# Architectural Decision Guidance across Projects

## Problem Space Modeling, Decision Backlog Management and Cloud Computing Knowledge

Olaf Zimmermann, Lukas Wegmann
Institute for Software
Hochschule für Technik (HSR FHO)
Rapperswil, Switzerland
{firstname.lastname}@hsr.ch

Heiko Koziolek, Thomas Goldschmidt
Research Area Software
ABB Corporate Research
Ladenburg, Germany
{firstname.lastname}@de.abb.com

*Abstract*— **Architectural Knowledge Management (AKM) has been a major topic in software architecture research since 2004. Open AKM problems include an effective, seamless transition from reusable knowledge found in patterns books and technology blogs to project-specific decision guidance and an efficient, practical approach to knowledge application and maintenance. We extended our previous work with concepts for problem space modeling, focusing on reusable knowledge, as well as solution space management, focusing on project-level decisions. We implemented these concepts in ADMentor, an extension of Sparx Enterprise Architect. ADMentor features rapid problem space modeling, UML model linkage, question-option-criteria diagram support, meta-information for model tailoring, as well as decision backlog management. We validated ADMentor by modeling and applying 85 cloud application design decisions and 75 workflow management decisions, creating one problem and three sample solution spaces covering control system architectures, and obtaining user feedback on tool and model content.**

***Keywords—agile practices; architectural synthesis; architectural decisions; cloud computing; knowledge management; patterns; UML;***

## I. INTRODUCTION

Lindvall and Rus stated that "the major problem with intellectual capital is that it has legs and walks home every day" [10]. In response to this threat, codification and personalization strategies can be applied. The Architectural Knowledge Management (AKM) community has proposed support for these strategies in metamodels, methods, and tools since 2004 [1]. Most AKM work focusses on capturing architectural decisions after the fact; e.g., the ISO/IEC/IEEE standard 42010:2011 recommends the capturing of decision rationale in architecture descriptions [8]. Such rationale answers why-questions concerning architectural synthesis. However, little attention has been paid to making the gathered knowledge applicable to multiple projects across organizations and to give decisions a guiding, lasting role in design processes.

According to our personal experiences gained on industrial research projects as well as many years in product development and professional services, the potential value of knowledge sharing is acknowledged by many architects (both junior and senior) – but still hard to realize in practice. Three reasons for these difficulties include a) the diversity in technology, b) the long lifetime of decisions throughout software evolution, and c) the differences in architecture design practices from agile to "big design upfront"; this is particularly relevant in global firms with many geographically distributed, federated and loosely coupled development organizations. Project constraints such as frequent due dates, budget limitations and stakeholder pressure cause a lot of precious knowledge to remain tacit (i.e., in people's heads) [20]. In our previous work, we have presented solutions to make architectural decisions sustainable [24] in response to issue b) and to organize the knowledge into abstraction-refinement levels [27] in response to issue a). However, issue c) remains open; let us investigate it further.

Most decision documentation tools suffer from the lack of incentives for mere decision capturing and other real-world inhibitors. Capturing decision knowledge in text documents after-the-fact is a promising start, but bound to fail in the long run as document-oriented decision logs are hard to process and maintain once they reach a certain size (say, 50 to 70 decisions captured). With such approach, sequential reading or full text searches are the main processing options. Decision documentation models promise to improve the situation, but have to be created, organized, and maintained over time as well.

Both textual decision logs and documentation models are unable to steer the initial design and review work on a project. For instance, development processes using a milestone-based approach such as Stage Gate® [19] may ask for a number of decisions to be made and documented before a milestone or gate can be passed (e.g., make-or-buy decisions and platform decisions that require product purchases). The existing decision capturing methods and tools do not make the need for such mandatory decisions explicit in the design process; they only allow decision makers to record (log) their decisions once they have identified and made them. As a consequence, the preparation for gate reviews or milestone approval meetings is time consuming and error prone, involving a lot of copy-paste and other manual content assembly work. To give an example: in cloud computing, architectural knowledge e.g. around workload exists in various forms from blog posts to white papers and patterns books [3]; this is precious, but passive knowledge. Related decisions are not made explicit; if the literature is not consulted, required decisions might be missed.

Decision guidance modeling practices and tools are still immature and confronted with a healthy amount of skepticism within the target audience, e.g., concerns about the organizational feasibility of cross-product, cross-unit reuse of knowledge as well as resulting maintenance efforts.

To overcome these problems, we separate *decisions required* from *decisions made*. The resulting research questions (RQs) to be investigated in this paper are:

*RQ 1:* How to model decisions required so that a) they are applicable to diverse projects, b) do not age fast e.g. due to technology evolution, and c) are simple to maintain over time?

To answer RQ1, we supersede previous metamodels for decision capturing and sharing with lean knowledge quadruples that give decisions a guiding role that works effectively and efficiently both in traditional and in agile settings.

*RQ 2:* How to integrate decision modeling concepts into architecture design practices and tools commonly used by architects to evolve their designs and record decisions made along the way, without creating more effort than gains?

To respond to RQ 2, we annotate the decision knowledge with meta-information, leveraging already existing organizing principles such as viewpoints, refinement levels, and project stages. Decision capturing is streamlined by leveraging lean documentation templates (from practitioner literature) flexibly.

We make our solutions (answers) to RQ 1 and RQ 2 available in a new add-in to the modeling tool platform Sparx Enterprise Architect, and demonstrate their value by creating decision knowledge for cloud computing and other domains.

The remainder of this paper is structured in the following way: Section II discusses the state of the art and the practice with respect to our research questions and derives requirements from it. Our novel research contributions towards completing the vision for an active, guiding role of architectural decisions to accelerate design work are introduced in Section III. Section IV presents an implementation of these concepts, Section V our further validation activities that include action research and experiments with architects from industry and their knowledge. Section VI then discusses the validation results and practicality of our approach. Section VII concludes and highlights future work and other opportunities.

## II. State of the Art and Practice (and Derivation of AKM Tool Requirements)

Since an inaugural workshop held in Groningen, NL in 2004, the software architecture community has advanced the state of the art in AKM significantly [1]. Some success with decision guidance modeling has been reported e.g. for enterprise applications and service-oriented architectures [27]; however, it is not a widely adopted practice yet.

To avoid unnecessary overlap with previous publications from the community, we structure this section into a comparison of seven templates that have been reported to be actively used in practice (Section II.A), briefly revisit research prototype tools from other researchers (II.B), and then summarize our previous work on the subject (II.C, II.D). Finally, we derive requirements for design and implementation of ADMentor from the state of the art and the practice.

