Abstract

Knowledge management has been playing an increasingly prominent role in software engineering in both academic and industrial perspectives. Meanwhile, the software architecture community has recently witnessed a paradigm shift towards managing architectural knowledge (AK), and consolidated research results have been achieved. Within the software development lifecycle, requirements engineering (RE) and architecting are two closely related activities, and so are their associated knowledge. This report makes a literature survey on RE from a knowledge perspective, and proposes to reuse the research outcomes for AK management in RE. We target specific RE problems and present knowledge-based ideas 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.

This research has been partially sponsored by the Dutch Joint Academic and Commercial Quality Research & Development (Jacquard) program on Software Engineering Research via contract 638.001.406 Griffin: a Grid for Information about Architectural Knowledge.
Contents

1 Introduction 3

2 AK and RK 4

3 AK and RK Management Strategies 4
   3.1 AK Management Strategies ................................. 5
   3.2 RK Management Strategies ................................. 5

4 Selected Problems in RE 7

5 Candidate Solutions 8
   5.1 Insufficient Requirements Traceability .................... 8
      5.1.1 Current Approach in RE ............................... 8
      5.1.2 AK Management Approach .............................. 8
      5.1.3 Application in RK Management ......................... 9
   5.2 Lack of Systematic Requirements Reuse .................... 9
      5.2.1 Current Approach in RE ............................... 9
      5.2.2 AK Management Approach .............................. 10
      5.2.3 Application in RK Management ......................... 10
   5.3 Lack of Integration of RE Activities ....................... 11
      5.3.1 Current Approach in RE ............................... 11
      5.3.2 AK Management Approach .............................. 11
      5.3.3 Application in RK Management ......................... 12
   5.4 Communication Problem in Distributed Development ........ 12
      5.4.1 Current Approach in RE ............................... 12
      5.4.2 AK Management Approach .............................. 13
      5.4.3 Application in RK Management ......................... 13

6 Conclusions and Research Agenda 14
1 Introduction

Software development is a knowledge-intensive business involving many stakeholders collaborating in different phases and activities, in which various types and huge amount of knowledge are produced and consumed. The main asset of a software organization is its intellectual capital, the “knowledge”. As identified in [84][3], Knowledge Management (KM) becomes a key issue in both business and technical aspects in software development: the involved knowledge (technical, managerial, domain, corporate, product, and project knowledge) is highly dynamic and difficult to manage due to the complexity and uncertainty caused by requirements change and systems evolution. This trend has already been witnessed in both academia [82][30] and the industry [31][80][75][85]. Targeted books [3][33], special issues [84], conferences (SEKE\(^1\)) and workshops (SHARK\(^2\), CHASE\(^3\), MaRK\(^4\), SWESE\(^5\), KCSD\(^6\), LSO+RE\(^7\)) focus on general or specific topics on KM in software engineering, such as requirements [65], architecture [6], design [40], maintenance [14], processes [77] and project-wide KM [54].

In the field of software architecture, there has been a paradigm shift from describing the outcome of the architecting process (e.g. components and connectors) to documenting Architectural Knowledge (AK) [56], such as design decisions and rationale [49]. In the SHARK workshop series [57][4][5], progress has been reported on addressing architecting challenges by AK sharing and reusing, including AK domain models, application methods and tools. Due to the interwaved relationship between RE and the architecting process [94][42][72], that 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 solved or mitigated by reusing the solutions for AK management.

The rest of this report is organized as follows. The general descriptions of AK and RK are presented in section 2 while the comparison of their specific KM strategies with a recommendation to the usage of KM strategies in RE activities is given in section 3. Selected problems identified in the current RE literature and practice that can be solved though a KM perspective are discussed in section 4. The current RE practice on the selected problems and the corresponding solutions from AK management with their potential applications are presented in section 5, with a discussion on pros and cons. The report concludes with a future research agenda in section 6.

