Managing Requirements Knowledge using Architectural Knowledge Management Approaches

Peng Liang
Wuhan University
liangp@sklse.org

Paris Avgeriou
University of Groningen
paris@cs.rug.nl

Chong Wang
Wuhan University
cwang@sklse.org

***Abstract*—The software architecture community has recently witnessed a paradigm shift towards managing architectural knowledge (AK), and consolidated research results have been achieved. Within the lifecycle of software development, requirements engineering (RE) and architecting are two closely related activities, and so are their associated knowledge. In this paper, we target specific RE problems and present knowledge-based approaches to tackle them by integrating concrete results from the AK community. This work can subsequently stimulate further research in knowledge-based RE, which may in turn feed its results back into software architecture.**

***Keywords*-knowledge management; requirements knowledge; architectural knowledge; requirements engineering**

I. Introduction

In the field of software architecture, there has been a paradigm shift from describing the outcome of architecting process (e.g., components and connectors) to documenting **Architectural Knowledge** (AK) [1], such as design decisions and rationale. In the SHARK workshop (http://www.shark-workshop.org/) series, progress has been reported on addressing architecting challenges by AK management, including AK models, application methods, and tools. Due to the interweaved relationship between RE and architecting process [2], there are fundamental commonalities (e.g., mostly text-based, decision intensive, significance of traceability etc.) and overlaps between AK and **Requirements Knowledge** (RK). Based on this premise, we argue that several RE problems can be partially addressed using AK management approaches.

The rest of this paper is organized as follows. Selected problems identified in RE literatures that can be potentially solved in a knowledge perspective are presented in Section II, with discussion of candidate solutions from AK management in Section III. The paper concludes with a future research agenda in Section IV.

II. Selected Problems in RE

We extensively reviewed the RE literatures published from 2000 to 2010, selecting four RE problems that can be potentially solved in a knowledge perspective based on our preference and understanding of RK. Note that the selection of the problems didn't follow a systematic approach, e.g., systematic literature review or mapping study, on which this work may get extended in the next step.

Insufficient requirements traceability. Traceability is critical to the success of a software project [3]. Current requirements traceability helps to scope the possible impact of requirements change, but it does not support automated reasoning about change, because the traceability links carry little semantic information [4].

Lack of systematic requirements reuse. The most strategic requirements reuse is in software product lines, in which the family's common requirements are collected in reusable templates. There are two key RE challenges for requirements reuse in product-line development: strategic techniques for domain analysis, and how to document requirements for product lines [5].

Lack of integration of RE activities. Most RE research focuses on individual RE problems, such as elicitation or traceability. As a result, the state of the art in RE is a collection of technologies that have been researched and evaluated in isolation, with little knowledge of how to effectively combine them [5].

Communication problem in distributed development. It is difficult to manage distributed requirements elicitation, specification, and negotiation due to the communication problem, and ill-defined requirements pose a high risk of project delay or failure [5].

III. Candidate Solutions

In this section, we present solutions from AK management to address the four RE problems. For each problem, we first present the current solutions in RE and discuss their inefficiencies. We then present how AK management approaches tackle similar problems, and how to apply the approaches in RE with potential issues of using AK management approaches in RE.

A. Insufficient Requirements Traceability

1) Current Approaches in RE: Requirements traceability has been identified as a significant quality factor that a system should possess. Most of existing work proposes various traceability models by following a heavyweight approach aligning with a full-fledged RE process. These

approaches are helpful in certain cases, but there are still issues: (1) lightweight requirements traceability methods are also desired, for example, to cope with rapid requirements change in agile processes; (2) semantic information about traceability links (e.g., the type of links) is needed to help distributed teams, who use different traceability models, to understand each other [3], and perform automated traceability reasoning (e.g., to detect conflicting requirements).

2) AK Management Approaches: In architecting process, similar problems arise. Various models for codifying AK are proposed either by industry or academia [6]. These models are composed of a set of AK concepts and the relationships between them (e.g., `Decision Topic` is *addressedby* `Alternative`). AK traceability management is challenging due to the heterogeneity of various AK models. We proposed an approach that employs semantic web technologies, e.g., RDF and OWL, for AK management [7]. First, the text in architecture documentation (source of most AK) is annotated into AK instances using an AK model (e.g., "*The programming language should be Java*" is annotated as an AK instance of AK concept `Architecture Design Decision`). Second, the annotated AK instances (in RDF) with the AK model (in OWL) are stored in an AK repository, in which various AK models can be integrated with the data (AK instance).