### A. *ISO/IEC/IEEE 42010 and Practitioner Templates*

Table 1 compares the guidelines in IEEE 42010 (and its companion template available under a Creative Commons License) with six other Architectural Decision (AD) capturing templates used in industry, ordered by their age: IBM UMF [25], the Tyree/Akerman template [22], a key decisions template from Bredemeyer Consulting [6], M. Nygard's ADR blog post [14], an arc42 resource by Hruschka/Starke [7], and our own Y-statements [24]. We selected these seven templates due to their public accessibility and their know uses on industry projects. The selection is not complete, but representative.[1]

The comparison in the table shows that there are many formats, with consensus about the core attributes/aspects (e.g., AD outcome and why-justification), but significant variability regarding traceability links and other types of attributes/aspects (e.g., status and owner of decision). The number of template attributes/aspects varies; hence, the effort to fill out varies too. IBM UMF and Tyree/Akerman are the most comprehensive templates; Nygard's ADRs, arc42 and Y-statements appear at the other end of the spectrum. Note that not all of the attributes/aspects are needed when predicting future decisions in decision guidance models (e.g., outcome, status); a generalized, forward-looking subset is enough (e.g., decision drivers, options to be considered). We can conclude that tools for decision capturing and sharing should be flexible and configurable to accommodate use of these (or other) templates.

### B. *Research Tools (Brief Recapitulation)*

AREL [21] is an add-in to Sparx Enterprise Architect, like Decision Architect (see Section II.C below) and ADMentor (described in this paper). Like most AKM tools, AREL focuses on decision rationale capturing. AREL defines a UML profile, but does not mandate the decision capturing attributes. Hence, our work can be seen as a continuation and extension of the research around AREL.

We refer the reader to Chapter 6 in [1] and two recent conference publications, a requirements-based tool evaluation [2] and a systematic literature review [24] for information about other AKM/AD tools; recent additions to the portfolio include ADvISE and SAW. Most of the existing tools focus on decision capturing, not on decision guidance; such tools do not answer our two research questions from Section I satisfyingly.

### C. *Decision Architect (Documentation Viewpoints)*

The work reported in this paper is connected to the formerly published Decision Architect [11], an add-in for Sparx Enterprise Architect implementing a conceptual framework for architecture decision documentation based on ISO/IEC/IEEE 42010. The framework consists of five decision viewpoints: relationship, chronology, stakeholder, forces and detailed viewpoint. With the implementation as an add-in to an UML editor, it is possible to link architectural decisions to UML model elements thereby realizing traceability. In [11], we reported the application of Decision Architect by five software architects with good feedback.

For the latest version 0.5, released as open source in September 2014, the Decision Architect features re-designed forces and detailed viewpoints and numerous bug fixes. In contrast to ADMentor as introduced in this paper, Decision Architect is meant mainly for documentation purposes, but not

[1] Note that we did not include any scientific approaches here; see e.g. the related work coverage in [2], [11] and [23] for analyses and comparisons.

TABLE I. COMPARISON OF PRACTITIONER TEMPLATES FOR ARCHITECTURAL DECISION (AD) CAPTURING

| **AD aspect (attribute)** | **IEEE 42010 (Template V2.2)** | **IBM UMF AD Table** | **Tyree/ Akerman** | **Bredemeyer Key Decisions** | **Nygard ADRs** | **arc42 Hruschka/Starke** | **Y-Statements (ABB, [24])** |
|---|---|---|---|---|---|---|---|
| *ID* | Unique Identifier | ID | (in D-Header) | / | (part of name) | (Section #) | (Id) |
| *Outcome* | Statement of the decision | Decision (Made) | Decision | Approach | Decision | Decision | we decided for |
| *Requirements trace (FRs, NFRs)* | Correspondence or linkage to concerns | (Derived requirements) | Related requirements | Business drivers, technical drivers | / | / | / |
| *Accountability (Role, Person)* | Owner of the decision | / | / | / | / | / | / |
| *Software architecture viewpoint trace* | Correspondence or linkage to elements | / | Related artifacts | / | / | / | In the context of |
| *Why-answers* | Rationale (linked entity) | Justification | Argument | Conclusion | / | (Question under Decision) | (optional "because" half sentence) |
| *Decision drivers* | Forces, constraints | / | (Constraints) | Benefits, Drawbacks | Context | Constraints | facing |
| *Assumptions* | Assumptions | Assumptions | Assumptions | / | / | Assumptions | / |
| *Options* | Considered Alternatives | Alternatives | Positions | / | / | Considered Alternatives | and neglected |
| *Problem* | / | Issue or Problem | Issue | / | / | Problem | / |
| *Decision dependencies* | (not in template, but in standard) | Related decisions | Related decisions | / | / | / | / |
| *Categorization, classification* | / | Subject Area, Topic | Group(ing) | / | / | (Decision Topic) | / |
| *Name* | / | Name | (in D-Header) | <<key decision>> | Title | (Section heading) | / |
| *State of AD making* | not in template, but in standard | (not in published example) | Status | / | Status | / | / |
| *Impact* | not in template, but in standard | Implications | Implications | Issues/ Considerations | Consequences | / | to achieve, accepting that |
| *Other entries* | Timestamps, Citations | Motivation | Notes, Related principles | Notes, Drivers realized | / | / | / |
| **Element count** | *9 (template), 11 (standard)* | *13* | *14* | *9 (plus 1-2 in header)* | *5* | *5 (with 14 questions)* | *6* |
| Scoping help (which ADs to capture?) | Yes | (not in table template, not published) | (anecdotal In article) | 2001 white paper | / | (ASRs mentioned) | / |
| Size (page or word limit) or other hints | / | not published | (example is half a page, table form) | / | 1-2 pages per ADR | to be ordered by importance | 1 (long) sentence per Y-statement |
| **Publication year** | *2011* | *1998 (internal)* | *2005* | *2005* | *2011* | *2012* | *2012* |

for architectural guidance: there is no separation between problem and solution spaces, and it is not possible to import generic problem domain models. However, Decision Architect and ADMentor can be connected within Enterprise Architect to allow both detailed documentation and architectural guidance.

## *D. SOAD and SDA (2006-2011)*

The SOA Decision (SOAD) Modeling project[2] [28] and Solution Decision Advisor (SDA) tool [13] addressed similar research problems as we do here, but chose different solutions both on the conceptual and on the technical level: for instance, options were modelled as child elements of problems, whereas ADMentor sees options as first class model elements; multiple options occurrences can be chosen per problem occurrence. Furthermore, the metamodel of ADMentor is not hardcoded or fixed otherwise, but extensible via typed meta-information attributes/annotations (see Section III). Moreover, ADMentor defines a UML Profile (see Section IV) and leverages Sparx Enterprise Architect as a UML and general modeling platform.

**Derivation of requirements.** Let us now establish additional requirements for ADMentor.

*Functional Requirements (FRs)***.** Tools for decision capturing and sharing must support:

- A user interface applying the master-details pattern so that both the big picture and the nuts and bolts of individual problems and options can be portrayed.
- Rich text editing (e.g., URIs, bullet lists, emphasis on certain words with italic and bold fonts, headings).
- Powerful model refactoring capabilities.
- Semantic queries (e.g., returning all problems to be solved for particular milestone or gate).
- Reporting, e.g. RTF/PDF/HTML exports that can be customized for project stakeholders that are involved in the decision making, but do not work with the tool.