\(^1\)http://www.ksi.edu/seke/skhistory.html
\(^2\)http://www.cs.rug.nl/paris/SHARK2008/
\(^3\)http://softwareresearch.ca/seg/CHASE/
\(^4\)http://www1.in.tum.de/static/mark08/
\(^5\)http://www.abdn.ac.uk/r01srfl/swese2008/
\(^6\)http://l3d.cs.colorado.edu/kcsd2006/
\(^7\)http://www.se.uni-hannover.de/LSOplusRE
2 AK and RK

As the two initial phases in the project lifecycle, RE and architecting are highly knowledge intensive processes due to the involvement of various stakeholders (e.g. customer, end-user, architect, requirements engineer and designer). The major outcomes of these two phases, requirements specifications and architecture documents, contain dense knowledge in various forms and determine the project success to a large extent. In this section, we present the general definition of AK and RK as we understand them, as there is no accepted definition of either of them [28].

Traditionally the architecting process only documented the outcome of the architecture design using Architecture Description Languages. The new paradigm in software architecture considers the documentation of all the produced knowledge: AK consists of the architecture design as well as the design decisions, rationale, assumptions, context, and other factors that together determine why a particular architecture solution is the way it is [56]. In different usage contexts, practitioners may address AK in various perspectives, including generalized, organizational, view-based, domain-specific and project-specific AK etc [26].

RK is generally defined as the knowledge used in the RE process [52], e.g. the rationale of requirements [15]. The most explicit part of RK is contained in the requirements specifications, including requirements stakeholders, system goals, functional and non-functional requirements and traceability information. However in current RE practice, there are other types of knowledge, that usually remains implicit in the heads of stakeholders, e.g. domain knowledge [89], the rationale leading up to requirements [44], and user experiences and expertise [21], etc. The knowledge types and KM strategies for AK and RK management are further discussed in section 3.

Within the project lifecycle, RE and architecting are closely related activities that take place simultaneously and affect each other [42]. The results of the RE process act as input for architecting, and the architecture design needs to be evaluated against the requirement specifications and may lead to new requirements. The activities of understanding the problem domain in RE and exploring the solution space during architecting, are highly overlapping [25], hence the knowledge related to these two phases is also interwoven with each other [72]. A typical example of the overlapped knowledge is architectural design decisions and architecture significant requirements. In both RE and architecting they are treated as first-class entities [49], and it is argued that there is no fundamental distinction between them [27]. The intrinsic relationship between RK and AK can improve the coordination between RE and the architecting process. In the next section, two types of KM strategies for AK and RK management are presented, which act as the basis for relating RE problems and AK solutions from a KM perspective.

3 AK and RK Management Strategies

Various types of knowledge are produced and consumed by stakeholders within the RE and architecting processes. A distinction is often made in KM between two types of knowledge: tacit and explicit knowledge [71]. Tacit (personalized) knowledge resides
in people’s head, whereas explicit (codified) knowledge is codified in some form, e.g. a document (documented knowledge) or knowledge based on formal models (formal knowledge). Based on that, Hansen et al. classify KM in two strategies: codification and personalization [43]. Codification aims on codifying knowledge and making it available for anyone through knowledge repositories [63]. Personalization, on the other hand, help people to communicate knowledge instead of storing it. In software engineering, both KM strategies are employed [84]: most research and industry practice has been associated with codification [31], while personalization has been given less attention. In this section, we present the current practice of AK management strategies, and discuss the implication in RK management.

3.1 AK Management Strategies

Babar et al. presented a preliminary review on the state-of-the-art of AK management strategies in research and the industry [6]. They discover that have an intentional focus on the codification strategy. However, most organizations seem to rely on unintentional personalization as the primary AK management strategy. Based on the findings, they propose a hybrid strategy which combines the two and provides a balance between codification and personalization, in order to get the best of both strategies [30]. For AK that is not subject to frequent changes (e.g. architecture patterns, approved design decisions), codification is useful, as it can be reused without too much effort to update it. On the other hand, much AK is not stable until consensus has been reached (e.g. AK produced in ad-hoc communications). For such knowledge, personalization is more appropriate, enabling stakeholders to find ”who knows what”.

