio are some of the more popular drawing tools used by architects. They by no means provide comprehensive AK capture and retrieval mechanisms. As discussed earlier, architects do not often use them or sometimes use them in an ad-doc fashion (Malavolta et al., 2013). Additionally, there is the issue of AK capture using file-based specifications and wikis. It has been found that file-based specification is not structured in a way that is effective for AK retrieval (de Graaf et al., 2012). Research tools are often immature and limited in their functions to be usable by practitioners. The existence of feature-rich and industry-standard tools is essential to managing AK.

Fourth, educating architects on AKM, its needs and its benefits are fundamental to changing the current practice. Most of the literature remains within the research community. AKM needs to enter into text books and training courses to train students and architects.

6. Conclusions

In this paper, we carried out a retrospective review of the architecture knowledge management for the past decade. We examined the fundamental ideas of architecture design as a set of design decisions and rationale. Many researchers came up with meta-models to represent the design decisions and explain the design artifacts, but new models have emerged to enhance the earlier ones. In this paper, we analyzed these models and their applications in AKM. We then evaluate AKM in practice and research from different angles: (a) What has been done? (b) What do practitioners need? (c) What are the barriers of AK implementation? (d) What is the future development?

First, we analyzed the existing meta-models supporting the AKM tools and we observed that most of the key concepts, despite of the variations in the terminology of each meta-model, comply with the ISO/IEC/IEEE 42010 standard, as it describes an agreement on the minimal set of concepts needed for AKM in software architecture. After that, models describing extensions and new capabilities rely on this standard as the core model to provide new functionalities for decision-makers, but all can be considered variations of the standard or just simply improvements to previous meta-models.

Second, we analyzed AKM tools from the past decade, and list their evolution using 11 generic features to evaluate the tools. From this analysis, we found that the capturing, sharing and documenting capabilities provided in the first generation of tools have been extended. In the second generation, the extensions include more advanced sharing and personalization mechanisms, a focus on evolution aspects and some basic assessment features. The third generation provides better assessment and advising mechanisms and some more elaborated reasoning features sometimes based on incomplete knowledge. The trend for a future and better AK management demands support for more sophisticated reasoning and collaborative mechanisms, better support for evolution and runtime decisions, and an answer to the claim for an integration with commercial modeling tools.

Third, we interviewed six practitioners to sample the AKM use. Despite of the amount of research and the large number of research tools that had been built, we did not find systematic uptake of AKM. Our interviewees told us that they captured AK depending on the contexts of the project, the tools they used are semi-formal and there are no systematic AKM approaches.

We also analyzed the barriers to AK practices. We found that the lack of understanding of the values of AK is a main issue. Management and practitioners do not appreciate what costs and benefits AK has, and therefore cannot decide how many resources to allocate for capturing AK. The lack of adequate for managing AKM is another main issue. Capturing AK is context dependent, the size of a project, the existing knowledge and familiarity of knowledge of existing personnel, global software development, lean and agile methodology are factors that influence how AK is valued and applied. We are aware of some experiences using AK in systems architectures or in specific application domains, but the lack of commercial tools supporting a wider use of this AK often hamper a systematic use of AK by companies. This is why we preferred to focus on the evolution, trends, barriers and advances on AK research in software architecture in general.

From our analysis, technical sustainability is an important factor to support evolvable AKM approaches, meta-models and architectures as well as a way to capture stable decisions. The sustainability of these models must be measured over time and metrics are needed to

estimate the changes in stable and volatile decisions. Another trend reflects the importance to educate software architects in the use of AK practice and the benefits gained documenting decisions. Additionally, business and project managers must perceive that AK training pays-off when decisions become a first-class entity that must be captured alongside trace links to other software artifacts in order to reduce maintenance efforts and increase the understandability of the system. The analyses show that AKM is now mainstream and active research field of major importance for modern software architecture practice. The richness of functionality of the many meta-models and tools provide support for AKM activities. They also support a wide variety of software development contexts. However, we observed two major barriers for a wider adoption of AKM practice. The first refers to a non-existing integration with commercial modeling tools. Second, we learned from the interviews that ad-hoc use, instead of systematic practice, of AK in the industry stems from a lack of understanding of how the values of AK can be harnessed. Having evaluated the progress of AKM practices in the past decade, we are optimistic with the future and we are hopeful that the results from some of these works will flow to software companies to provide better and integrated support for software designers.

