1 Riki: A System for Knowledge Transfer and Reuse in Software Engineering Projects

Inside Chapter

Many software organizations have a reputation for producing expensive, low-quality software systems. This results from the inherent complexity of software itself as well as the chaotic organization of developers building these systems. Therefore, we set a stage for software development based on social software for knowledge and learning management to support reuse in software engineering as well as knowledge sharing in and between projects. In the RISE (Reuse in Software Engineering) project, we worked with several German SMEs to develop a system for the reuse of software engineering products such as requirement documents. The methodology and technology developed in the RISE project makes it possible to share knowledge in the form of software artifacts, experiences, or best practices based on pedagogic approaches.

This chapter gives an overview of the reuse of knowledge and so-called Learning Components in software engineering projects and raises several requirements one should keep in mind when building such systems to support knowledge transfer and reuse.

1 Introduction

The software industry develops complex systems that often have a reputation of being expensive and of low quality. One approach for coping with the increasing complexity of these systems is software reuse – the sharing of knowledge about software products, processes, people, and projects in an organization.

But the poor and often non-existent documentation of this knowledge inhibits easy recording, collection, management, comprehension, and transfer. The knowledge, for example, in the form of requirement descriptions, architectural information, design decisions, or debugging experiences, needs a systematic, minimally-invasive, methodological, and technological basis to strengthen its reuse and transfer in software organizations.

Knowledge management (KM) and *learning management (LM)* seem to have the potential for building this basis if they are used in synergy. However, the relationships between these two promising fields have not been fully understood and harnessed yet. On the one hand, learning as the comprehension of knowledge is considered to be a fundamental part of KM, as employees must internalize shared knowledge before they can use it to perform a specific task. So far, research within KM has addressed learning mostly as part of knowledge sharing processes and focused on specific forms of informal learning (e.g., learning in a community of practice) or on providing access to learning resources or experts. On the other hand, LM includes techniques to preprocess and formalize knowledge and might also benefit from other KM approaches. Especially

those approaches that support technical and organizational aspects in an organization can be used in addition to professional e-Learning systems.

In this intersection between KM and LM, the Wiki technology (cf. http://en.wikipedia.org/wiki/Wiki) promises to be a lightweight basis to capture, organize, and distribute knowledge that is produced and used in organizations. Wikis are web-based knowledge repositories where every user can freely submit or retrieve knowledge.

The *RISE (Reuse In Software Engineering)* project was started to support software engineers via the reuse of didactically enriched software artifacts from all software development phases in SMEs (Small and Medium Enterprises). As the basis for knowledge transfer, we have developed a reuse-oriented Wiki (Riki) with fast and liberal access to deposit, mature, and reuse experiences made in software projects. Our 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. 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.

The RISE system sketches our approach for software reuse and tackles several problems of traditional KM and LM in learning software organizations. We use *semi-automatic indexing of pages* to improve retrieval and enable the semi-automatic creation and evolution of ontologies from Wikis (i.e., Wikitologies). The *cooperative adaptation* of knowledge to community needs, and the *didactic augmentation* of content and interface are targeted to improve the usability of lightweight KM applications in agile environments.

In this chapter, we give an overview of the methodology and technology developed in the RISE project to build the reuse-oriented Wiki framework named Riki. As depicted in Figure 1, we describe in section 2 the relevant background about knowledge and learning management as well as software engineering and Wikis. This is followed by section 3 containing the targeted problems and objectives examined in the RISE project.

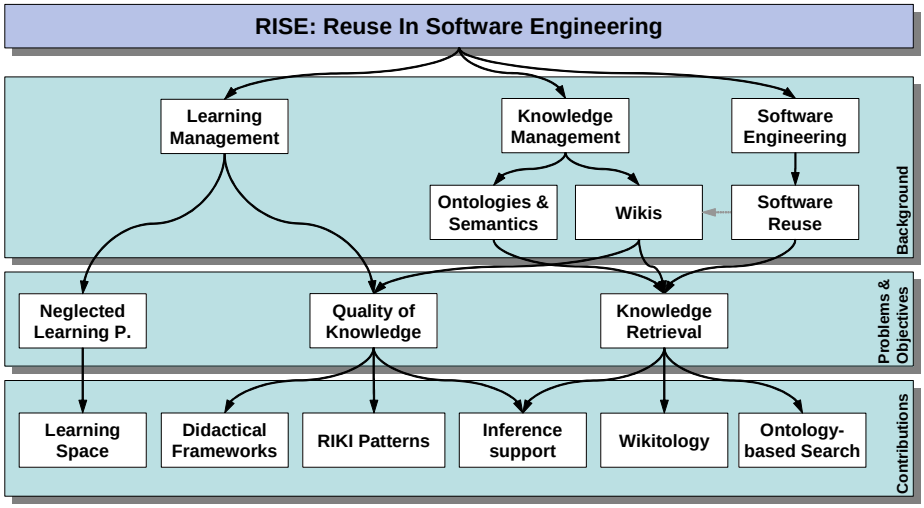

Figure 1. Outline and structure of the following chapter

Our main contribution for knowledge transfer and reuse in software engineering is summarized in section 4, which contains both the methodology in section 4.2 and the technology in section 4.3. Furthermore, several best practices, Wiki patterns, and lessons learned gathered during the project are listed in section 5. Finally, we conclude this chapter in section 5.

2 Background

This section is concerned with the background and related work in knowledge management with Wikis, problem-based and experiential e-Learning in Wiki systems, and reuse for software development artifacts. It gives an overview of related Wikis for SE, for example TRAC, Snip Snap, and MASE.

2.1 Software Engineering and Reusable Knowledge

The discipline of *software engineering (SE)* was born in 1968 at the NATO conference in Garmisch-Partenkirchen, Germany (Naur & Randell, 1968; Simons et al., 2003). At the same conference, the methodic reuse of software components was motivated by Dough McIllroy (McIllroy, 1968) to improve the quality of large software systems by reusing small, high-quality components. The main concern of software engineering is the efficient and effective development of high-qualitative and very large software systems. The objective is to support software engineers to develop better software faster with tools and methods.

2.1.1 Software Reuse

The reuse of existing knowledge and experience is a fundamental practice 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, engineers have to rebuild and relearn these components, methods, or tools again and again.

Today, *reuse-oriented software engineering* covers the process of development and evolution of software systems by reusing existing software artifacts. The goal is to develop complex software systems in shorter periods of time or with higher quality by reusing proven, verified, and tested components from internal or external sources. Trough systematic reuse of these components and feedback about their application, their internal quality (e.g., reliability) is continuously improved. But reuse of components is only appropriate if the cost of retrieving and adapting the component is either less costly or results in higher quality than a redeveloped component.

Since the 1980s, the systematic reuse and management of experiences, knowledge, products, and processes was refined and named *Experience Factory* (EF) (Basili et al., 1994). This field, also known as *Experience Management* (Jedlitschka et al., 2002) or *Learning Software Organization* (LSO) (Ruhe & Bomarius, 1999), researches methods and techniques for the management, elicitation, and adaptation of reusable artifacts from SE projects. The *Component Factory* (CF) as a specialization of the EF is concerned with the reuse of software artifacts (Basili et al., 1992) and builds the framework in which further analysis and retrieval techniques are embedded.

In the beginning, only the reuse of source code was the focus of reuse-oriented software engineering. Today, the comprehensive reuse of all software artifacts and experiences from the software development process is increasing in popularity (Basili & Rombach, 1991). Besides source code artifacts such as requirements, design document, test cases, process models, quality models, and best practices (e.g., Design Patterns) are used to support the development and evolution of software systems. These artifacts are collected during development or reengineering processes and typically stored in special artifact-specific repositories.

2.1.2 Software Engineering Experience and Knowledge

The terms knowledge, information, and experience are defined in multiple, more or less formal, and often contradictory ways. Models that define these terms and the processes that transit from one to another differentiate between tacit and implicit knowledge (Nonaka & Takeuchi, 1995) or between data, information, knowledge, ability, capability, and competence (North, 2002). There are positions such as by Stenmark (Stenmark, 2001) that consider the usage of the term “knowledge” for information stored in a computer inappropriate. In this model, tacit knowledge can, in fact, exist only in the heads of people, and explicit knowledge is actually information. However, the terminology used in the theory and practice of Knowledge-based Systems (KBS) and Knowledge-Discovery in Databases (KDD) considers knowledge to be information stored together with its context, and we follow this convention throughout this paper.

2.1.2.1 Types of Knowledge

More specifically we base our view of knowledge on the model of the architecture of human knowledge developed by Anderson. He classified knowledge not according to its content but according to its state in the person’s long-term memory. Two types of knowledge were defined (Anderson, 1993; Gagné et al., 1988):

- *Declarative knowledge* consists of 'knowing about' – e.g., facts, impressions, lists, objects and procedures, and 'knowing that' certain principles hold. Declarative knowledge is based on concepts that are connected by a set of relations forming a network that models the memory of a person. For instance, declarative knowledge

items in the domain of software engineering might be: a definition of ‘test case’, a listing of defect types, a detailed explanation of key testing principles.

- *Procedural knowledge* consists of 'knowing how' to do something, i.e., skills to construct, connect and use declarative knowledge. It contains the discrete steps or actions to be taken, and the available alternatives to perform a given task. For instance, procedural knowledge items in the domain of software engineering might be: a method for deriving test cases from requirements, a method for classifying defects choosing the right reading technique to perform an inspection.

Both declarative and procedural knowledge can be abstract or concrete. The knowledge can be connected to more or less concrete information that can be described technically, e.g., by semantic networks. Nevertheless, knowledge about situations experienced or about evaluating facts or determining circumstances in given situations cannot be classified as declarative or procedural knowledge. Therefore, a third form of knowledge has extended the spectrum of knowledge in cognitive science (Enns, 1993) when Tennyson and Rasch (Tennyson & Rasch, 1988) defined contextual knowledge as another type of knowledge:

- *Contextual knowledge* consists of 'knowing when, where and why' to use or apply declarative or procedural knowledge. Contextual knowledge is created by reflecting on the usage of declarative and procedural knowledge in practice in different contexts. Contextual knowledge enables the individual to be aware of commonalities between situations, and of the appropriateness or applicability of principles or procedures in a new context.

In summary, three types of knowledge have been presented. Documented and stored experiences consist mainly of contextual knowledge. They originate in most cases from expert memories and lack declarative basis background knowledge and detailed procedural knowledge, resulting in them being ineligible for learning purposes.

Furthermore, we identified the following *general artifacts* that are based on the six knowledge types from knowledge management (Mason, 2005):

- *Know-how*: Recorded information about how to do something. This can range from general *guidelines* about how to design a system to more specific and personal experiences about using a tool or software library.
- *Know-who*: Recorded information about a *person* (i.e., developer or maintainer). This can range from who designed a specific subsystem to who has knowledge about a customer.
- *Know-why*: Recorded *rationales* why something was done or decided. For example, why a design technique was applied or a defect ha not been removed from the system.
- *Know-what*: Recorded information about the *status* of something (e.g., the current situation). This includes information about the current status of a project or system that was elicited from the project manager.
- *Know-where*: Recorded information about the *location* of something (esp. knowledge). This includes where an algorithm is implemented in the source code as well as information about where to find the project plan or a process description.

- *Know-when*: Recorded information about the *situation* (e.g., time or version) when something happens or a process is applied. This can range from when defects are introduced into the system to when the quality assurance process should be applied.
- *Know-if*: Recorded information about the *consequences* of an action. This can range from what happens if a defect is removed to the effects of applying a specific quality assurance technique.
- *Know-that*: Recorded information about the *facts* of something. This can range from information about the characteristics of a quality defect to a model of the development process.