3.2 RK Management Strategies

Various RE processes [46][87][18] and requirements specification and modeling approaches [83][48][93][66][79][59][17][68] have been proposed. The purpose of this report is not to suggest a best RE process or specification approach for RK management, but to understand how RK management can improve the RE activities by taking existing RE processes and specification approaches as a starting point.

In current RE practice, codified knowledge is dominant due to focus on clearly specified requirements using natural language or models (e.g. documented requirements), which usually act as a project contractual baseline. The activity of requirements documentation [87] records the results of requirements elicitation, analysis, validation and negotiation. Similarly to AK management, industrial practice has an intentional focus towards codification for RK management [70] with tools support (e.g. DOORS, Requisite Pro), while little attention has been paid to personalization (e.g. [38]).

Based on the experience from AK management [6], we found that both KM strategies have their advantages in RE. Personalization is beneficial for the elicitation, analysis and negotiation when most of the RK is unstable and produced through ad-hoc communication. Codification, on the other hand, is suitable for documentation and management when RK is not subject to frequent changes, and for analysis and validation when requirements are formalized for requirements reasoning (e.g. [60]).
Table 1: Usage of personalization and codification in RE activities

<table>
<thead>
<tr>
<th>RE Activity</th>
<th>Personalization</th>
<th>Codification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elicitation</td>
<td>Communicate with stakeholders to understand the domain knowledge and get the sources of requirements</td>
<td>Document the mutual understandable domain knowledge and discovered requirements on which the requirements analysis is based</td>
</tr>
<tr>
<td>Analysis</td>
<td>Communicate with stakeholders to understand the requirements and explore the requirements alternatives</td>
<td>Document requirement analysis results, e.g. the selected requirements and their rationale (e.g. [69][15])</td>
</tr>
<tr>
<td>Validation</td>
<td>Communicate with stakeholders to ensure that the right requirements are defined</td>
<td>Document the validated requirements, as well as the validation issues and corrections</td>
</tr>
<tr>
<td>Negotiation</td>
<td>Negotiate with stakeholders in order to reconcile conflicting views on requirements in business and technical perspectives</td>
<td>Document agreed requirements and the negotiation issues, including involved stakeholders and trade-offs</td>
</tr>
<tr>
<td>Documentation</td>
<td>Document the personal information about who knows or is concerned with which requirements</td>
<td>Document the requirements and the traceability in requirement specifications</td>
</tr>
<tr>
<td>Management</td>
<td>Manage the personal information so that stakeholders can find the right person when they want to communicate requirements</td>
<td>Document the requirements change history for later use</td>
</tr>
</tbody>
</table>
The envisaged usage of these two KM strategies in RE activities is described in Table 1. For example, in requirements analysis, personalization (meeting and ad-hoc communication) is appropriate for discussing the system boundary, and exploring the requirement alternatives in the early phases. It may become prohibitively expensive to codify all the alternatives that stakeholders have considered, e.g. the goal-oriented method [93] can generate up to $n!$ (the number of leaf requirements in the model) potential requirements alternatives. Codification is suitable for capturing the discussion and exploration results, i.e. system boundary, confirmed requirements and the rationale of requirements (why requirements are selected and prioritized, what is the goal for achieving them) in the later phase of analysis. In this report, we mainly focus on codification in which fruitful research results have been achieved in AK management. We also discuss personalization to a small extent, as directions for future work.

4 Selected Problems in RE

We have extensively reviewed the RE literature published from 2000 to 2008\(^8\), selecting RE problems that can be potentially solved in a knowledge perspective. We identify the following problems related to knowledge-intensive activities.

**Insufficient requirements traceability.** Traceability is critical to the success of software development projects [19]. Current requirements traceability links help to scope the possible impact of requirements change, but they do not support automated reasoning about change, because the links carry little semantic information [73].

