Towards Effective Management of Software Knowledge Exploiting the Semantic Wiki Paradigm

Walid Maalej[1], Dimitris Panagiotou[2], Hans-Jörg Happel[3]

[1]Technische Universität München,
Boltzmannstraße 3, D-85748 Garching, Germany
maalejw@in.tum.de

[2]National Technical University of Athens,
9 Iroon Politechniou, 15780 Athens, Greece
dpana@mail.ntua.gr

[3]FZI Forschungszentrum Informatik,
Haid-und-Neu-Str. 10-14, D-76131 Karlsruhe, Germany
happel@fzi.de

Abstract: The increasing number of distributed software projects together with the success of agile development methodologies raise new challenges for collaboration and knowledge sharing. While traditional centralized knowledge management solutions fail to address these challenges, Semantic Wikis bring new potentials, providing lightweight, incremental and machine-readable knowledge articulation and sharing facilities. In this paper we survey the state-of-the-art of Semantic Wikis. We then discuss how they can cope with difficulties of conventional Wikis to efficiently manage knowledge created in software engineering projects.

1 Introduction

The increasing distribution of software projects together with the success of agile development methodologies raise new challenges for collaboration and knowledge sharing for software teams [PL06, Ram06]. On the one hand, distributed development hinders knowledge sharing due to reduced communication bandwidth. Context information is often not shared, e.g., problems encountered by a team member before making a specific design decision. Research has shown that distributed teams are less efficient than collocated ones [HM03]. On the other hand, agile methodologies embrace tight collaboration and informal communication [Be01], manifesting in practices such as frequent Scrum meetings [SB01] or pair programming [WK02]. Such practices dismiss "unnecessary", formal overhead as extensive documentation. Traditional knowledge management solutions like experience repositories [Ba92] or Rationale Management [DM06] fail to address these challenges. They require large upfront configuration and investment, as well as stable, long-term environments. In dynamic software projects, however, developers avoid spending extra effort for following formal and extensive knowledge management policies. A lightweight and continuously evolving knowledge articulation and sharing facilities are required. Semantic Wikis bring new potentials to address these issues.

Conventional Wikis have already been deployed successfully in software projects as interlinked web sites that can be collaboratively edited by anyone. The content is written in a simple syntax and can be easily edited even by novice users. The syntax consists of simple tags for creating links to other Wiki pages as well as textual markups such as lists and headings. Though the main disadvantage is that information accumulated in Wikis is unstructured. This makes the navigation and the retrieval of the right information difficult. Semantic Wikis should overcome these problems by combining Semantic Web and Wiki technologies. In this paper we present the state-of-the-art of Semantic Wikis, focussing on how they can be used in software engineering to overcome new knowledge management challenges for agile and/or distributed teams. First, we introduce Semantic Wikis (Section 2) and compare popular implementations (Section 3). Thereafter, we discuss advantages of Semantic Wikis over conventional ones to facilitate software engineering activities, proposing example of usage scenarios (Section 4). Finally, we conclude by discussing remaining problems in the State-of-the-Art (Section 5), giving an overview of our findings and proposing future research directions (Section 6).

2 The Ingredients

This section introduces Wikis, Ontologies and Semantic Web that make main ingredients of the Semantic Wiki. The latter is then described by its major features and advantages.

2.1 Wikis

The first Wiki, WikiWikiWeb, was introduced by Cunningham [LC01]. Wikis were increasingly adopted in enterprises as collaborative software. Their common uses include project communication, documentation and management of intranet sites. Wikis are interlinked web sites that can be collaboratively edited by anyone [Or05]. Pages are written in a simple syntax and can be edited even by novice users. The user interface of most Wikis consists of two modes: reading and editing modes. In reading mode users can navigate conventional web pages that can contain images, links, textual markup, etc. In editing mode, an editing box displays the page in the Wiki syntax.

There is a fast evolution in Wiki engines over the last few years. Most of them are open source and freeware, thus enabling every single user to setup a Wiki with zero cost. Many organizations and user communities capitalize on Wikis as a mean of collaboration. A typical example is a Wiki of an open source project, where users collaborate to add documentation about the project. In this manner, the “tedious” task of editing the content is shared among the members of the whole community, while still allowing everyone to quickly find relevant documentation. A popular Wiki like Wikipedia[1] can

[1] http://www.wikipedia.org/

grow at very fast rates, since any interested visitor can edit and create pages at will.

Since the highest amount of information in a Wiki is an unstructured and accumulated text, the only way to find it is through a keyword-search. Thus information retrieval easily gets inefficient in a large Wiki. Structural queries such as "Wiki pages describing lessons learned from projects conducted in the last year" are not supported. In addition the poor semantic support in Wiki might lead to problems in knowledge management and productivity in general. Consider for example a newcomer who is trying to find information in the Wiki about check-in policies, but who is not familiar with the synonym term "commit" used by the team. The only semantic of Wiki pages lies in the links between them. Most Wiki engines generate navigational structures from these links. Users can query pages that link to the current one and navigate to them. Nevertheless, navigation through related pages often fails to address advanced information retrieval needs, since only a single navigation path can be followed. It is therefore not possible for users to combine several retrieval criteria, e.g. browsing components developed together by Alice and Bob, if pages describing components are organized by single developers.