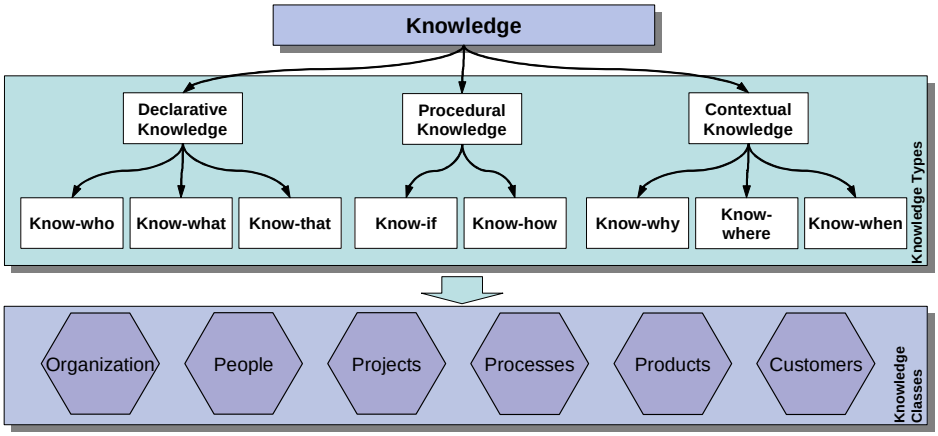

Figure 2. Types and characteristics of knowledge

The interconnections between these knowledge types are depicted in Figure 2. The organizational elements at the bottom of the picture are classes of information that are typically existent in (software) organizations. In a KM system, knowledge, or more specifically the know-how, know-who, etc. about the organization, products, projects, people, processes, customers, as well as further knowledge (e.g., about a technology such as Java or EJB) is stored. We refer to this group as the OP4C model (Organization, People, Processes, Projects, Products, and Customers).

2.1.3 Software Engineering Artifacts

Software engineering is concerned with the planning, management, design, development, and implementation of large-scale, complex software systems. During these processes, several documents, architectures, ideas, and experiences are created or made. In software reuse, the umbrella term *artifact* is used to describe explicit knowledge objects that are considered for reuse, such as:

- *Methods* and techniques from phases such as project planning, requirements analysis, or programming that describe the know-how.
- *Patterns* about the analysis and structure of software systems (i.e., analysis and design patterns) or the management of processes (i.e., process patterns). In general, these represent knowledge that was created by the aggregation of single experiences, either by human expert judgement or by methodological approaches (Basili et al., 2002).

- *Products* used in software projects such as project plans, product descriptions, demonstration installations, or presentation slides as well as products produced and consumed in these projects, such as requirement documentations, architectures, software libraries, or source code.
- *Models* about the software quality (i.e., quality models) or the execution of processes (i.e., process or lifecycle models) that were used in projects and adapted to the organization and the individual needs of the project and its team.
- *Defects* found in a software system or in a project that reduce its functionality, maintainability, or other aspects like performance. Associated information of this artifact is, for example, their location, characteristics, and reactive or preventive measures. If a specific type of defect is recorded over and over again, it might be summarized in a so called anti-pattern.
- *Descriptions* of the organization, people, projects, processes, products, and customers (O4PC) as well as technologies (e.g., EJB and JBoss), development rules, interface style guides, corporate quality assurance guidelines, or the corporate identity style.

2.2 Knowledge Management and Learning Management

Knowledge as the fourth factor of production is one of the most important assets for any kind of organization, and for all areas of science. Unfortunately, today few experts who have acquired valuable experience through their day-to-day work share this knowledge with other people in the organization. For example, experiences about how to solve complex problems in software development such as installation or optimization issues are typically not shared with colleagues.

2.2.1 Knowledge Management

Knowledge management is concerned with methods and technologies to enable an organization to record, inspect, adapt, and share knowledge cooperatively on a large scale. Recording, elicitation, and adaptation of knowledge are the most critical tasks in knowledge management with respect to further reuse of knowledge. The lower the quality of the knowledge is that is recorded (e.g., due to missing context information), the more complex it gets to understand and apply the knowledge in a new context. The participation of colleagues such as managers, employees, or external experts helps to record knowledge from multiple points of view.

In the domain of software engineering, software development can be considered as a human-based, knowledge-intensive activity. Together with sound methods, techniques, and tools, the success of a software project and the software quality itself strongly depends on the knowledge and experience brought to the project by its developers. This fact led to the development and use of experience-based information systems (EbIS) for capturing, storing, and transferring experience (Nick, 2005). Experience Management (EM) can be seen as a sub-field of KM that aims at supporting the management and transfer of relevant experiences (Bergmann, 2002; Tautz, 2001). The software system used for managing, storing, retrieving, and disseminating these experiences is called an Experience-based Information System (EbIS) (Jedlitschka & Nick, 2003), which is based on the Experience Factory concept. Another type of systems that are not based on the EF concept is Lessons Learned Systems (LLS) (Weber et al., 2001). Amongst the

definitions for lessons learned, the most complete definition as stated by Weber et al. (Weber et al., 2001) is: "A lesson learned is knowledge or understanding gained by experience. The experience may be positive, as in a successful test or mission, or negative, as in a mishap or failure. Successes are also considered sources of lessons learned. A lesson must be significant in that it has a real or assumed impact on operations; valid in that it is factually and technically correct; and applicable in that it identifies a specific design, process, or decision that reduces or eliminates the potential for failures and mishaps, or reinforces a positive result" (Secchi et al., 1999).

2.2.2 Learning Management

KM and learning management (LM) both serve the same purpose: facilitating learning and competence development of individuals, in projects, and in organizations. However, they follow two different perspectives. KM is related to an organizational perspective, because it addresses the lack of sharing knowledge among members of the organizations by encouraging individuals to make their knowledge explicit by creating Knowledge Elements, which can be stored in knowledge bases for later reuse or for participating in communities of practice. In contrast, e-Learning emphasizes an individual perspective, as it focuses on the individual acquisition of new knowledge and the socio-technical means to support this internalization process. In the following two sections, expectations on learning content and related specifications and standards that address these expectations are presented.

2.2.2.1 Expectations on Today's Learning Content

Especially in industrial training settings, learning objectives mostly correspond to concrete, well-defined job-related skills, specific tasks to be done, or problems to be solved. Hence, the delivered learning material and learning approach must suit the current situation that the learner finds himself in. The situation changes over time while the learner is performing his work. Nevertheless, conventional learning systems leave no space for dynamic selection and sequencing of learning material. In addition, the expectations on e-Learning content are high (cf. SCORM 2004 2nd Edition Overview page 1-22, http://www.adlnet.org/scorm/index.cfm):

- *Accessibility* – "the ability to locate and access instructional components from one remote location and deliver them to many other locations"
- *Adaptability* – "the ability to tailor instruction to individual and organizational needs"
- *Affordability* – "the ability to increase efficiency and productivity by reducing the time and costs involved in delivering instruction"
- *Durability* – "the ability to withstand technology evolution and changes without costly redesign, reconfiguration or recoding"
- *Interoperability* – "the ability to take instructional components developed in one location with one set of tools or platform and use them in another location with a different set of tools or platform"
- *Reusability* – "the flexibility to incorporate instructional components in multiple applications and contexts"

It is impossible to improve all aspects together. For example, in order to support high adaptability of learning content, more effort has to be spent on realizing adaptation

mechanisms and preparing learning content so that it can be adapted to specific learning situations. This will decrease the reusability of the content and its interoperability with other learning management systems. Therefore, tradeoffs have to be made by focusing on the most important aspects for a given purpose. Section 4 will elaborate on which aspects Riki will focus on. The requirement for content that fulfills the aims above leads to the concept of cutting learning material into so-called learning objects with associated metadata. Since then, many standards and specifications have been developed to offer a strong fundament for learning content with the previously listed characteristics.

2.2.2.2 E-Learning Standards and Specifications

Numerous initiatives like AICC (the Aviation Industry CBT Committee), ADL (Advanced Distributed Learning), IEEE LTSC (the Learning Technology Standards Committee of the IEEE) and IMS Global Learning Consortium have made efforts to establish standards. For several years, a number of initiatives have agreed to cooperate in the field of standards. Among the many available standards and specifications, the nes most relevant for Riki are described in the following.

The LTSC has developed the Learning Object Metadata Standard (LOM). This standard will specify the syntax and semantics of Learning Object Metadata, defined as the attributes required to fully/adequately describe a Learning Object. Learning Objects are defined here as “any entity, digital or non-digital, which can be used, re-used or referenced during technology supported learning”. A huge amount of specifications are being developed by the IMS consortium. Several of these specifications have been incorporated and in some cases been adapted by ADL to define the SCORM reference model. SCORM describes that technical framework by providing a harmonized set of guidelines, specifications, and standards based on the work of several distinct e-Learning specifications and standards bodies. These specifications have one aspect in common: by separating the content from the structure and layout, they enable the author to develop different variants of learning material very efficiently, while relying on the same set of learning objects. *SCORM Sequencing and Navigation* provides techniques to sequence learning objects by means of Learning Activity Trees, and the IMS Learning Design specification allows expressing more sophisticated pedagogical concepts by means of a more extensive role concept. However, they prescribe structures for expressing more or less generic instructional designs and do not provide possibilities for adapting instructional design during run-time. Nevertheless, despite their limitations, Riki will especially make use of specific concepts within SCORM (such as *Sharable Content Object, Organization, Manifest, Learning Activity Tree,* etc.) and parts of *Sequencing and Navigation*. Especially the contextualization of knowledge and experience (i.e., embedding the information in a context) plays a crucial role in Riki. Part of the context description could be done, for example, according to the IMS Learner Information Package specification in order to describing the content owner, or other content related roles. The Dublin Core Metadata Initiative (DCMI) has developed the *Dublin Core Metadata Element Set* outside of the e-Learning domain. DC defines a very simple set of metadata attributes that can be used for describing general resources. Since the content of a Riki should be annotated as unintrusive as possible, DC will be a starting point for describing RISE content.

2.3 Wikis and Semantics in Software Engineering

The semantic web initiative is concerned with the enrichment of information on the Internet trough the use of exchangeable and machine-readable metadata. To structure the knowledge in a "mini"-Internet as represented by a KM system such as a Wiki, we also planned to enrich the knowledge encoded on the Wiki pages with additional metadata. In the remainder of this section, we describe Wikis in general, followed by two overviews of Wikis for SE organizations and Wikis with integrated metadata support, which form the basis of our Riki system (i.e., semantic Wikis).

Wikis facilitate communication through a basic set of features, which allows the project team to coordinate their work in a flexible way. From the authors' point of view, these basic features are: *one place publishing*, meaning that there is only one version of a document available that is regarded as the current version; *simple and safe collaboration*, which refers to versioning and locking mechanisms that most Wikis provide; *easy linking*, meaning that documents within a Wiki can be linked by their title using a simple markup; *description on demand*, which means that links can be defined to pages that have not been created yet, but might be filled with content in the future. Furthermore, the simple mechanism of URL allows easy reference and thus traceability of Wiki content into other software documents like code. For a further discussion on Wiki features, refer to (Cunningham, 2005).

Besides those technical aspects, Wikis foster a mindset of a fit-for-use, evolutional approach to requirements documentation and management. The approach of Wikis – in particular, Wikipedia as the most prominent representative (Wikipedia, 2006) – demands that an initially created document is adequate for its intended usage (fit-for-use). This initial version is then extended based on the demand of the people using this document.