**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 and effective techniques for domain analysis, and how to document requirements for product lines [18].

**Lack of integration of RE activities.** Most RE research focuses on individual RE problems, such as elicitation or traceability management. 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 [18].

**Communication problem in distributed development.** Global software development (GSD) is an inevitable trend of economic globalization, but the downside is increased risk of communication gaps generated by the difference in geography, culture, languages and processes of distributed stakeholders [22]. It is difficult to manage distributed requirements elicitation, modeling and negotiation due to the communication problem, and ill-defined requirements pose a high risk of project delay or failure [18].

\(^8\)This duration of publications is selected due to the time limitation, and regarded as the latest work in RE research and practice.
5 Candidate Solutions

In this section, we present solutions from AK management to address the RE problems identified in section 4. For each problem, we first present the current solutions in RE and their inefficiencies. Subsequently we discuss how AK management solutions tackle similar problems, and how to apply these solutions in RE. Potential issues of using AK management solutions in RE are discussed in the end.

5.1 Insufficient Requirements Traceability

5.1.1 Current Approach in RE

Requirements traceability has been identified as a quality factor that a system should possess [78]. The significance of requirements traceability in supporting systems development has long been recognized in RE practice [41] with fruitful results [32][76][78][35][88]. However most of the existing work proposes various traceability models by following a heavyweight approach aligning with a full-fledged RE process. These results are definitely helpful in certain cases, but there are still issues: (1) lightweight requirements traceability methods are also desired, for example, to deal with rapid requirements change in agile processes; (2) semantic information about traceability links (e.g. the type of link, the stability of the link) is needed to help distributed teams, who use different traceability models, to understand each other [19], and perform automated traceability reasoning (e.g. for detection of conflicting requirements) [35].

5.1.2 AK Management Approach

In the architecting process, similar problems arise. Various domain models for codifying AK have been proposed either by industrial organizations (e.g. the Knowledge Architect [62] AK model for LOFAR\(^9\) project) or by researchers (e.g. Kruchten’s ontology [55], Tyree’s template [91], AREL [90], PAKME [2], ADDSS [16] and Archium [51] AK models). These domain models are composed of a set of AK concepts and the relationships between them (e.g. Decision Topic is addressed by Alternative). The traceability management of AK is rather challenging due to the heterogeneity of different AK domain models.

The semantic web has proven to be useful as a platform on which domain models can be created, shared and reused, and provides different levels of reasoning functionality based on the data semantics [39]. We have proposed an approach [62] that employs semantic web technologies, e.g. ontology description standards, RDF [58] and OWL [10] for AK management. We first annotate the text in the architecture documentation (source of most AK) into AK instances using an AK domain model (e.g. “The programming language should be Java” is an AK instance of AK concept Architecture Design Decision). Subsequently we store the annotated AK instances (in RDF) along with their domain model (in OWL) into Sesame [12] (an AK repository with semantic data), in which various AK domain models can be integrated with the data (AK instance).

\(^9\)http://www.lofar.org/
This approach can partially address the problems in current requirements traceability management in the following ways. 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 domain 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 development teams and organizations who employ different AK domain models. Third, automatic traceability reasoning can be performed (e.g. AK completeness), since the data (AK instances) stored in an AK repository have formally defined semantics using 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 the semantic data. If more advanced reasoning capabilities are required (e.g. consistency and correctness checking), then more expressive semantic information is needed.

