AMSE Association for the Advancement of Modelling & Simulation Techniques in Enterprises
IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Advances in Modelling and Analysis B
Vol. 62, No. 1, March, 2019, pp. 1-10
Journal homepage: http://iieta.org/journals/ama_b

A Recommender System for the Proactive Sharing of Architectural Knowledge

Buradagunta Suvarna*, Turimella Maruthi Padmaja

Department of CSE, VFSTR Deemed to be University, Vadlamudi, Guntur (Dt), India

Corresponding Author Email: sv1720@gmail.com

https://doi.org/10.18280/ama_b.620101

Received: 10 October 2018
Accepted: 5 February 2019

Keywords:
architectural knowledge, recommender systems, global projects, and similarity measures

ABSTRACT

Architectural Knowledge Management (AKM) is concerned with capturing, sharing and reusing the architectural knowledge. Design rationale, which constitutes the reasoning behind the software architecture, is a key component of architectural knowledge. Existing AKM support has been dedicated to capture and reuse of design rationale, however, automated and proactive knowledge-sharing has not been addressed well in the community. Hence, in this research, we address the issue of architectural knowledge sharing. We propose an enterprise-wide recommender system to enable proactive knowledge-sharing by extending a traditional guidance model with data mining techniques. The viability is demonstrated using synthetic data and Linkdin users of software architects. In particular, we focus on recommending an architect with items such as architects with similar interests, similar issues and alternatives. The key benefits of this approach are: improved reuse of design rationale in order to avoid several repetitive steps for deciding on architectural issues and enhancing knowledge transfer between global projects and departments. We believe that a similar recommender system could also be applied to other areas of software engineering.

1. INTRODUCTION

Architectural Knowledge Management (AKM) is a key activity in the design of complex software-intensive systems [1, 2]. AKM is mainly concerned with capturing, sharing and reusing knowledge about architectural design decisions. The knowledge on architectural design decisions has also been termed as Design Rationale (DR) [3] in literature. We note that a design rationale may also include design decisions that are not necessarily architectural e.g., technology platform choices; however, in this paper, we use the term Design Rationale as synonymous to architectural design decisions.

Companies often develop similar software systems. Examples of similar systems across the domains of energy and industrial automation are the software tools for configuring control and automation solutions for a process plant (e.g., oil & gas, chemicals, or cement industry plants) and for a power utility. For making a design decision, an architect of a software system could reuse DR information for the similar architectural issues of the other systems of a company. Such a reuse of information improves the efficiency of architecting because architects could avoid performing several repetitive steps for deciding on issues that consume significant time. Examples of key and time consuming steps of decision-making are eliciting alternatives, identifying selection criteria and risks, and evaluating alternatives [1, 3, 4]. Similarly, there are other applications on reusing DR. For example, by reviewing the existing DR documentation the architect would get insights on architectural decisions and could learn about them. Furthermore, DR also enables knowledge transfer to other global departments/projects as well as knowledge transition to new architects.

Recent contributions (e.g., [5-6]) highlighted the need for sustainable design decisions in order to maintain software architecture on a long-term basis. In particular, the contributions emphasize that the DR has to be maintained continuously like an asset of a company to evaluate the time period for which the design decisions remained meaningful and unchanged, and the costs associated with the required changes to those decisions. A well maintained DR management system would not only enable sustainability, but also help an architect in avoiding several repetitive and time consuming tasks for making an architectural decision.

In order to reuse DR, architects should be able to find issues similar to the architectural issues that they are facing in the current software's context. Traditionally, similar issues are identified manually and shared in meetings and email communications. However, there is little support to automatically and proactively share this architectural knowledge. Sharing DR in the context of global projects is even more critical nowadays, when lots of software development and architecting activities have been happening in the emerging economies. In this context, a new/less experienced architect in a country would benefit from the related DR information that was captured by experienced architects located in other countries.

In order to address the above mentioned challenges, we propose a Recommender System for Architectural Knowledge Management (RSAKM). We envision RSAKM to have the following capabilities: (i.) an enterprise wide social platform-like environment where architects share and rate issues and alternatives, (ii.) a database for modeling, architectural design decisions, and (iii.) a recommender system to aid decision making and AKM. This proposal is based on the synergies between software engineering and e-commerce and social computing [7].

Research on a recommender system for knowledge-sharing was already encouraged [8-9]. For example, the comparative review on AKM tools [10] identifies knowledge-sharing and support for recommendations as key research issues for the future. The literature review from Ding et al. [11] also suggested the need for using knowledge-based approaches such as a recommender system for AKM. Within the scope of this paper, we focus on three different use cases for RSAKM, which also highlight the properties of this proposed recommender system:

UC1: Identify and connect architects with similar interests

(1) An architect adds his profile to a RSAKM environment

(2) The environment recommends architects with similar profile so that he/she would be able to connect to the architects

(3) The environment recommends the architects with top rated issues (and other items such as current issues, searches for issues etc.) from the connected architects

(4) From the recommended issues, the architect would be able to find a set of related issues for DR reuse

UC2: Share similar issues

(1) The architect creates an issue in the environment and adds keywords

(2) The environment recommends a set of similar issues by searching in the database

(3) The architect would review the DR documentation of the similar issues and would reuse relevant information

UC3: Recommend alternatives for a decision

(1) The architect enters the issue and the alternatives in the environment

(2) The other architects with similar profiles rate the alternatives

(3) The environment recommends alternatives that might resolve the issue

As hinted in the recommendation strategy for each of the above use-cases, RSAKM would follow the following general principles [12]:

(1). Recommending architects profiles:
- ✓ RSAKM would recommend similar architect profiles by matching properties from user-profiles of architects in the system
- ✓ In the case where recommended profiles are accurate but very limited, RSAKM would make a trade-off in accuracy to provide multiple architect profiles ranked in order for decreasing similarity

(2). Recommending issues
- ✓ RSAKM would recommend issues that are similar to a user's issue based on key-word matching
- ✓ Top-rated issues would get more weight
- ✓ Issues that architects with similar profile looked at will also get more weight
- ✓ Current issues in the system that are based on trending key-word searches would also get relatively higher weight
- ✓ Issues that led other architects to look at a particular issue (i.e., user's issue) and the issues that other architects looked at after looking at user's issue would also be recommended

