

Qualitative Analysis of Semantically Enabled Knowledge Management Systems in Agile Software Engineering

Jörg Rech

Semantic Technologies
Kurt-Schumacher Str. 72, 67663 Kaiserslautern, Germany
joerg.rech@gmail.com

Christian Bogner

Technical University of Kaiserslautern
Erwin-Schrödinger-Str. 57, 67663 Kaiserslautern, Germany
christian.bogner@sowi.uni-kl.de

Abstract: In many agile software engineering organizations there is not enough time to follow knowledge management processes, to retrieve knowledge in complex processes, or to systematically elicit knowledge. This chapter gives an overview about the human-centered design of semantically-enabled knowledge management systems based on Wikis used in agile software engineering environments. The methodology – developed in the RISE (Reuse in Software Engineering) project – enables and supports the design of human-centered knowledge sharing platforms, such as Wikis. Furthermore, the paper specifies requirements one should keep in mind when building human-centered systems to support knowledge management. A two-phase qualitative analysis showed that our knowledge management system acts as a flexible and customizable view on the information needed during working-time which strongly relieves software engineers from time-consuming retrieval activities. Furthermore, our observations gave some hints about how that the software system supports the collection of vital working experiences and how it could be subsequently formed and refined.

Keywords: Semantically Enabled Knowledge Management Systems, Human-centered Design, Agile Software Engineering, Software Reuse, Experience Management, Wiki, Semantic Wiki, Riki

Introduction

The development of complex software systems is based on company- and domain-specific knowledge that has to be constantly cultivated among the employees, because the resulting quality of manufactured software systems depends on what degree the needed knowledge is actually available (Decker et al., 2005).

Typically, knowledge management (KM) platforms are used to support the knowledge capturing, management, or sharing processes within companies. However, in agile software engineering (SE) organizations there is not enough time to follow KM processes, retrieve knowledge in complex processes, or systematically elicit knowledge (e.g., using post-mortem analyses (Birk et al., 2002)). The technical KM platform for agile software organizations should release Software Engineers as much as possible from time-consuming retrieval processes.

Today, due to the various possibilities of searching and browsing through software engineering artifacts, it is assumed that users feel overwhelmed just by the flood of information: while the increasing amount of information itself might not be a problem, an unfiltered and unrated access to it is. When using KM systems, we should never forget that the main goal is to ensure that Software Engineers can deal with their daily tasks without burdening them with additional work.

Hence, we developed a semantically-enabled knowledge management system for SMEs based on a Wiki. This system not only supports the easy retrieval but also acquires valuable pieces of information from users, who publish their problems and experiences during work. Out of the day-to-day work, company- and product-specific knowledge can be built and refined without interfering too much with the daily work. Our approach, which concentrates on essential aspects of an artifact, considers the previous knowledge as well as the interests of the targeted users – provided by semantics encoded in metadata, concept structures and user profiles. It can be used to arrange artifacts in a flexible way depending on the users' needs (e.g., artifacts written/read by the user and given interests or tasks of the user). Besides that searching the whole system for certain artifacts will present results in a nested or context-based way (e.g. the artifact describing "Pair Programming" is arranged under its parent concept "Extreme Programming" (Beck, 1999a)). This solution is based on underlying Ontologies (Gruber, 1995). After the whole process of selecting certain aspects and restructuring them, users may be provided with complete, self-contained and motivating knowledge.

Our contribution touches several fields, such as Human-Centered Interfaces, Education, and knowledge management with a special focus on Wikis. We tackled the objectives to reduce learning barriers, improve the quality of knowledge, and improve the structure as well as interconnection of knowledge within KM systems. Our contributions comprises of knowledge elicitation techniques, SE-specific didactically improved templates, and semantic relations for SE documents.

In this paper, we present the research methodology (section 2) used in our project to build and evaluate a human-centered and semantically enabled knowledge management system (section 4) as well as the necessary background (section 3) from SE and KM. Several methods to support the users by means of an intelligent application are discussed within in the paper. The qualitative evaluation conducted in the project RISE – including baseline and delta evaluation – is then described in section 5. We give a summary and conclusion in section 6.

Methodology

In order to support the reuse of software engineering products such as requirement documents, we build the semantically enabled knowledge management system Riki (Reuse-oriented Wiki). The Riki was targeted towards ease of use and improved the acceptance by the users (Höcht & Rech, 2006). The methodology and technology developed make it possible to share knowledge in the form of software artifacts, experiences, or best practices based on pedagogic approaches.

Development Methodology

Our methodology to develop the Riki was based on the principles of a Human-Centered Design Process according to the international standard ISO 13407, as depicted in Figure

1. It expects that users take an active part in the development process in order to get a clear idea of what might be specific requirements (ISO-13407, 1999).

The early phase of our research project RISE (Rech et al., 2007b), therefore, was dominated by context-analyses within the participating companies. Qualitative-centered studies delivered deep insight into organizational aspects and users' special needs. Two focus groups dealt with the drawbacks and opportunities of using conventional Wiki-systems which were formerly introduced by the associated companies. Furthermore, the analysis of existing artifacts allowed allocating functions among the technical system and respectively the users themselves.

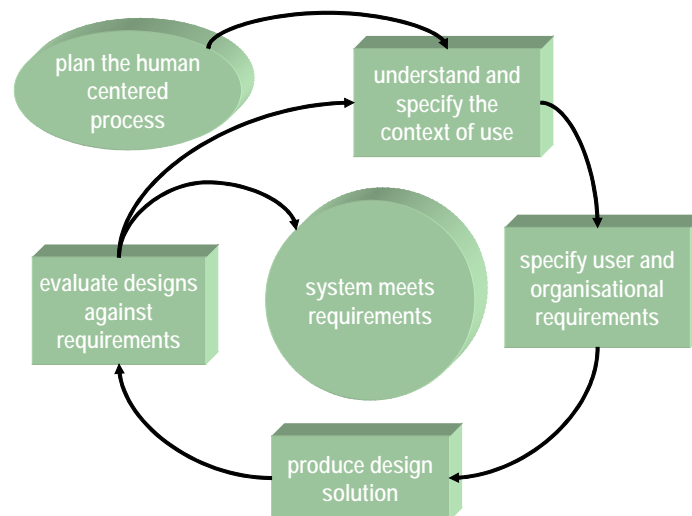


Figure 1. Human-centered design processes for interactive systems (ISO-13407, 1999)

For example, the analysis of individually built file structures showed that an automatically enabled structuring of existing artifacts might be critical because people tend to use structures in different ways (hierarchical, time-based, priority-based, people-based etc.) and with different granularity. This includes accessing existing artifacts as well as creating new ones. Teevan et al. discovered in a comparable study that people even use hierarchical organized structures if they do not maintain their individual structures hierarchically (Teevan et al., 2004). This example shows, how important it is, to allocate functions between users and intelligent systems. But it also gets clear that definite answers could not be found until final tests of the application.

Evaluation Methodology

The objective of our explorative evaluation was to show its effectiveness in a specific application context. In our evaluation the tailored instances of the Riki system were evaluated to identify the usefulness to the users, the examined applicability, the evolvability, as well as economic factors.