[2] The usage of SOAD concepts and method is not limited to SOA design. However, ADkwik, the tool for SOAD, is no longer available publicly.

- An API for semi-automatic model creation/updating and tool integration.

FRs for AKM tools, both for retrospective and for proactive methods and tools, have also been published in [2] (and other literature referenced in Section II.A to II.D).

*Other success criteria.* In our opinion, mere architectural decision trees resembling Unified Modeling Language (UML) visualizations are valuable, but not sufficient to represent the nature of design knowledge adequately. Successful decision capturing and sharing requires a combination of text processing capabilities (for a powerful, but still lean knowledge authoring) and graph-oriented visualizations (e.g., of dependencies).

A conceptual integration into both agile and more conservative design process practices also is required. This can be achieved with rich meta-information attribution/annotations with appropriate default values to minimize work (also including a confidentiality flag).

From the related work analysis and comparison of practiced approaches (template analysis in Section II.A), we can conclude that even when using light templates (e.g., Nygard's ADRs, arc42 pages, or Y-statements), decision documentation remains a lot of (typically unwelcome) work. AKM/AD tools like Decision Architect help, but do not reduce the effort sufficiently. According to our experience and community feedback, a  tool cannot dictate a single set of attributes (knowledge structure), but must be flexible and accommodate a variety of architectural capturing and sharing styles in a best-of-breed manner.

Before we introduce the contribution of this paper, let us now establish a running example indicating that many architectural decisions recur indeed. This example will serve as an exemplary instantiation of our concepts later in the paper.

**Modelling and usage example.** An example of a recurring design issue when moving a Web application to a cloud is session state management (e.g., think of a shopping session in an online store). The three top-level design options (patterns) are client session state, server session state and database session state [4]. Client session state scales well, but has security and possibly performance problems; this does not change when moving to a cloud platform. Server session state uses main memory or proprietary data stores in an application server (e.g., an HTTP session in a JEE servlet container); this approach is no longer recommended when deploying to a cloud due to scalability and reliability concerns. Finally, database session state is well supported in many clouds, e.g. via highly scalable key-value storages (NoSQL).

If session management is a requirement, this decision has to be made or revisited when designing a cloud-native application or when migrating a Web-based application to the cloud; while the decision outcomes may vary, the issues and options to be considered stay the same (i.e., they recur). Multiple instances of this decision may exist per project (e.g., in multi-channel applications that serve different user groups).

## III. Conceptual Design of ADMentor

This section presents our research contributions and is organized into six steps: (A) metamodelling and problem space creation, (B) problem space modeling, (C) meta-information annotation, (D) tailoring, (E) solution space creation and (F) solution space usage (decision backlog management).

### A. Metamodeling and Problem Space Creation

Figure 1 specifies our AKM model structuring. A four-quadrant design space structure is shown in the bottom half of the figure (in the last two table rows and columns). The upper part of the figure elaborates on the semantic differences between *problem spaces* and *solution spaces*, i.e., their different reach (i.e., scope and lifetime), owner role (i.e., model creator and maintainer), and purpose.

| *Model Type* | **Problem Space** | **Solution Space** |
|---|---|---|
| *Reach/Level* | Asset (Community) | Project |
| *Owner* | Knowledge Engineer | Software Architect |
| *Purpose* | Design Guidance | Decision (Back-)Log |

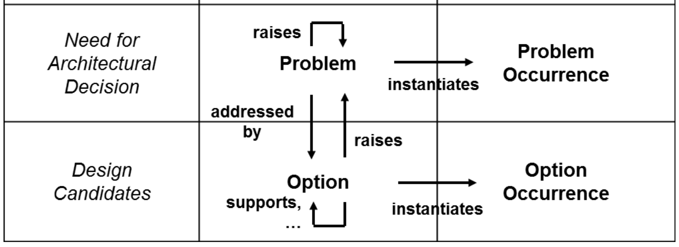


Fig. 1. Problem/solution space model elements and their link types

Problems are *addressed by* options. Problems and options can *raise* (i.e., lead to) further problems that need to be solved. Recurring problems can be instantiated one or more times in a solution space; option occurrences *instantiate* options (also one or more times). Additional, self-explanatory link types connecting options, not shown in Figure 1, are: *suggests*, *conflicts with*, and *bound to*. In our session state management example from above, there might be two problems, a) the conceptual decision which pattern to use (with the three state management patterns from above modeled as options) and b) a technology-level decision which storage medium to use for database session state (with options like relational MySQL database and MongoDB key-value store, among others). The database session state option of the conceptual pattern selection decision raises the technology-level decision about storage.

This approach is faithful to our earlier vision of architectural decision modeling with reuse, but also different and more advanced: we model problem spaces and solution spaces separately, and provide two abstractions each, problem and solution (yielding four structural elements instead of three). We also define novel link types connecting the four elements. We decided to define semantically rich links that can be used in queries etc., inspired by existing work in the AKM community (see Section II) and by REST maturity level three [5], which promotes hypermedia controls as/for typed link relations.

The following Figure 2 (on the next page) shows the six processing steps dealing with the elements from the quadrants from the bottom half of Figure 1. The processing steps are named A to F; they are associated with the owner roles from Figure 1, knowledge engineer and software architect (e.g., solution or product architect).

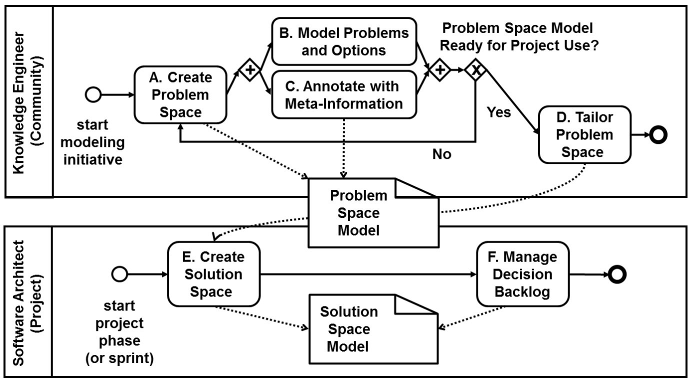

Knowledge Engineer (Community)
start modeling initiative
A. Create Problem Space
B. Model Problems and Options
C. Annotate with Meta-Information
Problem Space Model Ready for Project Use?
Yes
No
D. Tailor Problem Space
Problem Space Model
Software Architect (Project)
start project phase (or sprint)
E. Create Solution Space
Solution Space Model
F. Manage Decision Backlog

Fig. 2. Workflow for ADMentor users (notation: BPMN)

The remainder of the section follows the steps B to F from Figure 2, and specifies the Create, Read, Update, Delete (CRUD) operations that are performed on instances of the four structural elements from Figure 1. Feedback from solution space owners (on projects) to community-level knowledge engineers is out of scope of this paper; see [27] for a suggestion how to organize a continuous review-update loop.

