Recovering Software Architectural Knowledge from Documentation using Conceptual Model

Mojtaba Shahin[1,2], Peng Liang[1]*, Zengyang Li[3]
[1] State Key Lab of Software Engineering, Computer School, Wuhan University
[2] Department of Computer Engineering, Neyriz Branch, Islamic Azad University
[3] Department of Computing Science, University of Groningen
mojtabashahin@gmail.com, liangp@sklse.org, zengyang.li@rug.nl

***Abstract*—Software architectural knowledge (AK) is the integrated representation of the software architecture (SA) of a software-intensive system, the architectural design decisions, and the external context/environment. AK annotation using AK conceptual model is used to recover formal AK from SA documentation, including architecture design as well as the design decisions, rationale, context, and other factors that together determine architecture solutions. But there is no evidence on how architects, especially junior architects, understand and annotate SA documents and recover formal AK from the documents using an AK model, which is right the case when a new architect jumps into a project, trying to understand the SA documents created by previous architects. This paper first presents AKRCM (AK Recovery using Conceptual Model) approach for recovering AK from SA documents. Second, we conduct a descriptive study using experiment to investigate how junior architects annotate SA documents and recover AK using AKRCM approach. We found that an AK conceptual model is beneficial for junior architects to get a fair understanding of SA documents, and to recover better-quality AK from SA documents.**

Keywords - architectural knowledge; knowledge recovery; knowledge annotation; junior architect; conceptual model

I. INTRODUCTION

Software architecture is considered of paramount importance to the software development life cycle [1]. It is a key artifact for the early analysis of the system, as it facilitates stakeholders' communication and understanding, and drives both system construction and evolution. Software architecting is meanwhile a knowledge-intensive activity, in which a large amount of knowledge is being continuously produced and consumed. In the field of software architecture (SA), a paradigm shift has occurred from describing the outcome of architecting process to describing the Architectural Knowledge (AK) created and (re)used during architecting process, including architecture design as well as the design decisions, rationale, assumptions, context, and other factors that together determine architecture solutions [4]. Traditional approaches to documenting SA are limited to capture this knowledge and partially result in AK vaporization. The architecturally significant information is lost during architecting process, especially in architecture evaluation and maintenance activities. This situation leads to many severe problems, such as expensive system evolution, lack of stakeholders' communication, and limited reusability of architecture [2]. The SA community, both in industry and academia, is therefore gradually acknowledging that capturing and recovering explicit and valuable AK will lead to the improvement of architecting process, and of architecture documentation itself [3].

In knowledge management, a distinction is often made between two types of knowledge: implicit and explicit knowledge [7]. Implicit (or tacit) knowledge is the knowledge residing in people's heads, whereas explicit knowledge is the knowledge which has been codified in certain form (e.g., a document or a model). Two forms of explicit knowledge can be discerned: documented and formal knowledge. Documented knowledge is explicit knowledge which is expressed in natural language or images in documents. Typical examples of documented AK are Word documents that contain architecture descriptions. Formal knowledge is explicit knowledge codified using a formal language or conceptual model of which the exact semantics are defined. Typical examples of formal AK models include AK ontologies [5] or AK conceptual models [8] that formally define the AK concepts and their relationships, and aim at providing a common language for unambiguous interpretation by stakeholders. In comparison with documented AK, formal AK provides a clearer description of AK entities and their relationships in SA documents supported by conceptual model. This paper focuses on the recovery of formal AK that is codified in an AK conceptual model.

Formal AK recovery from SA documentation is critical to various architecting activities, e.g., architecture evaluation and maintenance. Jansen *et al.* stated that SA is often documented after most of AK have been produced and used [6]. Therefore, a large amount of AK may get lost during architecting process and SA documents get quickly outdated, also the trust to SA documents is decreased. To alleviate these problems, an architect needs to recover formal AK from SA documents and identify the gap in AK (e.g., an *architectural design decision* without supporting *design rationale*) with the support of formal AK management tool (e.g., Knowledge Architect [13]). In a human perspective, AK recovery activity might take place in the following situations: (1) AK recovery for understanding existing design and making a new design by architects. (2) AK recovery for communicating architecture design through AK sharing [11]. In both cases, architects need to first recover AK from SA documentation.