(3). Recommending Alternatives
- ✓ RSAKM would recommend alternatives that are rated higher in the community for a user-posted issue
- ✓ The alternatives that are rated higher by user's with similar profiles are ranked higher in the recommendation
- ✓ There may be many other scenarios which would need resolution to recommend a meaningful alternative. Here, RSAKM could consider multiple options. For example, if most of the architects slightly preferred alternative 1, but rest of the other architects with similar profile to the user had rated that alternative very low, RSAKM may not recommend alternative 1 as the best recommendation

To support the above described use cases, we propose RSAKM by extending a simple guidance model with concepts from the similarity measures of the statistical data mining community. In particular, the concrete contributions are: (i) A meta-model for RSAKM which provides a basis to model issues and to compute similarities between them for addressing UC1-UC3, (ii) An industrial illustration on how the use cases are addressed based on the meta-model and (iii) A summary of the initial evaluation.

The remainder of this paper is organized as follows. Section 2 presents a short overview of similarity measures while the second part introduces decisions and rationale. We describe related work in section 3. We propose the meta-model in section 4. Later, we apply the meta-model and illustrate the recommender system for architectural issues in section 5. After that, we present the initial evaluation results in section 6 and conclude the paper by indicating the limitations and future research in section 7.

2. BACKGROUND

2.1 Clustering and partitioning around K-Medoids (PAM) algorithm

As the data for the considered application is ordinal in nature, k medoids clustering [13] is employed to identify the similar group of profiles. The K-medoids are one of the partition based clustering algorithms and groups 'n' objects into k-clusters by minimizing the absolute error. Initially the algorithm considers random data objects as cluster representatives. Latter, the algorithm iteratively improves the quality of the clusters by replacing representative objects with the other objects. The algorithm terminates when the quality of the cluster alignment is not further improved. The quality of the cluster alignment is measured as an absolute error function of the average dissimilarity between all data objects of the clusters to its representative.

Algorithm:

(1). Initialize k medoids with any k random objects of the data of size n.

(2). Based on similarity assign each n - k medoids to the corresponding closest medoid

(3). Repeat the following steps until the cost of the cluster alignment decreases:

Do for each medoid object 'm' and for each non-medoid data point 'o'

a. Swap 'm' with 'o', and compute the cost of the new alignment.

b. If the total cost of the alignment decreases via swapping, the swap can be retained otherwise undo the swap.

2.2 Similarity measures

Finding similarity among the data is a key step in K-Medoids Partitioning Algorithm. Similarity measures [13] have been used in the data mining community to find similar/related data. These measures are behind today's e-commerce systems (e.g., Amazon [14]). The recommendations are based on similarity of products that the customer has bought before, or on similarity of profiles of other buyers when compared to the customer's profile.

A number of similarity measures have been proposed in the data mining literature [13]. For example, Jaccard similarity defines similarity based on relative size of intersection between two sets of data. Collaborative filtering is a method that uses Jaccard similarity to find similar profiles for users of a system. We used Jaccard similarity because we can identify stakeholders with similar interests by finding intersection between their profiles.

Similarities are generally measured on the basis of distance between two sets of data. In this respect, Jaccard similarity is represented by a distance measure called Jaccard distance (defined as: 1-Jaccard similarity). We use Jaccard Index, which measures the similarity between sample sets, and for sets A and B, it is defined as follows:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Some of the other distance measures described in literature are: Cosine Distance, Edit Distance, Hamming Distance etc. Cosine Distance is preferred when vectors have integer components, and we are interested in the direction of the vector.

$$\cos(A, B) = \frac{A.B}{||A|| * ||B||}$$

Edit Distance is beneficial when strings are compared. The distance between two strings is the least number of insertions/deletions of single string characters that will transform one string into the other. Hamming Distance makes sense when two vectors are Boolean. It is based on the number of vector components that differ when the Boolean components of two vectors are compared based on their position in their respective vectors.

2.3 Guidance model

A guidance model [3] is often used to guide stakeholders in making a decision. The goal is to guide stakeholders to make decisions while DR is captured as a byproduct. Questions Options and Criteria (QOC) [15] is a simple guidance model with basic DR concepts such as: an issue, or problem to be solved, alternatives to address the issue, arguments of stakeholders, criteria for selection and a decision, which is the outcome.

In addition, a guidance model also uses concepts such as goals, implications, etc. For example, decision representation language (DRL) [16] additionally models goals. Furthermore, a guidance model uses interdependencies between the concepts. For example, an issue can trigger another issue. Similarly, a decision can override another decision. A comprehensive set of relationships with formal semantics were already reported [17]. In this paper, we used QOC in order to simplify the description of this paper. However, the concepts proposed in this paper could be used with other guidance models.

3. RELATED WORK

Software Product Lines. The set of systems developed based on a reusable asset base is termed as a software product line. Variability is used as an abstraction to customize and reuse software. Reuse of knowledge across multiple systems of SPL is beneficial because of the potential to have similar issues across them. Thurimella and Bruegge propose a meta-model to capture rationale for variability and a pattern-based approach to reuse rationale [10]. The empirical study with students [6] identified strong empirical evidence on the reuse of rationale. However, both the contributions did not report on sharing rationale automatically. There were already attempts to model design rationale in the context of SPLs. For example, Lee and Kang add contextual information to feature models which are used to represent variability [18]. The Rationale has been modeled in the context of requirements engineering for SPLs [19] as well as design [20-21]. However, all the above described contributions, neither focuses on knowledge sharing nor on a recommender system.AKM. Design rationale has been considered as integral part of software design [1]. The industrial survey on the next-generation architectural languages elicited the need for supporting DR modeling as well as abstractions for design reuse [22]. Both these contributions [1, 22] view DR as a key topic for the future of software design. The need for knowledge sharing was already recognized in the community [1]. In the recent past, Zimmermann et al. proposed a reference architecture and formal semantics behind them in order to model design decisions and integrate them into software design [4]. The decision model is process-oriented which was applied for enterprise application development and outsourcing [4]. The reference architecture encourages the reuse of design rationale. Baber et al. [1] emphasized the value for architecture knowledge sharing and reuse. However, [1, 4] do not focus on a recommender system but encouraged DR sharing and reuse. Related papers on sustainability of a software design emphasized the need for the sustainability of DR [5, 23]. For example, DR has been considered as an asset of a company like code which has to be maintained continuously [5]. The metrics for the sustainability of the software architecture considered multiple artifacts including DR [23]. This research on recommender system would aid sustainability by sharing DR.