As *subjects* two companies participated in the evaluation: empolis GmbH – an enterprise with a development team of 40 developers and brainbot technologies AG – a micro-enterprise with 5 developers.

While we mainly conducted an explorative evaluation about the use of knowledge in agile organizations without specific hypotheses, one very important *hypothesis* of the RISE project was that software development teams collaboratively using a Riki, are actually about to create a semantic structure – with users not necessarily to be aware of. This structure called “Wikilogy” can be considered as a weakly formalized ontology with three main characteristics: a) the *maintenance of content and metadata* (i.e., the *ontology*) is not considered as extra activity, b) *ontologies* (as seen/build by the users) may always grow behind the content, and c) maintenance of these Ontologies can be *supported efficiently*.

The evaluation itself was split into two *phases*. The baseline evaluation was purely evaluative and used to elicit the context and current state of knowledge transfer. The delta evaluation later helped to measure the effect of the introduced Riki system. The baseline and delta evaluations have been carried out with the same teams and managers.

In both phases we used the following three *techniques* to elicit valuable information from the organizations and potential end-users:

- Goal-oriented, questionnaire-based *interviews* were used to query the previously listed questions with three to ten persons in two to four sessions. The collected answers were summarized and validated by the participants via email.
- *Group discussions* were done at every company to collect any additional information, opinions, ideas, etc. that were not covered by the interviews. The discussion was started with a specific topic (e.g., why had the old Wiki not worked for you as a KM system?) and every person could state what they expected from an improved knowledge management infrastructure.
- *Artifact analyses* were conducted to identify knowledge sources, the type of knowledge within, as well as how the people structure their documents and knowledge in existing storage systems (e.g., the hierarchy of directories in personal file systems or in pre-existing Wiki systems).

These evaluation techniques helped to cover the evaluation of a) the *technology* used by the companies (e.g., technical infrastructure, existing KM systems, and other software systems) that might be integrated into the Riki system, b) the *methodology* applied in the companies for software development and knowledge management, and c) the *knowledge* available in the companies as well as their characteristics and interrelation (e.g., for the development of an ontology).

A more detailed description of the baseline and delta evaluation as well as some results of the two evaluation phases are described in section 5.

Background

The RISE project

The research project “RISE” (Reuse In Software Engineering) was part of the research program “Software Engineering 2006” funded by the German Federal Ministry for Education and Research (BMBF) that started in June 2004 and ended in December 2005. The goal of this research project, in general, was to improve the reuse of artifacts in software engineering (with a focus on the requirement phase) and brought together researchers from social science (the Department of Educational Sciences and

Professional Development at the Technical University of Kaiserslautern) and computer science (Fraunhofer Institute for Experimental Software Engineering (IESE) and the German Research Center for Artificial Intelligence (DFKI)) with industrial partners (empolis GmbH – a small enterprise with a development team of about 40 developers and brainbot technologies AG – a micro-enterprise with approx. 4 developers).

Software Engineering and Reuse

Software engineering (SE) as a field in computer science is concerned with the systematic development of high-quality software systems. During the planning, definition, development and maintenance of software systems the people involved generate and require any information and knowledge to support them in their work or to back up their decisions. This includes source code as well as information about processes, products, or technologies. Knowledge reuse and code reuse, from our point of view, is equivalent as the user has to internalize the meaning of the reusable object (e.g., a code fragment or a how-to guideline) and apply it in a new context.

This reuse of existing knowledge and experience is one of the fundamental parts in many sciences. Engineers often use existing components and apply established processes to construct complex systems. Without the reuse of well proven components, methods, or tools we had to rebuild and relearn them again and again.

In the last thirty years the fields software reuse and experience management (EM) (Althoff et al., 2001) are gaining increasingly importance. The roots of EM lie in Experimental Software Engineering (e.g., the "Experience Factory"), in Artificial Intelligence (e.g., "Case-Based Reasoning"), and in knowledge management. EM is comprised of the dimensions methodology, technical realization, organization, and management. It includes technologies, methods, and tools for identifying, collecting, documenting, packaging, storing, generalizing, reusing, adapting, and evaluating experience knowledge, as well as for development, improvement, and execution of all knowledge-related processes.

The Experience Factory (EF) is an infrastructure designed to support experience management (i.e., the reuse of products, processes, and experiences from projects) in software organizations (Basili et al., 1994). It supports the collection, preprocessing, and dissemination of experiences the organizational learning and represents the physical or at least logical separation of the project and experience organization as shown in Figure 2. This separation is meant to relieve the project teams from the burden to find, adapt, and reuse knowledge from previous projects as well as to support them to collect, analyze, and package valuable new experiences that might be reused in later projects.

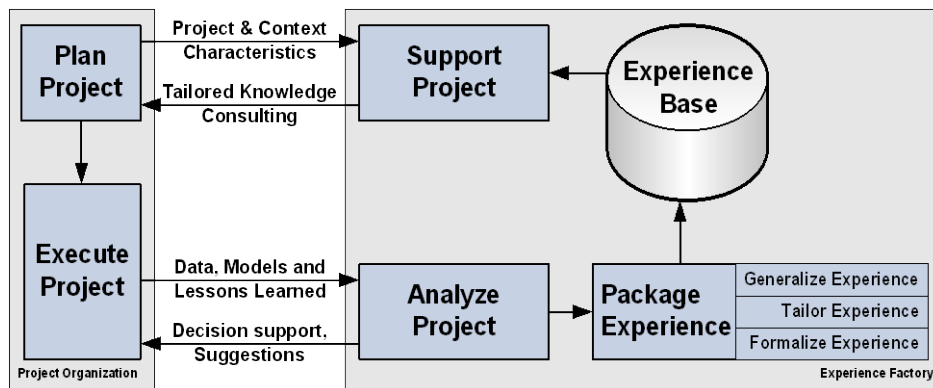


Figure 2. The Experience Factory

Traditional, process-oriented Software Development and Reuse

Since its beginning several research directions developed and matured in field SE. Figure 3 shows the software development reference model integrating important phases in a software lifecycle. *Project Engineering* is concerned with the acquisition, definition, management, monitoring, and controlling of software development projects as well as the management of risks emerging during project execution. Methods from *Requirements Engineering* are developed to support the formal and unambiguous elicitation of software requirements from the customers, to improve the usability of the systems, and to establish a binding and unambiguous definition of the resulting system during and after software project definition. The research for *Software Design & Architecture* advances techniques for the development, management, and analysis of (formal) descriptions of abstract representations of the software system as well as required tools and notations (e.g., UML). Techniques to support the professional *Programming* of software are advanced to develop highly maintainable, efficient, and effective source code. *Verification & Validation* is concerned with the planning, development, and execution of (automated) tests and inspections (formal and informal) in order to discover defects or estimate the quality of parts of the software.

Research for Implementation & Distribution is responsible for the development of methods for the introduction at the customer's site, support during operation, and integration in existing IT infrastructure.