2.3.1 Software Engineering-oriented Wikis

Wikis were initially used in a software engineering setting, namely the Portland Pattern Repository (Leuf & Cunningham, 2001). Furthermore, they are often used to support software development, in particular in the area of Open Source Software. The Wikis of the Apache Foundation are a prominent example of this application scenario. Some examples of Wikis offer specific functionality for software engineering:

- *Trac* (Alrubaie, 2006) is a Wiki written in python that integrates an issue tracker, allowing to relate Wikis pages to issues and vice versa. Furthermore, the python code of a project can be integrated as read-only documents.
- *MASE* (Maurer, 2002) is an extension to the JSP Wiki that offers plugins for agile software development, in particular for iteration planning and integration of automated measurement results.
- *SnipSnap* (John et al., 2005) is implemented in Java and allows read-only integration of code documentation. Furthermore, it offers support for the integration of Wiki entries into the integrated development environment eclipse.
- *Subwiki* (SubWiki, 2005) is a Wiki implementation that uses the versioning system subversion as a data repository. Since subversion allows attaching metadata to files, the resulting Wiki is supposed to have the same features. However, this project has not released a stable version yet.

- *EclipseWiki* (EclipseWiki, 2005) is a plugin for the eclipse software development platform. It uses the workspace of a project as (local) data storage. By using a versioning system, the Wiki files can be shared and edited by a project team.
- *WikiDoc* (Oezbek, 2005) is a conceptual work that supports adding java code documentation via a Wiki interface. This allows non-programmers to participate in the creation of code documentation.

All those examples show that Wikis are increasingly being used as a platform for software development. However, "regular" Wikis (see (WikiEngines, 2005) for an overview of most of the Wikis currently available) and (WikiMatrix, 2006) for a configurable overview) as well as software engineering-oriented Wikis build upon the fact that the relations between documents and further metadata are maintained by the users of those Wikis. This lack of explicit semantic information is addressed by an extension to the regular Wiki functionality that is developed in the RISE project. These so-called *semantic Wikis* are elaborated in the next section.

2.3.2 Semantic Wikis

In this section, we present further examples of semantic Wikis. Most of these examples are taken from the overviews presented in (Dahl & Eisenbach, 2005) and (Semantic Wikis, 2005). Those examples show that a) even "regular" Wikis offer some support for structuring their content and b) that semantic, RDF-based Wikis can be implemented. Most of those examples are general purpose Wikis that do not focus on software engineering in particular. A general overview of semantic Wikis can be found in (Völkel et al., 2005)

- The most common way to categorize within Wikis is the usage of the backlink function of a Wiki (Aumüller, 2005). Basically, a page is created that represents a certain category. Pages belonging to this category have a reference to this page. The backlink function lists all of these references. However, this approach has one major drawback: pages that are used to navigate to the category entry (and thus do not semantically belong to this category) are also included in the backlink list.
- Some Wikis offer additional support for structuring content. For example *TikiWiki,* (TikiWiki, 2005) allows assigning pages to structures (table of contents) and categories (taxonomy style classification). (XWiki, 2005) offers forms (templates) that contain metadata, which are instantiated in documents derived from this form.
- From the area of SE-specific Wikis, TRAC offers a labeling feature for pages (smart tags) that could be used for a facetted presentation of the pages annotated with those tags. SnipSnap allows determining the template of a document and offers RDF export.
- *Platypus* (Platypus, 2005), *SHAWN* (Aumüller, 2005), and *Wiki.Ont (2005)* allow adding RDF annotations for each page. Pages within Platypus represent a concept. While viewing a page, the RDF triples are displayed that have the current page as object (in particular, pages that reference this page) and as subject (in particular, metadata about a page). SHAWN also offers navigation support based on the ontology information added to a page. Wiki.Ont is still in a preliminary version.
- *Rhizome* (Rhizome, 2005; Souzis, 2001) and *RDF-Wiki* (Palmer, 2005) are Wikis that provide their content in RDF, thus allowing to reason about their context.

Only the Wikis mentioned under in last two bullets can be seen as "real" semantic Wikis, since they allow relating their content to an RDF-based ontology. However, all of them – at least in their current state – do not integrate their ontology into the Wiki, e.g., they neither provide metadata templates to be filled in based on an ontology nor do they check whether metadata entered is consistent with an ontology. Therefore, when related to the semantic web layer cake (Berners-Lee), all of these semantic Wikis implement the RDF and RDFS Layer. The vocabulary layer of these applications is not domain specific, and thus does not allow inferring about domain specific relations.

2.3.2.1 Ontology Development

The development of ontologies to classify and structure knowledge – often called knowledge engineering – is a rather mature field that has its origins in artificial intelligence. Several methodologies, approaches and standards have been developed that focus on ontology development in general or resulted in an ontology for software engineering knowledge. This work is implicitly included in the RODent method presented later, and developers of Riki-specific ontologies might find these works helpful as a source of inspiration for their own ontology development. Furthermore, by adhering to the following standards, the tools and framework based on (some of) these standards can be employed for reasoning specific tasks in RISE:

- *Ontology Development:* CommonKADS (Schreiber et al., 1994) is a methodology for developing Knowledge-based systems and contains guidelines for developing ontologies.
- *Ontology Description Standards*: The language set of RDF (Manola & Miller, 2004), RDF-Schema (Brickley & Guha, 2004) and OWL (Smith et al., 2004) defines language constructs that can be used to define ontology in a way suitable for machine reasoning.
- *General Ontology Standards*: Ontology Standards make use of those ontology description standards. In RISE, we referred to the Super Upper Merged Ontology (Legrand et al., 2003; *SUMO Ontology*), describing general relations (such as part-of) and the Dublin Core Metadata to have a general set of document metadata (such as author, creation date) (2005).
- *Software-Engineering-specific Ontologies*: These ontologies define a general set of concepts and their relations in the field of software engineering. This allows the entries within Riki to be annotated with domain-specific meaning. Examples of such Ontologies are the software engineering Body of Knowledge (Software Engineering Body of Knowledge, Iron Man Version, 2004) and the classification of the Association for Computing Machinery (2005).

This background section illuminated the background of our work and gave a short overview of the application field of software engineering and software reuse as well as the Wiki technology that was used in the RISE project. These fields represent the context in which several problems were encountered that are described in the following section 3.

3 Problems and Challenges

After the description of the background of software engineering and the application of techniques from knowledge management and learning management, we look at the

specific problems addressed within the RISE project. As depicted in Figure 3, we identified several problems of knowledge and learning management in software engineering.

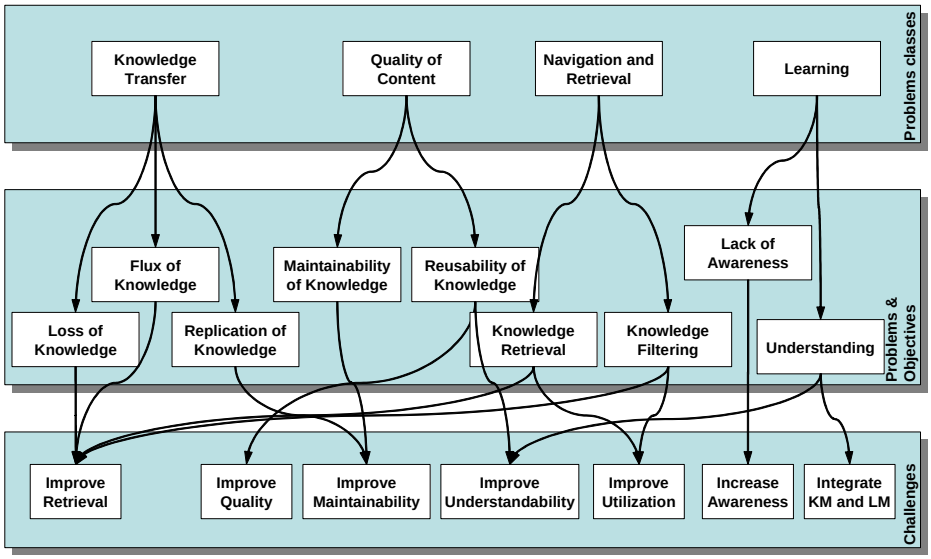

Figure 3. Problem overview

3.1 Problems of Knowledge Transfer and Reuse in Software Engineering

As motivated in section 2.1, the development of software is a highly knowledge-intensive activity that produces many reusable documents, ideas, and experiences. The productivity as well as the quality of the products developed heavily depends on the efficient and effective storage and retrieval of knowledge in a socio-technical system. A knowledge management system should at least reduce or prevent the occurrence of the following basic problems in KM:

- *Loss of knowledge*: Key personnel with valuable or even critical knowledge about products, processes or customers leave the organization and leave a knowledge gap. In a software organization, this might be a senior software developer who is the only person with critical information about the inner structure or function of a software system.
- *Replication of knowledge*: Products, intellectual property, or experiences are developed over and over again in slightly different versions. In a software organization, an algorithm might be developed twice because one project does not always know what other projects are doing. This might also apply to defects that are introduced to different parts of the software system and removed by different testers doing the same work (i.e., solving the same problem) twice.
- *Flux of knowledge*: The knowledge about new technologies, projects, processes, products, people, or standards is continuously subject to change. In order not to fall behind the innovations, competitors who are using these gaps have to be discovered

and handled. For example, software organizations have to cope with many new open source projects, competitive systems, development standards, or frameworks that are developed or changed every year.

In the following sections, we have grouped specific problems into classes regarding quality aspects, the integration of KM and LM, the learning processes, as well as retrieval and reasoning about the content.

3.1.1 Problems related to the Quality of Content

A knowledge base (KB) serves as an asynchronous communication medium or intermediary between people such as managers, developers, or even customers. The whole process is one of mediated communication (i.e., people are leaving messages to other people). Furthermore, there is the danger that *relevant content from the outside is not considered*, especially from outside the knowledge base (i.e., on the Intranet), from the Internet, and from the social system at large (i.e., the people). Until today, there are several problems related to the quality of the content in a knowledge base that still represent a barrier in knowledge management:

- *Problem of reusability of knowledge*: The quality of the reported and recorded knowledge and experience highly depends on the individual skills of the contributor, e.g., the ability to structure the content, to formulate the artifacts with accuracy, and to describe it properly according to the needs of the target audience. Without assistance, this leads to quality discrepancies and gaps in the knowledge base. To be included in the knowledge base, the new content must meet minimal quality requirements such as described in a correct, complete, consistent, concise, and non-ambiguous way, including information on the context of the specific content.
- *Problem of maintainability of the knowledge base*: A regular evaluation of all content and removal of outdated entries is required. Storing several contradictory solutions for a sole problem, originating from different persons at different points in time, is a source of confusion and mistrust. The capacity to retrieve previous related experiences exists in most KB approaches, but it requires time and effort to review them. The risk of applying outdated content is high if content does not have an attached expiry date, and on many occasions, these repositories become a sort of graveyards – some content is added, but nothing is ever thrown away. With the continually growing size of KBs, it is difficult to keep an overview in order to connect related packages and avoid inconsistencies.

While the use of Wiki systems as applied in our project facilitates and improves several of the above mentioned problems due to the increased flexibility, agility, and simplicity, they also bring along own new problems, such as the *motivation to contribute* or the *sustainability* of the Wiki system. Furthermore, the flexible and easy creation of links between Wiki pages (i.e., Knowledge Elements) leads to high cross-linking and hence to the *deterioration and loss of structure*.