An annotation is generally a comment, note, explanation, or other types of external remarks that can be attached to an object, i.e., a sentence in a SA document. Different from general text annotation, formal AK annotation uses controlled terms (i.e., the AK concepts from a specific AK model) to annotate the SA documents. We name this approach AKRCM

* Corresponding author

This work is partially sponsored by the NSFC under Grant No. 61170025 and AFR-Luxembourg under the contract No. 895528.

(AK Recovery using Conceptual Model). A sample of AK annotation using AKRCM approach in a SA document is shown in Figure 1. Note that practical AK annotation is generally made by tools, e.g., Word plug-in [13]. The snapshot in Figure 1 just shows how the subjects in this experiment annotate the SA document (See Section III). AK recovery requires a lot of design experiences and domain expertise of the architects that normally junior architects don't have. The other characteristic of junior architects is that they need more time to get familiar with a new system and may overlook some key architecture information when they jump into a new project. To the best of our knowledge, there is currently no any evidence on how software architects, especially junior architects, annotate the text in SA documents and recover formal AK using an AK conceptual model.

To this end, we conducted an experiment: asking the participants (a group of junior architects, in our setting, the master students following a SA course) to recover formal AK from a SA document using AKRCM approach based on their understanding of the SA document with the support of a specific AK model. We employ LOFAR AK conceptual model (the concepts and their relationships are shown in Figure 2 and detailed in Section III) to annotate and recover AK from an LOFAR SA document. The reason for choosing LOFAR AK model is that the SA document used for this experiment is from LOFAR (Low Frequency Array) project undertaken by Astron, the Dutch Astronomy Institute, which is involved in the development of large software-intensive systems used for astronomy research. It would be easier for participants to recover most AK using this model. In this work, we first investigate how junior architects understand AK concepts in LOFAR AK model, and how they annotate the SA document and recover AK using this conceptual model. Second, the effectiveness of an AK conceptual model in AK recovery is analyzed and evaluated.

The reminder of this paper is organized as a follows: Section II presents the challenge of AK recovery and research questions we try to answer by the experiment. Section III describes the experimental setup on AK recovery, with the threats to the validity of the experimental results. To answer the research questions, the experiment results are reported and analyzed in Section IV. Section V discusses the threats to the validity of this study. Section VI presents the lessons learned according to the experiment results. Section VII discusses related work on recovering AK. Section VIII concludes this paper with future work directions.

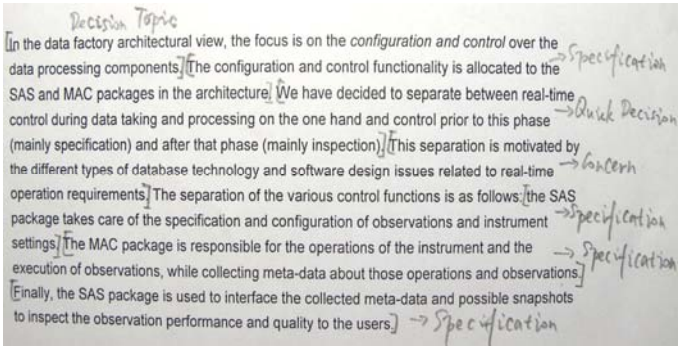

Figure 1. Snapshot of AK annotations in a SA document

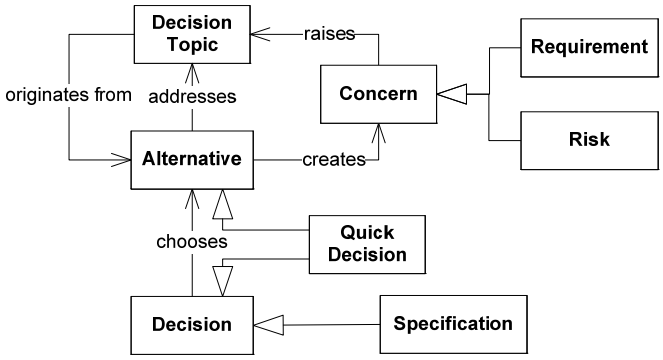

Figure 2. LOFAR AK conceptual model

II. GOAL AND CHALLENGE OF AK RECOVERY

The goal of this experiment is to present the findings on formal AK recovery using an AK model by junior architects. The next step, i.e., how to use recovered AK in SA activities, such as architecture evolution, is scheduled as the future work.