After delivery to the customer software systems typically switch into a *Software Evolution* phase. Here the focus of research lies on methods in order to add new and perfect existing functions of the system. Similarly, in the parallel phase *Software Maintenance* techniques are developed for the adaptation to environmental changes, prevention of foreseeable problems, and correction of noticed defects. If the environment changes dramatically or further enhancements are impossible the system either dies or enters a *Reengineering* phase. Here techniques for software understanding and reverse engineering of software design are used to port or migrate a system to a new technology (e.g., from Ada to Java or from a monolithic to a client/server architecture) and obtain a maintainable system.

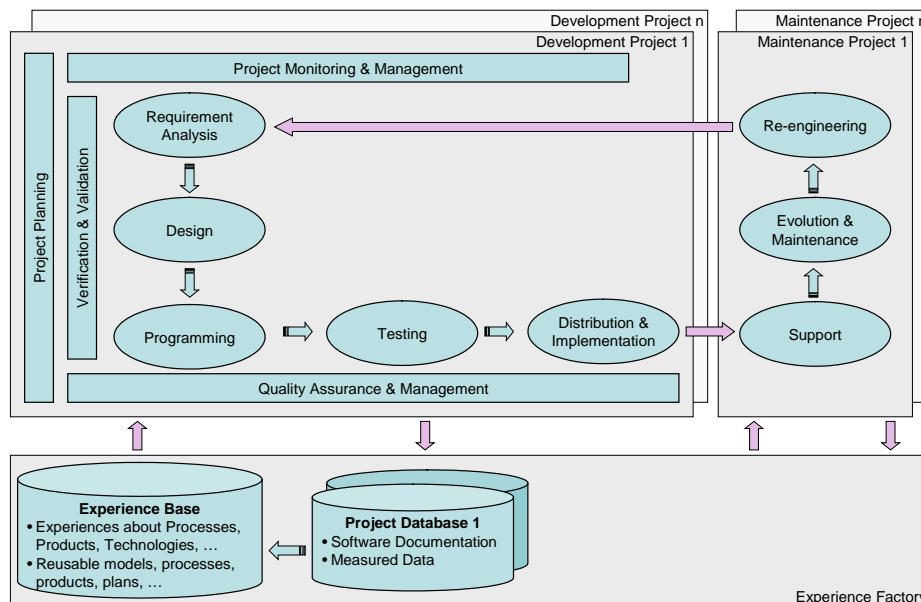


Figure 3. Software Development Reference Model

Agile Software Development and Reuse

Beside the traditional or process-oriented software development another trend arose during the last years. Agile software development methods such as Extreme Programming (Beck, 1999b), Scrum (Schwaber & Beedle, 2001) and others (Ambler, 2002; Cockburn, 2006; James A. Highsmith, 2000; Stapleton, 1999), impose as little overhead as possible in order to develop software as fast as possible and with continuous feedback from the customers. Agile methods have in common that small releases of the software system are developed in short iteration in order to give the customer a running system with a subset of the functionality needed. Therefore, the development phase is split into several activities that are followed by small maintenance phases.

Today, Extreme Programming (XP) (Beck, 1999a) is the best-known agile software development approach. Figure 4 shows the general process model of XP that is closely connected to refactoring and basically its cradle (Beck & Fowler, 1999). Extreme programming (XP) is based upon 12 principles (Beck, 1999a). We mention four of these principles as they are relevant to our work. *The Planning Game* is the collective planning of releases and iterations in the agile development process and necessary to quickly determine the scope of the next release. Knowledge about the existence of existing code elements or subsystems relevant to the project can be used to plan the scope of the next release. *Small releases* are used to develop a large system by first putting a simple system into production and then releasing new versions in short cycles. The more an engineer can reuse the faster his work is done and the quicker the customer gets feedback. *Simple design* means that systems are built as simple as possible, and complexity in the software system is removed if at all possible. The more libraries are used and identified (via code retrieval) the less functionality has to be implemented in the real system. *Refactoring* or the restructuring of the system without changing its

behavior is necessary to remove quality defects that are introduced by quick and often unsystematic development. Decision support during refactoring helps the software engineer to improve the system (Rech, 2007).

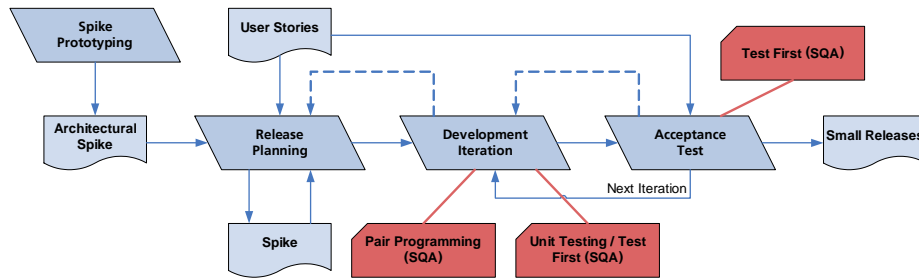


Figure 4. Agile Software Development (here the XP-Process)

In contrast to traditional, process-oriented SE where all requirements and use cases are elicited, the agile methods focuses on few essential requirements and incrementally develops a functional system in several short development iterations. Agile software processes, rely on direct face-to-face communication between customers and developers for knowledge sharing and, therefore, reduce the information loss in in-team and team-to-customer communication (Chau et al., 2003). Pair programming activities improve the exchange of knowledge and experience between two team members and results in fewer bugs, spreading code understanding, and producing overall higher quality code (Begel & Nagappan, 2008). In general, many positive benefits of agile approaches have been reported including shorter development cycle, higher customer satisfaction, lower bug rate, and quicker adaptation to changing business requirements (Barry & Richard, 2003). For the Scrum method, (Cho, 2008) identified the five challenges including reduced documentation, malfunctioning communication, missing user involvement, closed working environment, and superfluous team information exchange.

Traditional software reuse initiatives and approaches that were developed for process-driven software development are inadequate for highly dynamic and agile processes where the software cannot be developed for reuse and reuse cannot be planned in advance. KM in agile methods apply a personalization strategy with (1) reliance on proficient people (e.g., individual development), (2) orientation towards tacit knowledge (e.g., emergent solutions), (3) knowledge transfer through conversation (e.g., dialogue and discussion), (4) channeling of expertise (e.g., access to experts), and (5) focus on effectiveness (e.g., through flexibility) (Wendorff & Apshvalka, 2005). The main point why agile organizations need new KM approaches (and tools) are the lack of time and the rejection of formal processes by the developers.

Teams and organizations developing with agile methods need automated tools and techniques that support their work without consuming much time. Therefore, *agile software reuse* is a fairly new area where minimally invasive techniques are researched to support software engineers (Cinneide et al., 2004).

Human-centered Design of semantically-enabled KM Systems

Important Challenges in Software Development

The development of complex software systems requires specific knowledge. The quality of the outcomes depends strongly on how successful existing knowledge and expertise of the employees can be provided and actually applied.

Software engineering in general aims at engineering-like development, maintenance, adjustment and advancement of complex software systems. One characteristic of software engineering compared with classical engineering lies in the prevalent immateriality and the complexity and abstractness of software resulting from it. For this reason experts point out the difficulty to gain vital working-experiences and stress deficits in the field of further training (Broy & Rombach, 2002). Although many companies apply (agile) methods of software engineering, the transfer of the underlying complex knowledge still represents a crucial problem.