3.1.2 Problems related to the Navigation and Retrieval

A long running KM system that is accepted by the users and continuously increases the amount of knowledge stored within it typically amasses a plethora of knowledge that is unstructured, not connected, or outdated. The retrieval of a specific Knowledge Component either by navigation or search mechanisms typically leads to an information

flood. In order to enable and improve the elicitation, organization, retrieval, and usage of knowledge in software organizations, several problems have to be addressed:

- *Problem of knowledge retrieval*: Software development in general has to cope with the overwhelming amount of new processes, technologies, and tools that periodically appears. As a consequence, the fast retrieval of up-to-date information about new technologies, corporate software systems, or available experts becomes more and more important. Furthermore, there are many Knowledge Components that become outdated within a few years (e.g., information about Java frameworks).
- *Problem of knowledge filtering*: Searching for crucial information in a large knowledge base, especially if it is focused on a single topic such as software engineering, leads to a flood of results. The more similar the components are in a knowledge base, the more precise a search or navigational structure has to be in order to find the relevant information.

3.1.3 Problems related to Learning

KM systems focus mainly on *organizational learning*, i.e., where learning leads to collecting knowledge for the organization in order to be used by its employees or for the modifications of the software organization's processes, internal standards, objectives, or strategies. However, Rus and Lindvall stated that *individual learning* is considered to be a fundamental part of KM because employees must internalize (learn) shared knowledge before they can use it to perform specific tasks (Rus & Lindvall, 2002). KM systems make the assumption that the problem of continuous competence development can be partially solved by using intelligent retrieval mechanisms and benefitting from innovative presentations of retrieval results. As a result, knowledge-based systems (KBSs) focus mainly on knowledge acquisition, storage, and retrieval and less on the learning processes themselves, the integration with the work process, and the personal learning needs of the software developers. More specific problems related to experiential learning (i.e., learning from experience) are listed below:

- *Problem of understanding and applying of documented experience*: Experience is often documented by domain experts. A problem that occurs when expert knowledge is used for teaching novices is that there is a quantitative difference between expert and novice knowledge bases and also a qualitative difference, e.g., the way in which knowledge is organized (Ericsson et al., 1993). Novices lack SE background knowledge and are not able to connect the experience to their knowledge base. Hence, they often misinterpret other people's documented experience. The organization of knowledge at the experience provider's and at the consumer's makes the transfer of knowledge between different levels of expertise extremely difficult. Expert knowledge is somehow 'routine'. This makes it challenging for experts to document experiences appropriately and to make them reusable for others. A more detailed summary of problems related to learning from documented experience can be found in (Ras & Weibelzahl, 2004).
- *Problem of overview and lack of awareness*: Systemic thinking is the conceptual cornerstone of the 'Fifth Discipline' (Senge, 1990). It addresses the problem that we tend to focus on the parts rather than seeing the whole, and fail to see organizations as a dynamic process. Unfortunately, learning by documented experience is limited to a very focused view of SE and does not relate the experience to the whole SE context and related domains. Another problem is that

systems do not point explicitly to other available SE methods, techniques, and tools that could be useful for the work at hand, which leads to a lack of awareness.

3.2 Objectives, Challenges and Research Issues

Most of the above mentioned problems were tackled by several researchers in projects around the world, but a solution to them is still far in the future. We tried to improve the situation by using the open Wiki technology and integrated aspects of LM and KM to support reuse in software engineering. This system, which we call reuse-oriented Wiki (Riki), has to support the comprehensive reuse of the artifacts as described in Section 2, Background, and has to help the users to collect, record, retrieve, reuse, and learn from them. Furthermore, the administrative staff needs to be supported in the packaging, formalization, aggregation, and generalization of artifacts in this knowledge base.

The overall goal of the RISE project was to integrate lightweight experience management with agile software development. RISE pursued the following specific objectives:

- *Improvement of the retrieval of knowledge and orientation* in a body of knowledge to optimize the amount of knowledge and accelerate the time to access relevant knowledge.
- *Improvement of the quality of transferred knowledge* by assisting eoftware engineers in creating optimized artifacts (i.e., with optimized content and structure) based on didactical principles and by delivering didactically augmented experiences. These artifacts should easily be adopted and internalized by users of different expertise levels to support them in their daily work and performance.
- *Improvement of utilization and usability* of the KM systems to allow the user the goal-oriented search for suitable solutions to his problem in minimal time and to support him in the adaptation of the solution to his specific problems.
- *Integration of knowledge management and experience management with e-Learning and work* itself in order to improve the reuse of documented experience and to bring learning to the work place.
- *Improving the understandability* of documented expert experience by explicitly offering additional SE Learning Components and by explicitly stimulating learning activities through didactical principles.
- *Increasing the awareness of available but unknown SE topics* in order to increase the software quality based on the application of new software development approaches that have not been applied before.
- *Improve the maintenance of knowledge in KM systems* and especially Wikis, in order to optimize the amount of potentially relevant knowledge.

4 Semantic-based Reuse in SE: The RISE Approach

As we have argued, neither the technology nor the methodology currently available is sufficient to challenge the problems listed in section 3. After introducing the basic concepts of Riki, we describe the methodology and technology that was developed and used in the RISE project.

4.1 Basic Concepts in RISE

In general, we use the model defined by Tautz: "***knowledge*** is the range of learned information or understanding of a human or intelligent information system, experience is considered to be knowledge or practical wisdom gained through human senses, from directly observing, encountering, or undergoing things during the participation in events or in a particular activity" (Tautz, 2001).

4.1.1 Representation of Knowledge in Wiki Systems

After describing Wiki systems and the types of knowledge that might be stored in those systems (see section 2.3), we shed some light on the storage of knowledge artifacts in a Wiki system. We use the following concepts:

- **Knowledge Elements** are the most basic containers for knowledge and cannot be further divided without destroying the ability to understand them using other Knowledge Elements.
- **Knowledge Components** are complete and self-sufficient (i.e., independent of other Knowledge Elements) descriptions of knowledge (e.g., a SE artifact). A Knowledge Component consists of one or more Knowledge Elements.
- **Learning Elements** are the most basic learning resources. They are the electronic representation of media, such as images, text, sound, or any other piece of data that could serve as a learning resource when aggregated with other Learning Elements to form a Learning Component (Note: Learning Elements can be compared with assets of the SCORM Content Aggregation Model).
- **Learning Components** are units of instruction that contain at least one Learning Element. A Learning Component represents the lowest granularity of a learning resource that can be reused by the system for learning purposes. The difference between a Learning Component and a Learning Element is that a Learning Component is related to a learning activity and a learning objective. In addition, it can be referenced as a learning resource by the system (e.g., by using hyperlinks). Another difference is that a Learning Component could possess contractually specified interfaces and explicit context dependencies when these are used within a so-called Learning Space (Note: Learning Components are similar to Sharable Content Objects of the SCORM Content Aggregation Model).
- **Learning Space** consists of a hyperspace that contains at least one or more Learning Components that are presented, for example, by linked Wiki pages. A Learning Space follows a specific global learning goal and is created based on context information of the current situation (e.g., learner needs, working tasks the learner is currently performing, or attributes of software artifacts). The goal of a Learning Space is to provide a learning environment for self-directed situated learning (see Section 4.2.3.1 for more details).

As depicted in Figure 4, Knowledge Elements are the typical content of a Wiki page, while a short Knowledge Component might equally fit. Furthermore, a Knowledge Elements might have to be split into multiple Wiki pages if the content is too large or structured as a training course.

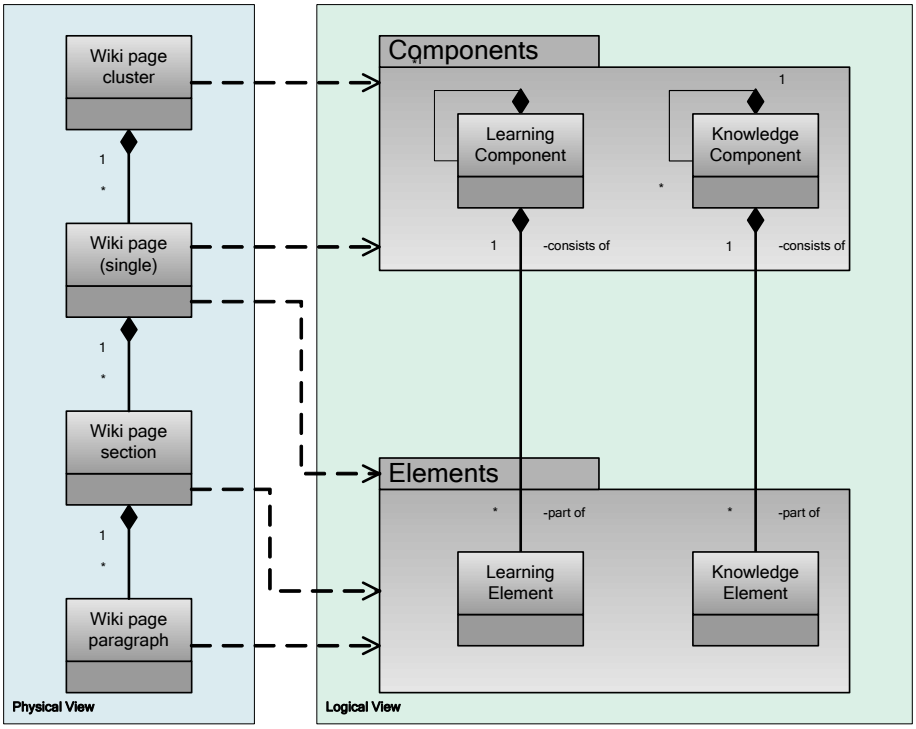

Figure 4. Knowledge in Wiki systems

4.1.2 Classes of Knowledge in a Riki system

The entrance of a Riki system consists of the six general classes of knowledge: projects, products, processes, people, customers, and (further) knowledge. Around these classes, several knowledge leaves are developed based on the knowledge types from section 2.1.2 (i.e., know-how, know-where, etc.). Therefore, a specific project page (e.g., about the RISE project) includes information about the requirements and designs of the changes planned for the products involved in the RISE project, but the product page (e.g., about the Riki) will include all requirements relevant for this specific product. However, a single knowledge leaf is not always purely classifiable as a single knowledge type (e.g., know-how), but might include other types of knowledge (e.g., source code developed in a project might include know-where (i.e., location) as well as know-when (i.e., version) knowledge.

As depicted in Figure 5, the artifacts from section 0 might be grouped around several core artifacts such as projects or products and represent connection points usable for further inference. For example, in the configuration shown, one could easily infer employees who have worked in a project that was initiated by a specific customer and therefore might have valuable knowledge for further acquisition activities.

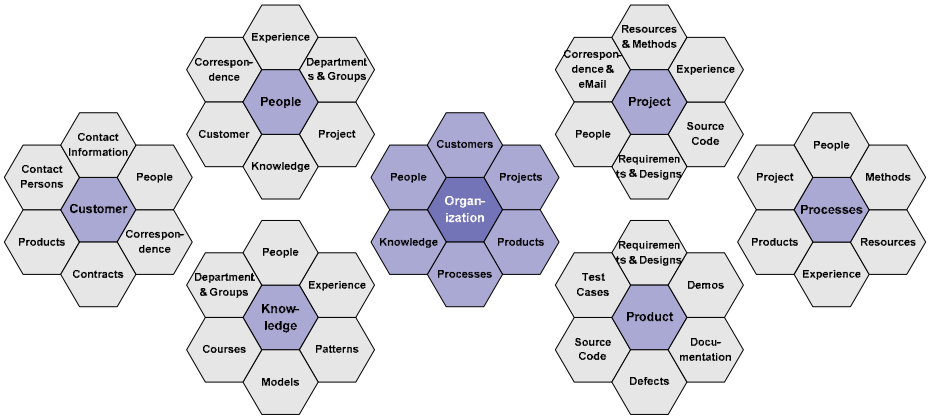

Figure 5. Knowledge component classes in an SE organization based upon O4PKC

The more often these or similar artifacts are generated, the more probable it is that they might be reused in a new project.