2.2 Ontologies

In the last decades ontologies won on popularity in several fields of computer science, mainly artificial intelligence, multi-agent systems, and Web technologies. The term ontology is borrowed from philosophy, where it stands for a systematic account of existence. In computer science ontology is formal, explicit specification of shared conceptualization [Gr93]. In other words, ontologies are models that explicitly define in machine-readable way concepts, their types and constraints on their use in a domain of interest. This domain might be a part of reality or an entirely fictitious environment. The universe of objects and the relationships that hold in the ontology are expressed in a declarative, formal vocabulary that collectively constitutes the knowledge about the domain [GN87]. The main difference between ontologies and conceptual models, such as entity-relationship-models or UML models, is the scope. Models are usually used in one particular project, not referring to areas beyond the project scope. Differently, ontologies supply a much bigger clientele, or even computer programs, which do not have to belong to the same project or same organization. As a result, ontologies represent universally valid truth, i.e. knowledge, about a restricted domain. It encompasses future projects and developments including potential, possibly still unknown users [Pr04].

2.3 Semantic Web

According to W3C [W3C07] the Semantic Web is about two things: "It is about common formats for integration and combination of data drawn from diverse sources, while the original Web mainly concentrated on the interchange of documents. It is also about language for recording how the data relates to real world objects. That allows a person, or a machine, to start off in one database, and then move through an unending set of databases which are connected not by wires but by being about the same thing." The Semantic Web effort [DF04] provides standards and technologies for the definition and exchange of metadata and ontologies. Available standard proposals provide ways to define the syntax (RDF[2]) and semantics of metadata based on ontologies (OWL[3]). There is an ongoing research covering data transfer, privacy and security issues.

2.4 Semantic Wikis

Semantic Wikis try to overcome the problems related to information retrieval by combining Semantic Web standards such as RDF/S and OWL with the Wiki paradigm. One idea is to annotate structure in the Wiki by providing metadata for existing features such as links and pages. These annotations are formal representation of additional resources that assist in describing the main resources in a Wiki page. In addition, one can strive to completely represent the Wiki content using instances of the respective ontology language [Ki06]. While the authoring effort is similar to that of regular Wikis, the main difference is that a Semantic Wiki enables users to additionally describe resources in a formal language, instead of natural language.

The formal annotation of resources allows Semantic Wikis to offer additional features. Users can query the annotations directly, e.g. “show me all authors”, or create views from such queries. In addition, users can navigate the Wiki using the annotated relations, e.g. “go to other books by Grady Booch”, and introduce background knowledge to the system, e.g. “all books are publications; show me all publications” [OB06].

The potential benefits of Semantic Wikis relative traditional ones are summarized in [Da06]. First, they provide concept-based rather than language-based searching, enabling question answering rather than simple information retrieval. Second, they provide rich-structured content navigation, including multiple perspectives, multiple levels of abstraction and relationships. Third, easy visualization and direct editing of content structure (categories, taxonomies, semantic nets) is possible in Semantic Wikis. Fourth, semantic relationships can be externalized from text-based content, e.g. by using data mining techniques, and then formalized as new knowledge. Fifth, Wiki content is linked to dynamic models, simulations and visualizations, as well as to external repositories and file systems such as personal desktop, enterprise servers, web sources or RSS feeds. Finally Semantic Wikis provide richer user access/rights models, including reputation systems.

3 Semantic Wiki: State-of-the-Art

In a separate survey [PM07], we have gathered relevant features of a variety of Semantic Wikis. We identified five types of activities that cover the whole operational lifecycle of a Semantic Wiki, being authoring, navigation, retrieval, reuse and social collaboration. Table 1 summarizes our findings. The following conclusions can be derived.

Commonalities and differences: Only few features are common and provided by most surveyed Semantic Wikis. Context-aware navigation and full-text search are implemen-

[2] http://www.w3.org/RDF/
[3] http://www.w3.org/TR/owl-features/

Table 1: Overview of state-of-the-art features of Semantic Wikis (● = supported)

Feature / Semantic Wiki	COW	IkeWiki	Kaukolu	Makna	OntoWiki	OpenRecord	Platypus Wiki	Rhizome	Semantic MediaWiki	SemperWiki	SweetWiki	WikSAR
Authoring												
ACID[4] transactions (4)[5]	●			●		●			●			
Auto-completion (6)		●	●	●	●						●	●
Ontology editor (4)	●	●			●						●	
WYSIWYG editor (4)		●			●	●					●	
Navigation												
Context-aware navigation (9)	●	●		●	●		●		●	●	●	●
Different views (3)					●			●				●
Faceted browsing (2)					●						●	
Interactive graph visualization (1)												●
Ontology browser (3)	●	●			●							
Retrieval												
Embedded query (2)	●											●
Full-text (12)	●	●	●	●	●	●	●	●	●	●	●	●
Inference (5)		●		●	●		●				●	
Query templates (2)	●			●								
Using query language (6)	●						●	●	●	●		●
Reuse												
Ontology export (5)	●		●	●				●	●			
Ontology import (3)			●	●					●			
Social collaboration												
Change tracking (5)		●			●	●		●	●			
Commenting (3)			●		●			●				
Popularity (1)					●							
Rating (2)					●		●					

mented by 9 respectively 12 from all 12 surveyed engines. All other features are not provided by more than half of the systems. We also observed that 10 from 12 surveyed Semantic Wikis provide at least two alternative ways for information retrieval. Some of them strive to provide alternative ways of navigation as well, such as interactive graph visualisation, ontology browsing or faceted browsing. In addition, from the release history of the surveyed systems we can conclude a trend for supporting auto-completion, inference, query-language-based search, ontology export and change tracking.