## *B. Problem Space Modelling (Problems, Options)*

The second modeling step after problem space creation is problem space modeling. Problem and option model elements are created and positioned into knowledge packages here.

Rich text support is one of the requirements we identified in Section II – for instance, the knowledge aspects appearing in the practitioner templates from Section II can be represented as rich text sections separated by headings. As motivated in Sections I and II, we do not want to dictate any particular decision capturing template (for the description of problems, options, and their occurrences). AKM usage and maintenance should be lean; hence, ADMentor encourages knowledge engineers not to copy much text into problem spaces, but to leverage the Web. Hence, Web links e.g. to pattern texts can be added in this step, and other model elements can be linked in.

With these concepts and standard UML tool extensibility features, QOC diagram support[3] almost comes for free: questions (Q) and options (O) are modelled with ADMentor, and criteria (C) are assumed to be supported by its host tool (here: Sparx Enterprise Architect, see Section IV), just like components and connectors in UML class or component diagrams. We added QOC diagram support rather late in our iterative development work, which demonstrates that such extensions and configurations are indeed feasible. Only minor additional customization effort had to be invested in the host tool (i.e., the naming of link types via UML stereotypes).

## *C. Meta-Information Annotation (a.k.a. Typed Tagging)*

Table 2 specifies the output of knowledge engineering activity B, annotate problems and options with ADK *meta-information*. To compile the meta-information annotations in the table, we reviewed the software architecture and method engineering literature and reflected on own project experiences (in software architect and project management roles):

TABLE 2. META-INFORMATION ABOUT RECURRING DESIGN PROBLEMS

| **Name** | **Purpose, Rationale** | **Sample Value(s)** |
|---|---|---|
| *Intellectual Property Rights* | Intellectual Property Rights (IPR) for model element, e.g. confidentiality level, copyright statement | Public, Company-Confidential, © Company X, 2015 |
| *Knowledge Provenance* | Reference to a cited source and/or acknowledgment of contributor | CCP book, PoEAA website, Project Y, Architect Z |
| *Refinement Level* | The abstraction level on which this problem typically occurs | Conceptual Level, Technology Level |
| *Project Stage* | Gate, milestone, phase and/or elaboration point in incremental and iterative design (in which this problem is typically tackled) | Inception, Elaboration, Construction (in OpenUP [15]) |
| *Organizational Reach* | Sphere of influence of the problem | Enterprise, Division, Business Unit, Project, Subsystem |
| *Owner Role* | The role (as defined e.g. in OpenUP) that is responsible and accountable for the decision | Application Architect, Integration Architect |
| *Stakeholder Roles* | People with an interest in this problem (note: the accountable person is annotated as owner role) | Enterprise Architects, Frontend Developers, Testers |
| *Viewpoint(s)* | e.g. one of the 4+1 views on software architecture or a Rozanski/Woods viewpoint [16] | Logical Viewpoint, Deployment Viewpoint |

Our compilation of meta-information does not claim to be complete; knowledge engineers can add and remove meta-information as desired within their community contexts (e.g., to streamline their problem space modeling activities). Activities D and F leverage this meta-information, which we implemented with UML Tagged Values (see Section IV).

## *D. Tailoring (from Problem Space to Solution Space)*

In the tailoring step, a problem space is trimmed down to the problems and options that are relevant for a particular project (context-specific filtering). The meta-information from activity C can support this filtering work; for instance, a role- or phase-specific problem space can be obtained this way.

This activity is not described in detail in this paper. Support for it requires tool engineering rather than design science work.

## *E. Solution Space Creation*

The following two user stories characterize this step:

- *Full copy/complete instantiation* story: "As a solution architect starting a project, I would like to create a fully populated solution space containing one open problem occurrence for each problem and one option occurrence for each option that came out of the tailoring so that I receive guidance for my design work and I do not forget to solve any problems.".
- *On demand instantiation* story: "As a solution architect, I would also like to be able to start with an empty solution space and create problem occurrences and option occurrences individually as needed during the project so that my decision log has a minimal size".

This activity also is straightforward to implement in tools.

[3] Question-option-criteria (QOC) diagrams were proposed in the HCI community in the 1990s [12] and later picked up by the AKM community.

## F. *Decsion Backlog Management (Solution Space Usage)*

Semantically rich meta-information as defined in activity C can be leveraged to search, filter and order solution space models in activity F. When combined with context and state information, a powerful and user-friendly representation of the solution space results – a *decision backlog* [9]. Architects do not have to follow any predefined decision making order (most architects probably would not do so anyway); they simply pick the decision backlog entries they deem to be particularly urgent (and decidable) in/for the current design iteration. Modeling tools can support such decision backlog elegantly in views and widgets with table layouts that are configurable w.r.t. filtering and ordering. Table 3 sketches such decision backlog view:

TABLE 3. DECISION BACKLOG (SIMPLE EXAMPLE/EXCERPT)

| Problem Occurrence | Status | Viewpoint | Owner Role | Comple-xity | … |
|---|---|---|---|---|---|
| Session State Management Occurrence 1 | Decided | Functional | Web architect | High | … |
| Session Database Provider Occurrence 1 | Open | Information | Data Architect | Medium | … |
| … | … | … | … | … | … |

The problem state (on the occurrence level) is aggregated from the associated option occurrence's states in the following way: A problem occurrence is in state *open* if all options are *eligible*; it is in state *not applicable* if all options are *neglected*, and in state *decided* if all options are either *chosen* or neglected; it is *partially decided* in all other cases (additional option states are *tentative* and *challenged*). Just like product backlogs in agile practices, the decision backlog never has to be emptied completely, as design time always is constrained.

**Transition to technology level.** The concepts introduced in this section are not specific any host platform, but can be realized in any extensible modelling tool. The following Section IV transcends from platform-independent concepts to one particular platform-specific design and its implementation.

## IV. IMPLEMENTATION OF ADMENTOR (VALIDATION PHASE 1)

ADMentor is an add-in to Sparx Enterprise Architect (EA) Version 10 (and higher). It is implemented in C# and uses .NET Version 4.5. ADMentor comes with an UML Profile and a MDG Technology (two extensibility concepts in Sparx EA) that carry Architectural Knowledge Management (AKM) semantics implementing the concepts from Section III.

Our rationale for the selection of EA as the UML tool and general-purpose modeling platform to be extended is threefold. First and foremost, it is used by many architects and therefore a preferred choice of our industry partner. Secondly, it has adequate support for rich text (e.g. Web links, bullet lists, etc.), model refactoring and UML tagged values, all of which is needed to implement our concepts. The third justification is that an early proof-of-technology and the predecessor project [11] demonstrated technical feasibility and usability of APIs and extensibility mechanisms of EA (some implementation difficulties had to be overcome).