Rationale has been used in the requirements engineering community similar to the related papers already discussed in the context of SPLs [24, 10]. Goal-oriented requirements engineering (GORE) [22] uses rationale-based techniques to elicit requirements. As the rationales are captured early in requirements engineering, GORE has positive impact towards software design. However, GORE does not directly focus on DR and software design.

Recommender Systems. Recommender systems are considered important in the context of software engineering [23]. Clustering and text-mining techniques are used to

recommend requirements [25]. Similarly, enhancing stakeholder profiles supports large-scale requirements elicitation [26]. Rating techniques are used for recommending product configurations [27]. Borsch et al. [27] use a similarity-aware graph technique to recommend conflict-resolution patterns for code. Similarly, Zang et al. recommend APIs [29]. All the above described attempts on recommender systems do not focus on AKM.

The edited book in the area of recommender systems on software engineering report on recommending source code/APIs, developer profiles, code-fragments for reuse, refactoring code and identifying requirements. However, these are not focused on DR. Moreover, the contribution encourages research on recommender systems in the software engineering context.

Knowledge extraction. Text mining and parsing techniques are used for extracting rationale from a document [30]. Similarly, text mining and natural language processing are also applied for extracting FAQs form emails, documents etc. [31]. A statistical learning model was proposed for performance prediction [32]. We differ from the papers on knowledge extraction [30-32] by focusing on sharing knowledge.

Knowledge sharing in other areas. Rodrigues et al. [33] propose a system to add end-users knowledge and enhance recommendations during business process modeling. Similarly, a know-how sharing is enabled based on content filtering [34]. Both these knowledge-sharing approaches do not focus on decision-support.

Requirements engineering. The book in the area of requirements knowledge [14] elicited the need to proactively share knowledge in the context of global projects. This finding is also important for the software architecture community because of the overlaps between design and requirements engineering [35].

4. META MODEL

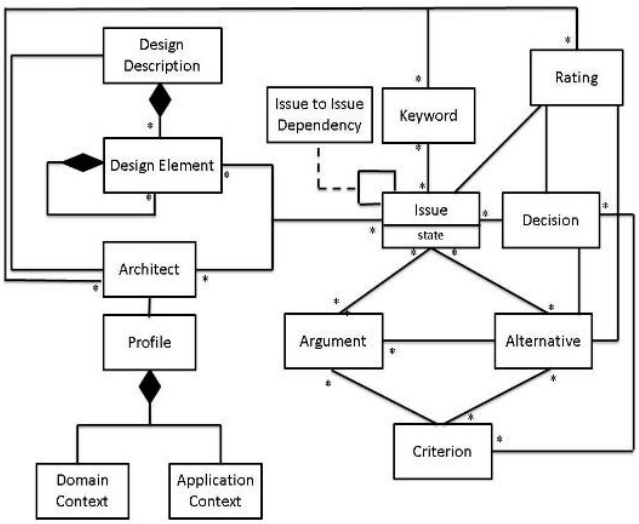

Figure 1. The meta-model for RSAKM

We represented the core guidance model with classes: Issue, Alternative, Criterion, Argument and Decision in Figure 1. We modeled many-to-many between an Issue and an Alternative because an issue can have multiple alternatives and an alternative can be involved in multiple issues. Similarly, we modeled the other many-to-many associations of the core guidance model. An Alternative can have multiple arguments across various Criteria while an instance of Argument can be dedicated to only one Alternative. This is the reason for "1 (default) to many" association between an Alternative and an Argument. Using a similar logic, we added multiplicities between Argument and Criterion as well between a Decision and Alternative. We propose a RSAKM meta-model (see Figure 1) by extending the core guidance model with several concepts related to a recommender system. The extensions are summarized below.

4.1 Architect & profile

An Architect has to add and maintain his/her Profile. The reason for introducing Profile is to compute similarities between two architects. Profile consists of an Application Context and a Domain Context.

Application Context describes high-level context of the application based on type of application e.g., Web/Desktop application/Integrated Development Environment (IDE) etc. This is the highest level classification for the software.

Domain Context describes high-level context of the domain for which the software application is targeted e.g., Industrial Automation, Utilities, Finance, Education etc.

A Profile would also contain concepts such as experience, skills, proficiency levels, organizational information etc. We used only contexts for simplicity reasons.

An architect can add keywords and ratings. This is the reason for the many to many association between an Architect and Keyword as well as an Architect and Rating.

4.2 Rating

An architect inputs a number between 1 and 5 to rate the quality of an item. On this scale, 1 would mean a poor decision, whereas 5 would mean good decision. Rating can also be left empty; hence, there is no constraint put on stakeholders to rate all the decisions in the repository although quality recommendations based on cosine distance would be made as more and more ratings are provided by more and more stakeholders. We propose using ratings for Issue, Alternative and Decision because they are the key concepts of the guidance model. Keyword. A set of keywords are used to contextualize an issue, which is represented by many-to-many association between Issue and Keyword. The purpose of introducing Keyword is to provide a basis for computing Jaccard similarity between issues.

4.3 Traceability

A Design Description is hierarchically structured based on objects of Design Element. The traceability between Issue and Design Element, and Architect and Issue are modeled based on many-to-many associations between the respective classes. Based on the traceability, an architect may trace between similar issues and corresponding design elements and architects in UC2 as well as from an architect profile to the issues and design elements in UC1.

4.4 State and interdependencies

The state of an issue is open (to be resolved) or closed (or already solved). Issues are also inter related. For example, addressing an issue may require other issues. Similarly, an

issue may exclude another issue. A network of interrelated issues is termed as an issue network. Based on an issue network, a stakeholder may trace though issues to find relevant information. Similarity computation Issues and the other related items based on the meta-model are stored in a centralized repository. In our previous contribution, we have detailed how to build such a repository [7] and therefore we do not repeat this in our paper.