References

Babar, M.A., Dingsøyr, T., Lago, P., Vliet, H.v. (Eds.), 2009, Software Architecture Knowledge Management: Theory and Practice. Springer-Verlag, Berlin, Heidelberg.

Babar, M.A., Boer, R.C.d., Dingsøyr, T., Farenhorst, R., 2007. Architectural knowledge management strategies: approaches in research and industry. In: Proceedings of Second Workshop on SHAring and Reusing architectural Knowledge - Architecture, Rationale, and Design Intent (SHARK/ADI 2007).

Babar, M.A., Gorton, I., 2007. A tool for managing software architecture knowledge. In: Proceedings of the 2nd Workshop On Sharing And Reusing Architectural Knowledge (ICSE Workshops).

Babar, M.A., Gorton, I., Kitchenham, B., 2006. A framework for supporting architecture knowledge and rationale management. In: Dutoit, A.H., McCall, R., Mistrik, I., Paech, B. (Eds.), Rationale Management in Software Engineering. Springer, pp. 237–254.

Baldwin, C.Y., 2010. When Open Architecture Beats Closed: The Entrepreneurial Use Of Architectural Knowledge. Harvard Business School, Massachusetts.

Becker, C., 2014. Sustainability and longevity: two sides of the same quality? Mental vol. 20, 21.

Berg, M.v.d., Tang, A., Farenhorst, R., 2009. A constraint-oriented approach to software architecture design. In: Proceedings Of The Quality Software International Conference (QSIC 2009), pp. 396–405.

Bonnema, G.M., 2014. Communication in multidisciplinary systems architecting. Procedia CIRP 21, 27–33.

Borches, P.D., Bonnema, G.M., 2010. A3 architecture overviews: focusing architectural knowledge to support evolution of complex systems. 20th Annual International Symposium of INCOSE, Chicago (USA), July 12–15.

Bosch, J., 2004. Software architecture: the next step. In: Proceedings of Software Architecture: First European Workshop, EWSA 2004. St Andrews, UK, pp. 194–199.

Burge, J., 2005. Software Engineering Using design RATionale. Doctor of Philosophy, Computer Science, Worcester Polytechnic Institute.

Burge, J.E., Brown, D.C., 2008. SEURAT: integrated rationale management. In: Proceedings of the 30th International Conference On Software Engineering, pp. 835–838.

Capilla, R., 2009. Embedded design rationale in software architecture. In: Proceedings of Joint Working IEEE/IFIP Conference on presented at the Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009.

Capilla, R., Nava, F., Carrillo, C., 2008. Effort estimation in capturing architectural knowledge. Proceedings of the Automated Software Engineering (ASE'08), pp. 208–217.

Capilla, R., Nava, F., Pérez, S., Dueñas, J.C., 2006. A web-based tool for managing architectural design decisions. In: Proceedings of the 1st Workshop on Sharing and Reusing Architectural Knowledge.

Capilla, R., Zimmermann, O., Zdun, U., Avgeriou, P., Küster, J.M., 2011. An enhanced architectural knowledge metamodel linking architectural design decisions to other artifacts in the software engineering lifecycle. Software Architecture. Springer, pp. 303–318.

Chen, L., Babar, M.A., Liang, H., 2010. Model-centered customizable architectural design decisions management. In: Proceedings of 21st Australian Software Engineering Conference (ASWEC), 2010, pp. 23–32.

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., et al., 2011. Documenting Software Architectures: Views and Beyond, 2nd ed. Addison Wesley.

Clements, P., Shaw, M., 2009. "The golden age of software architecture" revisited. IEEE Softw. 70–72.

Cloutier, R., Muller, G., Verma, D., Nilchiani, R., Hole, E., Bone, M., 2010. The concept of reference architectures. Syst. Eng. 13, 14–27.

Conklin, E., Burgess-Yakemovic, K.C., 1996. A process-oriented approach to design rationale. In: Moran, T., Carroll, J. (Eds.), Design Rationale: Concepts, Techniques and Use. Lawrence Erlbaum Associates, pp. 393–427.

Conklin, J., Begeman, M., 1988. gIBIS: a hypertext tool for exploratory policy discussion. In: Proceedings of the 1988 ACM Conference on Computer-Supported Cooperative Work, pp. 140–152.