The key features of ADMentor Version 1.1 are:

- Problem space modeling: recurring design decisions, options to be considered – providing a checklist effect
- Model tailoring and solution space modeling: decisions made and their rationale, decision backlog
- Model refactoring, reporting via Enterprise Architect integration; modeling patterns (template configuration)
- Rich text editing, decision capturing with lightweight decision capturing templates such as Y-statements
- Question, Option, Criteria (QOC) diagram support

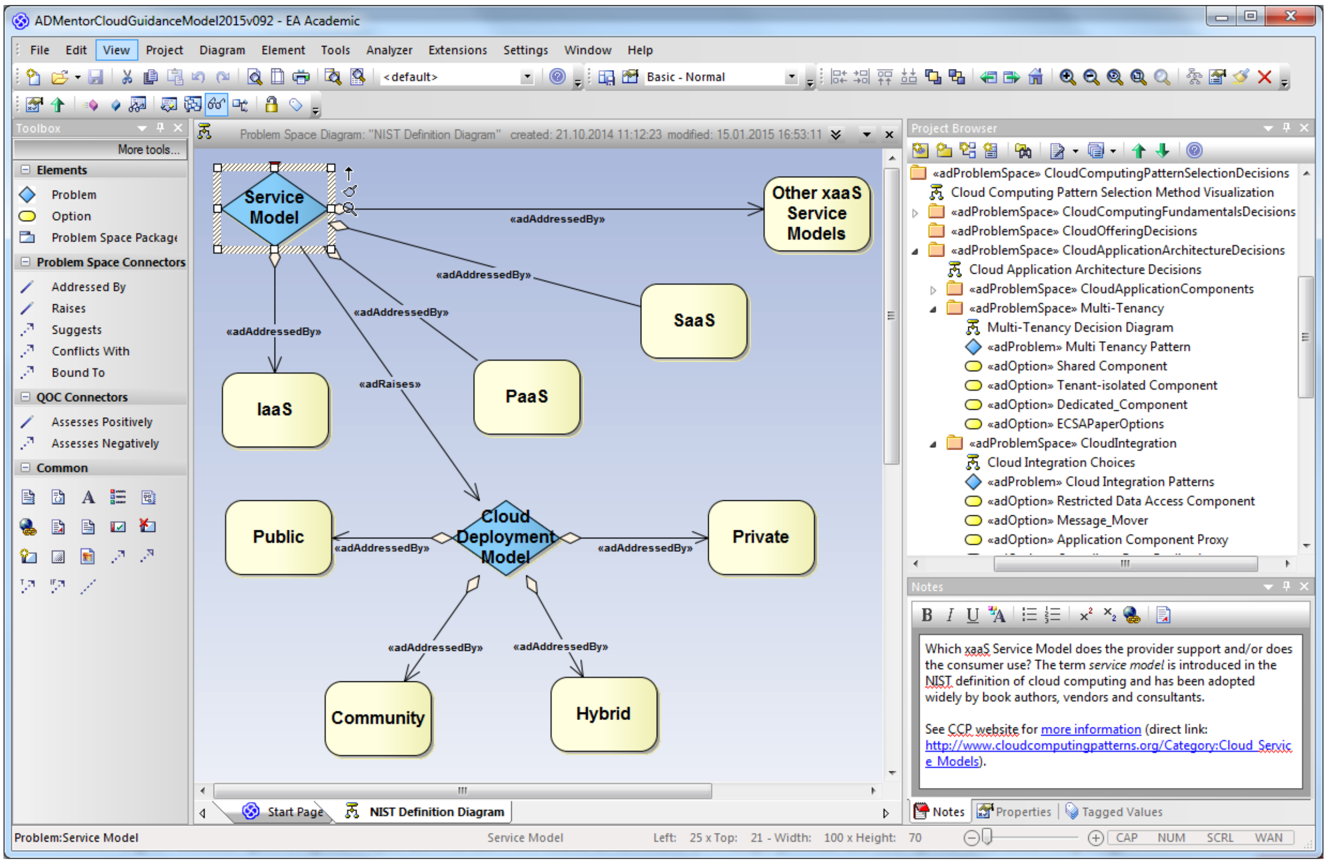
The above feature list shows that that our implementation stays very close to the concepts from Section III. The meta-information elements from Table 2 are realized as UML tagged values; the decision backlog is a customization of the package browser, a standard EA feature. Additionally, ADMentor also supports decision space analytics (e.g., number of options and problems per package and meta-information tag and breakdown of problem occurrences by state), model validation, and a RESTful HTTP interface for tool integration.

Problems/options and their respective occurrences are linked with the standard UML/EA concepts of classifiers and instances [18]. The model tailoring and filtering capabilities are based on tagged values (a standard UML element extension mechanism [18]). The typed links are realized as connector stereotypes that are defined in our UML profile. In addition to the concepts introduced in Section III, we also added two package stereotypes and additional links to our UML profile. The link definitions in ADMentor are compatible with other link modeling taxonomies; EA extensibility allows the user to add even more stereotypes, e.g. for relationships defined in the Kruchten/Lago/van Vliet ontology (see e.g. Chapter 3 in [1]).

Figure 3 on the next page shows a Problem Space Diagram (PSD) with fundamental cloud computing ADs modelled in ADMentor. Blue/dark diamonds represent recurring design problems, i.e., the need for a decision (which should not be confused with a quality attribute or stakeholder concern, e.g. in late design); options appear as yellow/light rounded ovals. The primary link between problems and options is an addressedBy relation. Two recurring problems are linked this way here; additional relationship links from Section III are shown in the EA toolbar (on the left side). The element notes view on the bottom right contains rich text including hyperlinks (to websites, but also other model elements, e.g., UML classes).

Figure 4, also on the next page, is an Architecture Overview Diagram (AOD) that illustrates the conceptual architecture of ADMentor in a functional and logical view. The figure shows the components of ADMentor (either through customization of EA when possible or, alternatively, through C# and .NET development), which can be traced back easily to the research contributions from the previous section and the tool requirements from Section II.

In the AOD, the components are placed in three logical layers (presentation, business logic, and data access/persistence layer) suggested by Fowler [4]. The left side of the AOD shows components supporting particularly relevant features in the standard EA product; the right side shows our extensions (add-in components). Call and dependencies links are not shown due to space constraints.



«adProblemSpace» CloudComputingPatternSelectionDecisions
Cloud Computing Pattern Selection Method Visualization
«adProblemSpace» CloudComputingFundamentalsDecisions
«adProblemSpace» CloudOfferingDecisions
«adProblemSpace» CloudApplicationArchitectureDecisions
Cloud Application Architecture Decisions
«adProblemSpace» CloudApplicationComponents
«adProblemSpace» Multi-Tenancy
Multi-Tenancy Decision Diagram
«adProblem» Multi Tenancy Pattern
«adOption» Shared Component
«adOption» Tenant-isolated Component
«adOption» Dedicated_Component
«adOption» ECSAPaperOptions
«adProblemSpace» CloudIntegration
Cloud Integration Choices
«adProblem» Cloud Integration Patterns
«adOption» Restricted Data Access Component
«adOption» Message_Mover
«adOption» Application Component Proxy

Which xaaS Service Model does the provider support and/or does the consumer use? The term *service model* is introduced in the NIST definition of cloud computing and has been adopted widely by book authors, vendors and consultants.