This approach can partially address the problems in current requirements traceability management. First, it provides a lightweight KM approach that codifies knowledge by annotation iteratively without following a specific architecting process. Second, it offers flexibility and scalability: the model can be easily changed and subsequently imported into the AK repository; AK instances annotated using different AK models can co-exist in a single AK repository. This facilitates AK traceability management in a central repository among multiple organizations who employ different AK models. Third, automatic traceability reasoning can be performed, since the data (AK instances) stored in an AK repository is semantically-enriched using RDF and OWL. For example, we can check that "*Each architecture Decision Topic is addressed by at least one Alternative design*". The capability of automatic reasoning relies on the expressiveness of semantic/annotated data. When advanced reasoning capabilities are required, more expressive semantic information is needed.

3) Application in RK Management: In RE, various methods (e.g., goal-oriented, value-based) have also been developed for requirements traceability management. The AK traceability management approach can fit well with requirements traceability management, in that requirements documentation is mostly text-based in current practice. Requirements documents can be annotated using various RK models for requirements traceability management, and the annotated RK can be stored and managed by a semantic data repository. This approach resolves the two aforementioned traceability problems: (1) It provides a lightweight and extensible approach to accommodate various requirements traceability models in a multiple site development context; (2) It provides semantic traceability information to support the understanding of requirements links and automated reasoning for the detection of e.g., conflicting requirements. The shortcoming is rather obvious: this approach is limited to be applied to KM based on formal models. In architecting practice, it was found that architects tend not to formalize (annotate) the AK, neither during architecture design, nor afterwards, since they consider knowledge formalization (annotation) a burden [8]. This problem can be possibly resolved by a similar solution to the Experience Factory Organization by Basili *et al*. Finally, process improvement and tool support are also critical for the acceptance of this approach.

B. Lack of Systematic Requirements Reuse

1) Current Approaches in RE: The reuse of various software artifacts on software design and implementation (e.g., code, functions, components, frameworks, patterns and models) has been extensively addressed. In software engineering, requirements reuse is regarded as a high level reuse activity, and can contribute significantly to the project success if it is applied effectively. As identified in section II, the most strategic form of requirements reuse is product lines, but two challenges exist: the lack of strategic techniques for analyzing domains, and methods to effectively document requirements for product lines. Feature models are commonly used to model the core features of a product line, but they quickly proliferate when used to model product-line instantiations, and therefore are difficult to manage. A promising but untested solution to this challenge is multi-level feature tree, which claims to be able to effectively manage the changes to the complex feature tree's structure generated by the feature model .

2) AK Management Approaches: In architecting process, methods with tool support for reusing AK have been proposed and validated in architecting activities, such as architecture analysis, evaluation, and documentation. Herold *et al.* tackle the problem of reusing architectural drivers in architecture analysis to support early requirements and architectural decisions [9]. They achieve this by combining goal-oriented requirements model with compositional architecture design model to make explicit the interactions between the early steps and artifacts in goal-oriented RE and architectural design, and then reuse architectural drivers (a type of AK). Babu *et al.* focus on application-generic AK that can be reused across application domains [10]. They present an ontology that captures general AK as an invaluable reusable asset for universal AK management. Babar and Gorton present tool support for capturing, presenting, and (re)using AK based on the PAKME AK model [11]. This tool can locate architecture design alternatives by keyword search

within the boundaries of the PAKME AK model.

All these methods for AK reuse are based on formal domain models, in which AK is formalized using domain concepts. On this basis, the activity of reuse can utilize the semantic information attached to the AK and query the AK repository to serve specific reuse purposes. For example, the semantic relationship (e.g., *positive/negative/noeffect*) between architecture patterns and quality attributes in architecture design (part of AK) can be reused to satisfy certain early requirements. The drawback of these methods is similar to the methods introduced in section III-A: they can only be applied to formal knowledge.

3) Application in RK Management: In RE for product lines, feature modeling is a kind of requirements modeling method, which focuses on the end-user visible characteristic of a system. As a conceptual model, feature models (and their corresponding RK) can be formally represented using ontology definition as described in section III-A. Organizations can define their specific RK ontologies (concepts, relationships, and constrains) for their specific reuse purposes. For example, to mitigate feature proliferation problem using multi-level feature trees, one can introduce three concepts in the RK ontology: `feature model`, `reference feature model`, and `referring feature model`. One can then define the *subClassOf* relationship from concepts `reference feature model` and `referring feature model` to `feature model`. The semantic constraints to the concept `reference feature model` (i.e., what deviations from it are allowed or disallowed) can also be formally represented in the ontology description. Based on the RK ontology and annotated RK instances, a reasoner can perform advanced queries which reveal explicit and implicit relationships among annotated RK instances, such as conflicting features. Furthermore, using semantic queries, the requirements that are common to all applications can be distinguished among alternative features in the RK repository, and can be further denoted as a commonality for requirements reuse in a software product line.

