AD_{kwik}: Web 2.0 Collaboration System for Architectural Decision Engineering

Nelly Schuster, Olaf Zimmermann, Cesare Pautasso
IBM Zurich Research Laboratory, Säumerstrasse 4, 8803 Rüschlikon, Switzerland
{nes, olz, cpa}@zurich.ibm.com

Abstract

Capturing and sharing software architecture design rationale has always been particularly challenging in complex application domains such as enterprise computing. Facing the ongoing acceleration of technology innovation and economic forces such as outsourcing and offshoring, conservative knowledge management practices and existing tools are no longer sufficient, offering only rudimentary support for knowledge exchange and collaboration on the Internet. In this paper, we present AD_{kwik}, a Web 2.0 collaboration system supporting the cooperative decision making work of software architects. We describe the realization of AD_{kwik} as a situational application wiki and discuss initial evaluation results. Thanks to its ease of use, AD_{kwik} has already shown concrete benefits to its users, including rapid team orientation, tangible decision making advice, and simplification of asset harvesting.

1. Introduction

Architectural decisions capture the rationale behind software architecture design. In current practice, this knowledge is tacit and rarely captured explicitly [1]. However, an explicit knowledge engineering approach to architectural decision capturing is beneficial, e.g., to attain *regulatory compliance*. Governance and maturity models such as Capability Maturity Model Integration (CMMI) desire architectural decisions to be captured and archived along with justifications.

A second motivation for explicit decision capturing is *team collaboration*. On large software development projects (e.g., in enterprise computing), architectural decision making is a team effort. Typically a lead architect has the overall technical responsibility, but delegates certain decisions to subsystem architects, chief developers, and platform specialists. Communication problems between these roles occur frequently; it is challenging to reach a shared view and a consensus over the architecture. This is even more difficult to achieve due to the current offshoring and outsourcing trends; more and more development teams are geographically distributed.

A third motivator for architectural decision capturing is *reuse*. A vast amount of architectural knowledge exists in practitioner networks such as company-wide *Community of Practice (CoP)* networks [2]. Often architectural knowledge is tacit or embedded in code. If documented at all, it resides in inappropriate, therefore rarely visited data stores such as personal mail archives and poorly structured team repositories.

In response to the regulatory compliance requirements, the need for collaboration during architectural decision making and the opportunities for reuse of architecture design rationale in CoPs, we have started to apply architectural decision trees as a fine-grained unit of knowledge exchange within and between project teams [3]. In this paper, we present how such a knowledge exchange can be facilitated by AD_{kwik}[1], a Web 2.0 collaboration system supporting the cooperative work of software architects. AD_{kwik} embeds a rich domain model into a situational application wiki, with the goal of making it easy for practitioners to share their knowledge about architectural decisions across project boundaries. The system is in use within a small community of software architects already. Their feedback has shown us the benefits of the approach, in terms of the acceleration of project initiation (team orientation), improvement of decision making quality, and simplification of project result sharing (asset harvesting). Users have also pointed out critical success factors: ease of use and dealing with extreme change dynamics.

The remainder of this paper is structured as follows: Section 2 introduces the context of this work and presents related work. Section 3 gathers requirements for AD_{kwik}. Section 4 discusses the knowledge engineering aspects of AD_{kwik}, while Section 5 focuses on its architecture and implementation. Section 6 presents our preliminary evaluation; Section 7 concludes the paper.

[1] AD_{kwik} stands for "Architectural Decision Knowledge Web Interchange Kit" and pronounces "AD-quick".

2. Background and Related Work

In the 1990s, *Design Decision Rationale (DDR)* research [4] proposed techniques such as QOC diagrams [5] structuring the decision making process for the general design process of software systems and human-computer interfaces. *Knowledge-Based Software Engineering* (KBSE) proposals such as Argo [6] stressed that tools for designers should support their cognitive needs such as "Reflection in Action, Opportunistic Design, Comprehension and Problem Solving". To achieve this, a "Managed To Do List" was seen as one of several key features. At that time, the regulatory compliance and team collaboration forces were not as dominating as today; therefore, aspects specific to these forces were not addressed. Furthermore, DDR and KBSE did not provide any support specific to architectural decision modeling.