The surveyed Semantic Wikis take different approaches concerning authoring, navigation and retrieval. Four of them provide WYSIWYG editors complementing the func-

[4] Atomicity, consistency, isolation and durability
[5] The number of Semantic Wikis supporting the feature from 12 surveyed systems

tionality of regular text editors. Wiki-based, easy content evolution is combined with ontology-based formal knowledge representation in various ways. For example, in COW [Fi06] users manipulate the ontology outside the Wiki, while OntoWiki [ADR06] offers an inline mode for editing RDF content and interactively propose already defined concepts to be added to the knowledge base. Besides OntoWiki, there is only a poor support for collaboration and social features. OntoWiki fosters collaboration by keeping track of changes, allowing commenting and discussing every single peace of knowledge, enabling to rate and measure the popularity of content and honouring the activity of users.

Open issues: From our survey we observed three main issues in current implementations. First, there seems to be no Semantic Wiki that allows true import of RDF data. Some Wikis allow usage of ontologies, but integration into the Wiki concepts seems to be amendable. For example, loaded ontologies typically do not show up in the Wiki since they are loaded into a separate repository. Thus, ontologies are deemed to remain static and cannot be edited by users. Tolksdorf and Simperl [TS06] argue that an essential limitation in conventional Wikis – which in our opinion, exists in Semantic Wikis as well – is the design decision considering a Wiki as a closed system with no interaction possibilities with other services. Second, several existing Semantic Wikis consider the URI of a page about a concept as its URI. This simplification is reasonable while working with Wiki based encyclopaedias such as Wikipedia. Typically, such Wikis have one page per concept. Thus making a link to a Wiki page expresses semantic statements about a concept. In addition, if users do not explicitly state what resource the statement is about, it can be assumed that it refers to the concept of the page where the statement appears. However, when using RDFS, Wiki pages must be both of the type `wiki:page` and of the resource type the Wiki page is supposed to describe. While this approach can be handy for generating RDF data of "shallow" ontologies with few classes and many relations, it reaches its limits as soon as more elaborate ontologies and structures are used. The simplified model restricts the granularity of possible annotations, since only the concept of the page and no fragment can be referenced. Finally, while few existing Semantic Wikis allow addition of semantic features to existing content, e.g. by typing previously untyped links in the Wiki, no one seems to assist users to extract further semantics from plain text. Besides, the only mean of querying semantics is either by simple queries built with a user interface, such as "Show a list of all publications to me", or complex queries entered manually in a query language such as SPARQL.

Conclusion. While we observed a broad feature spectrum in surveyed Semantic Wikis, common feature are still rare. The only common feature, full-text search, is indeed a standard functionality of conventional Wikis. The absence of a common "character" of Semantic Wikis can be interpreted by the absence of common requirements for this technology driven application. In addition the immaturity and experimentation nature of most available implementations explains their incompleteness. The most complete engine in terms of supported features is OntoWiki, followed by COW, IkeWiki and Makna.

4 Exploiting Semantic Wikis in Software Engineering

In this section we investigate potentials of Wiki-based technologies in the domain of software engineering. We first survey usage possibilities of conventional Wikis. Then we discuss new opportunities and challenges resulting from emergent Semantic Wikis.

4.1 Wikis in Software Engineering

The short history of Wikis is tightly coupled to software engineering. In fact, Wiki was invented in a software development context, when Ward Cunningham coined the term in 1995 for the website of the "Portland Pattern Repository"[6]. This first ever Wiki is dedicated to collecting and discussing design patterns. Since then, Wikis have been increasingly adopted in further communities. Especially distributed teams in the Open Source community appreciated Wikis as a lightweight solution for collaborative knowledge exchange. In such dynamic environments, Wikis act as flexible "glue" in the tool landscape, complementing mailing lists and configuration management tools. The rising availability of stable open source Wiki engines also fostered the adoption of Wikis in company settings. Again, tech savvy developers started to favour flexible Wiki technology against awkward traditional content management and groupware systems. Often, the rollout of Wiki software in those intranet settings started as a grassroots initiative without explicit management attention [Bu06, TW06]. Throughout the time, software engineering has remained one of the dominant application areas of Wiki technology [Lo06]. Wikis are often introduced by a small number of early adopters and then used by other employees. Besides open source development teams and small companies, also large companies such as SAP, Novell and Yahoo are using Wikis internally [TW06, Ba06]. Shashi Seth from Google even claims: "This company runs on Wikis"[7]. Currently, commercial vendors are addressing Wikis' technical drawbacks, poor usability support and low integration with other applications. For example, in its newest operating system, Apple Computer integrated an AJAX-based, user-friendly Wiki Server into several applications such as calendaring, mailing and content management ones. Besides general purpose "enterprise Wikis" other solutions focus on the software engineering domain. This includes integrating Wikis in IDEs [JJ05], source code documentation [AD05], bug tracking [Ed06a] or in collaborative development platforms[8]. In the following we discuss successful deployment fields and limitations of Wikis in software development.