Considering this challenging situation, the German research project RISE aimed at a holistic support of software engineering tasks. Software Engineers should be enabled to deal with their tasks and thus act more professional. Furthermore, the transfer of existing knowledge within software-related companies has to be optimized. This should be accomplished by systematic reuse of experiences, methods and models in different application contexts.

The vision of RISE is to develop a user-friendly knowledge management application for software developers that makes fun and requires minimum effort. Software developers are gently supported during teamwork and their re-use of working-experiences and knowledge. Consequently, the organization itself should profit from improved productivity increases.

Based on the stated special problems and situations, research also focuses on how high the impact of an Intranet-based platform could be, in order to ensure the individual capacity to act successfully through re-use of experiences, methods and models. Thus, RISE tries to find answers to the question what are main requirements for a knowledge management platform from the view of employees

- ...in order to keep complex knowledge manageable
- ...so that knowledge can be used individually and updated during work

It has to be clarified, in what respect a technical platform may allow an individually focused view of the information needed (knowledge management) and might facilitate (*usability*) use of it.

The participating companies already used wiki-systems (Leuf & Cunningham, 2001) as our project started. They provided initial data for the empirical study in the early project-phase. In our case, we had to deal with technically and functionally rather different wiki-systems, which can all be summarized under the wide term “Social Software”.

Such web-based content management systems allow users not only to access content but also to add own pieces of information or to edit already available artifacts with as little effort as possible.

The underlying rationale is based on a strongly decentralized way of editing content. Early results of the evaluation showed however that a sophisticated methodology is required to successfully implement “Social software” at the intranets of software-firms, in order to ensure a broad acceptance.

As observed, acceptance might decrease rapidly if the application system no longer represents a source of information which is usable as described within the international standard ISO 9241-11: “extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use” (ISO-9241-11, 1998). So, if the systems fail to be usable neither contents are published or edited nor even read in the worst case. The communication amongst software engineers will not be intensified – as intended – but stagnates to a total deadlock.

Understanding Context-of-use for KM Systems in Agile Organizations

It seems to be obvious that employees are responsible for an individual development of a vital knowledge basis. Not only that this requires special abilities, resources and strategies. Moreover, employees increasingly have to gather information and subsequently learn new things just besides their main work focus. These working-related information and knowledge gathering activities are conducted in peaks.

The main kind of peak being considered is caused by strict deadlines which temporarily raise the working load to a maximum. Besides those hard and rather ordinary kinds of peaks, a more elementary one is considered: the problem-orientation that many software engineers cling to. The term “problem-orientation” refers to the view that software development, viewed superficially, is a rather trivial process: developers are provided with certain requirements and try to realize them in the software development process.

But from time to time that “trivial” process gets disturbed by tricky problems developers stuck in. Those hard problems represent another kind of peak, which has fatal consequences for knowledge management in software companies: developers which run into that peak are often highly motivated to solve the problem and try to solve it on their own. They spend hours and hours with their problem and often don't bother about existing and proven solutions, concepts or methods. In this case, re-use does not exceed accidental or sporadic access to available knowledge in available knowledge repositories (e.g. internet forums). Considering that the knowledge collected by means of a content management system, it seems to be almost impossible to build a shared repository. The result would be a collection of various problems (hopefully including a solution) which might resemble each other more or less. The so called “Experience Factory” approach has to be mentioned which is considered as one answer to that challenge: “the Experience Factory approach was initially designed for software organizations and takes into account the software discipline's experimental, evolutionary, and non-repetitive characteristics“ (Basili et al. 2001, p. 1). Basili et al. propose that the company has to release its experts from the burden of managing the knowledge: “Experts in the organization have useful experience, but sharing experience consumes experts' time. The organization needs to systematically elicit and store experts' experience and make it available in order to unload the experts” ((Basili et al., 2001), p. 2).

Accordingly, employees with huge expertise are no longer considered like a means to an end but rather the core of the technical knowledge management system with the following characteristics:

- *First*, this application system acts as a flexible and customizable view (like a window) on the information needed during working-time. This system strongly relieves software engineers from time-consuming retrieval activities.
- *Second*, the system supports the collection and storage of vital working experience which has to be subsequently formed and refined.

These are the two main factors of the RISE methodology, which were optimized during project time.

Riki – A Semantically Enabled Knowledge Management System

We approached the before mentioned challenges by developing an enhanced Wiki-centered Framework (Louridas, 2006) that was targeted to act as a knowledge sharing platform for software development teams with fast and liberal access to deposit, mature, and reuse experiences made in software projects.

The content in the Riki (Reuse-oriented Wiki) consists of information on products (requirements, designs, documentations, component interactions, presentations, demo systems), projects (use cases, user stories, test cases, ideas), contacts (employees, customers, partners, suppliers), as well as individual blogs of the employees about software technologies, software development, etc. Furthermore, we extended it by two main components:

- First, the Riki was extended by search technologies using case-based reasoning technology and ontologies to provide a formal and consistent framework for the description of knowledge and experiences. As a consequence, software developers are now not only able to search the content of Wiki pages, but are also able to find files within other information repositories such as the shared file system.
- Second, the Riki contains a unified content system that allows software developers to annotate any type of content with so-called “tags”. Tags have become very popular with the rise of Web 2.0 and various new applications are adopting this principle. Using the Riki tags can be easily attached to any type of content. In our project, we decided to use shared tags: once a tag has been attached, it can be seen and modified by all users. This solution was favored in order to support a common vocabulary about artifacts within software development teams. Although this has been considered as being crucial for successful development processes, employees still have the opportunity to use their own “language” by using private tags.

Ontologies and templates enrich the Riki content with semantics that enable us to didactically augment the knowledge within the Riki with additional information and documented experiences. Additionally, the usage of metadata enables the users to build up and use their own individual ontology (in the form of individual tags) that is not bound to compromises or constraints from universal ontologies that might have been constructed in advance. Furthermore, metadata from the universal ontology (i.e., specified beforehand or as defined by other users) was partially used in their own ontology.

Within the Riki, the results are clustered by this metadata and the metadata can be used to refine search queries. Annotated pages that were listed in search results remind the

users that they have already read them or that they are highly valuable and should be read again.

Evaluation of semantically-enabled KM Systems

The objective of an explorative evaluation of a KM system such as the Riki is to show its effectiveness in a specific application context. In our evaluation the tailored instances of the Riki system were evaluated to identify the usefulness to the users, the examined applicability, the evolvability, as well as economic factors. As mentioned previously two companies participated in the evaluation: empolis GmbH – an enterprise with a development team of 40 developers and brainbot technologies AG – a micro-enterprise with 5 developers.