Recently, the DDR ideas were revived and applied to the particular domain of *architectural design decisions* [7][8]. Each decision describes a concrete, atomic design issue for which several alternate solutions with pros and cons exist. Examples for such decisions include: selection of programming language and tools for any development project, of communication protocols in client-server environments, of architectural patterns [9] in certain application domains, and of highly available network topologies in enterprise computing. In general, we defined architectural decisions as "conscious design decisions concerning a software system as a whole, or one or more of its core components" [3]. Many inhibitors for capturing such decisions have been reported, including no appreciation from project sponsors and missing short-term benefits, as well as lack of time, budget, and tool support [10]. Several architectural decision capturing tools have been proposed [11][12]. For example, PAKME [13] is the prototype of an architecture knowledge management system implemented on top of an existing groupware platform. It uses 25 tables to capture various forms of architectural artifacts, including design rationale. PAKME is populated from patterns repositories and the literature.

As potential building blocks for our solution, we also evaluated existing assets such as the Eclipse-based Architects' Workbench (AWB) [4], UML tools, plain HTML and wiki technologies. None of these met all of our requirements. Due to its powerful refactoring capabilities, AWB is well suited for architectural decision content capturing; it can generate reports, thus addressing the regulatory compliance issues. However, it was not designed for knowledge exchange and team collaboration over the Internet. UML tools are strong in capturing analysis- and design-level structure models such as use cases, class, activity, and sequence diagrams; they fall short when it comes to modeling knowledge comprising of text, often semi-structured, combined with other formats, e.g., images and Web links, that motivate and justify the rationale behind the designs captured in UML models.

Plain HTML and standard wiki engines provide flexible human user interfaces when designed and configured appropriately. One advantage is that many development projects already use plain wikis for collaboration and information sharing. Still, using plain wikis does not fully meet our needs. First, no explicit domain model exists since the content is left unstructured and blended with presentation elements in HTML or wiki code. Furthermore, there is no API to access the content apart from the HTML data sent to the browser via HTTP. Thus, it is difficult to populate the system from third party software, or to extract any well-structured knowledge content for further automatic processing.

3. Requirements and Use Cases

We believe that a lack of *collaboration and systematic knowledge reuse features* are key deficiencies of existing approaches. Unlike passive knowledge bases, we aim to guide the user through the content in the spirit of Argo's Managed To Do List, providing *team orientation*. This To Do list should be organized according to domain-specific engagement types and patterns, e.g., business process integration in enterprise computing. To keep its value, the knowledge base must be updated continuously with new decisions, experiences, and rationale gathered both on successful and failed projects. We refer to this collaborative knowledge maintenance activity as *asset harvesting*.

In response to these shortcomings, we propose AD_{kwik}, a Web-centric collaboration system, providing explicit support for sharing and reusing knowledge elements from the domain of architectural decision capturing. We see the following use cases for AD_{kwik}:

- *Obtain* architectural knowledge, captured in decision models, from other projects and CoP.
- *Tailor* imported decision models according to project-specific needs, e.g., filtering content.
- *Involve* experts from other projects and CoP leaders when looking for advice.
- *Manage* dependencies between correlated decisions automatically to guide the user.
- *Share* gained architectural knowledge with other projects and CoP (after sanitization).

Discussion and interaction support, e.g., via email, comments and issue tracking, *document management*, and *versioning* are additional functional requirements shared with existing wiki-like collaboration systems. Integration with other tools is also an important factor to ease the adoption of AD_{kwik}. An *Application Pro-*

gramming Interface (API) should be provided so that import and export mechanisms to automatically populate the AD_{kwik} content repository can be built, e.g., from requirements management systems and UML design tools. The system must be *highly usable*, as practitioners do not appreciate having to work with yet another tool to fulfill extra obligations. It must be intuitive to browse the content, and users should be attracted to contribute new knowledge. *User management*, including simple *workflow and basic security support* (authentication, authorization) is required if decision making responsibilities are shared within the team. A *thin client* eases deployment and remote access.

