Towards Reusing Architectural Knowledge as Design Guides

Functional Requirements, Tool Analysis and Research Roadmap

Mohsen Anvaari
Department of Computer and Information Science
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway
mohsena@idi.ntnu.no

Olaf Zimmermann
Institute for Software
University of Applied Sciences of Eastern Switzerland (HSR FHO)
Rapperswil, Switzerland
ozimmerm@hsr.ch

Abstract—In recent years, architectural knowledge management has demonstrated its potential to improve software development and evolution practices; various tools and research prototypes now exist for documenting architectural knowledge. However, capturing such knowledge is not enough: according to practitioners’ feedback, a certain amount of knowledge post-processing is required to make the captured knowledge consumable and stimulate reuse. In our previous work, we created a method for enhancing knowledge about the past (decisions made) into architectural guidance for the future (decisions required). However, additional concepts are required to let our method benefits from recent advances in architectural knowledge management tool engineering. In this paper we establish requirements for post-processing architectural knowledge captured on projects and enhancing the knowledge into architectural guidance. The requirements are derived from literature and industrial experiences. Next, we analyze existing tools with respect to these requirements. Finally, we establish a vision for an integrated method and tooling for architectural guidance modeling and outline a roadmap for future research and tool development towards this vision.

Keywords—Architectural knowledge; decision reuse; architectural synthesis; design guide; knowledge management tool

I. INTRODUCTION

Architectural decisions are considered a first class entity in software engineering now [7]; researchers define software architecture as a set of architectural design decisions [1]. Various tools and research prototypes exist (or are under development) for documenting the architectural knowledge. Although most practitioners are still reluctant to use formal templates and tools that academic researchers have developed [9], our observations show that many organizations have started to capture their architectural decisions [13]. This is often done in light and pragmatic ways, e.g., using simple wikis or chronological meeting minutes [13].

According to studies on inhibitors for knowledge reuse, documenting the knowledge is not enough; post-processing is required to stimulate the reuse and make the knowledge consumable [20]. Hence, we created a method for enhancing knowledge about the past (decisions made) into architectural

guidance for the future (decisions required) [19]. However, additional concepts are required to let our method benefits from recent advances in architectural knowledge management tool engineering. Therefore, the goals of this paper are:

1. To specify the requirements for tools that facilitate the post-processing of captured architectural knowledge from projects and enhancing such raw knowledge into design guides for future decision making activities.
2. To analyze existing tools and research prototypes with respect to the proposed requirements.
3. To establish a vision for an integrated method and tooling for architectural guidance modeling.

The rest of the paper is organized as follows: In Section 2, we present related work. Section 3 describes our research method. Section 4 specifies the functional requirements for tools that support enhancing architectural raw knowledge into reusable design guided. These requirements are derived from the authors’ industrial experience as well as a review of research prototypes and tools. Section 5 reports on the results of our analysis of existing tools and research prototypes with respect to the functional requirements from Section 4. Section 6 analyzes our results and establishes an architectural vision for a tool that supports architectural knowledge reuse. Section 7 summarizes the paper with conclusions.

II. RELATED WORK

The concept of architectural knowledge – defined as the integrated representation of the software architecture, the architectural decisions, and the external context/environment [23] – has been investigated by researchers since they started to consider the architectural decisions as important entities of a software system just like the architecture itself. In the last decade, the research community has elaborated the concept, clarified its definitions, terminologies and boundaries, established the ways of presenting the knowledge, and developed the approaches to manage the knowledge [12].

Applying general knowledge management principles [11] to the software engineering domain, two activities become essential for architectural knowledge management: creating (or capturing or documenting) knowledge and consuming (or
reusing or applying) knowledge. However, as we have shown in our previous work, the main focus of the software architecture community so far has been on knowledge capturing, not on knowledge reuse [20]. One may argue that when the knowledge is captured and made available to the others it is reusable; therefore any approach and tool that supports knowledge capturing also supports knowledge sharing and reusing automatically. But, according to practitioners’ feedback, capturing the knowledge is not enough: a certain amount of knowledge post-processing is required to make the captured knowledge consumable [20]. Examples of knowledge post-processing are anonymizing the knowledge (e.g., remove sensitive personal information such as names of actual people, or replace them with role definitions such as “application architect” or “integration architect”), connecting the related knowledge entities (e.g., a decision about a message exchange pattern with a decision about a messaging provider software product), and removing the project-specific knowledge (chosen alternative) to make the knowledge reusable for other projects.