While we mainly conducted an explorative evaluation about the use of knowledge in agile organizations without specific hypotheses, one very important hypothesis of the RISE project was that software development teams collaboratively using a Riki, are actually about to create a semantic structure – with users not necessarily to be aware of. This structure called “Wikilogy” can be considered as a weakly formalized ontology with three main characteristics: a) the *maintenance of content and metadata (i.e., the ontology)* is not considered as extra activity, b) *ontologies* (as seen/build by the users) may always grow behind the content, and c) maintenance of these Ontologies can be *supported efficiently*.

The evaluation itself was split into two *phases*. The first phase – called *baseline evaluation* – at the start of the project was purely evaluative and used to elicit the context and current state of knowledge transfer. From the information we gathered in this phase we designed and developed the Riki system considering the people, processes, and available technologies. The second phase – called *delta evaluation* – at the end of the project helped to measure the effect of the introduced Riki system. The baseline and delta evaluations have been carried out with the same project managers, who represented small teams of five to twenty people, and software developers of the companies. These people used the Riki system during real projects within their companies.

- Goal-oriented, questionnaire-based *interviews* were used to query the previously listed questions with three to ten persons in two to four sessions. The collected answers were summarized and validated by the participants via email.
- *Group discussions* were done at every company to collect any additional information, opinions, ideas, etc. that were not covered by the interviews. The discussion was started with a specific topic (e.g., why had the old Wiki not worked for you as a KM system?) and every person could state what they expected from an improved knowledge management infrastructure.
- *Artifact analyses* were conducted to identify knowledge sources, the type of knowledge within, as well as how the people structure their documents and knowledge in existing storage systems (e.g., the hierarchy of directories in personal file systems or in pre-existing Wiki systems).

These evaluation techniques helped to cover the evaluation of the following three topics:

- *Technology*: Elicitation of the existence and characteristics of the technical infrastructure, existing KM systems, and other software systems that might be integrated into or used as the KM system (i.e., the Riki system). Furthermore, these technological systems potentially have valuable information that can be utilized in a KM system.
- *Methodology*: Elicitation of the applied methodology for production (e.g., software development) and knowledge management (esp. knowledge transfer processes). This gives further information about how the KM system should be integrated into the social system of the organization and where, when and by whom knowledge is produced or consumed.
- *Knowledge*: Elicitation of the existing knowledge components available in the organization as well as their characteristics and interrelation (e.g., for the development of an ontology). Furthermore, the typical structure of documents that might be didactical enriched.

A more detailed description as well as some results of the two evaluation phases are described in this the following sections. They address persons, who want to evaluate a KM system such as the Riki in an organization.

Baseline Evaluation

The baseline evaluation is concerned with the determination of the organizational context a socio-technical knowledge management should be embedded in. The core goal of the baseline evaluation is to measure and analyze the current status of the implicitly or explicitly performed KM processes, the used knowledge carrying or KM systems as well as the knowledge culture itself. Baseline evaluations are typically applied only once to get a consistent view of the KM in the organization before larger changes. This section describes the basic process for the evaluation of a KM system as well as a summary of our baseline evaluation.

Baseline Evaluation process

The baseline evaluation process (and similarly the delta evaluation) is structured into three sub-phases problem-determination, context-determination, and knowledge-determination. The baseline evaluation was used to capture the working process before the intervention (i.e., the introduction of a Riki). As depicted in Figure 5 the steps in these sub-phases define a process that results in several documents (e.g., a problem description) usable for the specification of a socio-technical KM system integrated into the surrounding organizational context.

- *Problem-Determination*: The problem determination serves to identify existing knowledge sources (e.g., in the case of the partner empolis a Wiki system named MASE) as well as emerged problems and challenges with it. Step 0a is used to determine existing information systems and technical knowledge sources using a systematic analysis method. By means of a group discussion in step 0b the exchange of knowledge via existing technical (KM) systems in a typical project team is illuminated.
- *Context-Determination*: This sub-phase is concerned with the determination of the context the socio-technical KM system is embedded in. The context determination produces information about the production processes, roles, documents, or sites that

might be used in the KM system. The three steps in this sub-phase are concerned with the development processes which are to be supported by the KM initiative. Step 1a uses a group discussion technique to determine existing documents, templates, and other potentially reusable elements. To elicit or update the development process in use we applied an interview with several product and project managers in step 1b.

- *Knowledge-Determination*: The knowledge determination sub-phase targets the core information that should be made reusable via the KM system. The content, context, and structure of reusable elements is determined in step 2a using an interview. Based on this information the knowledge transfer processes and the knowledge culture itself is analyzed in step 2b using a group discussion with several members of project teams.

Finally, step 3 is used to identify the technical infrastructure of tools and systems not used for knowledge management but might be integrated and connected with the new or improved KM system (i.e., a new Riki or an improved Riki).

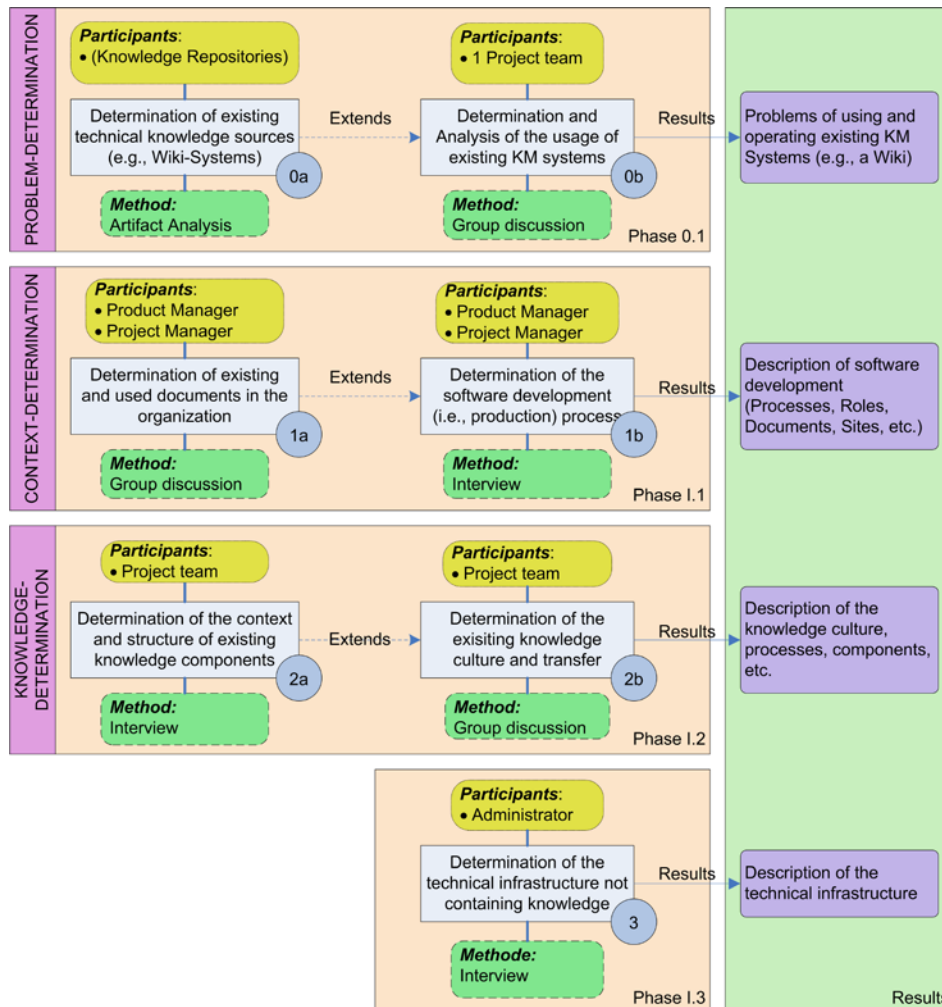


Figure 5. Plan for Baseline and Delta Evaluation

Results from the Baseline Evaluation

In the context of the RISE project we applied step 0a to step 3 of the evaluation plan as depicted in Figure 5. For the execution of the baseline evaluation one period we required a time -span of approx. 3 months taken into account that not all employees are continuously available. Based on different vacation planning, holidays in summer, autumn, and Christmas as well as other projects one should take into account more time during such an evaluation.