4. Knowledge Engineering in AD_{kwik}

In this section we present AD_{kwik} from the user's point of view, both including an overview of its domain model and also briefly describing the most important features of the user interface. For the organization of the knowledge content in AD_{kwik}, the main elements of the domain model are: *Architectural Decision (AD)*, *ADAlternative*, *ADOutcome*, and *ADTopic*. AD is the core entity describing the context of a decision including decision drivers [3] and relationships with other decisions [8]. ADAlternative instances present solution design options for ADs with their pros and cons. ADOutcome elements record the selection of ADAlternatives and the justification for decisions. ADTopic is a simple hierarchical grouping construct. We also define three *ADLevels* of abstraction in our domain model: *conceptual*, *technology*, and *assets*.

To give an example: on the conceptual level, the choice of programming language and runtime platforms such as application servers and databases are among the key *executive decisions* according to the ontology defined in [8]. A screenshot with this exemplary decision is shown in Figure 1. Several of the domain model elements, e.g., an AD (here: "Platform And Language Preferences") are visible at first glance (1). Applying the *master-details pattern*, ADs can be browsed using the hierarchical ADLevel/ADTopic Explorer (2). Clicking on entries then displays the details about the AD and its ADAlternatives in the main window (1,3). The *breadcrumb pattern* provides additional means of orientation, flattening the ADTopic hierarchy into a link list (4). The knowledge is organized and displayed in a hierarchical structure (2), but also tagged to enable searches (5).

The same user interface can be used for decision *identification*, decision *making*, and decision *enforcement* in a development team: Rather than identifying decisions from scratch, an initial set can be imported, e.g., from AWB, realizing the Obtain use case (6); export features also exist, supporting the Share use case (6). Decision drivers (forces) provide basic decision making support (1); more detailed documentation including scoring spreadsheets and DDR QOC diagrams can be attached to the page. ADs carry owner and status information to further facilitate team collaboration (7). In support of the Involve use case, there are literature links (8). The domain model is shared between projects so that knowledge can be exchanged, e.g., via generated e-mails.

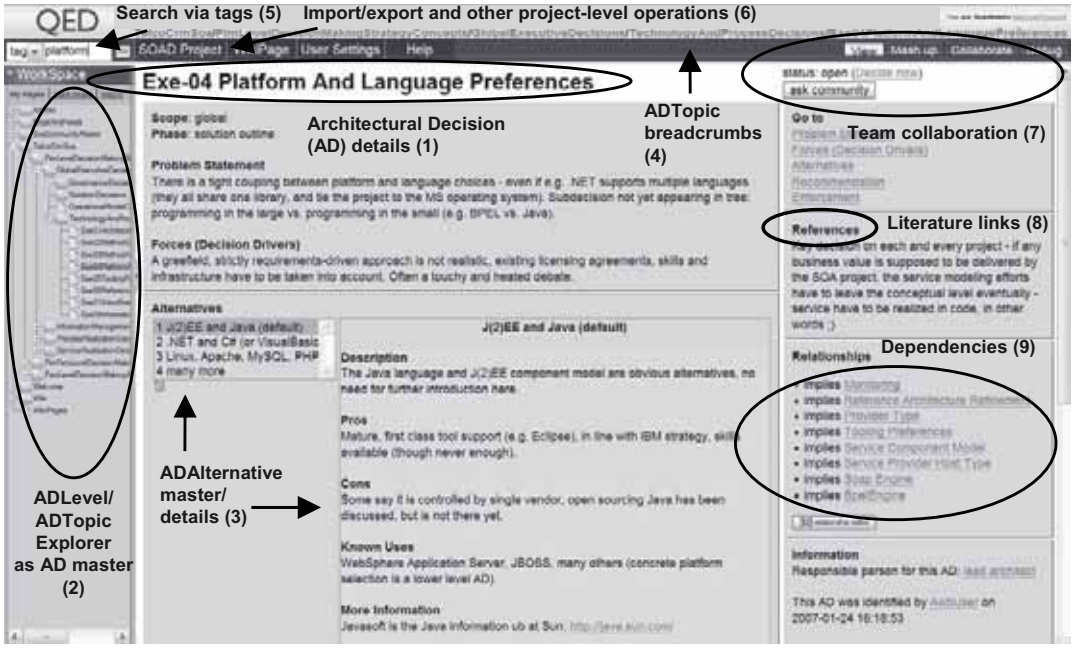

Fig. 1. User interface of AD_{kwik}: ADTopic Explorer as master, detail views organized according to domain model