While the method we have created in our previous work for reusing architectural knowledge and enhancing the captured architectural knowledge into design guides has demonstrated to be useful in the industry [19], better tool support is required to make the application of the method more efficient. As the first step, this paper explores the available architectural knowledge management tools and research prototypes to analyze how much they provide the required functionalities for architectural knowledge reuse and guidance development.

This paper is not the first survey in the software architecture domain to analyze and evaluate the architectural knowledge management tools. At least four preceding research papers have been published [2][14][16][25]. Although some papers have considered knowledge sharing as a functional requirement in their evaluation framework, their focus is not on knowledge post-processing and enhancing the knowledge into design guidance, which is the focus of our work. Furthermore, the last survey has been published in 2010 while in the last three years more tools and research prototypes have been developed or are under development. This paper covers these new tools as well. The mentioned research papers are valuable for our work; for instance, we have reused some of their functional requirements to establish the functional requirements in the section 4. In the next section, the method of the research will be explained.

III. RESEARCH METHOD

As Fig. 1 shows, we started the research by creating functional requirements. To do so we explored three sources: 1) industrial experiences that originate in the authors’ contribution in industrial projects and also their observations from various industrial domains (software development projects for Smart Grid, financial applications, etc.) 2) the tools and research prototypes that are not accessible but are specified in the literature 3) the tools that are accessible publicly on the internet. The main way to reach to the second source was exploring the literature that has reviewed and compared the architectural knowledge management tools. As we mentioned earlier, the last comparative study of the architectural knowledge management tools was conducted in 2010 [2]. It still is a valuable source to explore the tools that had been developed until then. We discovered the newer tools either through the literature that tool developers have published or by contacting the researchers that we were informed are developing a tool.

Figure 1. Research activities and contributions

The second step of the research was to analyze the tools and research prototypes with respect to the functional requirements we proposed in the first step. The tools that are publicly accessible (the third source) were a more valuable source for us, because we could actually use them and test their functionalities against the list of requirements. Some of the inaccessible tools were also valid for analysis since their functionalities have been described concretely in the literature. We finalized the list of tools for analysis using the following criteria:

- The tool should be publicly accessible. For example, ADkwik [18] that is covered by previous tool evaluation studies [25] would be a candidate for our analysis, but it is not accessible anymore. However, there are some tools that are under development and therefore are not released yet, but their functional requirements are concretely described in literature; such tools do meet the criterion. One such tool is analyzed in Section 5.
- The installation and usage of the tools should be straightforward.
- The tool should be representative for its domain. For example if tool A and B exist for capturing the decisions, and tool B covers all features of tool A, we just choose tool B.

The third and last step was to summarize the tool analysis results. Based on the analysis results, we establish a vision for developing a tool that supports post-processing the captured architectural knowledge and developing a design guide.
IV. FUNCTIONAL REQUIREMENTS

This section describes functional requirements for developing an architecture guidance modeling tool. We will use the functional requirements to analyze the existing tools later (in Section 5). In the following, we will define some terms that are essential for describing the functional requirements first. Next, we will describe the actual functional requirements.

A. Definitions

The required tool for supporting architecture guidance modeling should create and maintain a knowledge base (KB) that contains architectural entities. The tool should be able to post-process the entities (see Section 2 for examples) and enhance them into a design guide. Inspired by our previous work [19], we define the following entities that can be added to a KB:

- **Issue**: Any design issue that may occur in a software development project. It includes different properties mainly name, problem description, decision drivers and solution alternatives. Each alternative includes pros and cons, known uses and related background. For example in an enterprise architecture, an issue can be “enterprise integration pattern for designing the message channel between system A and system B”. The alternatives of the issue are “point-to-point-channel” and “publish-subscribe-channel”.
- **Decision**: A decision inherits its properties from an issue, but adds the outcome of the decision (chosen alternative and the rationale behind that). Therefore an issue can be converted to a decision by adding the outcome and vice versa. In the example we provided, the outcome may include “the publish-subscribe-channel” as the chosen alternative and “high number and change rate of the data sinks” as the rationale of the decision.
- **Group**: A group entity is an aggregation (or assembly) of other entities. An example for the usage of this container concept is a software project that includes some sub-projects and each sub-project includes issues and decisions.