C. Lack of Integration of RE Activities

1) Current Approaches in RE: A typical example of this problem is that despite significant advances in requirements modeling and notations, there has been little work on how to interconnect various types of requirement models. Well-defined approaches to interrelate requirements goals, scenarios, data, functions, state-based behavior, and constraints are needed to address this fundamental problem [5]. Some initial work has been done in terms of a modeling theory incorporating the aforementioned requirements modeling elements [12], and product-management techniques for using various requirement techniques integrally [13]. However further research is needed on how to integrate requirements technologies in various RE activities, so that practitioners know how to apply individual technologies

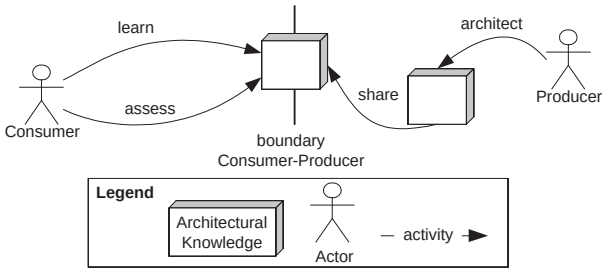

Figure 1. AK management in a consumer-producer perspective [1].

effectively and synergistically [5]. This problem is partially caused by the large number of heterogeneous requirement models, templates and specification languages employed in various RE activities that may hinder stakeholders from effectively communicating about a system. This has also been identified as a major factor for the discontinuity of information across different models in software lifecycle. In conclusion, bridging various requirements models and related RK across RE process is a major challenge.

2) AK Management Approaches: In architecting process, we look at AK management activities in a consumer-producer perspective (see Figure 1): the cube represents AK in various types, including *General* AK that helps architects to design a systems; *Design* AK, a collection of designs of a system; and *Reasoning* AK, a collection of reasoning information about a design. The arrows represent the AK consumption and production activities. The consumers and producers can be any stakeholders, such as architects and requirements engineers. We envision the AK management system (e.g., for using architectural decisions) as a knowledge grid that all stakeholders can access transparently. The AK grid provides AK integration by synthesizing multiple AK models into a common model and understanding a given subject (e.g., architectural decision) from different perspectives. Stakeholders working on this grid can manage their own AK and share part of their AK with (remote) counterparts in a collaborative social network. For the integration of architecting activities, we map AK management activities into architecting activities (i.e., architectural analysis, synthesis, evaluation). For instance, the AK management activities *Share* AK, *Learn* AK, *Search/Retrieve* AK, *Evaluate* AK, and *Distill* AK, are mapped to architectural evaluation. Based on the activity mapping within the AK grid, we can integrate heterogeneous AK models, which originate from different architecting activities, through the production and consumption of AK. The integration of architecting activities can be realized by the integration and communication of AK among stakeholders across architecting activities. This is also a flexible solution, as every organization can construct a specific AK model to adapt their individual architecting context.

3) Application in RK Management: The solution for the integration of architecting activities based on AK integration can be applied to RE in three steps. First, define RK management activities within RE activities, e.g., *Trace*, *Integrate*, *Evaluate* RK, and for each RK management activity, identify the types of RK (e.g., general, domain, and rationale RK) that the activities produce and consume. Second, classify and refine the types of RK identified in the previous step, and identify appropriate RK concepts belonging to these RK types (e.g., rationale RK can be composed of concepts `Goal`, `Softgoal`, `Business Driver`, etc.) and the relationships between concepts. Third, integrate and refine the RK concepts and their relationships from all the RK types, and come up with a common RK domain model. This model acts as the underlying model in the RK repository for the integration of RK, and further integration of RE activities in RE process. A major research challenge in this approach is how to integrate and harmonize the RK concepts and their relationships when inconsistencies appear. Also, the disadvantage of this solution is that it only applies to formal RK.