Cunningham, W., 1992. The WyCash portfolio management system. In: Proceedings of ACM SIGPLAN OOPS Messenger, pp. 29–30.

Curtis, B., Krasner, H., Iscoe, N., 1988. A field study of the software design process for large systems. Commun. ACM 31, 1268–1287.

de Boer, R.C., Farenhorst, R., Lago, P., van Vliet, H., Clerc, V., Jansen, A., 2007. Architectural knowledge: getting to the core. In: Proceedings of 3rd International Conference On The Quality Of Software Architectures (QOSA).

de Graaf, K.A., Tang, A., Liang, P., van Vliet, H., 2012. Ontology-based software architecture documentation. In: Proceedings Of The Joint Working Ieee/Ifip Conference On Software Architecture (WICSA) And European Conference On Software Architecture (ECSA), 2012, pp. 121–130.

de Graaf, K., Liang, P., Tang, A., van Hage, W., van Vliet, H., 2014. An exploratory study on ontology engineering for software architecture documentation. Comput. Ind.

Ding, W., Liang, P., Tang, A., Van Vliet, H., 2014. Knowledge-based approaches in software documentation: a systematic literature review. Inf. Softw. Technol. 56, 545–567.

Ding, W., Liang, P., Tang, A., Vliet, H.v., Shahin, M., 2014. How do open source communities document software architecture: an exploratory survey. In: Proceedings of the Presented at the 9th International Conference on Engineering of Complex Computer Systems (ICECCS). Tianjin.

Drury, M., Conboy, K., Power, K., 2011. Decision making in agile development: a focus group study of decisions and obstacles. In: Proceedings of Agile Conference (AGILE), 2011, pp. 39–47.

Dueñas, J.C., Capilla, R., 2005. The decision view of software architecture. In: Proceedings of the 2nd European Workshop on Software Architecture (EWSA 2005), pp. 222–230.

Dutoit, A.H., Johnstone, J., Bruegge, B., 2001. Knowledge scouts: reducing communication barriers in a distributed software development project. In: Proceedings of the Eighth Asia-Pacific Software Engineering Conference, 2001. APSEC 2001., pp. 427–430.

Falessi, D., Briand, L.C., Cantone, G., Capilla, R., Kruchten, P., 2013. The value of design rationale information. ACM Trans. Softw. Eng. Methodol. (TOSEM) 22, 21.

Farenhorst, R., Lago, P., Vliet, H.v., 2007. EAGLE: effective tool support for sharing architectural knowledge. Int. J. Cooperative Inf. Syst. 16, 413–437.

Feilkas, M., Ratiu, D., Jurgens, E., 2009. The loss of architectural knowledge during system evolution: an industrial case study. In: Proceedings of IEEE 17th International Conference on Program Comprehension, 2009. ICPC'09., pp. 188–197.

Fischer, G., Ostwald, J., 2001. Knowledge management: problems, promises, realities, and challenges. IEEE Intell. Syst. 16, 60–72.

Garlan, D., Shaw, M., 1993. An introduction to software architecture. Adv. Softw. Eng. Knowl. Eng. 2, 1–39.

Han, J., 2001. TRAM: a tool for requirements and architecture management. In: Proceedings of the 24th Australasian Computer Science Conference. Gold Coast, Australia, pp. 60–68.

Harrison, N., Avgeriou, P., Zdun, U., 2007. Using patterns to capture architectural decisions. Softw. IEEE 24, 38–45.

Haumer, P., Pohl, K., Weidenhaupt, K., Jarke, M., 1999. Improving reviews by extended traceability. In: Proceedings of The 32nd Hawaii International Conference On System Sciences.

IEEE, 1996. IEEE/EIA Standard - Industry Implementation of ISO/IEC 12207:1995, Information Technology - Software life cycle processes (IEEE/EIA Std 12207.0-1996). IEEE.

IEEE, 2000 IEEE Computer Society.

ISO/IEC/IEEE, "ISO/IEC/IEEE 42010:2010 Systems and Software Engineering - Architecture Description " Mar 2010.

Jansen, A., Avgeriou, P., Ven, J.S.v.d., 2009. Enriching software architecture documentation. J. Syst. Softw. 82, 1232–1248.

Jansen, A., Bosch, J., 2005. Software architecture as a set of architectural design decisions. In: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture.