For computing similar group of profiles to address UC1, we compute Jaccard similarity based PAM because the profiles are based on sets. In particular, the repository is iterated for all profiles and the Jaccard similarity is computed for each profile based on Eq. (1.1). Similarly for addressing UC2, we compute Jaccard similarity between keywords to find similar issues. Cosine similarity based on Eq. (1.2) is used to find similar issues and alternatives based on ratings in UC2 and UC3 respectively. This is because cosine similarity is used for integers.

4.5 Similar group identification for query

Figure 2 depicts the proposed methodology for retrieving top k similar profiles of the query profile. Initially the possible profiles extract from the world wide web. Latter, the extracted profiles are subjected to clustering using k-medoids data clustering algorithm. The outcome of clustering are the groups of profiles with similar characteristics. The characteristics of one group of profile are close with each other and are different with the profiles of other groups. As mentioned in section 2.1 as it is a partition clustering algorithm, K-medoid algorithm, partition the data around a center point (medoid / prototype). Obtained cluster and the corresponding medoids are stored in a (Central) repository. For a query point Q, top k similar profiles are retrieved on two phases, at first the group which is similar to the Q, is obtained by the similarity computation across Q to k medoids in the repository. Once, after the group is identified top K similar profiles for the query point are retrieved based on the similarity ranking of the profiles within that group. The main advantage of this approach for each query there is no need to compute similarity with all available profiles.

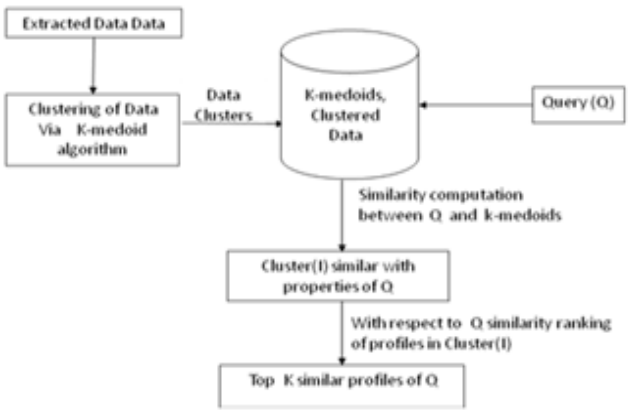

Figure 2. Flow diagram for proposed methodology

5. PROPOSED MODEL

In In this section, we consider an example to illustrate the proposed model. For the sake of a simple illustration, the example is explained with only essential elements needed to describe the recommendation system. These elements from the meta-model are also highlighted in the table headings and their instances

The illustration follows: For the last 5 years, Jenna was a developer of automotive software applications. Now, in her new job at a process automation company, she has been given the responsibility to architect the software of a new control logic application. Soon, she is surrounded by architectural issues that have various alternative solutions. She needs well-thought and reasoned decisions. Since, Jenna is working in a new domain, Jenna is unsure if she has covered all the relevant issues, alternatives and criteria.. In such a case, Jenna would look for experts and discuss with them to gain from their experience and make sound architectural decisions. However, the other architects have less time for Jenna and are located in other parts of the globe with time difference (e.g. half a day). In such a situation, RSAKM would aid her in the following way. Jenna wants to have a comprehensive view of architectural issues related to her software.

Jenna enters her profile by selecting Application Context and Domain Context (see Figure 1 and Table 1). Based on Jenna's updated profile, the recommender system can already provide her with an initial set of issues that may be of Jenna's interest. From the issues presented to Jenna, she could select an issue related to automation domain. She may also choose to search for issues in the automation domain. Based on her search term, the system answers the query with a set of issues. As Jenna selects an issue, related issues that other people looked into are also presented. In this manner, Jenna is able to look at various related issues in the automation domain.

Jenna may be interested in different kinds of related issues such as issues that

(1). 'Architects similar to Jenna's profile looked at'

(2). 'Architects who searched with the same term such as the term that Jenna used, looked at'

(3). 'Architects who looked at the issues that Jenna looked at, then looked at the following issues'

(4). 'Architects looked at before looking at the issue that Jenna just looked at'

(5). 'New issues that may interest Jenna'

The following is needed for supporting such recommendations: maintain architects' profiles, calculate similarity between profiles based on some distance measure, search terms and related ranking algorithm, and maintain data about clicks on the issues, out- and in-link data, timestamp of clicks and new issues and decisions in the repository. However, to first motivate the recommender system, we focus only on issues that are based on an architect profile. Moreover, the proposed concepts can be extended to the other data items using the same techniques. We use the following steps to recommend related issues based on Jenna's profile: (i.) Find a set of architects that have a similar profile to that of Jenna's. Let us call this set S. (ii.) Find issues that architects belonging to set S looked at. Let us call this set I. (iii.) Provide a subset of I as results to Jenna.

5.1 Recommending similar profiles (UC1)

Consider a set of architects Jenna, Molly, Scott, David and Nina. Table 1 shows the profiles that are maintained in the repository for each of these architects. The simplistic user profiles considered for the sake of this use case consist of an Application Context, and a Domain Context. Application

Context refers to the type of software application being developed, such as desktop or web-based application. Domain Context defines the domain for which the software is being developed e.g., retail, finance, or process industry.

Table 1. Architect profiles

Architect	Profile	
	Application Context	Domain Context
Jenna	Desktop Application	Process Automation
Molly	Integrated Desktop Environment	Process Automation
David	Graphics Engineering Application	Power Utility
Tom	Web-based Application	Health Care
Scott	Desktop Application	Automotive
Nina	Web-based Application	Retail Store

Initially the list of profiles that are similar to Jenna are calculated by PAM clustering using the Jaccard dissimilarity measure. Jaccard distance measures for dissimilarity between sample sets and is calculated by subtracting Jaccard index from 1. Based on the information provided in Table 1 for Molly: Jaccard similarity = 1 (Desktop Application = Integrated Desktop Environment) and Jaccard distance = 0. The Jaccard coefficient obtained via PAM clustering considering k=2 are shown in Table 2 and Table 3. Once after the medoids are stored in the repository profiles that are that are similar w.r.t. Jenna's profile is summarized (Cluster 1) in Table 3.