See CCP website for more information (direct link: http://www.cloudcomputingpatterns.org/Category:Cloud_Service_Models).

Fig. 3. Problem Space Diagram, Project Browser and Notes Editor in ADMentor (two sample problems and their options from cloud computing).

## V. Modelling Activities (Validation Phase 2)

Our main validation type in phase 2 was action research (i.e., use of concepts and tool on our own projects) [17]. One validation objective was to reconfirm that architecture design problem indeed recur (as already shown in a different technical domain in our previous work).

We also evaluated the expressivity and usability of our novel concepts (that address research questions 1 and 2 from Section I) and their implementation and measured the modelling and decision capturing efforts.

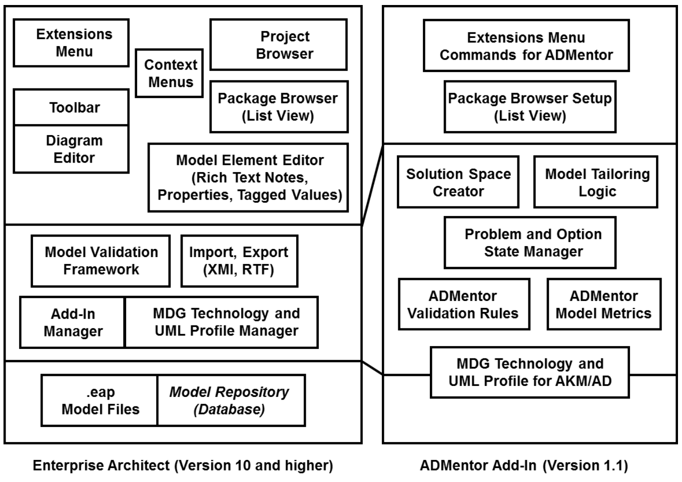

Extensions Menu
Context Menus
Project Browser
Toolbar
Package Browser (List View)
Diagram Editor
Model Element Editor (Rich Text Notes, Properties, Tagged Values)
Model Validation Framework
Import, Export (XMI, RTF)
Add-In Manager
MDG Technology and UML Profile Manager
.eap Model Files
Model Repository (Database)
Enterprise Architect (Version 10 and higher)
Extensions Menu Commands for ADMentor
Package Browser Setup (List View)
Solution Space Creator
Model Tailoring Logic
Problem and Option State Manager
ADMentor Validation Rules
ADMentor Model Metrics
MDG Technology and UML Profile for AKM/AD
ADMentor Add-In (Version 1.1)

Fig. 4. ADMentor architecture (in a high-level functional/logical viewpoint)

### A. A Problem Space for Cloud Computing

In parallel to concept creation and tool development, we created a problem space model for cloud computing, a technical domain that is of interest to many architects at present. This model has the primary goal to demonstrate that our modeling concepts work in practice and are beneficial to architects. Some examples were already given (session state management, session database provider, cloud service model, and cloud deployment model as defined by NIST and [3]). Other cloud design topics covered by the problem space are:

- Use of Cloud Computing Patterns (CCP) such as hypervisor, map-reduce, and key-value storage [3].
- Patterns for multi-tenancy, workload, message delivery.
- Cloud service management (e.g., watchdog pattern).

The patterns in the CCP book turned out to be well suited for a PSD representation in ADMentor: some patterns share the same problem statement and/or describe alternative solutions in a design category, e.g. multi tenancy. The CCP website can be linked to, which reduces the modeling effort significantly. No content was copy-pasted, but URIs added; the package structure in ADMentor mirrors that of book and website, e.g., the category "cloud application components" is found under "cloud application architecture". We blended in selected cloud knowledge from other sources, including books, but also blogs and own projects, and yielded 45 problems. All problems and options were modelled rather tersely to minimize creation and consumption effort (see Figure 4 in Section IV), but still be informative in the sense of a checklist and pattern language compass. The model was validated by creating a sample solution space (see below). In addition, one of the pattern book authors received a demonstration to confirm that the pattern knowledge in the book is represented properly in ADMentor.

In a second step, an existing decision log from a cloud project at our industry partner was transferred from Decision Architect (see Section II) to ADMentor to show feasibility and compatibility (of metamodels and tools) and to compare expressivity of concepts and their implementation. 14 problems were migrated (and re-modelled). We observed the same, or even better, expressivity of model elements and links; rich text notes in EA improve the user experience over plain text fields (as used in Decision Architect). Being inspired by the states suggested in the existing work [1], the state machine in Decision Architect initially was more powerful; when we realized this, we also integrated these concepts into ADMentor and aligned them with the problem and option states propagation from Section III. The linkage was achieved via the Relationship Viewpoint in Decision Architect; decisions can link to problem and option occurrences. No specific semantically rich link type was defined for that, as the generic *trace* links in UML/EA were deemed to be sufficient.

In a third step, 26 problems about enterprise application architecture and enterprise integration (messaging) were also modelled as recurring problems; while these decisions are not specific to cloud computing, they continue to be relevant and refine the ones in the CCP book. The session state management problem and its pattern options from [4] fall in this category.

In total, Version 1.0 of the reusable problem space for cloud computing compiles 85 problems and 226 options.

## B. Instantiation of Cloud Problem Space

This validation activity targeted the goals of validating the problem space instantiation of the ADMentor tool as well as the completeness and appropriateness of the cloud problem space that was created in validation activity A. This activity was performed by one of the authors of the paper, an industrial researcher who shadowed an architect of an ABB business unit. The approach to this validation was to import the problem space into the already existing UML model of a prototypical ABB cloud application that was built based on the conceptual architecture also used in step two of validation activity A. First, the problem space was tailored to fit the specific project, e.g., by omitting the Enterprise Application Decisions package (see step three in validation activity A), which was rated as not relevant for this project. The ADMentor user then instantiated a corresponding solution space and went through the problems step by step to document the decisions retrospectively. Of the total 48 problem occurrences in the joint cloud problem space 25 were completely solved, 4 partially solved, 8 deemed not applicable and 11 left open. Additionally, 17 key problem occurrences were linked to the corresponding model elements in the structural architecture model.

The results of the validation activity were as follows: the ADMentor user was able to complete the instantiation within a relatively short time (around 2.5 hours). The tool provided enough guidance to efficiently fulfill the task. In addition to the retrospective documentation of the actually made 14 problem occurrences, which had already partially been documented using a different tool, 23 further decisions were uncovered. Previously, these decisions were only implicitly documented or not made at all. With the help of the cloud guidance model it was possible to fill these gaps and capture this knowledge which might otherwise have been lost or only recoverable with significant additional effort.