4.2 RIME: The RISE Methodology for Knowledge Transfer and Reuse

In this section, we describe three parts of RIME, including the design and development methodology for Riki systems, ontologies in a Riki, and Learning Spaces in a Riki. As depicted in Figure 6, all three are based on a detailed context evaluation in software organizations whose results were used for deriving the requirements for a Riki in the context to be addressed.

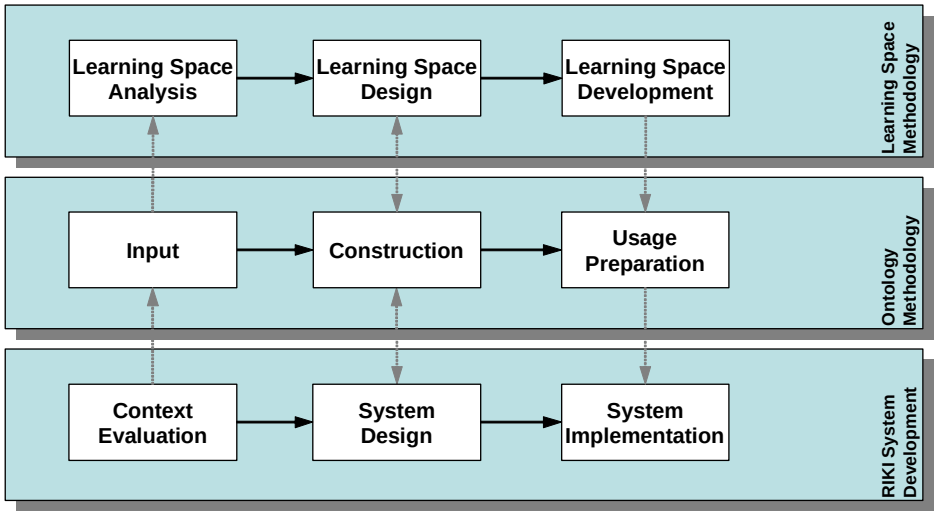

Figure 6. The RIME components (Ontology, Learning Space, and Riki System Development)

The RISE methodology represents a concept for the structuring, implementation, and operation of the Riki system.

4.2.1 Riki System Development

To develop a KM system such as Riki, we first analyzed and evaluated the context in two SMEs that both develop search technologies systems. From the information we gathered in these context evaluations, we designed and developed the Riki system, considering the people, processes, and available technologies. Finally we developed, tailored, and implemented these systems in the organizations.

4.2.1.1 Context Evaluation: Analyzing the existing socio-technical infrastructure

The goal of our context evaluation step was to elicit the existing socio-technical infrastructure of the companies. Therefore, we developed a catalogue of questions to elicit information about concrete topics that we wanted to know about (e.g., what technical systems are in use). In order to answer the following questions, and therefore elicit knowledge about the context the RISE system will be embedded in, we conducted an interview using a refined questionnaire based on the know questions from section 2.1.2. The basic topics of the questions were:

- Where and what are technical systems with potentially valuable knowledge?
- Who are the experts for specific topics such as products or technologies?
- How is knowledge transferred from tools to persons? What technical systems for knowledge transfer are available?
- How is knowledge transferred from person to person (interpersonal)? What social knowledge transfer techniques are used?
- How is knowledge collected from persons to tools?
- How is knowledge collected or transformed from tools to tools?
- Who needs and who produces new knowledge?
- What processes and tools are used to develop the software? Where can knowledge be collected and injected?
- When is knowledge collected, transferred, reused, needed, or produced?
- Why is knowledge currently being collected, transferred, or reused?
- What knowledge is (seemed to be) valuable for collection, transfer, or reuse? Which artifacts (as in section 0) are used and should be reused?

Beside the use of questionnaires to query the employees of the companies and elicit knowledge about the context a social-technical system has to be embedded in, we also conducted open group discussions and analyzed the existing artifacts and storage system ourselves. Overall, we used the following techniques to gather the knowledge:

- Goal-oriented, questionnaire-based *interviews* were used to put the previously listed questions to 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 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).

Using the information extracted with these techniques, we derived and created tailored versions for each company of:

- *Ontology* with metadata for all pages in the KM system (i.e., the Riki) and relations between them. This metadata is used for inference, search and presentation purposes. The ontology is based upon information from the existing artifacts of the company, the knowledge lifecycle, and standards of knowledge description such as LOM.
- *Templates* for every artifact type with a special focus on requirement documents. These help to record new knowledge of that type and are used in inference, for example, to find similar documents of the same type. The templates represent the inner structure of artifacts and esp. Knowledge Elements on a Wiki page. Information to define the templates came from the identified knowledge artifacts, the knowledge lifecycle (i.e., how would a user read the document in a specific activity of the process?), and industrial best practices such as the Volere requirement template (Robertson & Robertson, 2005a) or the ReadySET requirements-engineering template.
- *Navigational structures* within the KM system (Riki) were created to support the goal-oriented and artifact-centric access to the knowledge. The structures are based upon information from the existing structures used by the employees of the company and the processes, products, and groups of the company.
- *Riki software system* with plugins to support the users with additional information from inference via the currently viewed knowledge as well as connectors to the technical infrastructure. The system design is based upon information from available software systems in the organization, established knowledge transfer techniques, and usability aspects.

4.2.1.2 System Design: Tailoring a Riki for a Learning Software Organization

After we elicited the information about the context our socio-technical system should be embedded in, we started the design of the system. Beside the information about the context, we needed further information about the technical features and the extensibility of the available systems. The following information was found to be useful for the design of the system:

- *Survey* of systems (e.g., open source), extensible without too much work, and flexible enough to realize the planned systems.
- *Selection* of an appropriate system that fulfills the optimal set of requirements by the specific customer based on the use cases and scenarios as defined by the customer's business or development processes.
- *Design of plugins and portlets* that give specific views on related articles, similar pages (of the same document type) as well as the plugin interface. In general, every know-question from section 2.1.2 represents one plugin that shows information related to the currently viewed Knowledge Element, for example, the persons who

have further knowledge (i.e., know-who) or methods for post-processing (know-how).

- *Design of templates and interface masks* for every specific knowledge type (e.g., templates and masks for use cases (requirements)).

Furthermore, based on the features already existent in Wikis and extensions planned for the Riki as well as on the features of the search and inference technology, we shaped the ontology and vice versa.

4.2.1.3 System Implementation: Introducing a Riki into a Software Organization

Based upon the information from the design, the system and required plugins are built using the chosen Wiki systems. During operation, and the experiences acquired from the daily work (e.g., projects) are seized and didactically augmented to support users in current or future projects.

The operation and maintenance of the Riki system, its underlying ontology, and the knowledge stored within it is currently being done by the software organizations themselves.

4.2.2 RODent: Riki Ontology Development

This section explains the RODent method for the design and development of Riki ontologies. In parallel with systems building, we need to develop an ontology to be used by the Riki. In general, an ontology is defined as "an explicit and formal specification of a conceptualization" (Gruber, 1995).

In RISE, we instantiate this definition: An ontology is a set of templates, their metadata ,and the relations among those templates. The role of the resulting ontology is to structure and relate the information captured inside the Riki. Therefore, the ontology can be seen as a link between the implementation of a Riki and the content of a Riki. The guiding idea is *Wikitology*, i.e., that the Wiki is the physical representation of the ontology with pages as concepts and links as relations (Decker, Ras et al., 2005; Decker, Rech et al., 2005).

Besides the ontology itself, the result of this method is an initial set of document templates and content to seed a Riki. In the following sections an overview of RODent is presented, using Requirements Engineering as an example application.

4.2.2.1 Outline of RODent

As depicted below, RODent is divided into two groups of subtasks: *Ontology Construction*, where the concepts and their relations are identified, and *Usage Preparation*, where the usage of the ontology is defined. A more detailed description of the following tasks is presented in the subsequent sections:

- *Input* to RODent comes from two sources: The first source is the result of the context analysis with the prioritized challenges of the organization and the currently used structures and documents, defining the “asIs” status of the organization. The second source is an analysis of domain-specific literature relevant for the domain where the ontology should be built (in the example, from the area of Requirements Engineering). This literature will provide an initial set of templates.

- *Construction* as the first group of tasks uses this input as follows: First, the actual templates used in Riki are identified. Second, those templates are related with each other and "equipped" with metadata.
- *Usage Preparation* as the second group is about setting up a Riki to use the ontology defined in the subsequent phase. The task "Determine Relation Requirements and Default Relation" is a link between those phases, since it refines ontology relations, but also covers aspects of ontology usage in the Riki. "Identify views and consistency checks" takes care of using the ontology for navigational and editing support as well as to define inconsistencies within the knowledge captured in Riki. In "Technology Deployment", it is defined where the ontology, the views and consistency checks are put in the Riki.

The application of RODent – provided that no increments or iterations occur – is depicted from left to right in the illustration below.

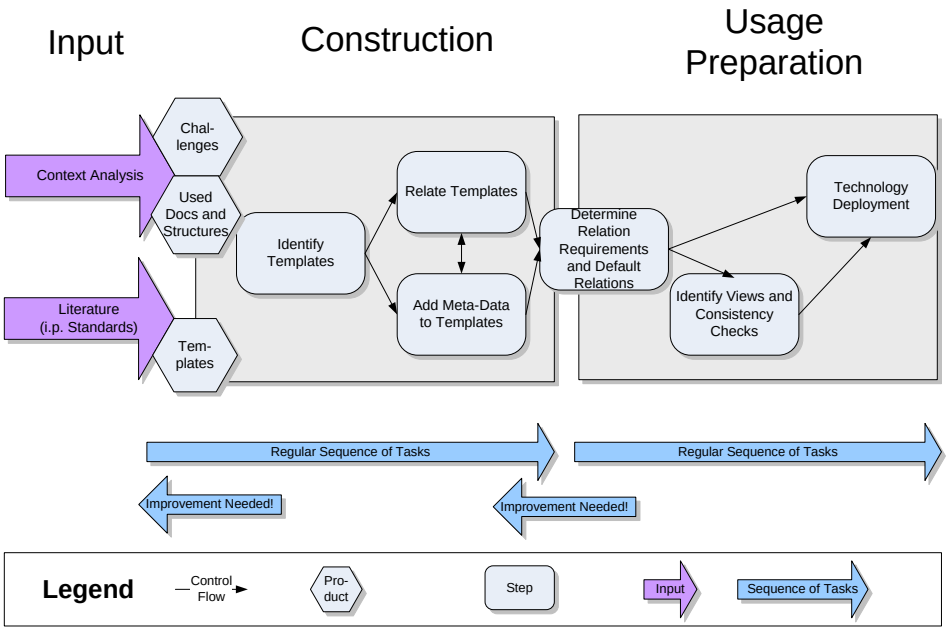

Figure 7. Overview of RODent

The challenges as well as the feedback when developing and using the systems guide the execution of the method:

- In the first execution of the method, one has to concentrate on solving the most relevant challenges most of the people using the Riki are interested in. In particular, one should try to identify multipliers, i.e., people in the organization who have a special stake in the Riki usage and support their daily work. For example, if inconsistent and outdated documentation is a challenge, then the resulting ontology should support the documentation author.
- During development and usage of the systems, increments and iterations are likely to occur. Since those terms are sometimes used with interchangeable meaning, we define their meaning within RODent and their impact on ontology development:

Increments are additions to the current ontology without the need to change further parts. Iterations also include a change within the existing ontology.