This structure is not the only way of modeling the architectural entities. Tools can apply other metamodels such as those presented in [24] or [27].

B. Functional Requirements

We categorize the functional requirements in two groups: 1) create and maintain knowledge and 2) consume knowledge. However, these categories have some overlaps and some requirements can belong to more than one category.

1) Create and Maintain Knowledge

- **AddE** – Add an entity [6][17][25]: Insert an entity to KB. The following features are required:
  - **Rich text editor**
  - **A tag field (or a semantic-based approach) to make search easier**

2) Consume Knowledge

- **Entity identification**
- **Entity name**
- **Entity description**
- **Entity stakeholders**
- **Entity version**
- **Entity confidentiality**
- **Issue level** (e.g. conceptual, technology, vendor asset [21])

Before the tool inserts an entity to KB, it should first search for available related entities and if there are some, it should suggest them to the user. If the user finds that the entity is already available in KB, (s)he can decide to cancel the procedure. This helps to reduce the redundancy. Sometimes the entity is not already available, but the search brings some related entities and the user can connect the new entity to the related ones (CnctE).

- **UpdE** – Update an entity [17][25]: Update an available entity. The features that are required for AddE apply here as well.
- **RmvE** – Remove an entity [17]: Remove an entity from KB. It should clean up all of the relations of the entity to the other entities.
- **MovE** – Move an entity [17]: Move an entity from one group to another group.
- **CnctE** – Connect (Relate) to an entity: Connect an entity to other entities (examples are issue to issue, issue to decision, issue to group).
- **UnlkE** – Unlink an entity: Unlink an entity from its parent or from its related entities without deleting the entity.
- **RevE** – Review an entity [6][25]: If an entity is supposed to be reviewed before inserting to KB, it should be sent to the reviewers. The reviewers should validate the entity before a specific time. Then based on the rates or opinions the reviewers give to the entity, the entity can be rejected, inserted (or edited and inserted) to KB.
- **ImpE** – Import an entity [17]: Import an entity from a file (for example a XML or JSON file) or a URL.
- **AnmE** – Anonymize an entity [17]: Sometimes an entity can be reused or shared or exported, but the project-specific information should not be shared with the others. This feature replaces the project-specific terms with a pseudonym.
- **MkeD** – Make a decision (convert an issue to a decision) [17]: When a decision on an issue is made, the decision should be inserted to KB. This functionality adds an outcome part to the issue and converts it into a decision.
- **Gnrl** – Generate an issue from a decision: Sometimes a decision is made, but the related issue is not in KB. This functionality generates an issue from a decision by removing the outcome part of the decision entity.
(and adding a possible recommendation). This feature is essential for upgrading decisions to guides (UpgG).

- **UpdD** – Upload documents [17]: An entity may include background information either as a link or as a document. This feature uploads a document to the entity.

- **NtS** – Notify stakeholders [25]: Sometimes an entity would have more than one owner (stakeholder). This feature reports any changes to the entity to all stakeholders. Stakeholders should be able to disable/enable this feature. The implementation examples of this requirement are RSS (Rich Site Summary) and Atom [4].

- **Conf** – Configure the tool: The metamodel (default profile) that are required to insert an entity (such as entity version, entity stakeholders, entity confidentiality, issue level and so on; check AddE for more details), should not be fixed and unchangeable. This functionality customizes the attributes based on the project or organization needs.

2) **Consume Knowledge**

- **SrE** – Search an entity [17][25]: Search KB for a group, an issue or a decision. The search can be done based on an entered text or by choosing an entity to find the relevant entities. Advanced search to limit the search results based on level, confidentiality, project, etc. should be supported.

- **LstE** – List entities: Make a list of all entities of a group.

- **NKB** – Navigate the knowledge base [17]: The user should be able to navigate between the groups, their sub-groups and their issues and decisions.

- **VwE** – View an entity [6][17][25]: When the user finds the list of entities by searching them, or navigating KB, (s)he should be able to view each entity and its properties.

