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