5.1.3 Application in RK Management

In RE, various methods (e.g. goal-oriented, value-based) have also been developed for requirements traceability modeling as mentioned in section 3.2. The aforementioned AK traceability management approach can fit well with requirements traceability management, which is also mostly text-based in current practice. The requirements specifications can be annotated using RK domain models with special focus on requirements traceability, and then the annotated RK can be stored and managed by a semantic data repository. This approach resolves the two aforementioned traceability problems as follows: (1) It provides a lightweight and extensible approach to accommodate various requirements traceability models in the agile or the multiple site development context; (2) It provides semantic traceability information to support automated reasoning for the detection of conflicting requirements. The shortcoming is rather obvious: This approach can only be applied to formal knowledge based on formal models. In practice, it was found [6] that architects tend not to formalize (annotate) the AK, neither during the architecture design, nor afterwards, since they consider the knowledge formalization a burden and interruption to their daily activities. This problem can be possibly resolved by a similar solution to the Experience Factory Organization [9]. Finally, process improvement and tool support are also critical for the acceptance of this approach.

5.2 Lack of Systematic Requirements Reuse

5.2.1 Current Approach 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 successfully. As identified in section 4, the most strategic form of requirements reuse is product lines, but two challenges exist: the lack of strategic and effective techniques
for analyzing domains, as well as methods to effectively document requirements for product lines. Feature models [74] 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 trees [81], which claims to be able to effectively manage the changes to the complex feature tree’s structure generated by the feature model.

5.2.2 AK Management Approach

In the architecting process, some methods with tool support for reusing AK have been proposed and validated in architecting activities, such as architecture analysis [45], evaluation [36], documentation [8] and the whole architecting process [7]. Herold et al. [45] tackle the problem of reusing architectural drivers in architecture analysis to support early requirements and architectural decisions. They achieve this by combining the goal-oriented requirements model with the 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 the architectural drivers (a type of AK). Babu et al. [8] focus on application-generic AK that can be reused across application domains. They present an ontology that captures the general AK as an invaluable reusable asset for universal AK management. Babar and Gorton [7] present tool support for capturing, presenting and (re)using AK based on the PAKME AK domain model. This tool can locate architecture design alternatives by keyword search within the boundaries of the PAKME AK domain 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 knowledge to query the AK repository to server specific reuse purposes. For example, the semantic relationship (positive/negative/noeffect) between the patterns and quality attributes in architecture design (part of AK) can be reused to satisfy certain early requirements. The drawback of these methods is the same as the methods introduced in section 5.1: they can only be applied to formal knowledge.

5.2.3 Application in RK Management

In RE for software product lines, feature modeling is a kind of requirements modeling method, which focuses on the end-user visible characteristic of a system [53]. As a conceptual model, feature models (and their corresponding RK) can be formally represented using ontology definitions as described in section 5.1. Organizations can define their specific RK ontologies (concepts, relationships and constrains) for their specific reuse purposes. For example, to mitigate the 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 (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, the 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.

5.3 Lack of Integration of RE Activities

5.3.1 Current Approach in RE

A typical example of this problem is that despite the 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 [18]. Some initial work has been done in terms of a modeling theory incorporating the aforementioned requirements modeling elements [13], and product-management techniques for using various requirement techniques integrally [34]. However, further research is needed on how to integrate requirements technologies in various RE activities, so that practitioners know how to apply individual technologies effectively and synergistically [18]. 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 the software lifecycle [67]. In conclusion, bridging the requirements models and related RK across the RE process is a major challenge.

5.3.2 AK Management Approach

In the architecting process, we look at the AK management activities in a consumer-producer perspective (Figure 1): the cube represents AK in various types, including General AK that helps architects to design a system; 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 [95] that all stakeholders can access transparently. The AK grid provides AK integration by synthesizing multiple AK models into a common model [26] 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 the AK management activities into architecting activities [47] (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 the 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 the architecting activities. Several concrete cases can be found in [90][2][16]. This is also a flexible solution, as every organization can construct a specific AK domain model to adapt their individual architecting context.

5.3.3 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 the RK management activities within the 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 the RK concepts and their relationships from all the RK types, and come up with a common RK domain model. This domain model acts as the common model in the RK repository for the integration of RK, and further integration of RE activities in the RE process. A major research challenge in this approach is how to integrate and harmonize the RK concepts and their relationships when inconsistencies appears. Also, as in the previous subsections, the disadvantage of this solution is that it only applies to formal RK.