Hence, profiles of Molly, Scott and David are relatively closer to Jenna's when compared with Nina's profile. So, let us say the set of profiles similar to Jenna's is, S: {Molly, David, Scott}.

Table 2. Jaccard Index for various architect profiles

Architect	Jaccard Index (Similarity)	Jaccard Distance (Dissimilarity)
Molly	2/2 = 1	0
David	½	½
Scott	½	½

Table 3. Jaccard Index for various architect profiles

Architect	Jaccard Index (Similarity)	Jaccard Distance (Dissimilarity)
Tom	0	1
Nina	0	1

Table 4. Jaccard Index for various architect profiles

Architect	Jaccard Index (Similarity)	Jaccard Distance (Dissimilarity)
Molly	2/2 = 1	0
David	½	½
Scott	½	½

The environment would recommend these profiles (i.e., set S: {Molly, David, Scott}) to Jenna, who would have possibilities to connect and follow these architects in the environment. Given the set S of similar profiles to Jenna, related issues are identified by looking at the issues visited by each of the members of set S. The interesting case are top 10 most looked at issues and maximum visited issues, which would be computed and recommended.

5.2 Recommending similar issues (UC2)

The assumptions for recommending similar issues are (i.) key words are to be added (or, selected) for issues in order to provide context, and (ii.) architects actively rate issues on the five point scale. Jenna creates an issue "Ix: Decide on extending the automation framework for the next-generation functionalities?" in the environment and adds the keywords below.

Keywords: extensibility, plugins, framework, process automation

The meta-model proposed in figure 1, models an issue with its state. This state could be 'open' or 'closed'. In the current example, the status of the issue is "Open" because the issue has been newly created and not solved. An issue is also associated with keywords (Figure 1). So, in order to find similar issues, we calculate Jaccard similarity between keywords of the other issues in the database with the status "Closed". The computation is similar to the example of Table 4. (Another extension of the model would be to also include keywords from "Open" issues. This could provide opportunity for teams working on similar issues to collaborate, or at least, share the knowledge (rationale) behind their decisions.

We recommend a list of similar issues that are close to Ix based on the Jaccard distance. In the running example, Ix1 and Ix2 (Figure 3) are recommended to Jenna as similar issues. As shown in the meta-model (Figure 1), Issues are also associated with criterion. Jenna reviews Ix1 and Ix2 and reuses relevant content from the DR documentation of similar issues for deciding on Ix. For example, Jenna finds criteria of Ix1 helpful for deciding on IX

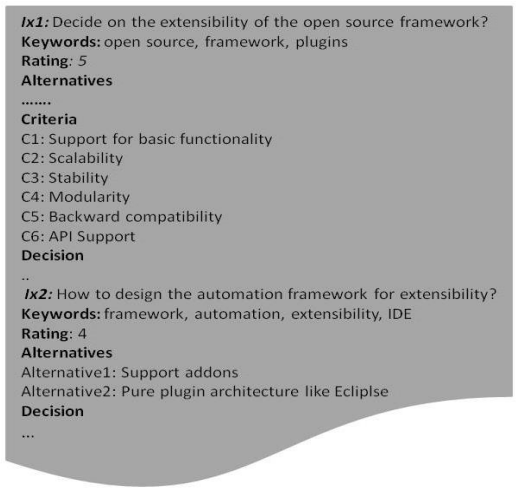

Figure 3. Dr for related issues for Ix in the running example

Given that the issues are also (individually) rated by various architects, once we have identified issues that are close to Jenna's issue (based on Jaccard similarity as described above); we could now make use of cosine similarity to recommend related issues to that created by Jenna. Continuing from the example in the last section, we found that a set of profiles that are relevant to Jenna's profile consists of profiles of Molly, David and Scott. Table 5 shows that each of these architects has rated Ix1 and Ix2.

Table 5. Ratings given by various architects for two issues

Architect	Issue	
	Ix1	Ix2
Molly	5	0
David	2	4
Scott	2	2

Later, we normalize the matrix in Table 4 by subtracting from each rating the average rating given by the architect. This way (Table 5) we nullify the impact of non-differentiating ratings given to various issues (e.g., ratings given by Scott for Issue 1 alternatives).

Table 6. Normalized ratings for Issue 1

Architect	Issue	
	Ix1	Ix2
Molly	5/2	-2.5
David	-1	1
Scott	0	0

From the normalized matrix in Table 6, now we can make a choice of Ix1 as a recommended issue. We can also do more: Since, we know that Molly's profile is relatively closer to Jenna's within the set S itself (see Table 2); we can try to find relative closeness of David or Scott to Molly's decisions. This way we could choose to present issues based on ratings that are closer to Molly, and transitively, closer to Jenna.

To illustrate this, let us modify the Table 6 such that Scott's normalized ratings are non-zero, say, 1/2 and -1/2 for issue1 and issue2 respectively (Table 7).

Table 7. Issues with differentiating ratings from Scott

Architect	Issue	
	Ix1	Ix2
Molly	5/2	-2.5
David	-1	1
Scott	½	-0.5

We can now consider the cosine similarities (where ratings provided to each of the issues are part of a ratings vector that is specific to each architect) between Molly and Scott, as well as Molly and David.

Molly and David:

$$\frac{\left(\frac{5}{2}\right)*(-1)+\left(-\frac{5}{2}\right)*(1)}{\sqrt{\left(\frac{5}{2}\right)^2+\left(-\frac{5}{2}\right)^2}*\sqrt{(1)^2+(-1)^2}}=-1$$

Molly and Scott:

$$\frac{\left(\left(\frac{5}{2}\right)*\left(\frac{1}{2}\right)+\left(-\frac{5}{2}\right)*\left(-\frac{1}{2}\right)\right)}{\sqrt{\left(\frac{5}{2}\right)^2+\left(-\frac{5}{2}\right)^2}*\sqrt{\left(\frac{1}{2}\right)^2+\left(-\frac{1}{2}\right)^2}}=1$$

The cosine angle value is greater for Molly and Scott; Molly's preferences of important issues are closer to Scott's then to David's preferences. In fact, in this example, Molly and Scott's ratings are same because the cosine of angle between Molly's rating vector and Scott's rating vector is 1 (i.e., angle is 0).