The challenge this paper tries to address is how an AK conceptual model can help junior architects to recover AK from SA documents. This challenge can be further detailed in three research questions:

RQ1: How junior architects understand AK concepts in an AK conceptual model? A domain-specific AK conceptual model is normally co-constructed by experienced architects and domain experts, but junior architects with few background and experiences may have different understanding to the same AK concept.

RQ2: How junior architects annotate SA documents and recover AK? Experienced architects may recover AK from SA documents by employing their tacit knowledge (e.g., experience on architecture patterns, domain knowledge, architecture reasoning knowledge, etc.) in the recovery process [6]. In other words, AK recovery results may heavily depend on the skills and experiences of software architects, which junior architects normally lack of.

RQ3: Is the AK conceptual model helpful for junior architects to recover better-quality AK from SA documents? To answer this question, we try to compare the results of AK recovery by junior architects and experienced architects to investigate the effectiveness of an AK conceptual model in AK recovery.

III. EXPERIMENT SETUP FOR AK RECOVERY

We conducted a descriptive study using experiment to answer the research questions in Section II. Descriptive study is an observational study that tries to determine the distribution of certain characteristics or attributes. The focus of a descriptive study is not about why the observed distribution exists, rather what it is [12].

At Wuhan University, SA course is introduced as an optional course in the curriculum for the first-year master students major in software engineering. 36 students participated in this course in the academic year of 2010-2011. During this course, the students learned fundamental knowledge about architecture design, process, and evaluation by conducting an

architecture course project. We regard these students as junior architects in this experimental context since Host *et al.* found that students are suitable replacements for industry professionals if performing small tasks of judgment [14]. The task of this experiment is to ask students to annotate one page (about 400 words) of corpus selected from the LOFAR SA document[1] using LOFAR AK conceptual model, which are both produced in an industrial project. A snapshot of the AK annotations by annotators (i.e., the subjects in this experiment) is shown in Figure 1. The experiment steps are following:

- **Step1**: hand out the printed text in one page selected from a SA document to the students (i.e., the subjects);
- **Step2**: present a short tutorial in 10 minutes on the AK conceptual model (i.e., LOFAR AK model) that are used for the AK annotation with AK annotation examples;
- **Step3**: ask the subjects to annotate the printed text (using pen) with the AK concepts of the introduced AK conceptual model within limited time (15 minutes).

A short description about the LOFAR AK model shown in Figure 2: A *Concern* is an interest to the systems development. A *Requirement* represents something demanded from the system, which is a specific type of *Concern*. A *Risk* is also a special type of *Concern*, which expresses a potential hazard. A *Decision Topic* is a certain problem that a *Decision* should be made to solve it. *Alternative* concept represents one or more potential candidate solutions to address the described problem. A *Decision* is chosen from the *Alternatives* to address the described *Decision Topic*. If only one *Alternative* is proposed and no (sufficient) motivation is given why this *Alternative* is chosen, we then have a *Quick Decision*. A *Specification* is the lowest level architectural *Design Decision* that is being made. The refinement process of architecture documentation is finished when it comes to *Specifications*. The concepts *Concern*, *Decision*, *Alternative* and the like are not specific concepts to the LOFAR AK model. These concepts exist in most AK conceptual models although different AK models might employ these concepts with different terms and with different relationships [9].

IV. EXPERIMENT RESULTS AND ANALYSIS

There were 36 subjects (students), and 35 AK annotation results were returned, in which 33 results were valid with 2 invalid results: one was no any AK annotations, and the other one provided annotated text without annotation concepts. The SA document (in one page) is composed of 16 sentences and 1 diagram, which are potential annotatable AK entities. Some subjects also annotated part of a sentence as an AK entity or annotated more than one sentences as an AK entity, in which the unit of AK annotation can be a phrase, a paragraph, or a sentence. In this section, we try to answer the research questions presented in Section II.

RQ1: How junior architects understand AK concepts in an AK conceptual model?