Documentation: Changes, e.g. in requirements, technologies, architectures or resources, belong to the everyday life of a software developer. Thereby a main challenge is to keep documentation up-to-date. In addition, in large projects with overlapping responsibilities, information can get redundant, and then quickly inconsistent. Wikis are a convenient platform to ensure information up-to-datedness and consistency. First, the easy-to-use editing mechanism enables all stakeholders to update information as soon as changes occur. In such way documentation also gets distributed and simultaneous to development and management. Second, the easy linking in Wikis reduces inconsistencies by keeping the information in a unique Wiki page, and referencing it if required. WikiDoc [WZ07] and XSDoc [AD03, AD05] demonstrate benefits of Wikis for software documentation.

[6] http://c2.com/cgi/wiki?WikiHistory
[7] http://google.wikia.com/wiki/Goowiki
[8] E.g., Polarion (http://www.polarion.com) or CodeBeamer (http://www.codebeamer.com)

In WikiDoc java code documentation can be added via a Wiki interface – also from non-programmers. XSDoc enables the creation and annotation of software framework documents as well as the integration of different content types (text, models and source code).

Using Wikis as a documentation infrastructure might be inefficient, if paper-based, high quality documentation such as user manuals is needed. In this case, the organic structure of Wiki pages, due to the spontaneous and non-predefined growth, as well as the informality of the content might affect the document quality. Wiki pages needs to be restructured, serialized and adapted to target readers. The missing structure in Wikis might also cause knowledge acquisition problems for newcomers. For example, navigating a Wiki of an Open Source project at the first time, it is difficult to know where to start and in which sequence to read. Large projects such as the Eclipse project spend extra effort for structuring the content after pages has been created.

Reuse: Wikis are an intuitive platform for exchanging reusable knowledge within and between software projects. Their support for reuse goes beyond its origin for supporting pattern reuse. Easy linking enables reusing, e.g. requirements, test instructions or lessons learned. The DRY principle (Don't Repeat Yourself) is easy to comply within a Wiki.

However, reuse opportunities are limited to either information textually represented in the Wiki, or to artefacts linked from or within the Wiki. Even if a Wiki captures significant information about a problem domain, identifying reusable parts in later projects is a non-trivial task. This is mostly due to limited expressiveness, e.g. categorization or formal description information. Again, this limitation makes advanced, structured retrieval of information hard. Learning about a component design or querying available interfaces, usage parameters, pre- and post-conditions is, e.g., easier in JavaDocs than in a Wiki.

Issue tracking: Wikis have been used to capture and manage bugs and issues of software systems, especially in distributed open source projects. As bug tracking is a collaborative, knowledge-intensive and interdisciplinary task, it can be supported by Wikis. Again, the linking feature and the easy posting are the key success factors. Developers as well as system users are able to post, comment, link, describe and collaborate in order to detect and fix issues. Trac [Ed06b] is an example of a Wiki-based bug-tracker that allows relating Wiki pages to issues, and integrating source code as read-only documents.

The lack of attributes with specific semantics is at a disadvantage of using Wiki for bug tracking. Unlike conventional bug repositories, Wikis do not provide features to annotate the bug with well-defined attributes such as status, priority or affected subsystem. Such attributes are required by development teams and need to be queried and processed. For example, shortly before a release, the integrator needs to know all closed bugs, in order to integrate them into the new version. Statistics, e.g. about most "buggy" subsystem or duration of fixing bugs, are also required for project management reasons.

Requirement and Traceability management: Traceability enables to trace requirements back to stakeholders or forth to design concepts, source code and test cases. In projects with frequent changes, participants also need to trace other artefacts than requirements to further artefacts, decisions, models and participants [DP03]. It is useful to, e.g., trace test cases to the related components that should be tested, or a design goal to non-functional requirements. Also management decisions might be traced to a risk, to a design decision or to a stakeholder. Wikis offer an intuitive way to associate "everything with everything". Links between Wiki pages can be seen as associations between information pieces. This makes information traceable to other information across the project. Using XSDoc [AD03], in [SF05] the authors present a Wiki-based traceability management approach for customizable software. Relationships between specific requirements and generic product characteristics are established via configuration parameters and configuration questions. Viewing facilities support traceability analysis.

However, the linking in Wikis can be simplistic for advanced traceability requirements. Though, Wiki offers only one semantic of links, i.e. "associated to". Developers cannot define in Wikis further link semantics like *Feature F* "is required by" *Customer C*, *Instruction Sequence I* "tests" *Feature F* or *Decision D* "solves" the *Problem P*. In incremental development such missing semantics bring difficulties to deal with change. For example a new project participant cannot easily query previous design decisions resulting from specific requirements. Also a tester cannot easily retrieve test cases that have to be conducted based on features and changes included in the next release.