Thus, since, Molly is closer in her profile to Jenna, and Molly's issue preferences are closer to Scott's than to David's preferences; issues rated by Scott are also made available to Jenna.

Hence, we present three methods (of different granularities) to provide recommendations to Jenna when she poses an issue: (i.) Recommend Molly's issues to Jenna based on similarity of profiles. (ii.) Recommend issues based on similarity of keywords. (iii.) Recommend Scott's issues to Jenna along with the above recommendations.

5.3 Recommending alternatives (UC3)

We propose an approach similar to the above to recommend alternatives for decision based on the following steps:

(1). Find profiles that match with Jenna's profile (e.g., Molly in the above described example).

(2). Consider the matching profiles that have rated the architectural decision alternatives for the issue (e.g. Issue1 and, Molly and Scott's profiles in the above described example).

(3). For the selected profiles, find the decision ratings that are most relevant (e.g., Molly and Scott's profiles in the above described example).

(4). Recommend a decision based on the most relevant ratings.

Continuing from the example in the last section, we found that a set of profiles that are relevant to Jenna's profile consists of profiles of Molly, David and Scott. Table 8 shows that each of these architects has rated alternatives for Issue 1 along with ratings for the other issue and alternatives

Similar to the example in Table 6 and 7, alternatives would be recommended by normalizing Table 6 based on average ratings and computing cosine similarity.

Table 8. Ratings given by various architects for two issues

Architect	Issue				
	Issue 1		Issue 2		
	Alternative				
	alt11	alt12	alt21	alt22	alt23
Molly	5	0			
David	3	1	3	5	2
Scott	2	2	3	4	0

6. INITIAL EVALUATION

As a first step, we evaluated the concept of recommending stakeholders with similar interests (UC1) because without sufficient stakeholder data the recommender system will not be feasible. We implemented the support for UC1 by extending an existing DB (database) application containing details about project participants. The DB itself has more than 20K users and also contains keywords, role of the participant and contact information. We extend the DB application because it provided an initial data to test UC1. After implementing the recommender capability, we asked 30 participants to use the recommender system. All the participants had prior software development experience. Those participants were obtained based on convenient sampling, that is, their participation was voluntary. After that we asked the participants to rate the quality of recommendations. For the rating we used a 5 point linear scale with 1 as the best rating and 5 as the worst rating. Particularly,

we stated and communicated the scale explicitly to the participants to avoid any confusion during measurements: 5 (Very high), 4 (High), 3 (Fair), 2 (Low) and 1 (Very low). The mean quality obtained was around 3, which implies that the quality of recommendations higher than the average quality (2.5). Based on this we believe that UC1 works reasonably with around 20K stakeholders.

7. VALIDATION OF LINKEDIN SOCIAL NETWORK PROFILES

We have collected the data of 1000 architects from Linkidin profiles. The data contains the architect names and on which application and domain he /she is working. Further, based on their area of expertise (domain context) they are categorized in to two broad categories of Application context i.e. Web-based Application and Desktop Applications. As per the space limitation here we are illustrating the scenario for 10 profiles (See Table 9). Sanjeev was a developer working on Web-based applications in an automation company and he has changed the company and was assigned by the new domain. He got several issues in the domain because he is new to this domain. So he wants to know that who others working in the same or related issues. Proposed system (See Section 4. & 4.2) identifies Cluster 1 of profiles that are similar to Sanjeev using k-medoids algorithm with Jaccard similarity (Table 10 & 11)

Table 9. Linkdin architect profiles

Architect	Application Context	Domain Context
Sanjeev	Web-based application	Finance
Rakesh	Web-based application	Telecom
Thiru	Web-based application	Finance
Sudha	Web-based application	Healthcare
Sreekanth	Desktop application	Healthcare
Rajneesh	Web-based application	Finance
Madhu	Web-based application	Telecom
Manju	Web-based application	Finance
Gautham	Web-based application	Telecom
Niran	Desktop application	Telecom

Table 10. Jaccard index of architects for cluster 1

Architect	Jaccard Index (similarity)	Jaccard Distance (Dissimilarity)
Rakesh	1/2	1/2
Thiru	1	0
Sudhakar	1/2	1/2
Rajneesh	1	0
Madhu	1/2	1/2
Manuju	1	0
Gautham	1/2	1/2

From the Table 10, based on the similarity ranking within Cluster 1 using Jaccard Index, we can say that Thiru, Rajneesh, Manju are most similar to Sanjeev and Rakesh, Sudhakar, Madhu, Gautham are a bit similar to Sanjeev. As the profiles of Sreekanth, Niran (Table 9) belongs to different cluster as their profiles are not similar to Sanjeev.

Table 11. Jaccard index architects for cluster 2

Architect	Jaccard Index (similarity)	Jaccard Distance (Dissimilarity)
Sreekanth	0	1
Niranjan	0	1

7.1 Recommending similar issues

Continuing from the example in the last section, we found that a set of profiles that are relevant to Sanjeev are of Rakesh, Thirumal Bandi and Sudhakar Anivella. Table 12 shows that each of these architects has rated Ix1 and Ix2 which are related to Sanjeev. normalizing the rating given by the architects are listed in the Table 13.

Table 12. Ratings given by various architects for two issues

Architect	ISSUE	
	IX1	IX2
Rakesh	43	**10**
Thiru	83	15
Sudhakar	8	61

By finding the cosine similarity between the Rakesh & Thiru and Rakesh & Sudhakar we can mathematically say that the Rakesh & Thiru profiles are relatively closer to Sanjeev profile. Because the cosine angle between the Rakesh & Thirumal Bandi is greater i.e. 1.

Cosine similarity

Rakesh & Sudhakar:

$$\frac{(16.5)*(-26.5)+(-16.5)*(26.5)}{\sqrt{(16.5)^2+(-16.5)^2}*\sqrt{(26.5)^2+(-26.5)^2}}=-1$$

Rakesh & Thiru:

$$\frac{(16.5)*(34)+(-16.5)*(-34)}{\sqrt{(16.5)^2+(-16.5)^2}*\sqrt{(34)^2+(-34)^2}}=1$$

From the normalized ratings table and cosine similarities we can say that Rakesh and Thiru are closer to Sanjeev.