D. Communication Problem in Distributed Development

1) Current Approaches in RE: The RE activities require intensive communication from requirements elicitation down to requirements analysis, implementation, and testing. Software development in a distributed context increases the risk of the communication gaps as discussed in section II. To address this problem, Bhat *et al.* proposed a framework based on a people-process-technology paradigm that describes best practices for negotiating goals, culture, processes, and responsibilities across a global organization [14]. Damian *et al.* investigated how to use and combine media technologies to facilitate requirements negotiations and quality agreements [15]. Sinha *et al.* developed an Eclipse-based tool for distributed requirements engineering collaboration by adding collaboration capabilities to the traditional requirements management tools [16]. These solutions address partially the challenges in the perspective of best practices, communication techniques, and tool support respectively for distributed RE. However, distributed projects need more investment upfront to make requirements precise for correct interpretation and perception by different stakeholders, and to address the problem of ineffective knowledge sharing and communication in a distributed context.

2) AK Management Approaches: For the last five years, the Griffin (http://griffin.cs.vu.nl/) project has been working on methods, tools, and techniques to manage AK with a special focus on AK sharing and reusing among distributed organizations. We envision AK sharing and reusing in a grid environment as described in section III-C2. A number of use cases for using architecture decisions (most important type of AK) has been identified, an AK core model to harmonize the different AK models has been constructed, and a method on how to select a high quality core model for AK sharing is proposed and validated [17]. Using the core model and various AK models, AK can be described precisely to ease the (re)using and sharing with correct interpretation. Furthermore, best practices on AK management for GSD have been identified and validated, e.g., establish a shared AK repository to obtain common understanding of architecture across development sites. Moreover, a web-based AK sharing portal was developed to share AK among distributed teams using hybrid strategy [18], which is more suitable in an industrial environment. Another tool, the Knowledge Architect emphasizes on the formal AK management based on semantic data repository, including a set of interconnected utilities for AK annotation, integration, validation, and translation [7]. It is capable of defining precise semantics of AK and using them to communicate the AK effectively, e.g., share AK for quantitative architectural analysis.

3) Application in RK Management: Both personalization and codification KM strategy are valuable in various RE activities, which holds true for distributed RE context. To manage distributed requirements elicitation, analysis, modeling, and negotiation, a common understanding on requirements among distributed organizations should be established, either through semantic annotation (for formal RK), document classification (for documented RK), or personal communications (for personalized RK) about the requirements. An RK grid can contribute to the management of formal RK. In addition, a requirements sharing portal can ease the communication of requirements through lightweight RK management methods, such as classified documents for browsing and experts yellow page to find out who knows what. For formal RK communication, a core RK model should be constructed by following the steps in section III-C3, while concept mappings between the core RK model and specific RK models should be defined by RE experts. These solutions can partially mitigate the problems of requirements communication in distributed development: formal RK models can cope with imprecise requirements that may lead to misunderstanding; a central RK grid and a web portal can deal with ineffective requirements communication. Except for the methods, techniques, and tool support, a healthy culture of collaboration should also exist in order to promote the communication and make distributed projects work.

IV. Conclusions and Research Agenda

This work makes a first step in investigating how the research outcomes from AK management can be applied to RE. The major contribution of this paper is proposition of candidate solutions for four RE problems in a KM perspective. The envisaged usage of KM approaches in RE is discussed based on the research results from AK management. Some solutions can be directly reused in RE, while others need to be extended to accommodate the

specific characteristics of RE. We recognize that knowledge formalization is a hard prerequisite for applying the proposed approaches, which may limit the application of the approaches, and we argue that documented and tacit knowledge in RK management need further attention.

Besides the future research work mentioned in each RE problem, the following research work are scheduled: (1) Systematic approaches to assist RE process through RK management in three aspects: KM strategies, KM activities, and RK types and concepts; (2) How personalization KM strategy can fit various RE processes and activities, for example, in agile development when requirements change frequently and codification strategy is not suitable; moreover more insight is needed on the balance between personalization and codification strategy in RK management (i.e., a hybrid strategy); (3) Evaluation of the ROI (cost vs. benefit) of RK management in productivity, quality, and risk management for software development; (4) Due to the interweaved relationship between RE and architecting process, the overlap and coordination between RK and AK needs further investigation for better cooperation and smooth transition from requirements to architecture design.

KM is beneficial but also costly, and resources, time, and effort are required before benefits become visible. This situation is also true for RK management since project managers who focus on completing the current project on time, but not helping the next project succeed, often consider RK management a burden. It is crucial for the acceptance of RK management in practice to leverage the benefit and cost, e.g., through lightweight and social-web based KM methods (e.g., wiki and semantic wiki) in which RK can be codified collaboratively and shared as soon as they are available.