ADs can be related to each other. Dependencies between them are shown (9), e.g., "Tooling Preferences". For example it is no longer required to select between C# and Visual Basic as a programming language if it has already been decided that .NET will not be used as a platform. On the technology and asset level, many more such constraining relationships exist, often buried in vendor information and best practices documents. Explicit, fine-grained representation of decision dependency relationships helps uncovering implicit assumptions, contradictions, and implementation limitations so that a more objective technical discussion becomes possible (Manage use case). Active dependency management leads to a more dynamic and therefore up-to-date knowledge base than static content repositories can provide, which is very important when dealing with the complexity and change dynamics of current enterprise computing environments. It also helps attaining regulatory compliance.

5. Software Architecture of AD_{kwik}

Conceptual Design. AD_{kwik} combines the benefits of a *rich Web 2.0 front end* [14] with those of the *domain model pattern* from [9]. The use cases from Section 3 and the user interface design from Section 4 lead to a logical decomposition as shown in Figure 2.

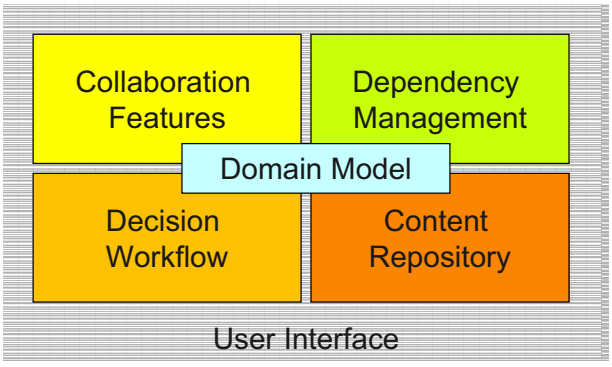

Fig. 2. High-level building blocks of AD_{kwik}

There are four functional building blocks: *Collaboration Features, Decision Workflow*, *Content Repository*, and *Dependency Management*. A *Domain Model* is orthogonal to these four building blocks. The Collaboration Features and Decision Workflow building blocks realize the Obtain, Involve, and Share use cases. The Content Repository provides Create, Read, Update, Delete, and Search (CRUDS) operations for the Domain Model elements.

Dependency Management structures the knowledge into a graph. As opposed to a simple decision catalog, the graph improves the user's navigation across related decisions and provides the basis for advanced features such as context-specific, dynamic decision tree *morphing* and what-if *simulations*.

An example of an abstract conceptual decision in the Web services integration domain is the message exchange pattern (request-response vs. one-way) [3], which can be refined into a technology decision dealing with Web Services Description Language (WSDL) contract design (in and out message vs. in message only), which in turn lead to two asset decisions (which SOAP engine and WSDL tool to use when realizing the abstract pattern as a WSDL-described service that can be invoked via SOAP). These decision dependencies modify the asset-level To Do List for the user depending on the outcome of the conceptual and the technology decisions.

To refine this functional view into a logical component model we use logical layering [9] as our governing architectural pattern. The three layers of AD_{kwik} are: *Presentation*, *Domain*, and *Persistence Layer* (Figure 3). The Presentation Layer supports all functional building blocks, e.g., Collaboration Features. Blogs and feeds from vendor forums such as IBM developerWorks [15] and industry thought leaders [16] can be integrated here without development effort. Dependency Management and Decision Workflow are key Domain Layer responsibilities. The Persistence Layer implements the Content Repository as a Relational Database Management System (RDBMS); hence, the full power of the RDBMS technology can be leveraged, e.g., for reporting purposes during technical audits (in response to the regulatory requirements). The Domain Model affects all layers: each model element is represented by one user interface component, related domain layer logic, and a corresponding database entity.

Implementation. The design of AD_{kwik} resembles traditional enterprise application architectures. However, using wiki technology as the presentation layer of such an enterprise application is a new approach requiring an *application wiki* rather than a plain wiki engine. Application wikis extend the user and page management capabilities of plain wikis with application server features and a mash-up API. This allows us to create and manage content programmatically.