Jansen, A., d. Vries, T., Avgeriou, P., v. Veelen, M., 2008. Sharing the architectural knowledge of quantitative analysis. In: Proceedings of the Quality of Software-Architectures (QoSA 2008).

Koziolek, H., Domis, D., Goldschmidt, T., Vorst, P., 2013. Measuring architecture sustainability. IEEE Software 30 (6), 54–62.

Kruchten, P., 2004. The Rational Unified Process: An Introduction. Addison-Wesley Professional.

Kruchten, P., 2011. Games architects play. Available: http://www.cs.rug.nl/~matthias/pages/workshop_april_18_2011/slides_kruchten.pdf. (accessed 15.09.15.).

Kruchten, P., Lago, P., Vliet, H.v., 2006. Building up and reasoning about architectural knowledge. Quality of Software Architecture (QoSA) 43–58.

Kruchten, P., Lago, P., Vliet, H.v., Wolf, T., 2005. Building up and exploiting architectural knowledge. In: Proceedings Of 5th Working Ieee/Ifip Conference On Software Architecture.

Lange, C.F., Chaudron, M.R., Muskens, J., 2006. In practice: UML software architecture and design description. Softw. IEEE 23, 40–46.

Lee, J., Lai, K., 1996. What is design rationale? In: Moran, T., Carroll, J. (Eds.), Design Rationale - Concepts, Techniques, and Use. Lawrence Erlbaum, New Jersey, pp. 21–51.

Lee, L., Kruchten, P., 2007. Capturing software architectural design decisions. In: Proceedings of Canadian Conference on Electrical and Computer Engineering, 2007. CCECE 2007., pp. 686–689.