To comment on the feasibility to implement other use cases of the system, we need to have an estimate about the number of issues that have been created by the stakeholders. Because, the mathematical models behind the approach work appropriately in case of a large number of issues. We did text analysis in the specifications, searched in the existing tools to identify the number of architectural issues that were created across the projects. We identified that around over 80,000 architectural issues were identified and documented in a form in the last 2 years. Therefore, we believe that sufficient data points would be created for using the recommender system. In the following, we summarize major limitations of our evaluation: (i) In order to evaluate the recommender system, the architects need to generate a large number of issues, which would take several years. Therefore, we validated a restricted system in order to generate an initial feasibility report with minimal efforts. A major limitation is that this would not

provide a comprehensive evaluation of the system. (ii) Furthermore, we only measured quality of recommendations, but did not observe how the recommendations are behind used. (ii) All the participants are taken with convenience sampling. Therefore, all our findings would be biased.

Table 13. Normalized ratings for issue1

Architect	ISSUE	
	IX1	IX2
Rakesh	16.5	-16.5
Thiru	34	-34
Sudhakar	-26.5	26.5

8. CONCLUSION

In this paper, we identified synergies between AKM and e-commerce and social computing. We have proposed and illustrated an enterprise wide recommender system for sharing architectural knowledge called RSAKM. The system recommends similar profiles, similar issues and alternatives for a decision. Based on the recommendations, architects could reuse DR for resolving architectural issues quickly and effectively.

Limitations**.** In the following, we summarize the major limitations (L1-L6) for RSAKM.

L1 RSAKM requires enthusiastic architects as well as active involvement of architects for AKM. For example, architects have to add and update their profiles and have to rate issues and alternatives. For example, without keywords and ratings identifying similar issues would not be possible.

L2 In general, recommender systems are used in case where there is huge data. The proposal would make sense in global organizations with large projects so that there is good chance of having a large number of issues.

L3 The approach makes sense for enterprise-wide usage because cross-project knowledge sharing would not be enabled while using the approach for only one project. Therefore, multiple departments of the organization should have commitment for using RSAKM.

L4 The departments/projects using RSAKM should be willing to share data with the other departments/projects within the company.

L5 The system is suitable for a company/organization where there is a need for transparency and well-thought decisions. However, it is not well suited organizations that work based on quick and rapidly changing decisions as well as gamble with decisions. In some environments, managers that encourage well documented decisions are discredited. Implementing our guidelines in those environments is difficult as well.

L6 Furthermore, the above described limitation (L5) would happen partially, that is, the, stakeholders might not work the guidance model for some issues or use it wrongly. In this case, the quality of data for those issues would be poor (e.g. low quality ratings) which influences the overall recommendation items. Those factors were not considered by our model.

L7 The proposed system does not resolve lack of consensus among stakeholders, since, it does not make decisions on an issue. It just recommends decision (alternatives)/decision rationale based on similar issues/resolution strategies that were previously used in the organization. Stakeholders should use RSAKM only as an aid to enable themselves in making a decision by considering pros and cons of the decisions made in the present issue's context.

Future work**.** We suggest the following research items (RI1-RI5) for extending RSAKM:

R11 The RSAKM has to be implemented. The first alternative is to develop an individual tool. Another alternative is to develop a recommender system plugin for an established issue tracking tool (e.g. Jira) so that companies do not need to introduce a new tool throughout the organization. The challenges are migration of legacy issues and decisions as well as integration of the tool with the design environment for maintaining traceability between design elements and issues (see Figure 1).

R12 RSAKM has to be evaluated in a global software development environment of a company. Major considerations for the evaluations are the accuracy of recommendations, identifying minimum number of issues for the recommender system to work, user acceptance for RSAKM as well as for the recommended items, studying improvements in DR sharing and reuse.

R13 Identifying knowledge-sharing and reuse patterns so that architects and the other stakeholders would be focus on the patterns for sharing and reusing as much DR as possible.

R14 RSAKM could be applicable to the other area decisions such as requirements decisions, planning decisions etc. In this paper we focused on the design phase only. This has to be researched how to extend RSAKM for issues related to the complete software lifecycle. The benefits are: (i) large number of issues which aids providing quality recommendations and (ii) improve knowledge management between various life cycle areas.

R15 Decision-making is a complex activity with several considerations. Recommending alternatives based on ratings would not be sufficient. Therefore, the rating based method for UC3 has to be enhanced with qualitative and quantitative methods.

REFERENCES

[1] Lago, P., van Vliet, H., Ali Babar, M., Dingsoyr, T. (2009). Software Architecture Knowledge Management, Theory and Practice. 1st edition. Springer.

[2] Koziolek, H., Domis, D., Goldschmidt, T., Vorst, P. (2013). Measuring architecture sustainability. IEEE Software, 30(6): 54-62.

[3] Dutoit, A., McCall, R., Mistrik, I., Paech, B. (2006). Rationale Management in Software Engineering. Springer.

[4] Zimmermann, O., Miksovic, C., Küster, J.M. (2013). Reference architecture metamodel, and modeling principles for architectural knowledge management in information services. Journal of Systems and Software, 85(9): 2014-2033.

[5] Zdun, U., Capilla, R., Tran, H., Zimmermann, O. (2013). Sustainable architectural design decisions. IEEE Software, 30(6): 46-53.

[6] Thurimella, A.K., Brügge, B. (2013). A mixed-method approach for the empirical evaluation of the issue-based variability modeling. Journal of Systems and Software, 86(7): 1831-1849. https://doi.org/10.1016/j.jss.2013.01.038

[7] Korper, S., Ellis, J. (2001). The E-commerce Book. Second Edition: Building the E-Empire

(Communications, Networking and Multimedia), Morgan Kaufmann.