Within the RISE project, we encountered several increments (e.g., adding an architectural description) and some iterations (e.g., re-arranging Use Cases), but no severe change of the ontology occurred. This experience supports the demand-oriented approach that is the basis of RODent. However, this stability of the resulting ontology might be a result of the comprehensive context analysis and literature survey.

The following sections describe the task of RODent in details and provide an example of the usage of this method for requirements engineering.

4.2.2.2 Identify Templates

The main objective of this task is to identify the templates capturing knowledge inside a Riki and to create prototypes of the templates needed. As outlined in the method's overview, this task has the following two inputs:

- The result of the context analysis with its overview of the currently used documents and the challenges of the organization.
- The literature survey identifying standards and other useful information to be used in template development.

During identification and development of those templates, we found the following guidelines to be helpful:

- *Split and link self-contained documents:* Resources found in the literature (e,g., standards) are normally part of a self-contained document written in a linear way. Since Riki is a hypertext-system, one needs split and link coherent information chunks found in those documents. Potential candidates are the different sections within these documents. For example, in the Volere Requirement Specification Template (Robertson & Robertson, 2005b), we identified the different user descriptions based on this approach.
- *Refactor documents covering multiple aspects:* The resulting templates should cover one aspect of information. Given a readable font, this guideline boils down to the following operational rules: 1) When a template is completed, it should not be shorter than one and not longer than five screens. 2) Each (main) section of the template should fit on one screen. Adherence to this guideline is the basis for the reuse of knowledge, since it allows referring to specific information.

When all templates concerning the current challenges are addressed in the first execution of RODent, further templates demanding too much additional effort should be left out. To find out about those templates, one should ask the user whether the information is relevant and whether the template would be completed.

As a preparation to the following steps "Relate Templates" and "Add Metadata to Templates", the templates should be arranged into main categories according to their purpose. In RISE, we found the following categories helpful:

- *Context* templates like project and homepage of employees (people) based on O4PKC.
- *Navigation* templates like overviews about certain types of documents for specific reader groups.

- *Core* templates that define the structure of SE artifacts (cf. 0) directly related to software engineering activities (in the example, requirements engineering documents).

Finally, in order to test the templates, one should try to map some of the available information in the organization to the developed templates. This test might indicate hints that some of the information required by the templates is not available within the organization.

4.2.2.3 Add Metadata to Templates

After the templates and their contents are identified, they are annotated with metadata to support filtering and finding of relevant documents. To test which set of metadata is suitable, one should test it with a template. To identify relevant metadata, the following sources should be considered:

- *General standards*: Domain-independent, standardized metadata-sets like Dublin Core and Learning Object Standards (e.g., LOM) provide general purpose metadata. This allows other applications that rely on these standards to reuse the content of a Riki.
- *Template-specific metadata*: Some templates derived from existing standards and documents might already contain a set of specific metadata to classify their content. Furthermore, similar templates that are on different sections of the outline of the document are candidates for identifing metadata. One example is the description of stakeholders and users in the Volere Requirement Template (Robertson & Robertson, 2005b), which has a similar structure.
- *Method-specific metadata*: Some metadata might not be directly related to a template, but to the underlying method. Using this metadata allows providing method-specific support in the form of know-how documents. An example from RISE is the TORE method (Paech & Kohler, 2003), which classifies requirements engineering documents (e.g., as description of the current state “As Is” and the system to be developed “to Be”).

For adding metadata in general, remember the following guidelines: *Automatically derivable metadata* (like the author’s name) can be added without further consideration. *Manually entered metadata* should only be considered if it is of direct benefit to the person who will enter it. In our experience, it is no sufficient benefit that the author is able to find this document based on the metadata entered: This search will be performed in later stages of the project. Therefore, improving the searchability of the document does not provide any direct benefit. An example of metadata with direct use is the use case classification, which can be used to generate overviews. Whether this direct use is actually taking place should be checked in the task “Identify view and consistency checks”.

4.2.2.4 Relate Templates

The tasks before focused on identifying and creating single documents that have a high internal coherence. In this task, these templates are linked in order to define their semantical relations, which are later used by a Riki to identify relevant documents.

In the work, several general types of relations are identified that form the foundation for defining the relations within RISE:

- *Inheritance / Subclass*: This relation denotes that one sub-concept (template) inherits the properties (in RODent, the metadata) of a super-concept (a generalized template).
- *Instantiation*: This relation denotes that an object (in Riki: a document) belongs to a certain class (in Riki: a template) and thus, inherits the properties of this class.
- *Part-of / composition* : This relation denotes that an object derived from a concept is supposed to be part of another object.
- *Temporal*: This relation denotes that there is a relation within time between one concept to another concept. For example, it is a temporal relation that one object is created before another one.

Within RODent, those general relations are refined further according to their interrelations between templates and their documents. The actual use of these relations for reasoning is described in the task "identify views and relations":

- *Hierarchy* among documents of the same templates (part-of type): Instances of templates might have a hierarchical relation. For example, one use case might have several sub-use cases. It might be necessary to distinguish between different types of hierarchies for documents derived from one template: However, in Rise, we found one hierarchy to be sufficient.
- *Used in* (part-of type): This relation describes a "strong" relation between documents of different templates: If the content of a document is needed to understand another document. For example, the description of an Actor is part of the description of a use case. Without knowing the description of the Actor, the use case is not sufficiently understandable.
- *Refined in* (temporal): Refined in defines a temporal relation between documents. For example, a user story (a prose text of a certain requirement) is refined in a use case (a more structured representation).
- *Of Interest* (part-of type): This relation is used for weak relations between documents of different templates, i.e., where information relevant to one document can be found in another one..
- *Context definition* (inheritance type, relation between context documents and other documents): This relation is used when additional context of a document is defined in another document. For example, the project context (like size, programming language) of a use case is defined in the project homepage. Simplified, when such a relation is established, the referencing document "inherits" the metadata of the document (in this case, the metadata about the project).
- IsA (inheritance type, between templates and documents): The IsA relation is reserved for denoting the relation between a template and an instance of this template.

4.2.2.5 Determine Relation Requirements and Default Relations

In the previous task, (relevant) potential relations between documents have been identified. In this task, those relations are annotated to define requirements concerning the relation itself. These relation requirements are used in the subsequent task to derive views and consistency checks. In addition to these additional relation requirements, a default relation between documents derived from a certain template is defined.

The additional requirements define the nature of a relation between templates and thus, the nature of the link between documents derived from those templates that have this relation. In other words, they define the nature of a link between the source document (i.e., where the link is defined) and the target document (where the links points to). In RISE, the following additional requirements concerning relations were identified:

- *"Required", "recommended", "optional" and "not allowed"* denotes the degree to which a link is supposed to be established between documents.
- "*Unique"* and "*multiple*" denotes the cardinality of the relation, i.e., the source document can have only one relation to a target document, or it may have multiple relations.

Based on this overview of relations between these templates, the default relation – i.e., the one used most often – is determined. For example, the default relation between use cases and actors is Used In. As long as no other relation is stated, a Riki assumes that the link between those documents is the default relation. Therefore, a Riki must state less explicit information about the type of the link. In addition, a user not aware of the additional Riki features can use a Riki like any other Wiki system.

4.2.2.6 Identify Views and Consistency Checks

In the previous steps, the templates, their metadata, and relations were defined. In the task "Identify views and consistency checks", the actual use of these results is determined. In particular, the definition of views and consistency checks is based on the general types of relations and the additional requirements presented in the previous two tasks. For metadata, relation type and additional potential requirement views and consistency checks are presented in Table 1.

Table 1. Ontology elements

Ontology element / structure	Views	Consistency Check
Metadata	Present metadata annotations	n/a
Inheritance	n/a	Is there a cyclic relation among subclasses?
Instantiation (IsA)	Show all documents belonging to a template	Does a template have documents? Does each document belong to a template?
Part-of	Show documents that are the target of a part-of relation	Are there cyclic relations among documents derived from templates that have a part-of relation (i.e., is the "higher" document part-of of a "lower" document")?
Temporal relationship	Show timeline of creation of documents	Does the document derived from the source template have an earlier editing date than the target document?
Quality of relation (i.e., required and not allowed)	Show documents derived from templates that are the target of the required relation.	Are all required or not allowed relations satisfied?
Cardinality of	n/a	Are cardinality constraints violated?

Ontology element / structure	Views	Consistency Check
relations		

4.2.2.7 Technology Deployment

The task "technology deployment" covers how the templates, metadata and their relations are implemented within Riki. The task itself is subdivided into two sequentially performed subtasks: 1) Deployment of templates, metadata and ontology upon distinct Wikis (if any) and 2) the deployment of ontology concepts on native Wiki syntax, i.e., how parts of the metadata and relations are expressed by using naming conventions.

The *deployment on different Wikis* is trivial if only one Wiki is used. However, based on the experience from RISE, it is sensible to use – at least in larger organizations – one Wiki for each group (in particular, projects) and one main Wiki for the whole organization. This separation between different Wikis has several advantages independent of whether or not Riki is used: First, it reduces the complexity of access right administration by dividing it into several Sub-Wikis. Second, the possibility of name collision is lowered. An example of such a naming collision would be if two projects using the same Wiki have a meeting on the same day. If both want to use "meeting<date>" as the name for the document containing the meeting minutes, a name collision occurs. Furthermore, if there is no syntactical collision (i.e., the same name), a semantic collision might happen: Members of one project might link to the entry of the other project, because they cannot differentiate between the two documents (e.g., "meeting<date>" and "<date>meeting"). Second, it is more likely that the group responsible for a Wiki accepts it as "their" Wiki. The separation into several Wikis also has an advantage for Riki: Since each group owns "their" Wiki, contextual information about this group can be derived. An example are project Wikis, where context information about the project could be derived from the metadata defined on the project homepage.

The *deployment on native Wiki syntax* is partially based on the deployment on several Wikis. By using the interwiki feature, the context of a document can be derived even if referred from a different Wiki. Another example for deployment of parts of the Ontology into native Wiki concepts is the instantiation (IsA) relation. Within Riki, we had the following naming convention to denote this relation without the need to explicitly create a link: The instances of a template have the name of the template as prefix. For instance, documents of the type "Actor" are named "Actor_<NameOfActor>".

When an ontology developed using RODent is finished, it should contain the relevant information to execute the Software Engineering activities it is intended for. How it is actually used and gradually enriched with experience and further information on how to perform Software Engineering tasks is covered by the Learning Spaces described in the following section. These are an unintrusive approach for providing the procedural knowledge needed to perform a certain task.

4.2.3 Riki Learning Space Development

In many publications about learning objects, terminological issues are discussed, because there is a lack of solid theoretical foundation in the field of e-Learning (Self,

1992), especially concerning learning objects. It seems that the main reason why learning objects have been invented is reuse and the desire for more flexible, adaptive learning systems. Current development efforts with learning objects are mainly concerned with metadata and content packaging aspects. Current object metadata says little about how to combine learning objects with others, and this will limit the success of the numerous repositories of learning objects that are being developed. Nevertheless, a model such as the SCORM *Sequencing and Navigation* is a first step towards a more systematic way for defining the sequencing of learning content and navigating through it.