5.4 Communication Problem in Distributed Development

5.4.1 Current Approach in RE

The RE activities require intensive communication from the elicitation [21] down to the analysis, implementation and testing phases [1]. Software development in a distributed context increases the risk of the communication gaps as discussed in section 4.
address this problem, Bhat et al. [11] have 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. Damian et al. have investigated how to use and combine media technologies to facilitate requirements negotiations and quality agreements [23]. Sinha et al. have developed an Eclipse-based tool for distributed requirements engineering collaboration by adding collaboration capabilities to the traditional requirements management tools [86]. 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 the correct interpretation and perception by different stakeholders, and to address the problem of ineffective knowledge sharing (communication) in a distributed context [24].

5.4.2 AK Management Approach

For the last four years, the Griffin project has been working on methods, tools, and techniques to manage AK with special focus on AK sharing and reusing among distributed organizations. We envision AK sharing and reusing in a grid environment as described in section 5.3.2. A number of use cases for using architecture decisions (most important type of AK) has been identified [92], an AK core model to harmonize the different AK domain models has been constructed [26], and a method on how to select a high quality core model for AK sharing is proposed and validated [61]. Using the core model and different AK domain models, AK can be described precisely to ease the (re)using and sharing with correct interpretation. Furthermore, some best practices on AK management for GSD have been identified and validated [20], 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 development teams using hybrid strategy [37], which is more suitable in the industrial environment. A second one, the Knowledge Architect [62] tooling suite emphasizes on the formal AK management based on semantic data repository, including a set of interconnected utilities for AK annotation, integration, validation and translation. It is capable of defining the precise semantics of AK and using them to communicate the AK effectively, e.g. share AK for quantitative architectural analysis [50].

5.4.3 Application in RK Management

As specified in section 3.2, both personalization and codification are valuable in various RE activities; the same is true for the 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 manage-
ment methods, such as classified documents for browsing and experts yellow page to 
find out who knows what. For formal RK communication, a core RK domain model 
should be constructed by following the steps in section 5.3.3, while concept mappings 
between the core RK model and specific RK domain models should be defined by RE 
experts. These solutions can partially mitigate the problems of requirements commu-
ication in distributed development: formal RK domain models can cope with imprecise 
requirements that may cause; a central RK grid and a web portal can deal with in-
effective requirements communication. Except for the methods, techniques and tools 
support, a healthy culture of collaboration should also exist in order to promote the 
communication and make distributed projects work.

6 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 two-fold: it 
presents a literature survey on knowledge-based RE and proposes a research agenda for 
the major RE challenges from a KM perspective. The envisaged usage of KM strategies 
in RE activities is outlined, while few selected problems from the RE literature and 
practice are addressed based on 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 acknowledge that knowledge formalization is a hard 
prerequisite for applying the proposed solutions, which limits their scope. For this 
reason, we point out that documented and tacit knowledge in RK management need 
further attention.

Besides the future research work mentioned in each RE problem, the following 
research directions are scheduled: (1) Systematic approaches to assist RE processes 
through RK management in three aspects: KM strategies, KM activities, and RK types 
and concepts; (2) More focus on how personalization strategies can fit various RE pro-
cesses and activities, for example, in the agile development when requirements changes 
frequently and codification strategy is not suitable; moreover more insight is needed on 
the right balance between personalization and codification 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 the architecting process, the overlap and co-
ordination between RK and AK needs further investigation for better cooperation and 
smooth transition from RE to architecting.

KM is beneficial but also costly, and resources, time and effort are required be-
fore benefits become visible [84]. This situation is also true for RK management since 
project managers who focuses on completing the current project on time, but not help-
ing the next project succeed, often considers 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 [29] and semantic 
wiki [64]) in which RK can be codified collaboratively and shared as soon as they are 
codified.
References


16


17