[8] Thung, F., Wang, S., Lo, D., Lawall, J.L. (2013). Automatic recommendation of API methods from feature requests. ASE, 290-300. https://doi.org/10.1109/ASE.2013.6693088

[9] Robillard, M.P., Maalej, W., Walker, R.J., Zimmermann, T. (2014). Recommendation Systems in Software Engineering. Springer. https://doi.org/10.1007/978-3-642-45135-5

[10] Tang, A., Avgeriou, P., Jansen, A., Capilla, R., Babar, M.A. (2010). A comparative study of architecture knowledge management tools. Journal of Systems and Software, 83(3): 352-370. https://doi.org/10.1016/j.jss.2009.08.032

[11] Ding, W., Liang, P., Tang, A., Vliet, H. (2014). Knowledge-based approaches in software documentation: A systematic literature review. Information & Software Technology, 56(6): 545-567. https://doi.org/10.1016/j.infsof.2014.01.008

[12] Shani, G., Gunawardana, A. (2011). Evaluating Recommendation Systems. Recommender Systems Handbook, 257-297. https://doi.org/10.1007/978-0-387-85820-3_8

[13] Rajaraman, A., Leskovec, J., Ullman, J.D. (2014). Mining of Massive Data Sets. http://infolab.stanford.edu/~ullman/mmds.html

[14] Linden, G., Smith, B., York, J. (2003). Amazon.com recommendations item-to-item collaborative filtering. IEEE Internet Computing, 76-80. https://doi.org/10.1109/MIC.2003.1167344

[15] MacLean, A., Young, R.M., Bellotti, V.M.E., Moran, T.P. (1991). Questions, options 1and criteria. Elements of design space analysis. Human–Computer Interaction, 6(3-4): 201-250.

[16] Lee, J. (1991). Extending the Potts and Bruns model for recording design rationale. [1991 Proceedings] 13th International Conference on Software Engineering, TX, USA. https://doi.org/10.1109/ICSE.1991.130629

[17] Zimmermann, O., Koehler, J., Leymann, F., Polley, R., Schuster, N. (2009). Managing architectural decision models with dependency relations, integrity constraints, and production rules. Journal of Systems and Software, 82(8): 1249-1267. https://doi.org/10.1016/j.jss.2009.01.039

[18] Thurimella, A.K., Bruegge, B. (2012). Issue-based variability management. Information & Software Technology, 54(9): 933-950. https://doi.org/10.1016/j.infsof.2012.02.005

[19] Lee, K., Kang, K.C. (2010). Usage context as key driver for feature selection. Software Product Lines: Going Beyond. Springer Berlin Heidelberg, 32-46. https://doi.org/10.1007/978-3-642-15579-6_3

[20] Stoiber, R., Glinz, M. (2009): Modeling and managing tacit product line requirements knowledge. Second International Workshop on Managing Requirements Knowledge (MARK). https://doi.org/10.1109/MARK.2009.8

[21] Galvão, I., van den Broek, P., Akşit, M. (2012). A model for variability design rationale in SPL. Proceedings of the Fourth European Conference on Software Architecture: Companion Volume, pp. 332-335. ACM. https://doi.org/10.1145/1842752.1842813

[22] Capilla, R., Bosch, J. (2013). Software Variability and Design Decisions. Systems and Software Variability Management. Springer Berlin Heidelberg, 287-292

[23] Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., Tang, A. (2013). What Industry Needs from Architectural Languages: A Survey. IEEE Trans. Software Eng., 39(6): 869-891. https://doi.org/10.1109/TSE.2012.74

[24] Robillard, M., Walker, R., Zimmermann, T. (2010). Recommender systems for software engineering. IEEE Software, 80–86.

[25] Dumitru, H., Gibiec, M., Hariri, N., Cleland-Huang, J., Mobasher, B., Castro-Herrera, C., Mirakhorli, M. (2011). On-demand feature recommendations derived from mining public product descriptions. ICSE, 11: 181-190. https://doi.org/ 10.1145/1985793.1985819

[26] Castro-Herrera, C., Cleland-Huang, J., Mobasher, B. (2009). Enhancing stakeholder profiles to improve recommendations in online requirements elicitation. 2009 17th IEEE International Requirements Engineering Conference, Atlanta, GA, USA. https://doi.org/10.1109/RE.2009.20

[27] Falkner, A., Felfernig A., Haag, A. (2011). Recommendation technologies for configurable products. AI Magazine, 32(3). 99-108. https://doi.org/10.1609/aimag.v32i3.2369

[28] Brosch, P., Seidl, M., Kappel, G. (2010). A recommender for conflict resolution support in optimistic model versioning. In Proceedings SPLASH, 10: 43-50. https://doi.org/ 10.1145/1869542.1869549

[29] Zhang, C., Yang, Y., Zhang, Y., Fan, J., Zhang, X., Zhao, J., Ou, P. (2012). Automatic parameter recommendation for practical API usage. In proceedings ICSE, 12: 826-836.

[30] Rogers, B., Gung, J., Qiao, Y., Burge, J.E. (2012). Exploring techniques for rationale extraction from existing documents. ICSE, 1313-1316. https://doi.org/10.1109/ICSE.2012.6227091

[31] Henss, S., Monperrus, M., Mezini, M. (2012). Semi-automatically extracting FAQs to improve accessibility of software development knowledge. ICSE, 793-803. https://doi.org/0.1109/ICSE.2012.6227139

[32] Guo, J., Czarnecki, K., Apel, S., Siegmund, N. (2013). A variability-aware performance prediction: A statistical learning approach. ASE, 301-311. https://doi.org/10.1109/ASE.2013.6693089

[33] Rodrigues, J.A., Tomaz, L.F.C., Souza, J.M.D., Xexéo, G. (2012). Bringing knowledge into recommender systems. Journal of Systems and Software, 86(7): 1751-1758. https://doi.org/10.1016/j.jss.2012.10.002

[34] Kawai, R., Hazeyama, A. (2010). A know-how recommender system for a software engineering project course by using the content filtering technique. IEEE COMPAC, 547-548. https://doi.org/10.1109/COMPSAC.2010.63

[35] Regev, G., Wegmann, A. (2005). Where do goals come from: The underlying principles of goal-oriented requirements engineering. 13th IEEE International Conference on Requirements Engineering, pp. 353-362. https://doi.org/10.1109/RE.2005.80