Communication and collaboration: Wikis support both planed and unplanned communication. A successful example for planned communication is the Wiki-based management of meeting protocols [MB06]. Decker et al. [DR05] state four main advantages of Wikis for supporting collaboration and knowledge exchange in software projects: *One place publishing*. A central place for publishing information often simplifies the communication between participants. *Simple and safe communication*. The Wiki syntax is trivial, especially to software developers who are used to source code editing. Simple access right mechanisms make the communication more secure. *Easy linking*. This is a major strength of Wikis, especially when applied in the domain of software engineering. *Description on demand*. A user might link an object to a target, although the target does not exist at the "linking" time, and will be created whenever it is needed. In [Ba06] the authors conclude that the main success factors for introducing Wikis at SAP were easy accessibility, low entry barriers and to "convince the hackers". Users are allowed to edit most pages while the versioning mechanism avoids vandalism – social protocols replace technical controls like access rights. Chau and Maurer [CM05] found out, that the paradigm of decentralized contributors adding unstructured knowledge improves knowledge sharing in contrast to structured knowledge provided by a centralized team.

Nevertheless, low entry barriers and easy content editing requires a high discipline from developers. This is analogue to Extreme Programming, where continuous code refactoring requires highly disciplined team members. Besides, Wikis do not offer support for synchronous communication and limited notification mechanisms. Users can get notified only if a Wiki page has been changed, leading in some cases to an information overload. It is for example not possible to receive a notification only if an "important" requirement has been changed. Important here depends on developer's interests and activities.

Agility support: Wiki represent a convenient knowledge-sharing infrastructure for agile teams, because it supports three main agile values: informal communication, responding to change and developers' motivation. First, the low barriers to create and update content in Wiki encourages informal communication [Be01]. Similar to Web forums or mailing lists, every project participant can require or share knowledge by creating or editing a Wiki page. Interested participants can also watch the page for updates and contribute to the knowledge in it. For example few Open Source libraries such as PHP include developers' questions and answers as well as usage examples at the bottom of the Wiki page of the corresponding library function. Second, unlike conventional processes that strive toward minimizing and controlling changes, e.g. through change control boards, agile teams consider change as a "mutual" part of daily work. Wikis make up a flexible and quick mechanism of carrying out changes. Third, managing project knowledge in a Wiki requires developers to personally propagate their knowledge to the team, which augments their motivations, as they become "valuable" members in the project. Unlike central communication mechanisms with tight control, in Wiki everyone is responsible for his content, which increases the information quality. Several studies demonstrated the usefulness of Wikis in agile projects [SF05, CM04, CM06]. In [CM06] the authors present the results of an exploratory case study, arguing for the need of knowledge sharing tools that support not only structured but also unstructured knowledge representation. Wiki-based informal knowledge authoring tools can be used for sharing content about problem understanding, instrumental, projective, social, expertise location and content navigation purposes. The authors also observed self-organized maintenance of the repository content as a result of the open-edit nature of the Wiki-based repository.

Conclusion: While Wikis are used as a universal tool for a wide range of software engineering tasks, their full potentials are not taped yet. Wikis procure convenient mechanisms for managing informal, unstructured content and linking different information pieces. They support the fact that knowledge matures as it evolves and discussed in a community [HSc07]. Low entry barriers and easy editing allow for small contributions and an incremental content improvement. In early project stages developers can easily interpret the implicit semantics in Wiki pages. However, in later development stages, as content in a Wiki grows and software artefacts get more formal, maintaining and accessing information becomes difficult. In the following section we discuss, how Wikis can be extended to express formal semantic statements to cope with these difficulties.

4.2 Semantic Wikis in Software Engineering

While classical Wikis are well suited for working collaboratively on unstructured information, they are limited concerning fine-grained structured content. Conventional Wikis offer limited support for storing machine-interpretable knowledge and for answering structured queries. Semantic Wikis provide a general approach for the acquisition and evolution of structured knowledge.

Since mature implementations of Semantic Wikis are still rare, there are no established applications in software engineering yet. However we want to provide a grasp of concrete visionary scenarios. We revisit application areas in software engineering, where Wikis have been successful, and discuss the new potentials of Semantic Wikis.

Documentation: Since Semantic Wikis support structural queries as well as reasoning, specific document such as user manuals or interface specifications can be automatically created by using query templates. Semantic Wikis also offer various navigation features, such as context-aware navigation or interactive graph visualisation, as seen in Section 3. Users with different roles can be provided with personalized views according to their specific interests. This is beneficial if different stakeholders with different perspectives need to exchange information, which is the case in the documentation of Service-Oriented Architectures (SOA). These systems consist of a large number of heterogeneous artefacts and documents and involve diverse stakeholders such as developers and business experts. The situation of both, technical e.g. interfaces specifications, and textual documentation leads to scattering of information into different spaces. Substantial management and coordination effort arise. Semantic Wikis can serve as a foundation for a lightweight solution to document SOA [Hse07]. They allow for informal documentation with low entry barriers, while providing means for specifying formal relations among concepts where needed. Additional information can be derived by combining the asserted knowledge with existing specification documents taken from external service repositories. The structure of technical artefacts like interface descriptions can be imported into a knowledge base, serving as the basic structure for browsing the SOA. This leads to clear overview and efficient navigation for developers and business experts.