During the group discussion concerning the usage of existing KM systems (cf. Figure 5, 0b) as well as the knowledge culture and transfer (cf. Figure 5, 2b) we determined the following status in the organization:

- The organization used a Wiki that few people used and contained partially documented concepts, ideas, comments, and larger documentations about planned

as well as already developed software systems. Other information in the Wiki included: Customer information, dates, To-do-lists, addresses, and absent lists.

- Important information about the software system itself is documented in the source code (in this case: Python and ePyDoc (ePyDoc, 2005)). To extract this information one has to check-out the whole software system from the versioning system and search for relevant information on file level.
- Important information about changes to the software system is documented in the used versioning system (in this case: Subversion). Changes at the software system are sent to all interested users by email. To extract this information without the eMails one has to analyze every check-in into the system and the appropriate comments.
- Internal information sources (i.e., Repositories) with other project-relevant information are: Change tracking system, project folders, universal and private network drives, eMails, and chat tools (e.g. ICQ) with information in files such as plain text or MS Word documents. Furthermore, task cards at a physical blackboard.
- External information sources are distributed over the whole internet but the employees had a focus on MSDN and Google if they required further information.

Positive Characteristics: What runs well?

- The co-operation between two people closely working together, but who were distributed physically over two cities, worked very well over the old Wiki.
- The local teams worked on one floor and had only short distances to their colleagues. The face-to-face communication was very good (i.e. everybody was in “shouting” distance).

Negative Characteristics: What runs badly?

- The Wiki was not used anymore at the start of the evaluation and the feedback indicates that most people were not pleased with the structure of the knowledge base.
- Neither the old WIKI nor the documentation language for Python (ePyDoc) permitted the integration of pictures or graphics.
- The search for information in the WIKI and file system is term-based (i.e., no stemming or Boolean operators). This was perceived as insufficient and demotivating by the users. Furthermore, the search in the Wiki was impeded by the use of camel case (e.g., “MyProjectDescription”) in the page names.
- The discovery of relevant information is perceived as complicated as they are distributed over multiple repositories that all use different (or none) search mechanisms.
- Access to the Wiki from a text editor (e.g. emacs under Linux/Unix) or a shell (i.e. command line interface) is impossible and/or uncomfortable. Nevertheless, some developer are bias or required to use these and are not willing to install or use other operating systems (or web browsers such as internet explorer™).
- Neither were the authors (resp. other observers) informed about changes done to the content in the Wiki nor were they informed about changes to the navigational or divisional structures (i.e., chapters and sections) of the content.

- It was not clear were to store information as they could be spread or duplicated in multiple repositories, for example, in the versioning system, change tracking system, or the code itself.

In summary, the knowledge transfer and management processes as they were lived in the organization previously to the Riki introduction were determined by the baseline evaluation as follows:

- *Recording* of information is limited to few people in the organization and the documented information is only partially complete, consistent, or valid. The informal storage, e.g., of user requirements, opens the door for inconsistent, incomplete, and generally low-quality documents. Nevertheless, most users embraced the Wiki idea and recorded even preliminary information that is revised and improved by themselves or other users over time. In order to improve this quality, approach such as incentive systems (Feurstein et al., 2001) can be used to motivate the sharing as well as improvement of knowledge. Paul Duguid examined the “laws of quality” and found that peer-produced projects constantly change: “What is flawed today may be flawless tomorrow” (Duguid, 2006). However, during the RISE project, we could not address this topic thoroughly enough. But our findings indicate that the overall quality or value of an information space is judged by the users themselves and was acceptable. The users were able to comment about artifacts and incomprehensible, out-of-date, misleading, or unhelpful documents were often directly improved or commented.
- *Reuse* of content is minimal as the information is distributed over several sources with different search interfaces and techniques. Furthermore the content of the documents have inconsistent structures, incomplete descriptions, or are plain outdated.
- *Workflow* for reuse of content and getting an *overview* is slow and typically demotivating as multiple sources have to be searched manually and documents belonging together are not grouped or linked.
- *Sharing knowledge* is cumbersome, there are no templates, guidelines, or checklists to validate if the recorded information has some quality and might be easily reused by the colleagues.
- *Confidence* in the knowledge transfer system and *motivation* to share is low, as only few people are creating shareable documents and the documents are sometimes not accurate or up-to-date. Nevertheless the participants stated that they would like to share their knowledge in a more persistent way.
- *Face-to-face communication* is strong esp. as most employees have short distances to their colleagues, are roughly of the same age, and see not need to hide their information (i.e., egghead’s syndrome).

Delta Evaluation

The core goal of a delta evaluation is to measure and analyze the changes a KM system has inflicted on the affected organization. Delta evaluations can be applied multiple times during the lifecycle of a KM system (e.g., every year) to evaluate the effect on the organization.

Delta Evaluation types

Since KM systems are usually used only irregularly in the beginning and not yet a firm part of the working process, the delta evaluation phase can be applied in three different types:

1. *Applicability*: The first type serves to examine how frequently the system is used, if it integrates into the socio-technical infrastructure and if it helps the users in their daily work. In order to keep the costs of the evaluation down to a minimum, a light-weight evaluation is planned by reusing existing plans and other technology assessment or acceptance models.
2. *Usefulness*: In the second type the results and experiences from the first type serve to concretize the requirements and improvement goals. The system has already demonstrated that it is applicable in the organization and represents an integral component of the regular work routine. By means of structured evaluation processes a formal evaluation plan is constructed to examine, in particular, the aspect of the usefulness, ease-of-use, and usability for the direct users and management.
3. *Economy*: Finally the third type serves to examine the economical aspects of the KM system. The system is already integrated into the regular work routine and users share and reuse the knowledge within. An adaptive evaluation, reaction, and risk plan are developed to continuously monitor and improve the system. The focus of this type is the evaluation of the system and forecast of the usefulness regarding the Return on Investment (ROI) and Total Cost of Ownership (TCO).

To identify existing or potential problems the entire system is continuously monitored during the start-up phase. Log-files help to drill-down into specific problems if the users observe a problem while using the system.