## C. Workflow Decision Modeling (with Industry Participants)

Our third validation activity was the participation in a two-day community meeting of 26 software architects from various German companies (including banks, insurance firms, telecommunications service providers, and professional IT consulting services). The action researcher prepared an initial problem space for the design of workflows, via reflection of own experience and an ad hoc literature review. He presented the ADMentor vision as well as sample content from the initial problem space to the meeting participants (at the start of the meeting); e.g., the shown content included decisions about transaction boundaries and service granularity. The action researcher then facilitated a short exercise in which participants were asked to identify recurring decisions themselves; selected decisions were then discussed in the plenum. The attendee feedback included general agreement as well as four additional problems with options (e.g. on placement of business data in workflow, process instance migration, and interface signature sourcing). This activity reconfirmed the general hypothesis that architectural decisions recur; as the modeling work was continued during subsequent workshop presentations, it also showed that the ADMentor tool can be used in meeting and design workshop situations (for decision modeling on the fly). The final problem space model for workflow design comprises 75 recurring problems with 150 options.

## D. Third-party Problem Space Modeling

This validation activity concerned the retrospective modeling of a problem space and several solution spaces for industrial control systems. The goals were a) to validate ADMentor's problem space modeling capabilities concerning expressiveness and efficiency , b) to verify the traceability features to other model elements, and c) to gain experience on working with multiple solution spaces for a given problem space. In this case, an industrial researcher with no prior experience with ADMentor created the problem space from company-internal input, thus this validation activity concerns an external application of ADMentor as problem space modeling tool other than the tool authors themselves. However, only qualitative statements about the application can be made.

In this validation activity the approach was to exploit an existing technical report surveying the design concepts of different control systems and to model the included problem space. The problem space consisted of 16 problems and 36 options grouped into three different packages. Explanations for the options were copied over from the document into the model in some cases; in other cases, direct references to the technical report were created because reading lots of text inside Enterprise Architect proved to be cumbersome. From the problem space, three solutions spaces for three different control systems were derived. Existing UML models for these systems were then copied into the same Enterprise Architect project so that trace links between UML elements and ADMentor elements could be created. For example, traces were modelled from decisions to software components that had been implemented as a consequence of a particular decision.

This validation activity provided the following results: we noticed reported the ability to quickly (i.e., less than two hours) to create the problem space with ADMentor based on the available reference material. The expressiveness of ADMentor was sufficient to capture the required information. In addition the problem and solution space models provided a condensed overview of the made decisions and neglected options which was deemed useful. The creation of trace links between the modeling elements worked successfully. This ability was well received as tracing both from a particular UML element to an ADMentor element and vice versa is supported. Thus, it is convenient to derive the decisions attached to a specific component in the model or to identify the components affected by a specific decision. One challenge is that the solution spaces are not updated when the problem space is extended, thus the user needs to manually update the solution spaces. The interplay between the different viewpoints of the Decision Architect and ADMentor elements worked well. Architects from an ABB business unit reviewed the resulting solution space model informally and provided positive feedback about accuracy and usefulness of the models.

Although ADMentor was not used for forward engineering in this validation activity, its application proved valuable for our industrial partner. The usability of the tooling was deemed satisfactory and the instantiation of the problem space in three different solution spaces was achieved quickly. The model can potentially be used in the future for forward engineering when new systems of the same class are designed.

**Validation summary.** The evaluation results demonstrate that our concepts and their implementation work in practice; users have to invest relatively little effort to be productive and experience benefits. Hence, our research contributions and their implementation answer research the questions from Section I.

## VI. Discussion

**Contributions and their novelty.** The lessons learned on previous AKM projects were taken into account during design and development of ADMentor. In response to RQ1 and RQ 2 from Section 1, a leaner approach is now promoted, which means less modelling and model maintenance effort (for knowledge engineers), and also less consumption effort (for project architects). A decision backlog is available; meta-information is more comprehensive and more flexible so that it can be extended. We define AKM quadruples (i.e., problem, option, problem occurrence, option occurrence) and a workflow for processing them. The quadruples make the architectural decision knowledge more consumable, reusable and manageable. They are created and consumed in the context of a general-purpose modeling tool. In this approach, problems and options are harvested and curated as reusable assets; problem and option occurrences are then created by project architects as needed. This separation of knowledge management into 2x2 dimensions (and/or perspectives) is a key differentiator between our approach and the related work. Our meta-information attribution and its implementation with UML tagged values differs both from the universal/fixed modeling approach in SOAD and the findings and results of the GRIFFIN project (see Chapters 6 and 8 in [1]): unlike, SOAD, the GRIFFIN core model only defined knowledge entities, but not their attributes. For ADMentor, we found a "meet-in-the-middle" compromise: decision templates can be flexibly configured via the rich text editor; additional tagged values (meta-information annotations) can also be defined.

ADMentor provides a superset of the functionality in previous research prototypes, but has a very different technical design and implementation (e.g., metamodel, tool architecture, and codification in .NET/C#). The most significant differences to previous implementations are: a) any decision capturing template can be used in the rich text notes, b) an option is not physically contained in a problem, c) multiple options can be chosen, d) there is rich but flexible meta-information tagging, and e) the list view of the package browser serves as a full-fledged decision backlog now. Our approach does not mandate UML, but can be implemented in any extensible modeling tool that meets the requirements from Section II. Our UML profile can be exported via a platform feature for reuse.

**Threats to validity of validation.** We aimed at capturing both the quality of the tool and the created guidance models as well as the perceived efficiency in working with the tool. However, threats to the validity of our validation activities still exist: The researchers performed the majority of the validation activities themselves (action research). Furthermore, the amount of case studies was of course not statistically relevant, so all our results reside on the qualitative level. Thus, the external validity of our validation results, at least regarding efficiency, might be reduced. Finally, a threat regarding the created problem spaces model exists: the cloud computing domain is still evolving, and other architects might find other sources of more suitable knowledge and guidance than the ones that we picked.

We countered these three threats by using our approach in different contexts and with different people. We did not rely on a single case study, but performed several quite different modeling and decision making activities to broaden the validation scope. Additionally, we collected feedback on our approach and the resulting models from external stakeholders.

**Impact on practice.** We foresee problem space models to become virtual mentors making formerly tacit knowledge explicit in an easy-to-consume way; this concept is well suited for globally distributed development organization (with some amount of coordination). As a result, better decisions can be made in less time; the decision makers are empowered, but also held accountable. Deviations from group-level standards around patterns, technologies, and products can be chosen; but such new, non-standard solutions to known problems are now documented along with justifications. To increase the reuse potential and the longevity of the architectural knowledge, problems and options are pointed out (as an incentive to the architect/decision maker to get engaged), but not blindly promoted. We do not to try to make the chosen solutions reusable (as this would lead to an unrealistic one-size-fits-all approach to architecting systems that ignores project context, requirements, and constraints). As a result, architects remain masters of their project's destiny with ADMentor, but their decisions are backed by the guidance and recommendations given by the community that contributed to the problem space models used. For instance, the cloud guidance model can be applied and extended on future research and development projects. We hope that it will also serve as a steward for future decision modeling work in the community.