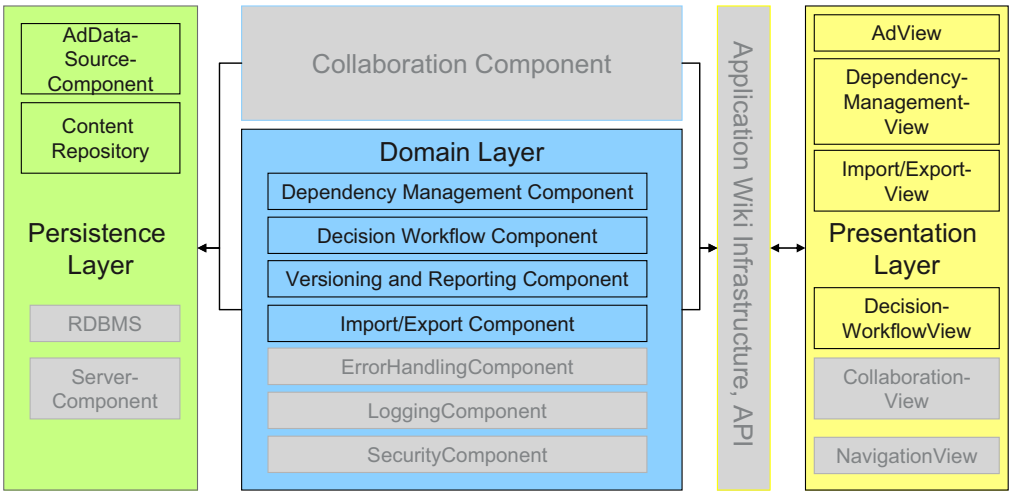

Fig. 3. Logical view on architecture of AD_{kwik}, our Web 2.0 collaboration platform for architectural knowledge exchange

More specifically, AD_{kwik} is implemented in a situational application and Web 2.0 mashup environment called *QEDWiki* [14]. QEDWiki can be characterized as a hybrid wiki engine and PHP application server, providing access to incoming HTTP request data via a command interface. QEDWiki extends the Zend PHP Framework and uses the XAMPP distribution from apachefriends.org. It includes the Apache HTTP server and the MySQL RDBMS. HTTP server and QEDWiki provide the required user management.

Through predefined commands, QEDWiki provides out-of-the-box support for adding comments, attachments, and email threads. We customized and extended these commands to provide native support for our domain model, also using the Dojo JavaScript library in order to provide a user experience as attractive as that of rich clients. The domain layer – comprising the model elements outlined in Section 3 – is implemented in PHP. It accesses the persistence layer via the *active record pattern* [9], requiring little coding effort. The integration with other tools is realized via file import, and a RESTful interface that can be accessed remotely via HTTP.

6. Evaluation

AD_{kwik} has knowledge acquisition and presentation capabilities similar to, for example, PAKME [13]. AD_{kwik} also provides support for dependency management, decision workflow, and decision maker guidance. Its initial content comes from large-scale industry projects conducted since 2001. Since then we have updated the repository continuously with input from additional projects and refactored it many times according to the needs of practitioners [3]. At present, AD_{kwik} contains 130 decision nodes capturing reusable knowledge about enterprise application architectures and Web services integration.

We started to make the system available to selected colleagues and clients in December 2006. To obtain usability feedback, we conducted several workshops. AD_{kwik} already is in use within one industry project. From these engagements, the initial user feedback regarding the value and usability of AD_{kwik} is encouraging: users appreciate that all knowledge required during architectural decision making can be conveniently located in a single place, and that the system comes with a rich set of initial content. Despite the large size of the decision space, early users reported to be productive without major training efforts. AD_{kwik} leverages Web 2.0 application wiki technologies; first users perceive the HTML-based user interface to be compelling and well designed. Thanks to the modeling and collaboration features, we can already report improvements in the quality of the decision making experienced by several AD_{kwik} users. For example, one architect consulting to an IBM client in a SOA coach role reported that he could locate and reuse detailed advice regarding 13 of 15 required decisions related to the usage of Web services [3].