Results from the Delta Evaluation

In the context of the RISE project we could only apply the first sub-phase and evaluate the applicability of the Riki system. Due to the short amount of time the system was used (2 month) we could only get little insight into the usefulness of the system for the users in their daily routine. During the group discussion concerning the usage of the Riki (cf. Figure 5, 0b) as well as knowledge culture and transfer (cf. Figure 5, 2b) we noted the following characteristics the users liked very much about the Riki:

- Metadata elements that can be placed by every user (such as keywords in form of tags) can be very helpful in indexing the content of a Riki and the users reacted very positively that they were able to *index every page* with their own metadata.
- The usage of metadata enabled the users to build up and use their own *individual ontology* (in form of individual tags) that is not bound to compromises or constraints from universal ontologies that might have been constructed in advance. Furthermore, metadata from the universal ontology (i.e., specified beforehand or as defined by other users) was partially used in their own ontology.
- *Searching* the information stored in the Riki is more accepted by the users when the metadata might be integrated into the search process. In the Riki the results are *clustered* by this metadata and the metadata might be used to *refine* the search query. The search technology exploit co-occurring metadata from multiple users

and applies “collaborative filtering” techniques (i.e., “metadata x you search for is also called y”).

- Annotated pages that were listed in search results reminded the users that they already read them or were highly valuable and should be read again. Similar to the Memex concept by Vannevar Bush (Bush, 1945) (cf. http://en.wikipedia.org/wiki/Vannevar_Bush) the metadata might even be used to record a reading sequence for oneself.
- The integrated view a Riki gives over the information in the organization enables the user to send links that are not subject to change as, for example, mounted devices in the windows file system.

In comparison to the status as determined by the baseline evaluation the usage of the Riki system had the following subjective effect on the organization:

- *More recording* of information into the Riki than before as barriers (technological and social) are reduced.
- *More reuse* of content from the Riki as users are more likely to share and search for content.
- *Faster workflow* for reuse of content and an *improved overview* due to the integrative view over multiple systems (e.g., file system, eMail, etc.) in a central integrated repository.
- *Less barriers for sharing knowledge* as it easier and faster to enter information but this typically results in a *low quality* of the content. Most users embrace the Wiki idea and record even preliminary information that is revised by oneself or others over time.
- *Higher confidence* in the system and *increased motivation* as content is easier to find and more people are participating in the sharing process.
- *Consistent face-to-face communication* even as more information is reused from the technical system.

Further and more quantitative results about the usefulness and economical aspects esp. of the more SE specific features of a Riki such as ontology-based templates for requirements will be elicited in later evaluations.

Evaluation of Design Solutions against Requirements

The evaluation of the design solutions was dominated by qualitative methods and revealed valuable insights about requirements of users and organizations. Two focus groups were dedicated to explore usage problems with the knowledge management platforms introduced in earlier stages. The gathered qualitative data is characterized by high validity, which means that a common understanding of user requirements was developed and the involved users agreed which user requirements should be realized in favor (DA-Tech 2004, P. 76).

In order to classify the collected data, the so called “Munich knowledge management model” has been used with its four categories: knowledge-representation, knowledge-usage, knowledge-communication and knowledge-creation (Reinmann-Rothmeier 2000, pp. 18).

Knowledge-Representation: Some employees tend to write down a minimum of information. It was observed that some of them document just as little as possible so

that they will be able to find out later what was meant by the given artifact. The main disadvantage of that behavior is that the saved data becomes hard or even almost impossible to understand for other employees accessing that content because they don't have the necessary knowledge to »decode« and to classify it. However, it seems to be a kind of game those developers are playing when doing some documentation: they rather puzzle together pieces of information than writing down supposedly redundant or superfluous information.

Use of Knowledge: Crucial information about projects, products or customers is distributed across various databases. This means, that useful knowledge is hard to find and thus hard to be transferred in other contexts like new projects for example.

Communication about Knowledge: Employees who browse or search through wiki-content in order to find some special piece of information have to spend a lot of time. Moreover, editing content in most Wikis is rather uncomfortable: for example, adding a chart, a table or even a picture to wiki pages requires a special handling if it is not even impossible. As a consequence, information is being transferred by mail under pressure of time and not added to the wiki-site.

Creation of Knowledge: With increasing activity in a wiki, it becomes harder for authors to integrate their piece of information into the existing structure. So, it is very likely that no new content is being added and already saved content gets orphaned.

Basically, using metadata is a good idea to keep heterogeneous and complex structures describable and thus technically manageable. But the problem is that users often fail to provide useful metadata or simply don't bother about complex sets of metadata.

Discussion

During the RISE project, qualitative evaluation activities have been preferred, because they helped to specify the context of use as well as user and organizational requirements. In general, empirical data gathered by means of qualitative research methods is characterized by its high validity. Unfortunately, it is rather difficult to extract requirements out of those data or to check to what extent those "implicit" requirements have been considered during the development process of the software system.

Following user-centered design approaches that problem could not be bypassed. It is rather necessary to provide prototypes as soon as possible which contain as much functionality as necessary in order to check current implementations against more or less "implicit" requirements.

In terms of a rather comprehensive meaning of *usability* it was not only about developing user interfaces and improving them. In fact, usability engineering is considered as a vital concept in software development. So, it is not only about to describe, design or produce design solutions for user interfaces. It is rather about to understand and facilitate human-engineered socio-technical systems (DATEch, p. 77).

Conclusion

One very important hypothesis of the RISE project was, that software development teams that use a wiki, collaboratively, to build knowledge repositories are actually about

to **create a semantic structure** – with users not necessarily to be aware of. This structure called “Wikilogy” can be considered as a weakly formalized ontology. This user-generated content and ontology has three main advantages.

- The *maintenance of content and metadata (i.e., the ontology)* is not considered as extra activity.
- *Ontologies* (as seen/build by the users) may always grow behind the content.
- Maintenance of these Ontologies can be *supported efficiently*

Nevertheless, the quality of the ontology – as well as the content – depends always on the expertise and knowledge of the contributors. Therefore, similar to Wikipedia or the Experience Factory system special administrators are required that inspect, assess, and correct (i.e., “refactor” (Rech et al., 2007a)) the ontology and content from time to time.

Another crucial approach of the RISE project is the **use of tags** as a very minimalistic but rather effective way of providing metadata. Any tags may be assigned individually to content-pages in the Riki. So, each user is able to create his view on any artifact by using his own depiction. Since in our approach all users are able to see the tags provided by the others, it is of course very probable that teams members manage to create a shared set of tags as well. Later these tags can be analyzed (e.g., using data mining techniques) in order to automatically extract or semi-automatically build ontologies for specific topics (e.g., software engineering or specific software projects)

Furthermore, **templates** for special types of content (bug report, use case, user story etc.) help to provide a minimum of required information. Readers as “consumers” of the artifact can be satisfied, too. For example, the given structure helps them to browse faster through the content in order to find only some pieces of information being of interest. Finally, this results in a higher acceptance of the system which motivates the employees of adding more content to the repository.

Additionally, a **blog-component** was added to the system. The blog contains not only news. In fact, the blog can be considered as an active filter on the wiki-content. For example users might link to a certain Riki-page and provide some extra information along with the link. So, other users get useful assistance with the evaluation of artifacts.