Reuse: As stated in Section 2.4, knowledge can be externalized in Semantic Wikis. Results of analysis and mining of software artefacts, such as source code can be used to automatically annotate information in semantic Wikis. JavaDocs can, e.g. be linked to Wiki-based documentation, allowing advanced, structural queries. Moreover, through semantic connections between software components like services in SOA, and application domain concepts like business objects and rules in the stock exchange domain, it is possible to gain a quick overview about components that are using certain business objects. This enables exploring the list of available components, as well as advanced retrieval of relevant ones, for example those with appropriate licence for the current project context. Reusing component or services in SOA becomes easier since information about them can be efficiently retrieved. Decker et al. [DR05] illustrated how augmenting a Wiki with application and solution domain ontologies, can actually foster reuse. When used to annotating the knowledge created in requirements elicitation, their system is able to offer relevant recommendations based on scenarios that have been captured in earlier designs. For instance, explicitly “telling” the Wiki that an object is an actor and that an actor instantiates a use case, allows Wikis to recommend possible actors.

Issue tracking: It is generally difficult to define the exact scope and detail of an issue when it is reported for the first time. Often, bugs “emerge” from informal discussions on mailing lists or Wikis. Formally describing an issue results in a disruption from its original context. The main benefit in using Semantic Wikis is the support for a smooth transition from vague, ill-defined problems to more formal issue descriptions. Semantic Wikis thus combine advantages of conventional Wikis with those of bug repositories. They offer a usable and flexible bug description mechanisms, as well as features to semantically annotate specific bug attributes such as responsible person or affected version.

Requirements and traceability management: Semantic Wikis allow for ontology-based annotations of links between pages. Instead of creating a new “associated to” link, developers can create, e.g. “requires”, “extends” or “has higher priority” links between features. Requirements can also be semantically linked to other resources, e.g. to a user who "is concerned by" this requirement or to a developer how "is responsible for the implementation". Dealing with daily changes becomes easier since advanced retrieval functionalities can be automated. Affected test cases or components for the next release can be retried automatically every time requirement priorities change. Moreover consistency checks, e.g. each actor should initiate at least one use case, as well as cross project offering based on the capture of the user context might add value to classical Wikis.

Communication and collaboration: Wiki-based collaboration can get more precise and efficient by including semantic annotations to collaboration content. Information can be shared precisely to stakeholders depending on their interests. For example, a project manager is interested in deadline information for the next release, unlike an architect, who is more concerned about provisional design decisions. Linking content with user interests can be realized by an explicit, "semantic" registration of the user. Instead of watching pages as in conventional Wikis, users register themselves to watch concepts like a requirement or a release. Another example is the notification of a component owner if a component user adds an experience report. In addition, within large distributed projects more awareness can be achieved by automatically building interest groups based on linking semantically annotated content to an ontology-based user profile. Based on component dependencies, Wiki users such as component owners and component users can build a group of interest about this component.

Agility support: In addition to advantages that agile teams can take from conventional Wikis (Section 4.2), Semantic Wikis achieve more flexibility in release planning. In a project with frequent releases, test and integration checklists, release notes as well as management reports can be automatically extracted from the Wiki, using query templates and reasoning mechanisms. A query like "list manual tests that need to be conducted for the next release" can be supplied by selecting fixed issues, affected components and requirements and then retrieving related tests. Less effort is required for non-development, but essential activities such as quality and change management. In addition Semantic Wikis help in bringing customers closer to development teams. Customer and developers can edit the same Wiki, while information is still differentiated based on stakeholders interests. Different vocabularies used by customers and developers can be easily mapped by using "equivalent" annotations. Finally, implicit knowledge such as, the models included in the source code, e.g. in Extreme Programming (XP), can be extracted and formalized based on the ontology used by the Semantic-Wiki and then linked to the concepts described in the Wiki.

Conclusion: Semantic Wikis entail various advantages for software engineering projects. They enable an incremental formalisation of underlying knowledge across various software engineering activities. Thereby, they increment results from the underlying ontologies that can be extended and customized in different contexts. Unlike a development or a collaboration infrastructure with a fixed scheme, e.g. a bug repository where bugs must occur in a version, Semantic Wikis provide a simultaneously growing scheme and data. Thus a higher flexibility for project rules is supported. In addition, linking capabilities of Semantic Wikis enable the association of various artefacts and system models. Unlike in classic Wikis these associations are typed in Semantic Wikis. In conventional Wikis the only semantics of the link between two pages is that pages are "related to" each other. In real life, developers need to define a design pattern that traces a non-functional requirement, a change request that changes a core feature, or a design concept that implements a requirement. Annotation of data and links between data in Semantic Wikis satisfy such needs. Thereby, rules might be defined based on these typed associations as well as the information pieces, which also acquire types and semantic specifications. Conventional Wikis lack machine-understandable semantics. Thus structuring the Wiki content by machine reasoning is difficult. These semantics are especially valuable for linking the various data objects.