Sometimes, learning objects are put into relation to object-oriented programming objects. The term 'object' is somehow misleading. Some authors toss around theoretical connections to object-oriented theory that stem from computer science. One reason why many of us attempt to connect learning objects to code objects is that there is a grammatical affinity between 'object' as used in 'learning object' and object-oriented programming theory. It is not wrong to refer to concepts from object-oriented theory in order to increase our understanding of learning objects and our belief in successful reuse. Friesen states that there is not only a conceptual confusion in the literature between software objects and learning objects, it also seems that object-orientated programming objects and learning objects do not fit together at all (Friesen, 2001).

Therefore, we will use the term *Learning Component* instead of learning object. They should be considered as components instead of objects because their characteristics are similar to software components. A software component is a 'unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition of third parties' (Szyperski & Pfister, 1997). Component-based software development is based on the principles of 'separation of concerns' in order to decrease the complexity of system design and its latter embodiment with components. Therefore, a set of formal description models for components, their interfaces, and context dependencies are defined in a metamodel in order to develop components from an abstract, generic, and coarse-grained view of a conceptual component model into concrete, specific, and fine-grained learning material. This material is composed of Learning Components that can be integrated into a Learning Space (see Figure 9).

In order to develop Learning Spaces (see Section 4.1.1 for the definition of concepts used in these sections), the following three phases can be identified (see numbering in Figure 8):

- During the *Learning Space Analysis* phase, the results of the context evaluation are analyzed. The main goals are to find out what types of content are available and useful for developing appropriate Learning Spaces, how learning needs can be identified based on analyzing the produced software artifacts, and in which way Learning Spaces can be connected to the working environment.
- During the *Learning Space Design* phase, the analysis results are used to describe the Learning Space from a conceptual point of view. This means that appropriate learning methods, learning goals, and a content classification have to be chosen. Based on this information, the metamodels of the Learning Components and the instructional design are adapted to the current learning context.
- During the *Learning Space Development* phase, the physical Learning Space is created to be explored by the learner by transforming, in a first step, the

metamodels into a framework with 'empty' placeholders for Learning Components and Learning Elements (see Section 4.1.1 for the definition of these concepts). The automatic embodiment replaces the placeholders within the framework with concrete Learning Components and Elements, i.e., Wiki pages with appropriate links and navigation structures are created during run-time.

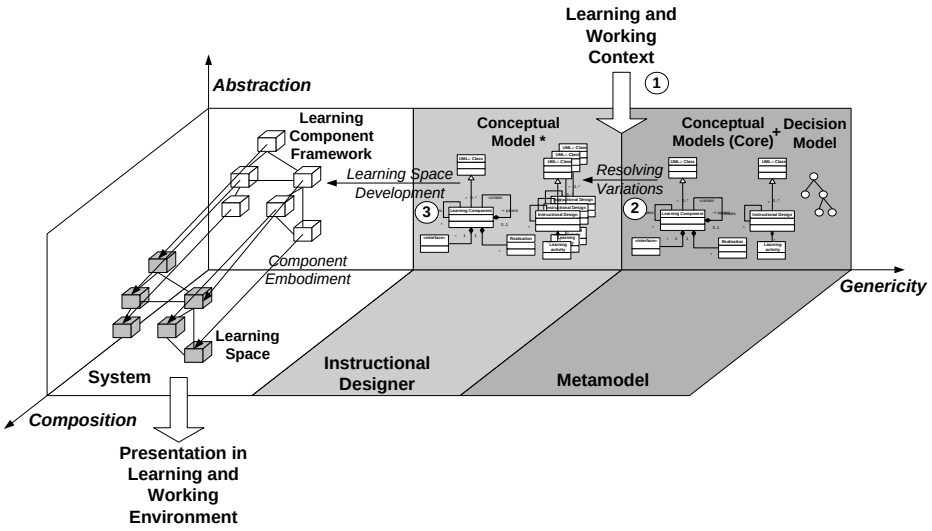

Figure 8. Development Dimension of Learning Space Engineering

The subsequent sections explain the three phases, which have to be performed for setting up a Learning Space, in more detail. Each phase is described with a few main questions that cover the goal of that phase and end with a concrete example.

4.2.3.1 Learning Space Analysis

This phase can be motivated around the following four questions:

- What is the most relevant domain for improving task performance and competence development, and which target group do we address?
- What type of knowledge that is suitable for learning is available or created by the software engineers during work?
- Where are the triggers for Learning Space creation and how can a Learning Space be related to the working context?
- Which software artifacts could be used to identify (e.g., automatically) competence and skill gaps to be solved by a Learning Space/ Learning Arrangement?

In general, Learning Spaces can have two different purposes. First, a Learning Space can improve short-term task performance, i.e., by providing solutions in order to solve problems more efficiently or by offering different methods or tools that enhance a specific well-known task. The domain under consideration is very narrow. Second, long-term competence development involves an analysis of a much broader domain software engineers are working in. Here, we would like to find our competence gaps that cover perhaps more than one software development phase, such as general software engineering principles or completely new technologies or development approaches.

Usually, a software project covers the whole development cycle from requirements elicitation to programming, testing, and delivery of the software to the customer. The aim of the Riki methodology is not only to address the whole development process but to focus on very specific tasks first and to extend the scope of the Riki later. This has the advantage that early successes on important and difficult tasks, in terms of better task performance and competence development can be used to extend the system's scope. This step is called *domain scoping* and will also influence the development of the domain ontology. We found out that especially during the requirements and programming phase, suitable tasks can be found to get the system started. The information that is necessary to choose such tasks was gathered through personal interviews, by analyzing content available in knowledge repositories, and by looking at artifacts produced such as code or software documentation. In addition, a rough process model and a role model are derived in order to relate the Learning Space to the well-known processes and roles. It is essential to ask software engineers what kind of problems bother them the most, and which tools, methods, and techniques they apply to create software artifacts.

During the context evaluation available content was already analyzed and software engineering challenges were identified. In this phase, we check especially if this knowledge could be used for Learning Components and Learning Elements. The content is classified according to the knowledge dimensions and types (see Table 2 for details). Furthermore, metadata and granularity (i.e., size) are investigated in order to make first decisions about metadata extensions, respectively adaptations, to the metamodel of learning content. The idea is not to change existing policies and rules for content authoring. Learning Spaces and hence the Learning Components should be adapted to the existing situation and content within the organization.

Another important issue is to find out which situations trigger a learning need. The generation of a Leaning Space is demand-driven, i.e., Learning Spaces are created based on a trigger. We distinguish between two types of triggers. The human-based trigger is created by the software engineer himself, e.g., by searching for specific information, documented experience, or solutions. These search results could be embedded in a Learning Space. The system-based trigger is related to an internal system event, e.g., the technical software development environment analyzes a code artifact and finds a defect, which leads to a Learning Space where the engineer could learn how to remove this defect. Human triggers can be found by interviewing the engineers. System triggers have to de defined by Riki experts. Those triggers strongly depend on the capability of analyzing produced software artifacts such as code or other formally described documents. Based on those findings, appropriate Learning Spaces could be created in order to improve the quality of those artifacts or their development process.

The last aspect in this phase is to relate Learning Spaces directly to the working context. Since the Learning Space should be delivered as close to the work place as possible, connection points have to be found. In the domain where Riki can be implemented, those points are usually located within the technical development environment (e.g., Eclipse IDE) or within the knowledge management environment.

4.2.3.2 Learning Space Design

After finding out what domains are the most promising for learning (i.e., both long term as well as short term learning), and what kind of artifacts could be used to identify skill

and competence gaps, this phase defines learning goals and appropriate methods to be applied for the Learning Space development in the last phase.

This phase can be motivated around the following four questions:

- What kind of learning goals could be defined and how are they related to the knowledge dimensions/types and learning content?
- What are appropriate learning methods for the identified learning needs, triggers, and competence levels?
- How is the domain ontology used or adapted to meet the domain context and requirements for creating Learning Spaces?
- How is the instructional designer supported in adapting the core metamodels to the requirements for creating Learning Spaces?

Our Learning Spaces follow the constructivist learning theory and are related to the current working situation as closely as possible. Before we provide answers to the above questions, we would like to say some words about learning theories and learning approaches that are suitable for the software engineering context.

First, following the constructivist learning theory, learning can be seen as a self-directed process where the focus lies on opening up information and on constructing individual knowledge, e.g., (Bruner, 1973). Constructive learning theories criticize former learning methods for ignoring almost completely the transfer of lessons learned to new situations in practice, i.e., knowledge remained in a latent condition and was not applicable.

Second, situated learning approaches developed mainly at the end of the 1980s emphasize that a human's tasks always depend on the situation they are performed in, i.e., they are influenced by the characteristics and relationships of the context (Brown et al., 1989). Because of the relation between cognition and context, knowledge and the cognitive activities meant to create, adapt, and restructure the knowledge cannot be seen as isolated psychological products – they all depend on the situation in which they take place. Learning involves interpreting individual situations in the world based on one's own subjective experience structures. Learners have an active role and derive most of the knowledge from real situations by themselves, and this knowledge is afterwards integrated into their own knowledge structures. Learning and applying the lessons taught should happen in a situated context, i.e., during the development of software artifacts.

Third, research in cognitive psychology has shown that students learn better when involved in solving problems. Collins' Cognitive Apprenticeship (Collins et al., 1989), Schank's Goal Based Scenarios (Schank et al., 1999), and the 4 Component Instructional Design (4C/ID) Model of Merriënboer (Merriënboer, 1997) are just three of the instructional models that address problem-based learning. Merrill proposed the *First Principles of Instruction*: Learning is facilitated when previous experience is *activated*, when the lessons learned are *demonstrated* to the learners instead of just being presented to them, when the learners are required to *apply* their knowledge or skill to solve a problem, and when the learners are motivated to *integrate* the new knowledge or skill into their daily work (Merrill, 2000).

In fact, instructional design could be handled in two obvious places: embedded within a Learning Component or as a separate object (e.g., by using specifications such as SCORM Sequencing and Navigation or IMS Learning Design). Riki handles the pedagogical rules of instructional design *outside* of the Learning Components in a

metamodel for instructional design. By applying those didactical rules to this metadata, adequate Learning Spaces can be created that fit the current situation of the user.

Based on the identified triggers, appropriate learning methods are chosen. In the domain of software engineering, problem-based learning methods as listed above and experiential learning (i.e., compared to experience-based learning, experiential learning integrates elements of reflection, support and transfer) methods are a good starting set for creating Learning Spaces. The utilization of a knowledge management system is usually problem-driven, i.e., a problem arising during the completion of a software engineering task motivates the software developer to search for suitable information or even complete solutions in the repository. When reusing an experience, a developer is usually engaged in active problem-solving while reading, understanding, abstracting or instantiating the experience, and trying to apply the gathered knowledge to the real problem situation. Ideally, software engineers could learn effectively from experiences when all four phases of Kolb's Experiential Learning Circle (Kolb, 1984) are passed: making a concrete experience, observing and reflecting about the occurrence, forming abstract concepts, and testing these concepts in new situations. When a software engineer documents an experience for later reuse (i.e., this is usually done by creating abstractions), he or she profits from being involved in the situation that leads to the experience, and his own observation and reflection about the happening. When a software engineer other than the experience's provider wants to reuse this documented experience, he or she will lack specific knowledge about the event that led to the experience, and the knowledge that results from observation and reflection. Hence, the Learning Space should focus on the delivery of appropriate content in addition to the experience in order to support knowledge construction as described in Kolb's learning cycle.

In addition to the triggers, the role and the competence level of the software developer play a crucial role in how learning methods are implemented in a Learning Space. We follow a self-directed learning approach with a specific amount of guidance based on the capabilities of the learners, i.e., learners proceed through the Learning Space at their own pace and decide by themselves which Learning Components they want to access in which order. Nevertheless, when Learning Spaces are created, a certain amount of guidance and suitable content is provided to learners depending on their competence level. Riki distinguishes between three competence levels: novice (knowledge dimension: declarative knowledge), practitioner (declarative and procedural knowledge), and expert (all kinds of knowledge).