Acknowledgements

Our work is part of the project RISE (Reuse in Software Engineering), funded by the German Ministry of education and science (BMBF) grant number 01ISC13D.

References

Althoff, K.-D., Decker, B., Hartkopf, S., Jedlitschka, A., Nick, M., & Rech, J. (2001, 24.-25. Juli 2001). Experience Management: The Fraunhofer Institute for Experimental Software Engineering (IESE) Experience Factory. Paper presented at the Industrial Conference Data Mining: Data Mining, Data Warehouse and Knowledge Management, Leipzig, Germany.

Ambler, S. (2002). Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process: John Wiley & Sons, Inc.

- Barry, B., & Richard, T. (2003). Using Risk to Balance Agile and Plan-Driven Methods. *Computer*, 36(6), pp. 57-66,
- Basili, V. R., Caldiera, G., & Rombach, H. D. (1994). Experience Factory. In J. J. Marciniak (Ed.), *Encyclopedia of Software Engineering* (Vol. 1, pp. 469-476). New York: John Wiley & Sons.
- Basili, V. R., Lindvall, M., & Costa, P. (2001, June 13-15, 2001). Implementing the Experience Factory concepts as a set of Experience Bases. Paper presented at the Proc. 13th Int. Conf. on Software Engineering & Knowledge Engineering, Buenos Aires, Argentina.
- Beck, K. (1999a). *Extreme programming explained: embrace change*. Harlow: Addison-Wesley.
- Beck, K. (1999b). *eXtreme Programming eXplained: Embrace Change*. Reading: Addison-Wesley.
- Beck, K., & Fowler, M. (1999). Bad Smells in Code. In G. Booch, I. Jacobson & J. Rumbaugh (Eds.), *Refactoring: Improving the Design of Existing Code* (1st ed., pp. 75-88): Addison-Wesley Object Technology Series.
- Begel, A., & Nagappan, N. (2008). *Pair programming: what's in it for me?* Kaiserslautern, Germany: ACM.
- Birk, A., Dingsøyr, T., & Stålhane, T. (2002). Postmortem: Never Leave a Project without It. *IEEE Software*, 19(3), pp. 43-45,
- Broy, M., & Rombach, H. D. (2002). Software Engineering. Wurzeln, Stand und Perspektiven. *Informatik-Spektrum*, 25(6), pp. 438-451,
- Bush, V. (1945). As We May Think. *The Atlantic Online*, 176, pp. 101-108, (Reprinted in: *ACM interactions*, Volume 3, No. 2, pp. 35-46, 1996, ISSN:1072-5520).
- Chau, T., Maurer, F., & Melnik, G. (2003). *Knowledge Sharing: Agile Methods vs. Tayloristic Methods*: IEEE Computer Society.
- Cho, J. (2008). Issues and Challenges of Agile Software Development with Scrum. *Issues in Information Systems*, IX(2), pp. 188-195,
- Cinneide, M. O., Kushmerick, N., & Veale, T. (2004, July 2004). *Automated Support for Agile Software Reuse*. ERCIM News.
- Cockburn, A. (2006). *Agile Software Development: The Cooperative Game* (2nd Edition) (Agile Software Development Series): Addison-Wesley Professional.
- Decker, B., Ras, E., Rech, J., Klein, B., Reuschling, C., Höcht, C., et al. (2005). A Framework for Agile Reuse in Software Engineering using Wiki Technology. Paper presented at the Conference Professional Knowledge Management - Experiences and Visions., Kaiserslautern.
- Duguid, P. (2006). Limits of self-organization: Peer production and 'laws of quality'. *First Monday*, 11(10),

ePyDoc. (2005). Epydoc website. Retrieved 5 October, 2005, from <http://epydoc.sourceforge.net/>

Feurstein, M., Natter, M., Mild, A., & Taudes, A. (2001). Incentives to share knowledge. Proceedings of Hawaii International Conference on System Sciences. HICSS 34, Maui, HI, USA, 3 6 Jan. 2001 * Los Alamitos, CA, USA: IEEE Comput. Soc, 2001, p 8 pp.,

Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing. International Journal of Human Computer Studies, UK * vol 43 (Nov. Dec. 1995), no 5 6, p 907 28, 56 refs.,

Höcht, C., & Rech, J. (2006). Human-centered Design of a Semantically Enabled Knowledge Management System for Agile Software Engineering. In M. D. Lytras & A. Naeve (Eds.), Open Source for Knowledge and Learning Management: Strategies beyond Tools: IDEA Group Publishing.

ISO-9241-11. (1998). Ergonomic requirements for office work with visual display terminals (VDTs) -- Part 11: Guidance on usability (Standard No. ISO 9241-11:1998(E), ISO TC 159/SC 4/WG 5): ISO (the International Organization for Standardization).

ISO-13407. (1999). Human-centred design processes for interactive systems (Standard): ISO (International Organization for Standardization).

James A. Highsmith, III. (2000). Adaptive software development: a collaborative approach to managing complex systems: Dorset House Publishing Co., Inc.

Leuf, B., & Cunningham, W. (2001). The Wiki Way: Quick Collaboration on the Web: Addison-Wesley Professional.

Louridas, P. (2006). Using Wikis in Software Development. IEEE Software, 23(2), pp. 88-91, Software, IEEE.

Rech, J. (2007). Handling of Software Quality Defects in Agile Software Development. In I. Stamelos & P. Sfetsos (Eds.), Agile Software Development Quality Assurance: Idea Group Inc.

Rech, J., Decker, B., Ras, E., Jedlitschka, A., & Feldmann, R. L. (2007a). The Quality of Knowledge: Knowledge Patterns and Knowledge Refactorings. International Journal on Knowledge Management (IJKM), 3(3), pp. 74-103,

Rech, J., Ras, E., & Decker, B. (2007b). RIKI: A System for Knowledge Transfer and Reuse in Software Engineering Projects. In M. D. Lytras & A. Naeve (Eds.), Open Source for Knowledge and Learning Management: IDEA Group Publishing.

Schwaber, K., & Beedle, M. (2001). Agile Software Development with Scrum: Prentice Hall PTR.

Stapleton, J. (1999). DSDM: Dynamic Systems Development Method: IEEE Computer Society.

Teevan, J., Alvarado, C., Ackerman, M. S., & Karger, D. R. (2004). The perfect search engine is not enough: a study of orienteering behavior in directed search. Vienna, Austria: ACM.

Wendorff, P., & Apshvalka, D. (2005). The Knowledge Management Strategy of Agile Software Development.

Reinmann-Rothmeier, G. (2000): Wissen managen: Das Münchener Modell. http://www.wissensmanagement.net/download/muenchener_modell.pdf (letzter Abruf: 01.09.2005)

DATech (2004): DATech-Prüfungshandbuch. Usability-Engineering-Prozess. Leitfaden für die Evaluierung des Usability-Engineering-Prozesses bei der Herstellung und Pflege von interaktiven Systemen auf der Grundlage von DIN EN ISO 13407. Frankfurt a. M.: Deutsche Akkreditierungsstelle Technik e.V.