The acquired collaborative formal knowledge rather complements informal human communication than replacing it. Rule-based systems, reasoners and declarative query languages might be used for a semiautomatic processing of such knowledge and creation of new one. New content like implicit associations between artefacts can be generated. Additionally, flexible and declarative model-checking and -validation mechanisms become easy to implement. When common ontologies are deployed, project knowledge can be much easier imported from and exported to other projects and repositories. This is obviously valuable for software development organisations and communities. Reuse and knowledge exchange becomes more technology, organisation, process and methodology independent. A user-story in Extreme Programming (XP) can be considered, for example, semantically equivalent to a use case instance in Rational Unified Process (RUP).

5. Conclusion and Future Work

In this paper we presented the state of the art of Semantic Wikis, comparing different implementations and giving an overview of related challenges. This remains valid in the software engineering domain, where conventional Wikis has already shown many advantages. In documentation, reuse, issue tracking, requirement and traceability management as well as communication and collaboration, several success stories have been associated to Wikis, in particular in agile and distributed teams. However, Wiki's limited support for handling machine-interpretable information, answering structured queries and for structuring collaboration content also showed several disadvantages compared to other project infrastructures such as bug repositories or groupware. These difficulties can be eliminated by expressing formal semantic statements, as Semantic Wikis do. The incremental formalisation of knowledge, as well as the parallel growth of scheme and data represent the main arguments for considering Semantic Wikis as promising technology. When integrated to further software engineering tools a Semantic Wiki can serve as an integrated "Knowledge Desktop" for developers and provide a lightweight knowledge articulation and sharing facilities. This is especially relevant for sharing knowledge across distributed teams, and can therefore enable agility in non-collocated projects. Empirical studies should investigate this in more detail. Semantic Wikis can also serve as an integration point to other tools and resources. They provide flexible and intuitive ways for authoring, manipulating, retrieving and navigating through structured knowledge. The following features can then be offered to developers:

- Knowledge articulation facilities like semantic search and recommendation.

- Context-aware navigation through information objects.
- Faceted navigation and support of different views of data types.
- Ontology import and export to enhance reuse in software engineering.
- Integration of several data types to support different types of documents.
- Facilities to interrelate information objects and semantic traceability.
- Semantic recommendation functionalities.

Nevertheless, several research and technical issues like interoperability, distinctiveness and identification of resources, as well as methodology issues need to be addressed first.

6 Acknowledgments

This work has been supported in part by the TEAM project, which is funded the EU-IST programme under grant FP6-35111 and the BMBF-funded project WAVES.

References

[AD03] Aguiar, A.; David, G.; Padilha, M.: XSDoc: an Extensible Wiki-based Infrastructure for Framework Documentation, in Proceedings of JISBD, pp. 11–24, 2003.

[AD05] Aguiar, A.; David, G.: WikiWiki weaving heterogeneous software artifact, In Proceedings of the 2005 International Symposium on Wikis, San Diego, CA, pp. 67-74, 2005.

[ADR06] Auer, S.; Dietzold, S.; Riechert, T.: OntoWiki - A Tool for Social, Semantic Collaboration, in I. Cruz et al. (Eds.): Proceedings of 5th ISWC, GA, USA, LNCS 4273, 2006.

[Ba06] Barlas, D.: SAP Using Twiki, 01.2006, http://www.destinationkm.com/articles/default.asp?ArticleID=1127

[Ba92] Basili, V.; Caldiera, G.; McGarry, F.; Pajerski, R.; Page, G.; Waligora, S.: The software engineering laboratory: an operational software experience factory, In Proceedings of ICSE'92. ACM Press, 1992.

[Be01] Manifesto for Agile Software Development (2001) http://agilemanifesto.org/

[Bu06] Buffa, M.: Intranet Wikis, In Proceedings IntraWeb Workshop WWW2006, 2006.

[CM04] Chau, T.; Maurer, F.; Grigori, M.; Harald, H.: Proceedings of the 6th International Workshop on Advances in learning software organizations, Springer, 2004.

[CM05] Chau, T.; Maurer, F.: A case study of wiki-based experience repository at a medium-sized software company, K-CAP 2005, pp. 185-186, 2005.

[CM06] Chau, T.; Maurer, F.: A case study of a Wiki-based experience repository at a medium-sized software company, In Proceedings Kumar Jain, R.; Prabhakar, R. (Eds.): Wiki - A new wave in web collaboration, The ICFAI University Press, India, pp. 60-79, 2006.

[CRP06] Calero, C., Ruiz, F., Piattini, M.: Ontologies for Software Engineering and Software Technology, Springer Verlag, 2006.

[Da06] Davis, M.: Semantic Wave 2006 Part-1: Executive Guide to Billion Dollar Markets, A Project10X Special Report, 2006.

[DF04] Decker, S.; Frank, M.: The Networked Semantic Desktop, In Proc. of the WWW, 2004.

[DM06] Dutoit, A.H.; McCall, R.; Mistrik, I. ; Paech B.: Rationale Management in Software Engineering: Concepts and Techniques, In: Rationale Management in Software Engineering, 2006.