## VII. Summary and Conclusions

To overcome inhibitors for decision capturing and sharing, we conceptualized and implemented novel approaches to problem space modeling, solution space modelling, and decision backlog management. Problem spaces look forward and anticipate decisions to be made along with their options; solution spaces offer a lightweight form of decision guidance and capturing of past decisions to project architects. We implemented our modeling concepts and applied them to craft problem and solution space exemplars for cloud computing, workflow management, and distributed control systems.

Problem and solution spaces present decision knowledge to architects in a form that can be seen as checklist with work items. This approach to knowledge reuse makes weaker assumptions about target environment and target audience than previous knowledge sharing attempts – and therefore promises to produce more timeless knowledge and guidance that is easier to accept. As a result, poor decisions can be avoided and no important decisions are missed; faster time-to-market and less technical risk on development projects can be achieved.

The extensible UML modeling tool Sparx Enterprise Architect is the carrier platform of ADMentor; however, our concepts are designed and presented in such a way that they also work in other infrastructures (e.g. project wikis, decision capturing spreadsheets and other semi-structured decision logs).[4] We consider integrating ADMentor with project management and team collaboration tools. ADRepo, a Web repository with a publicly accessible RESTful HTTP interface, will simplify such integration. Potential future work also includes an extension of ADMentor to represent architectural refactorings as architectural decisions to be revisited.

## Acknowledgment

The research work presented in this paper was supported by an ABB Research Grant 2014. We would like to thank Tobias Blaser, Zoya Durdik, Christoph Fehling, Laurin Murer, Martin Naedele, Roland Weiss and the participants of the 8th workshop on software architectures at Softwareforen Leipzig for their input, support and/or constructive criticism.

## References

[1] M. Ali Babar, T. Dingsøyr, P. Lago, H. van Vliet (eds.), Software Architecture Knowledge Management: Theory and Practice, Springer-Verlag, 2009.

[2] M. Anvaari, O. Zimmermann, Towards Reusing Architectural Knowledge as Design Guides, Proc. of SEKE 2014, KSI, 2014.

[4] We plan to make the ADMentor tool and a cloud problem space that is not company-specific available publicly in the future. The current project website is: http://www.ifs.hsr.ch/ADMentor-Tool.13201.0.html?&L=4

[3] C. Fehling, F. Leymann., R. Retter, W. Schupeck, P. Arbitter, P., Cloud Computing Patterns, Springer 2013.

[4] M. Fowler, Patterns of Enterprise Application Architecture. Addison Wesley, 2003.

[5] M. Fowler, Richardson Maturity Model, http://martinfowler.com/articles/richardsonMaturityModel.html

[6] A. Gaind., Key Architecture Decisions Template, October 2005, Available via http://www.bredemeyer.com

[7] P. Hruschka, G. Starke, arc42, Resources for software architects, http://arc42.org/

[8] ISO/IEC/IEEE, Systems and software engineering – Architecture description, ISO/IEC/IEEE 42010:2011(E), Dec. 1 2011, http://www.iso-architecture.org/ieee-1471/templates/

[9] P. Kruchten, What color is your backlog?, via InfoQ blog post, 2010.

[10] M. Lindvall, I. Rus, R. Jammalamadaka, and R. Thakker. Software Tools for Knowledge Management: A DACS State-of-the-Art Report. Technical report, Fraunhofer Center for Experimental Software Engineering Maryland and The University of Maryland, 2001.

[11] C. Manteuffel, D. Tofan, H. Koziolek, T. Goldschmidt, P. Avgeriou: Industrial Implementation of a Documentation Framework for Architectural Decisions. Proc. Of IEEE/IFIP WICSA 2014, IEEE Computer Society, Los Alamitos (2014).

[12] A. MacLean, R. Young, V. Bellotti, and T. Moran, Questions, Options, and Criteria: Elements of Design Space Analysis, Human-Computer Interaction, 6 (3&4), 1991.

[13] C. Miksovic, O. Zimmermann, Architecturally Significant Requirements, Reference Architecture and Metamodel for Knowledge Management in Information Technology Services. Proc. of IEEE/IFIP WICSA 2011 (2011), IEEE Computer Society, Los Alamitos (2011).

[14] M. Nygard, Documenting architecture decisions, http://thinkrelevance.com/blog/2011/11/15/documenting-architecture-decisions

[15] Open Unified Process (OpenUP), http://epf.eclipse.org/wikis/openup/

[16] N. Rozanski, E. Woods, Software Systems Architecture : Working With Stakeholders Using Viewpoints and Perspectives, Addison Wesley, 2005.

[17] M. Shaw, Writing Good Software Engineering Research Papers: Minitutorial. Proceedings of the 25th International Conference on Software Engineering. IEEE Computer Society, 2003.

[18] Sparx Enterprise Architect Version 11 Users Guide, http://www.sparxsystems.com/enterprise_architect_user_guide/11/

[19] The official site of the Stage-Gate®, http://www.stage-gate.com/

[20] A. Tang,, M. Ali Babar,, I. Gorton., and J. Han, 2005. A Survey of the Use and Documentation of Architecture Design Rationale. Proc. of the 5th Working IEEE/IFIP Conference on Software Architecture. IEEE Computer Society, 2005.

[21] A. Tang, Y. Jin, J. Han: A rationale-based architecture model for design traceability and reasoning. Journal of Systems and Software 80(6): 918-934 (2007).

[22] J. Tyree, A. Akerman: Architecture Decisions: Demystifying Architecture. IEEE Software 22(2): 19-27 (2005)

[23] R. Weinreich, I. Groher. A Fresh Look at Codification Approaches for SAKM: A Systematic Literature Review. Proc. of ECSA 2014, Springer.

[24] U. Zdun, R. Capilla, H. Tran, O. Zimmermann, Sustainable Architectural Design Decisions, IEEE Software, Volume 30, Number 6 (2013).

[25] O. Zimmermann, An Architectural Decision Modeling Framework for SOA and Cloud Design, SEI SATURN 2010 Tutorial, page 14. SEI Techncial Library, http://resources.sei.cmu.edu/library/

[26] O. Zimmermann, C. Miksovic, Decisions Required vs. Decisions Made: Connecting Enterprise Architects and Solution Architects via Guidance Models, in: I. Mistrík, A. Tang, R. Bahsoon, J. Stafford (eds.), Aligning Enterprise, System, and Software Architectures. IGI Global (2013).

[27] O. Zimmermann, U. Zdun, T. Gschwind, F. Leymann, Combining Pattern Languages and Architectural Decision Models into a Comprehensive and Comprehensible Design Method. Proc. of IEEE/IFIP WICSA 2008. IEEE Computer Society, Los Alamitos (2008).