Li, Z., Liang, P., Avgeriou, P., 2013. Application of knowledge-based approaches in software architecture: a systematic mapping study. Inf. Softw. Technol. 55, 777–794.
Lytra, I., Tran, H., Zdun, U., 2013. Supporting consistency between architectural design decisions and component models through reusable architectural knowledge transformations. Software Architecture. Springer, pp. 224–239.
Maclean, A., Young, R., Bellotti, V., Moran, T., 1996. Questions, options and criteria: elements of design space analysis. In: Moran, T., Carroll, J. (Eds.), Design Rationale - Concepts, Techniques, and Use. Lawrence Erlbaum, New Jersey, pp. 53–105.
Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., Tang, A., 2013. What industry needs from architectural languages: a survey. IEEE Trans. Softw. Eng. 39, 869–891.
Manteuffel, C., Tofan, D., Koziolek, H., Goldschmidt, T., Avgeriou, P., 2014. Industrial implementation of a documentation framework for architectural decisions. In: Proceedings of 2014 IEEE/IFIP Conference on Software Architecture (WICSA), pp. 225–234.
McConnell, S., 2008. Construx Software Builders, Inc.
Miksovic, C., Zimmermann, O., 2011. Architecturally significant requirements, reference architecture, and metamodel for knowledge management in information technology services. In: Proceedings of 9th Working IEEE/IFIP Conference on Software Architecture (WICSA), 2011, pp. 270–279.
Nonaka, I., Takeuchi, H., 1995. The Knowledge-Creating Company: How Japanese Companies Create The Dynamics Of Innovation. Oxford university press.
Nowak, M., Pautasso, C., 2013. Team situational awareness and architectural decision making with the software architecture warehouse. Software Architecture. Springer, pp. 146–161.
Perry, D.E., Wolf, A.L., 1992. Foundation for the study of software architecture. ACM SIGSOFT Softw. Eng. Notes 17, 40–52.
Petre, M., 2013. UML in practice. In: Proceedings of the 2013 International Conference on Software Engineering, pp. 722–731.
Proper, D.G.E., 2011. Architecture Principles: The Cornerstones of Enterprise Architecture. Springer Verlag.
Ramesh, B., Jarke, M., 2001. Towards reference models for requirements traceability. IEEE Trans. Softw. Eng. 27, 58–93.
Rittel, H.W.J., Webber, M.M., 1973. Dilemmas in a general theory of planning. Policy Sci. 4, 155–169.
Schuster, N., Zimmermann, O., Pautasso, C., 2007. ADkwik: Web 2.0 collaboration system for architectural decision engineering. SEKE 255–260.
Shaw, M., Clements, P., 2006. The golden age of software architecture. Softw. IEEE 23, 31–39.
Stacy, W., MacMillan, J., 1995. Cognitive bias in software engineering. Commun. ACM 38, 57–63.
Stamelos, I., Kakarontzas, G., 2009. AKM in open source communities. Software Architecture Knowledge Management. Springer, pp. 199–215.
Tang, A., 2011. Software designers, are you biased? In: Proceeding of the 6th International Workshop On Sharing And Reusing Architectural Knowledge. Waikiki, Honolulu, HI, USA.
Tang, A., Avgeriou, P., Jansen, A., Capilla, R., Ali Babar, M., 2010. A comparative study of architecture knowledge management tools. J. Syst. Softw. 83, 352–370. doi:10.1016/j.jss.2009.08.032.
Tang, A., Barbar, M.A., Gorton, I., Han, J., 2006. A survey of architecture design rationale. J. Syst. Softw. 79, 1792–1804.
Tang, A., Jin, Y., Han, J., 2007. A rationale-based architecture model for design traceability and reasoning. J. Syst. Softw. 80, 918–934.
Tang, A., Lau, M.F., 2014. Software architecture review by association. J. Syst. Softw. 88, 87–101 2.
Tofan, D., Galster, M., 2014. Capturing and making architectural decisions: an open source online tool. In: Proceedings of the 2014 European Conference on Software Architecture Workshops, p. 33.
Tofan, D., Galster, M., Avgeriou, P., 2011. Capturing tacit architectural knowledge using the repertory grid technique (NIER track). In: Proceedings of the 33rd International Conference on Software Engineering, pp. 916–919.
Tofan, D., Galster, M., Avgeriou, P., Schuitema, W., 2014. Past and future of software architectural decisions–a systematic mapping study. Inf. Softw. Technol. 56, 850–872.
Tyree, J., Akerman, A., 2005. Architecture decisions: demystifying architecture. IEEE Softw. 22, 19–27.
van der Ven, J.S., Bosch, J., 2013. Making the right decision: supporting architects with design decision data. Software Architecture. Springer, pp. 176–183.
van Heesch, U., Avgeriou, P., 2011. Mature architecting-a survey about the reasoning process of professional architects. In: Proceedings of 2011 9th Working IEEE/IFIP Conference On Software Architecture (WICSA), pp. 260–269.
van Heesch, U., Avgeriou, P., Hilliard, R., 2012a. A documentation framework for architecture decisions. J. Syst. Softw. 85, 795–820.
van Heesch, U., Avgeriou, P., Hilliard, R., 2012b. Forces on architecture decisions-a viewpoint. In: Proceedings of 2012 Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), pp. 101–110.
van Heesch, U., Avgeriou, P., Tang, A., 2013. Does decision documentation help junior designers rationalize their decisions?-A comparative multiple-case study. J. Syst. Softw. 86, 1545–1565.
van Heesch, U., Eloranta, V.P., Avgeriou, P., Koskimies, K., Harrison, N., 2014. Decision-centric architecture reviews. Softw. IEEE 31, 69–76.
Venters, C.C., Griffiths, M.K., Holmes, V., Ward, R.R., Cooke, D.J., Huddersfield, U., et al., 2014. The Nebuchadnezzar effect: dreaming of sustainable software through sustainable software architectures. In: Proceedings of the Second Workshop on Sustainable Software for Science: Practice and Experiences.
Vliet, H.v., Avgeriou, P., Boer, R.d., Clerc, V., Farenhorst, R., Jansen, A., et al., 2009. The GRIFFIN project: lessons learned. In: Babar, M.A., Dingsøyr, T., Lago, P., Vliet, H.v. (Eds.), Software Architecture Knowledge Management: Theory and Practice, pp. 137–154.
Vogel, O., Arnold, I., Chughtai, A., Kehrer, T., 2011. Software Architecture. Springer.
Whittom, A., Roy, M., 2009. Considering participant motivation in knowledge management projects. J.Knowl. Manag. Pract. 10, 1–13.
Wolf, T., Dutoit, A.H., 2004. A rationale-based analysis tool. IASSE 4, 209–214.
Zdun, U., Capilla, R., Tran, H., Zimmermann, O., 2013. Making architectural design decisions sustainable: challenges, solutions, and lessons learned. IEEE Softw. 30, 46–53.
Zimmermann, O., Miksovic, C., Küster, J.M., 2012. Reference architecture, metamodel, and modeling principles for architectural knowledge management in information technology services. J. Syst. Softw. 85, 2014–2033.
Zimmermann, O., Zdun, U., Gschwind, T., Leymann, F., 2008. Combining pattern languages and reusable architectural decision models into a comprehensive and comprehensible design method. In: Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture, 2008. WICSA 2008, pp. 157–166.