[DP03] Dutoit, A.H.; Päch, B.: Eliciting and Maintaining Knowledge for Requirement Evolution, In Proceedings Managing Software Engineering Knowledge, Springer, 2003.

[DR05] Decker, B.; Rech, J.; Ras, E.; Klein, B.; Hoecht, C.: Self-organized Reuse of Software

Engineering Knowledge supported by Semantic Wikis, In Proceedings of SWESE, 2005.
[Ed06a] Edgewell.org: The Trac User and Administration Guide, http://trac.edgewall.org/wiki/TracGuide, 09.2006.
[Ed06b] Edgewell.org: Trac Project Homepage, http://trac.edgewall.org/, 09.2006.
[Fi06] Fischer, J et al.: Ideas and Improvements for Semantic Wikis, in Proceedings of ESWC 2006, LNCS 4011.
[GN87] Genesereth, M.R.; Nilsson, N.J.: Logical Foundations of Artificial Intelligence, Morgan Kaufman Publishers, 1987.
[Gr93] Gruber, T.: A Translation Approach to Portable Ontology Specifications, Knowledge Acquisition, 5(2), 199-220, 1993.
[JJ05] John, M.; Jugel, M.; Schmidt, S.; Wloka, J.: Wikis in der Softwareentwicklung helfen, Java Magazin 7, pp. 88-91, 2005.
[HM03] Herbsleb, J.; Mockus, A.: Formulation and preliminary test of an empirical theory of coordination in software engineering. ESEC / SIGSOFT FSE. 2003.
[HSc07] Happel, H.-J. and Schmidt, A.: Knowledge maturing as a process model for describing software reuse. Proceedings of LSO 2007. 2007.
[Hse07] Happel, H.-J., Seedorf, S.: Ontobrowse: A Semantic Wiki for Sharing Knowledge about Software Architectures. In: Proceedings of the 19th SEKE, Boston, 2007.
[Ki06] Kiesel, M.: Kaukolu: Hub of the Semantic Corporate Intranet, SemWiki Workshop, ESWC 2006.
[LC01] Leuf, B.; Cunningham, W.: The Wiki Way: Collaboration and Sharing on the Internet, Addison-Wesley, 2001.
[Lo06] Louridas, P.: Using Wikis in Software Development, IEEE Software, 23, 2, 2006.
[MB06] Mullick, N.; Bass, M.; Houda, Z.; Paulish, P.; Cataldo, M.: Siemens Global Studio Project: Experiences Adopting an Integrated GSD Infrastructure, In Proceedings of ICGSE'06, 2006.
[OB06] Oren, E.; Breslin, J. G.; Decker, S.: How semantics make better Wikis, in Proceedings of WWW '06, ACM Press, New York, NY, pp. 1071-1072, 2006.
[Or05] Oren, E.: SemperWiki: a semantic personal Wiki, in Proceedings of the 1st Workshop on The Semantic Desktop - Next Generation Personal Information Management and Collaboration Infrastructure, Galway, Ireland (ISWC 2005).
[PL06] Paasivaara, M.; Lassenius, C.: Could Global Software Development Benefit from Agile Methods? In Proceedings of ICGSE’06, IEEE Computer Society, 2006.
[PM07] Panagiotou. D. and Maalej. W. (ed.): Report describing state-of-the-art in SE Knowledge Desktop. 2007.
[Pr04] Pretorius, A.: Ontologies – Introduction and Overview, 2004. http://www.starlab.vub.ac.be/teaching/Ontologies_Intr_Overv.pdf.
[Ram06] Ramesh, B.; Cao, L.; Mohan, K.; Xu, P.: Can distributed software development be agile? Commun. ACM 49, 10 (Oct. 2006), 41-46, 2006.
[SF05] Silveira, C.; Faria, J.P.; Aguiar, A.; Vidal, R.: Wiki Based Requirements Documentation of Generic Software Products, AWRE’05, Melbourne, Australia, pp. 42-51, 2005.
[SM07] Schmidt, R.; Maalej, W.; Happel, H-J.; Kögel, M.; Narendula, R.; Panagiotou, D.; Wolf, T.: Report describing State-of-the-Art in Metadata Management, 2007.
[SB01] Schwaber, K. and Beedle, M.: Agile Software Development with SCRUM. 2001.
[TS06] Tolksdorf, R.; Simperl, E. P.: Towards Wikis as Semantic Hypermedia, in Proceedings of the WikiSym '06, ACM Press, New York, NY, pp. 79-88, 2006.
[TW06] TWiki Success Stories, http://twiki.org/cgi-bin/view/Main/TWikiSuccessStories 2006.
[W3C07] Semantic Web Official Homepage 12.2007. http://www.w3.org/2001/sw/
[WK02] Williams, L. and Kessler, R. Pair Programming Illuminated. 2002.
[WZ07] Wolf, R.; Zhao, J.: JavaDoc + Wiki = WikiDoc - Kollaborative Dokumentationssystem für Java, 2007. http://projects.mi.fu-berlin.de/w/bin/view/SE/ThesisWikiDoc