As shown in Table 1, different software architects may annotate same unit of text with different AK concepts, because they have different understanding about specific AK concept, or some AK concepts are actually overlapped with each other, e.g., *Specification* is a `subClassOf` *Decision*, *Decision Topic* is `raisedFrom` *Requirement*, etc. One of best practices to resolve this issue is to combine two AK concepts into one AK concept when most of architects annotate the same unit using the two AK concepts alternatively. For example, in LOFRA AK model, the concept *Requirement* is a `subClassOf` *Concern* and most of junior architects annotate the same unit using these two AK concepts alternatively. Table 1 shows that most of junior architects (78.8%) annotate the Sentence 1 and 2 with *Requirement* and *Concern* alternatively. Therefore, it is suggested that these two concepts are combined as one concept, and the merging of AK concepts is considered as a post-processing to annotated AK. Note that, the focus of this work tries to investigate how junior architects recover AK with the support of an AK model, but not to evaluate the effectiveness of an AK model for AK annotation. As a side-effect of AK recovery activity, the AK model can be improved according to the AK annotations, which is not discussed in details in this paper.

RQ2: How junior architects annotate SA document and recover AK?

Junior architects annotate the text in different granularity, e.g., one junior architect annotates two sentences as a *Requirement*, and the other one annotates these two sentences as two *Requirements*. For example, Table 1 shows that Subject 26 annotated Sentence 1 and 2 as one AK entity of *Requirement*, while Subject 16 annotates the two sentences as two different AK entities of *Requirement* and *Specification* respectively. Most of junior architects did not consider the figure (and related caption) in the SA document as an AK entity. Figure 3 shows what kind of text units (phrase, sentence, and paragraph) are mostly annotated by junior architects. The result shows that *sentence* is mostly used as the unit of AK annotation. We assume that this is partially because sentence is a unit that has a logical-sound meaning.

RQ3: Is the AK model helpful for junior architects to recover better-quality AK from SA documents?

A consistent annotation in this experiment refers to the AK concept used by the most of subjects to annotate AK entities. Figure 4 and Table 1 show how junior architects annotate the 16 sentences. For example, 9, 16, 1, and 7 out of 33 subjects annotated Sentence 1 as *Requirement*, *Concern*, *Decision Topic*, and *No-AK entity* (No-AK entity means that subjects did not consider the sentence as any AK concept) respectively. Therefore, we regard that junior architects get a consistent AK annotation of Sentence 1 as *Concern*, which is listed in the column 4 of Table 2.

To evaluate the quality of AK annotation and recovery results by junior architects, the results by junior architects are compared to the AK recovery results by experienced architects. We asked two LOFAR architecture experts to annotate the same SA corpus. Table 2 (columns 2 and 3) shows that the annotation results by LOFAR experts 1 and 2 are largely overlapped with each others, i.e., **62%** of AK annotations by experts 1 and 2 are similar (in yellow rows). The comparison of the annotation results between junior architects (column 4 of Table 2) and two experts (columns 2 and 3 of Table 2) shows that **57%** of AK annotations by them are identical (in

[1] The corpus of the LOFAR SA document is available at http://www.cs.vu.nl/~liangp/project/AKRv/LOFARdoc.pdf

yellow rows). The comparison result supports our hypothesis that a pre-defined AK model (e.g., the LOFAR AK model in this experiment) has a very positive impact to help junior architects to get a fair understanding of SA documents and recover better-quality AK.