- **RseE** – Reuse an entity [6][25]: Choose an entity in group A and copy it into group B. The confidentiality (access permission) of the entity should be checked first. The owner of group B might be able to view the entity of group A, but not to reuse it.

- **ExpE** – Export an entity [17]: Export an entity to a file. The confidentiality of the entity should be checked first: if export is not allowed by the owner of the entity, the export procedure should be rejected. If anonymized export is allowed, the entity should be anonymized first and then be exported. If export is allowed unconditionally, the entity can be exported without any pre-processing.

- **ShrE** [6][25] – Share an entity: Send the link of an entity to other stakeholders by email notification. First the confidentiality of the entity should be checked. Sharing is possible only if the entity is public or other stakeholders have access to the group.

- **UpgG** – Upgrade to guide: Convert past architectural knowledge into guidance for the future. First, the tool will ask about anonymization. If the user requests anonymization, the feature will anonymize all entities of the group (AnmE), otherwise leave them unchanged. The next step is to look up each decision and its related issue. If both a decision and its related issue are available, the feature will remove the decision (RmvE). If only the decision is available, it will generate an issue from the decision (GnrI) and remove the decision (RmvE) afterwards. The final result is a group and all of its sub-groups and each sub-group includes a list of design issues and each issue has some alternatives. This can be shared (ShrE) or exported (ExpE) as a design guide. Assume that a software developing firm has a project for developing a system for customer A. They have captured the decisions in a group called project A. The group includes various sub-groups (A1, A2, A3, etc.). Now by using this feature, they will have a list of issues and alternatives for each of the sub-groups and they can use it as a guide for making decisions in a similar project for developing a system for customer B. The architects and designers involved in the new project could be different, but the knowledge from previous project is reused in a structured manner.

- **DAPI** – Documented application programming interface: The tool should provide a public documented API to make it possible to be integrated with other architectural knowledge management or design modeling tools.

In the next section, we present the results of analysis of existing tools with respect to the mentioned functional requirements.

V. **Analysis Results**

This section reports on the results of our analysis of existing tools and research prototypes with respect to the functional requirements from Section 4. First, we briefly introduce the five tools that are nominated for the analysis, and then we present the functional requirements that are satisfied by these tools.

1) **SAW.** Software Architecture Warehouse (SAW) is a Web-based tool to capture, manage and analyze architectural knowledge. It is implemented to help the entire software architecture design team achieve situational awareness about architectural decisions [15].

2) **Decision Viewpoints.** Decision Viewpoints is a documentation framework for architecture decisions. It uses the conventions of ISO/IEC/IEEE 42010 [26]. A tool is developed supporting the framework as an add-in for Sparx Systems’ Enterprise Architect [5].

3) **AREL.** Architecture Rationale and Elements Linkage (AREL) is a Sparx Systems’ Enterprise Architect plug-in that
creates architectural design with a focus on design rationale [3].

4) SEURAT. Software Engineering Using RATionale system (SEURAT) is an Eclipse plug-in that aims to manage architectural knowledge from requirements to source code [8].

5) Eclipse Process Framework (EPF)². EPF is an Eclipse-based method creation tool. In EPF, knowledge creation takes place in the tool; knowledge consumption, on the other hand, can be done in a Web browser.

Table I. shows which functional requirements are supported and which are not supported or partially supported by the introduced tools.

<table>
<thead>
<tr>
<th>FR</th>
<th>SAW</th>
<th>Decision VP</th>
<th>AREL</th>
<th>SEURAT</th>
<th>EPF</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddE</td>
<td>Partially</td>
<td>Partially</td>
<td>Partially</td>
<td>Partially</td>
<td>Partially</td>
</tr>
<tr>
<td>UpdE</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>RemE</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MovE</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>UlnkE</td>
<td>Yes</td>
<td>Yes</td>
<td>Partially</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>CnctE</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>RevE</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Partially</td>
</tr>
<tr>
<td>ImpE</td>
<td>Partially</td>
<td>Partially</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>AnmE</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>MkeD</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>GenI</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>UplD</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>NtfS</td>
<td>Partially</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Conf</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Partially</td>
<td>No</td>
</tr>
<tr>
<td>SrhE</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>LstE</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>NKB</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>VwE</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>RuseE</td>
<td>Partially</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>ExpE</td>
<td>Partially</td>
<td>Partially</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>ShrE</td>
<td>Partially</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>UpgG</td>
<td>Partially</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>DAPI</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table I. Analysis of Tools in a Nutshell³

² http://projects.eclipse.org/projects/technology.epf
³ Detailed evaluation results omitted due to space constraints, but are available upon request.