Dr. Rafael capilla holds a Ph.D. in Computer Science and is associate professor in the department of Computer Science at Rey Juan Carlos University of Madrid (Spain). Currently, he heads the Software Architecture & Internet Technologies (SAIT) research group and the IEEE Computer Science Spanish chapter. Rafael is co-author of more than 80 peer-reviewed articles in international journals, conference proceedings and books and also co-author of the Springer book (2013) "Systems and Software Variability Management. Concepts, Tools, and Experiences". He has edited special issues in IEEE Software, Wiley JSEP and Springer REJ. His research interests focus on Software Architecture and Architecture Knowledge, Software Product Line Engineering, SOA & Cloud Computing, and Technical Debt among others. He also serves as regular reviewer in conferences and top journals and he co-organized several international workshops. He served as general chair in the XIV European Conference on Software Maintenance an Reengineering (2010). Rafael is IEEE Senior member. Contact him at rafael.capilla@urjc.es.

Dr. Anton Jansen is an architecture consultant at Philips Innovation services since 2015. Before this, he was senior scientist at ABB corporate research in the software architecture & usability (SARU) group in Västerås, Sweden. Between 2002 and 2009, he was a member of the Software Engineering and Architecture (SEARCH) research group at the University of Groningen, the Netherlands. He received a master of science degree in computing science in 2002, as well as a Ph.D. in Software Architecture in 2008 from the University of Groningen, the Netherlands. He has worked as a Ph.D. associate (2002–2006) on architectural decisions under supervision of Jan Bosch, and as a postdoc (2006–2008) on the Dutch Joint Academic and Commercial Quality Research & Development (Jacquard) project GRIFFIN. His research interests are software architecture and architectural knowledge. In his spare time, he likes to play multiplayer computer games, cycling, and reading books.

Dr. Antony Tang is an associate professor of computer science in the Swinburne University of Technology's Faculty of Science, Engineering, and Technology. His research interests include software architecture design reasoning, software development processes, software architecture, and knowledge engineering. Tang received a Ph.D. in information technology from the same university. He's a member of ACM and IEEE. Contact him at atang@swin.edu.au.

Dr. Paris Avgeriou is Professor of Software Engineering in the Johann Bernoulli Institute for Mathematics and Computer Science, University of Groningen, the Netherlands where he has led the Software Engineering research group since September 2006. Before joining Groningen, he was a post-doctoral Fellow of the European Research Consortium for Informatics and Mathematics (ERCIM). He has participated in a number of national and European research projects directly related to the European industry of Software-intensive systems. He has co-organized several international conferences and workshops (mainly at the International Conference on Software Engineering - ICSE). He sits on the editorial board of IEEE Software and Springer Transactions on Pattern Languages of Programming (TPLOP). He has edited special issues in IEEE Software, Elsevier Journal of Systems and Software and Springer TPLOP. He has published more than 130 peer-reviewed articles in international journals, conference proceedings and books. His research interests lie in the area of software architecture, with strong emphasis on architecture modeling, knowledge, evolution, patterns and link to requirements. He champions the evidence-based paradigm in Software Engineering research.

M.A. Babar is a Professor and Chair of Software Engineering in the School of Computer Science, the University of Adelaide, Australia. Prior to this, he was an Associate Professor (Reader) in Software Engineering at Lancaster University UK. Previously, he worked as a researcher and project leaders in different research centres in Ireland and Australia. His research projects have attracted funding from various agencies in Denmark, UK, Ireland, and Australia. He has authored/co-authored more than 150 peer-reviewed research papers at premier software engineering journals and conferences such as ACM Trans. on Software Engineering and Methods (TOSEM), IEEE Software, and ICSE. He has recently co-edited a book on Agile Architecting published by Morgan Kaufmann, Elsevier. He is a member of the steering committees of several international software engineering and architecture conferences such as WICSA, ECSA, and ICGSE. He regularly runs tutorials and gives talks on topics related to cloud computing, software architecture and empirical approaches at various international conferences. More information on Prof. M. Ali Babar can be found at http://malibabar.wordpress.com.