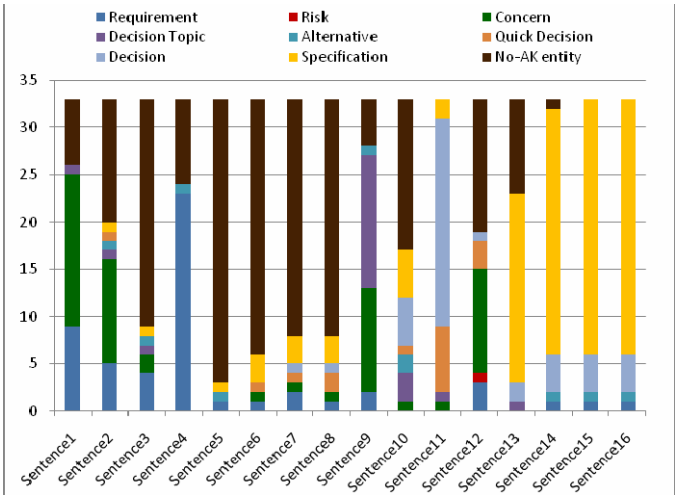

Figure 4. Consistent annotations with the support of an AK conceptual model

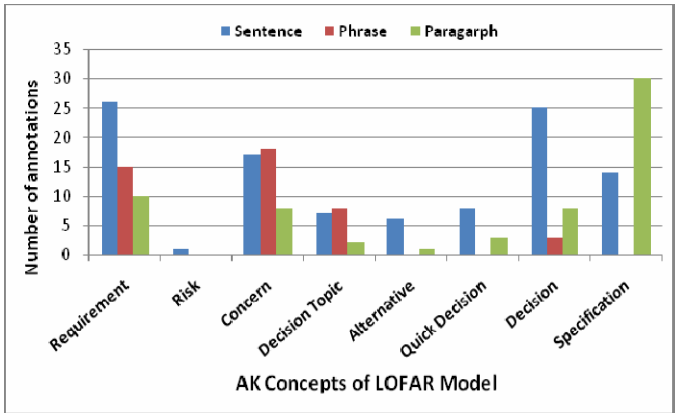

Figure 3. Units of AK entities annotated in LOFAR AK concepts by junior architects

As shown in Table 2, the LOFAR architecture experts annotated more AK entities (e.g., *Requirement*, *Quick Decision* in Sentence 5~8) than junior architects did. The implication of this difference is that the AK conceptual model can do help junior architects to understand/annotate/recover AK in SA documents, but domain knowledge about projects is also necessary/helpful and sometimes critical to recover AK.

Table 1. AK annotation results by junior architects

	Sentence															
Subject	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**	**S10**	**S11**	**S12**	**S13**	**S14**	**S15**	**S16**
Subject 01				R			R	QD	DT	QD	D		S			
Subject 02	R			R					C		D		S			
Subject 03	R								C	DT	S		D			
Subject 04	C			R					DT		D	QD	S			
Subject 05	R	A				QD			DT	D				A	A	A
Subject 06		C		R			D				QD				S	
Subject 07	DT			R					C		D	D	S			
Subject 08	R								C	DT	QD			S		
Subject 09	C		R			S			A		QD		S			
Subject 10	C			R					DT	A	D	C	S			
Subject 11	C			R					DT		D			S		
Subject 12	C			R					DT	A	D		S			
Subject 13	C	C		R					C	C	QD			S		
Subject 14				R					C		C	Rs	S			
Subject 15	R			R					DT		QD	R	S			
Subject 16	R	S		R					DT	S	QD	R	S			
Subject 17	C									S	S		DT	D		
Subject 18	C			R							D			S		
Subject 19									R	S	D	C	S			
Subject 20	C								C		D	C	S			
Subject 21									C		D		S			
Subject 22	C			R					DT		D	C		S	S	S
Subject 23	C					S			DT	D	D	R		R	R	R
Subject 24	C		R						C		D		S			
Subject 25	R	C	DT	R	S				R	S	D	C	S			
Subject 26	R					S					D	C	S			
Subject 27	C		R						DT		D		S			
Subject 28	C		R						DT	D	QD	C	S			
Subject 29				R					DT		D		S			
Subject 30	C	QD		R					C	S	DT	C		S		

Subject 31				**R**		**C**			**DT**		**D**	**C**	**S**
Subject 32	**R**	**R**		**R**		**R**		**R**		**D**	**D**	**C**		**D**
Subject 33	**C**		**S**	**R**					**C**	**D**	**D**	**C**	**D**

Requirement (R), Concern (C), Decision Topic (DT), Quick Decision (QD), Decision (D), Alternative (A), Risk (Rs), Specification (S)

Table 2. Comparison of AK annotations between LOFAR experts and junior architects

	LOFAR Expert 1	**LOFAR Expert 2**	**Consistent Annotations by Junior architects**
Sentence 1	Concern	Concern	Concern
Sentence 2	Concern	Requirement	Concern
Sentence 3	No-AK entity	No-AK entity	No-AK entity
Sentence 4	Requirement	Requirement	Requirement
Sentence 5	No-AK entity	Quick Decision	No-AK entity
Sentence 6	No-AK entity	Quick Decision	No-AK entity
Sentence 7	Requirement	No-AK entity	No-AK entity
Sentence 8	Requirement	No-AK entity	No-AK entity
Sentence 9	Decision Topic	Decision Topic	Decision Topic
Sentence 10	Specification	Quick Decision	No-AK entity
Sentence 11	Quick Decision	Quick Decision/Decision	Decision
Sentence 12	Concern	Concern	Concern
Sentence 13	No-AK entity	No-AK entity	Specification
Sentence 14	Specification	Specification	Specification
Sentence 15	Specification	Specification	Specification
Sentence 16	Specification	Specification	Specification

V. THREATS TO VALIDITY

Due to the limitations of this experiment, there are several threats to the validity of the experiment results:

The size of the SA document used in this experiment is relatively small due to the time limitation of the experiment since this experiment was conducted during a two-hour course session. We plan to extend the size of the experimental SA document to cover all the LOFAR AK concepts.