VI. Vision for Future Research

In the previous section, we presented the results of our analysis. It showed the functional requirements that are supported by the available tools and research prototypes. Based on the data Table I. provides (the functionalities that are not focused by the available tools), we establish some directions for the next steps towards tool developing for architectural knowledge reuse and architecture guidance modeling:

1) Separating issue (decision required) from decision (decision made): A design guide is mainly a list of design issues (decisions required) and their possible solutions (alternatives). To create a guide from captured decisions (decisions made), issue and decision should be separated and the tool should provide the possibility to generate an issue from a decision (Gnrl). This will also make organizations less reluctant to share their knowledge with a community; because it guarantees that only the issue and its alternatives will be shared with the community and others will not be informed about their decision (chosen alternative).

2) Providing default profile: One of the main reasons architects state for their unwillingness to use architectural capturing tools is the time limitations [9]. To overcome this, tools should make capturing knowledge less time consuming. One of the solutions is providing a default profile for adding entities to the knowledge base.

3) Providing knowledge confidentiality: As we mentioned earlier, organizations are not eager to share all of their architectural knowledge with the community. There can be even a situation that in one organization, the knowledge of one project should not be shared with other projects. Therefore, the confidentiality level of an entity should be defined for adding the entity to the knowledge base. The tool should always consider the confidentiality and intellectual property rights level of an entity before sharing, reusing or exporting the entity or creating architecture guidance (e.g., “open”, “copyright protected”, “company-internal”, and “confidential”).

4) Configuring metadata: Users should be able to customize the metadata (attributes profile) based on their organizational policies and concerns. IEC/IEEE/ISO 42010 is one, but not the only template to be supported (many more have been defined, e.g. [10]).

5) Considering semantic tags: The architectural knowledge base can grow very fast. Navigating a large knowledge base can be painful, e.g. if it takes a long time to find a knowledge entity. Providing semantic tags will make searching the knowledge base easier and more precise.

6) Searching the knowledge base before inserting new knowledge: To create a useful yet concise architecture guidance it is essential to reduce the redundancy of knowledge. To reach that, the tool should search the knowledge base in advance to inserting any new knowledge. It is also useful for finding relevant knowledge and connecting them together.

7) Being consistent with real world situation: In reality, large organizations develop software within various projects and sub-projects. The design guide would be more usable if it
was categorized into projects and sub-projects. Therefore grouping the entities of knowledge base to projects and sub-projects should be provided.

8) Anonymizing the knowledge: Rather than separating issues from decisions, anonymizing the knowledge also makes organizations more eager to share their knowledge with the community (see Section 2 for an example of a required anonymization).

9) Providing programming interface: The activities related to architectural knowledge management are very wide and it is not possible to have a holistic tool that supports all activities. The focus of the proposed tool in this research is on reusing architectural knowledge and enhancing a design guide. The tool should therefore provide an interface (API) to make it possible to be integrated with other architectural knowledge management or design modeling tools such as general-purpose wiki engines and Unified Modeling Language (UML) tools.

VII. CONCLUSIONS

Reusing architectural decisions as design guides gives these decisions a more proactive role and therefore makes decision management more appealing and relevant to practitioners. In this paper, we leveraged our industrial experiences, our previous research work and also the current literature in the architectural knowledge community to establish functional requirements for future knowledge management tools that enhance architectural decisions to design guides. With respect to the functional requirements, we analyze representative tools and research prototypes. We reported that the available tools and research prototypes have made significant contributions in the area of architectural knowledge capturing, but still require a number of extensions so that the captured decision can serve as design guides in practice. We finalized the paper with a vision for method integration and tool improvement.

In the next step, we are going to evolve our design guidance enhancing framework to decrease the time and effort of design guidance generating by applying automatic information extraction approaches. The extracted architectural entities will feed the knowledge base (KB) in a more efficient way. We also intend to extend and integrate our method into existing and emerging tools (our own tools and those developed in the research community) – applying the vision we established in this paper.

REFERENCES