We also have received constructive criticism regarding the challenges of modeling a large and complex decision space facing a high degree of change. Numerous new ADs and even more new ADAlternatives become available almost on a daily basis. If we aimed for completeness, just for enterprise applications organized according to SOA principles, we estimated that AD_{kwik} would contain thousands of decision nodes with numerous dependencies and alternatives. While this complexity is inherent to the problem domain, we run the risk of being criticized for exposing it. However, experienced practitioners report that they prefer to be made aware of this complexity and to have a system that manages it collaboratively, rather than to let the knowledge remain tacit and unma-

naged. As we further improve AD_{kwik} based on the feedback of this initial evaluation, we will continue the usability studies on a larger scale.

7. Conclusion

In this paper, we described the conceptual design and implementation of AD_{kwik}, a Web-centric collaboration platform for architecture knowledge capturing and exchange. AD_{kwik} supports five use cases, Obtain-Tailor-Involve-Manage-Share. Its design employs both domain modeling concepts and a layered architecture. AD_{kwik} features an API to populate the initial decision model from existing requirements models, reference architectures, and other community assets. The presentation layer is a Web 2.0 application wiki facilitating community collaboration. The persistence layer is implemented as a RDBMS so that database reports can be generated to meet regulatory requirements.

Using Web 2.0 technology for team collaboration and project internal documentation purposes is state-of-the-practice; layered software architectures and domain modeling are known concepts as well. We combine these technologies in a novel way, and apply them to the domain of architectural decision knowledge exchange. Key features of the AD_{kwik} knowledge management approach include pre-population of content, a rich domain model with decision relationship management, support for collaborative decision making, and project result sharing over the Internet. AD_{kwik} has been tested by a small number of early adopters. Extended user tests are planned already.

A critical success factor for AD_{kwik} is to create incentives for users to contribute, not only consume, content, which according to our experience has been a challenge for many industrial knowledge management approaches in the past. Through close contacts with practicing architects, e.g., via a CoP, we have continuous access to up-to-date project results and lessons learned which have to be quality assured and generalized before they can be added to the knowledge base. Architectural decision engineering is a broad, complex, and continuously changing domain. Therefore, keeping the organization of the hierarchical classification of the decision space consistent and manageable is another important factor for future success.

Future research work will investigate additional use cases. For instance, we plan to study design space pruning and recommendation making algorithms. When team collaboration support becomes available on top of the Eclipse platform, additional integration opportunities will arise. Improving usability even further is another focus area. Finally, we are interested in the interdisciplinary aspects of architectural decision making. For instance, will investigate the relationship of architectural decision knowledge with project management concerns such as effort estimations, status reporting, and work breakdown structure creation.

References

[1] Tyree, J., Akerman, A., Architecture Decisions: Demystifying Architecture, IEEE Software, 22 (2005)

[2] Gongla P., Rizzuto C.R., Evolving Communities of Practice: IBM Global Services Experience, IBM Systems Journal Vol. 40, 4/2001

[3] Zimmermann O., Koehler J., Leymann F., The Role of Architectural Decisions in Model-Driven Service-Oriented Architecture Construction, Workshop on Best Practices and Methodologies in SOA, OOPSLA 2006

[4] Lee J., Lai, K, What's in Design Rationale?, Human-Computer Interaction, 6(3&4), 1991

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

[6] Robbins J. E. , Hilbert D. M. , and Redmiles D. F.: Extending Design Environments to Software Architecture Design, KBSE 1996

[7] Farenhorst R., de Boer R., Deckers R., Lago P., van Vliet H., What's in a Domain Model for Sharing Architectural Knowledge?, SEKE 2006

[8] Kruchten P., Lago P., van Vliet H, Building Up and Reasoning About Architectural Knowledge, QOSA 2006

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

[10] Tang W., Ali Babar M., Gorton I., Han J., A Survey of the Use and Documentation of Architecture Design Rationale, WICSA 2005

[11] Abrams S. et al, Architectural thinking and modeling with the Architects' Workbench, IBM Systems Journal Vol. 45, 3/2006

[12] Jansen A., Bosch, J., Software Architecture as a Set of Architectural Design Decisions, WICSA 2005

[13] Ali Babar M., Gorton I., Jeffery R., Capturing and Using Software Architecture Knowledge for Architecture-Based Software Development, QSIC 2005

[14] IBM QEDWiki, http://services.alphaworks.ibm.com/qedwiki

[15] IBM developerWorks, http://www.ibm.com/developerworks

[16] Booch G., Handbook of Software Architecture, http://www.booch.com/architecture