In this experiment, we use one AK model (i.e., LOFAR AK model). The LOFAR model is a specific AK model for the SA documentation of our industrial partner, and may not cover the AK concepts in other SA documents. We plan to repeat this experiment with more AK models, and the general AK core model proposed in [8], in order to investigate how junior architects annotate AK using various AK models, and what kind of model is more cost-effective for AK recovery.

The third threat of validity is related to the selected SA document, which is from the domain of astronomy research, and the subjects of this experiment are junior architects who are not the experts in that domain. Therefore, the subjects (students) may have some difficulties to understand correctly the SA document of the LOFAR system (e.g., this SA document may include specific and domain information which is only understandable after special training).

VI. LESSONS LEARNED

The experiences and issues during applying AK recovery method using AK conceptual model (AKRCM) are discussed:

(1) The AK conceptual model used for AK recovery is not fixed. An AK conceptual model can evolve (e.g., merging, splitting, modifying, and removing AK concepts) according to users annotations, in order to get a wider consensus on annotation results and get more AK being recovered.

(2) The AK recovery method using conceptual model focuses on explicit AK (i.e., from documented AK to formal AK) in SA documents without considering tacit AK (e.g., implicit design rationale and decisions). The method for recovering architectural design decisions (an important type of AK) proposed in [6] can be complementary to our method since the two methods cover both explicit and tacit knowledge in architecture design, documentation, and maintenance.

(3) In current software development practices, most of software artifacts are recorded and managed in the form of file-based documents besides SA documents, including requirements documents, test specifications, etc. The preliminary results of AK recovery from SA documents suggest that knowledge recovery using specific conceptual models can be extended and employed in the formal knowledge recovery of other file-based software artifacts. Furthermore, we propose that software knowledge recovery using conceptual models, as part of knowledge reengineering activity in software development, can be synthesized as a best practice in reverse engineering of software-intensive systems.

VII. RELATED WORK

Capturing and recovering AK are two complementary ways to record AK during architecting process. Capturing AK is different from recovering AK in that AK capture normally

takes places during architecture design activity, while AK recovery is an activity after the fact [6]. ADDRA (Architectural Design Decision Recovery Approach) is an approach to recover AK especially architectural design decisions and document them [6]. Archium conceptual model is used as input in ADDRA to document the recovered AK. The concern of this approach is that it is heavily based on tacit knowledge of original architect to recover AK. Roeller *et al.* proposed an approach to recover architectural assumptions (the architectural design decisions that are implicit and undocumented) from existing software artifacts of software products [10]. The approach, RAAM (Recovering Architectural assumptions Method), uses *documentation*, *financial reports*, *free interviews*, *version control*, and *source code* as inputs, and then produces a list of assumptions.

ADDRA and RAAM are both heavyweight approaches that try to recover tacit knowledge from existing SA documents and transform them into explicit knowledge. The AK recovery method in our work is based on annotations of existing SA documents to transform the documented AK to formal AK, which is more meaningful than documented AK for understanding and communicating architecture design, and facilitates AK reasoning for e.g., design maturity assessment.

There are also some work on recovering implicit AK (e.g., architecture layers and externally visible features) at source code level using knowledge annotations, e.g., in [15]. The advantage of this method is that it does not mandate a fixed conceptual model for knowledge annotation in advance, but employs an iterative process to refine flexible annotation types (i.e., concepts in AK models), which introduces a viable solution for iteratively refining AK models using the results of AK annotations (see the point 2 of the future work).

VIII. Conclusions and Future Work

In this paper, we conducted a descriptive study using experiment to investigate how an AK conceptual model can help junior architects to recover AK from existing SA documents in order to further support other architecting activities. Experiment results indicate that an AK model can do help junior architects in AK recovery in two aspects: (1) it helps junior architects get a fair understanding of SA documents and recover better-quality AK; (2) it has a very positive impact to junior architects to make AK annotation.