ACKNOWLEDGEMENTS

This research has been partially supported by the Natural Science Foundation of China (NSFC) under Grant No. 60950110352, STAND: Semantic-enabled collaboration Towards Analysis, Negotiation and Documentation on distributed requirements engineering.

REFERENCES

[1] P. Lago and P. Avgeriou, "First workshop on sharing and reusing architectural knowledge," *ACM SIGSOFT Soft. Eng. Notes*, vol. 31, no. 5, pp. 32–36, 2006.

[2] B. Nuseibeh, "Weaving together requirements and architectures," *IEEE Soft.*, vol. 34, no. 3, pp. 115–117, 2001.

[3] J. Cleland-Huang, A. Dekhtyar, J. Hayes, G. Antoniol, B. Berenbach, A. Eyged, S. Ferguson, J. Maletic, and A. Zisman, "Grand challenges in traceability," Center of Excellence for Traceability, Tech. Rep. COET-GCT-06-01-0.9, 2006.

[4] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap," *Proc. of the 22nd Int. Conf. on Future of Soft. Eng. (FOSE)*, pp. 35–46, 2000.

[5] B. Cheng and J. Atlee, "Research directions in requirements engineering," *Proc. of the 29th Int. Conf. on Soft. Eng. (ICSE)*, pp. 285–303, 2007.

[6] M. Shahin, P. Liang, and M. R. Khayyambashi, "Architectural design decision: Existing models and tools," *Proc. of the 8th IEEE/IFIP Working Conf. on Soft. Architecture (WICSA)*, pp. 293–296, 2009.

[7] P. Liang, A. Jansen, and P. Avgeriou, "Collaborative software architecting through knowledge sharing," *Collaborative Software Engineering, Mistrik, I. et al., Eds*, pp. 343–368, 2010.

[8] M. Babar, R. de Boer, T. Dingsoyr, and R. Farenhorst, "Architectural knowledge management strategies: approaches in research and industry," *Proc. of the 2nd Int. Workshop on SHAring and Reusing architectural Knowledge-Architecture, rationale, and Design Intent (SHARK/ADI)*, pp. 2–8, 2007.

[9] S. Herold, A. Metzger, A. Rausch, and H. Stallbaum, "Towards bridging the gap between goal-oriented requirements engineering and compositional architecture development," *Proc. of the 2nd Int. Workshop on SHAring and Reusing architectural Knowledge-Architecture, rationale, and Design Intent (SHARK/ADI)*, pp. 15–20, 2007.

[10] L. Babu, S. Ramaiah, T. Prabhakar, and D. Rambabu, "ArchVoc - towards an ontology for software architecture," *Proc. of the 2nd Int. Workshop on SHAring and Reusing architectural Knowledge-Architecture, rationale, and Design Intent (SHARK/ADI)*, pp. 9–14, 2007.

[11] M. A. Babar and I. Gorton, "A tool for managing software architecture knowledge," *Proc. of the 2nd Int. Workshop on SHAring and Reusing architectural Knowledge-Architecture, rationale, and Design Intent (SHARK/ADI)*, pp. 21–26, 2007.

[12] M. Broy, "The 'grand challenge' in informatics: engineering software-intensive systems," *IEEE Comp.*, vol. 39, no. 10, pp. 72–80, 2006.

[13] C. Ebert, "Understanding the product life cycle: four key requirements engineering techniques," *IEEE Soft.*, vol. 23, no. 3, pp. 19–25, 2006.

[14] J. Bhat, M. Gupta, and S. Murthy, "Overcoming requirements engineering challenges: lessons from offshore outsourcing," *IEEE Soft.*, vol. 23, no. 5, pp. 38–44, 2006.

[15] D. Damian, F. Lanubile, and T. Mallardo, "An empirical study of the impact of asynchronous discussions on remote synchronous requirements meetings," *Proc. of the 9th Int. Conf. on Fund. Appr. to Soft. Eng. (FASE)*, pp. 155–169, 2006.

[16] V. Sinha, B. Sengupta, and S. Chandra, "Enabling collaboration in distributed requirements management," *IEEE Soft.*, vol. 23, no. 5, pp. 52–61, 2006.

[17] P. Liang, A. Jansen, and P. Avgeriou, "Selecting a high-quality central model for sharing architectural knowledge," *Proc. of the 8th Int. Conf. on Quality Software (QSIC)*, pp. 357–365, 2008.

[18] R. Farenhorst, P. Lago, and H. van Vliet, "EAGLE: effective tool support for sharing architectural knowledge," *Int. J. of Cooperative Inf. Sys.*, vol. 16, no. 3/4, pp. 413–437, 2007.