Based on the experiment results and analysis, we outline the promising future work in several points: (1) Is an AK conceptual model useful for AK recovery by experienced/expert architects even if little impact is identified to them? (2) Can recovered AK and unrecovered AK (i.e., No-AK entities) helps to refine the domain-specific AK conceptual model (e.g., add, remove, or modify AK concepts according to the AK annotations)? (3) How architects are going to annotate SA documents, if an AK conceptual model is not presented to them? AK annotation without following an AK conceptual model is called AK tagging, which produces various AK tags we can not expect since architects can use whatever tags they like, including domain-independent AK tags (e.g., *Design Decision*, *Pros*, and *Cons*) and domain-dependent AK tags (e.g., pattern tags, function-related tags, performance, security, etc.). It is useful to investigate how the two methods (i.e., AK annotation and tagging) can be combined to achieve better AK recovery results. (4) Conduct this experiment in an industrial context with experienced architects and industrial-size SA documents to understand the challenges of AK recovery that architects face in architecting process. (5) Can this AK recovery method using conceptual model be employed and beneficial to other knowledge recovery activities in software development, and further facilitate reverse engineering of software-intensive systems? For example, requirements rationale knowledge recovery from requirements specifications to support evolution of requirements [16].

References

[1] Bass, L., Clements, P., and Kazman, R., 2003. Software Architecture in Practice, 2nd edition. Addison-Wesley Professional.

[2] Jansen, A. and Bosch, J., 2005. Software architecture as a set of architectural design decisions. In: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA), pages 109-120.

[3] Lago, P. and Avgeriou, P., 2006. First workshop on sharing and reusing architectural knowledge. *ACM SIGSOFT Software Engineering Notes*, 31(5):32-36.

[4] Kruchten, P., Lago, P., van Vliet, H., and Wolf, T., 2005. Building up and exploiting architectural knowledge. In: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA), pages 291-292.

[5] Kruchten, P., 2004. An ontology of architectural design decisions in software intensive systems. In Proceedings of the 2nd Groningen Workshop on Software Variability Management (SVM), pages 54-61.

[6] Jansen, A., Bosch, J., and Avgeriou, P., 2008. Documenting after the fact: recovering architectural design decisions. *Journal of Systems and Software*, 81(4):536-557.

[7] Nonaka, I. and Takeuchi, H., 1995. The Knowledge-Creating Company, How Japanese Companies Create the Dynamics of Innovation. Oxford University Press.

[8] de Boer, R., Farenhorst, R., Lago, P., van Vliet, H., Clerc, V., and Jansen, A., 2007. Architectural knowledge: Getting to the core. In: Proceedings of the 3rd International Conference on the Quality of Software-Architectures (QoSA), pages 197-214.

[9] Shahin, M., Liang, P., and Mohammad, R. K., 2009. Architectural design decision: existing models and tools. In: Proceedings of the 8th Working IEEE/IFIP Conference on Software Architecture (WICSA), pages 293-296.

[10] Roeller, R., Lago, P., and van Vliet, H., 2006. Recovering architectural assumptions. *Journal of Systems and Software*, 79(4):552-573.

[11] Liang, P., Jansen, A., and Avgeriou, P., 2009. Sharing architecture knowledge through models: quality and cost. *The Knowledge Engineering Review*, 24(3):225-244.

[12] Wohlin, C., Host, M., and Henningsson, K., 2003. Empirical research methods in software engineering. Empirical Methods and Studies in Software Engineering, Springer, pages 145-165.

[13] Liang, P., Jansen, A., and Avgeriou, P., 2009. Knowledge Architect: A Tool Suite for Managing Software Architecture Knowledge. Technical report RUG-SEARCH-09-L01, SEARCH, University of Groningen, February 2009.

[14] Host, M., Regnell, B., and Wohlin, C., 2000. Using students as subjects - a comparative study of students and professionals in lead-time impact assessment. *Empirical Software Engineering*, 5(3):201-214.

[15] Brühlmann, A., Gîrba, T., Greevy, O., and Nierstrasz, O., 2008. Enriching reverse engineering with annotations. In: Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS), pages 660-674.

[16] Liang, P., Avgeriou, P., and He, K., 2010. Rationale management challenges in requirements engineering. In: Proceedings of the 3rd International Workshop on Managing Requirements Knowledge (MaRK), pages 16-21.