Content for learning has been identified in the first phase and domain related basic software engineering content that is not available has to be added to the repository. This content is now classified according to Learning Element types. A basic set as listed in Table 2 is used to categorize the learning content. The set is a mixture of different instructional designs as well as software engineering specific types. This set could be extended if necessary. Learning elements could not be related directly to the cognitive processes categories, respectively learning objectives, because many Learning Elements could be used in different cognitive processes such as 'Example'.

We refer to Bloom's taxonomy of educational goals (Bloom et al., 1956), which is widely accepted and applied in various topic areas including software engineering (Dupuis et al., 2003). In addition, we refer especially to the revision of the original taxonomy by Anderson and Krathwohl (Anderson & Krathwohl, 2001), which is briefly explained next. The new taxonomy can be explained by two dimensions: the knowledge

and the cognitive processes dimension. Our work is based on a similar knowledge dimension as the one in this taxonomy. Regarding the cognitive process dimension, Anderson and Krathwohl distinguish between six different categories:

- *Remembering* is to promote the retention of the presented material, i.e., the leaner is able to retrieve relevant knowledge from long-term memory. The associated cognitive processes are *recognizing* and *recalling*.
- *Understanding* is the first level to promote transfer, i.e., the learner is able to construct meaning from instructional messages. He builds a connection between the "new" knowledge to be gained and his prior knowledge. Conceptual knowledge provides the basis for understanding. The associated cognitive processes are *interpreting*, *exemplifying*, *classifying*, *summarizing*, *inferring*, *comparing*, and *explaining*.
- *Applying* also promotes transfer and means carrying out or using a procedure in a given situation to perform exercises or solve problems. An exercise can be done by using a well-known procedure that the learner has developed a fairly routinized approach to. A problem is a task for which the learner must locate a procedure to solve the problem. *Applying* is closely related to procedural knowledge. The associated cognitive processes are *executing* and *implementing*.
- *Analyzing* also promotes transfer and means breaking material into its constituent parts and determining how the parts are related to one another as well as to an overall structure or purpose. *Analyzing* could be considered as an extension of *Understanding* and a prelude to *Evaluating* and *Creating*. The associated cognitive processes are *differentiating*, *organizing*, and *attributing*.
- *Evaluating* also promotes transfer and means making judgments based on criteria and/or standards. The criteria used are mostly quality, effectiveness, efficiency, and consistency. The associated cognitive processes are *checking* and *critiquing*.
- *Creating* also promotes transfer and is putting elements together to form a coherent whole or to make a product. Learners are involved in making a new product by mentally reorganizing some elements or parts into a pattern or structure not clearly presented before. The associated cognitive processes are *generating*, *planning*, and *producing*.

Riki addresses all the categories of these dimensions, with the focus being on the first three categories, because these are important for reaching the upper levels and can be taught directly, while the fourth to sixth levels require a longer term and deeper understanding of a subject matter.

Common instructional design theories often speak of the following elements in the design of instruction: generalities, examples, explanations, practice items, test items, overviews, advance organizers, and analogies, among others (Yacci, 1999). Table 2 shows the educational goals and the related Learning Component types that were selected as a first set for a Riki in the domain of software engineering.

Table 2. Overview of educational goals and associated Learning Element types

Knowledge Type	***Knowledge Dimension***	*The Cognitive Process Dimension (Learning Objectives)*						**Learning Element Types**
		1. Remember	2. Understand	3. Apply	4. Analyze	5. Evaluate	6. Create	• Definition • Description • Example • Counterexp. • Analogy • History • Overview • Summary • Scenario • Procedure • Explanation • Theorem • Rule • Law • Principle • Model • Practice item • Checklist • Strategy • Reference • Learning Space Map • …
Know-What Know-who	A. Factual Knowledge	LA 1						
Know-That	B. Conceptual Knowledge		LA2	LA4				
Know-how	C. Procedural Knowledge		LA3	Overall Learning Goal	LA5			
Know-if Know-when Know-where Know-why	D. Meta-Cognitive Knowledge			LA6				

Although a lot of effort has been put into the definition of standards, and although the LOM standard seems to be widely accepted by now, ″key issues related to global content classification such as a common schema, ontology, granularity, taxonomy and semantics of learning objects which are critical to the design and implementation of learning objects remain unsolved″ (Mohan & Daniel, 2004). Hence, one of the key issues of this phase is to adapt the domain ontology in order to describe the semantics of Learning Components/Elements and their relations, and to find a common vocabulary for describing Learning Components/Elements.

The resulting ontology created by RODent covers the domain that was identified in the analysis phase. Learning Spaces tie into upon this ontology. After defining the types of Learning Components, each Learning Component/Element has to be related to the concepts of this ontology. This involves that the ontology concepts are used for describing the Learning Components/Elements with metadata. Riki will use LOM as metadata description. The suggested LOM vocabularies are adapted to the types of Table 2 (right column) in order to specify the value range of the LOM attributes.

The most complex task in this phase is the adaptation of both metamodels to the working and learning context where the Riki should be implemented. There exists one Learning Component core metamodel for defining the Learning Component and its composition of Learning Elements on a conceptual level, and one core metamodel for instructional design applied to this domain. The adapted metamodels are used to produce a Learning Space framework (i.e., a concrete implementation framework to create Learning Spaces). The types and relations between Learning Components (i.e., according to the relations defined by RODent) are explicitly modeled in the

metamodels. The conceptual model covers aspects such as the specification of metadata (i.e., by using LOM), containment rules that specify the parent-child relationships between Learning Components/Elements within a containment tree (e.g., for modeling the location of Learning Components), specialization rules that define the types of Learning Components and their specialization (e.g., definitions-, examples-, table of content-, overview-, and summary-components). Furthermore, this model could contain elements that specify the kind of interaction between the system and the user, and adaptation rules for adapting the Learning Components to learner types or context aspects. These rules are very similar to the contracts used as specifications for interfaces. Beside the conceptual core model for instructional design that is based on learning objectives with related learning activities (see Table 2 right section), a decision model exists that enables the instructional designer to adapt the model. The decision model contains so-called variation points, their resolution space, and their effects on the conceptual model. The variation points mark variable parts of the model that are resolved by questions. The instructional designer uses these questions in order to change the conceptual model in a systematic way. The questions refer to the categorization of Learning Components (i.e., the instructional designer adapts the categories or the specialization structure), instructional design strategies (i.e., the instructional designer adapts the containment rules, for example, to an *experiential learning* strategy), or the questions consider adaptation aspects (i.e., the instructional designer changes the variable parts of the logical Learning Component such as their composition of Learning Elements, see Table 2, middle section). The answers to the questions include solution packages, so that the instructional designer gets support on how to adapt the model. One possibility for defining such solution packages is to use design patterns. They are very useful for describing instructional design strategies in a comprehensive manner. A design pattern can be understood as a transformation process from one conceptual model state to a new state. Each transformation step relates to specific parts of the model and tells the instructional designer how to change those parts.

Based on these models, frameworks can be derived that can be used for creating Learning Spaces. The next section describes how a pedagogical agent creates Learning Spaces.

4.2.3.3 Learning Space Development

After the selecting of learning goals and appropriate methods as well as the generation of the metamodels, this section explains how a Learning Space could be created based on a framework from the metamodels.

This last phase can be motivated around the following four questions:

- How can a framework be built from the adapted metamodels in order to develop Learning Spaces?
- How can this framework be embodied with Learning Components and Elements?
- How can the Learning Space be presented by means of Wiki pages?
- How can sequencing of Learning Activities and navigation through a Learning Space be realized?

Before we answer those questions, a small excursus about pedagogical agents is given. A Learning Space is created automatically by a so-called pedagogical information agent. Information agents are a special kind of intelligent software agents (Wooldridge

& Jennings, 1995). Software agent technology itself originates from distributed artificial intelligence. Software agents have access to multiple, heterogeneous, and geographically distributed information sources. Klusch provides an overview of information agents and describes their main tasks as performing pro-active searches as well as maintaining and communicating relevant information on behalf of their users or other agents. This includes skills such as retrieving, analyzing, manipulating, and fusing heterogeneous information as well as visualizing and guiding the user through the available individual space (Klusch, 2001).

The pedagogical agent is a special type of information agent: it puts its emphasis especially on the mediation of information by taking into account learner profiles' learning preferences, such as preferred learning styles, presentation modes, etc. and creates their Learning Space based on the metamodel. The difference as compared to Instructional Tutoring Systems (ITS) lies in the fact that agents react proactively and take into account the current environment the learner is working in, instead of simply analyzing the current status of the learner's knowledge as ITS do. In our approach, a task agent observes specific software engineering tasks that are suitable for monitoring. For example, the activity of programming in an integrated development environment, such as the *Eclipse IDE*, can be monitored. Once a trigger condition as specified in the design phase has been observed, the task agent sends a notification message to the pedagogical information agent that has registered interest in the occurrence of particular triggers during the monitored task.

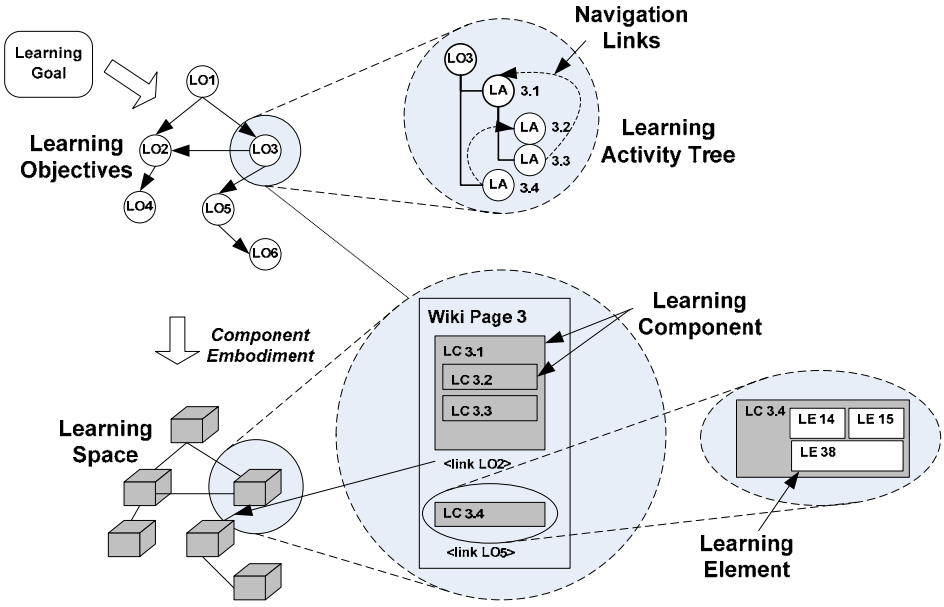

Figure 9. Details of an embodied Learning Space

For each instructional design metamodel, a framework is derived that is consistent with the metamodel for Learning Components, which was developed in the previous phase. An instructional design refines a learning goal into several objectives (see values in the cells of Table 2). Each learning objective refers to a cognitive process (e.g., remember the task of refactoring). As illustrated in Figure 9, arrows between the learning objectives show how the learning objectives should be sequenced in a Learning Space.

The difference between a learning goal and a learning objective is that usually, learning goals are broad, often imprecise statements of what learners will be able to do when they have completed the Learning Space. Learning objectives are more specific, have a finer granularity, and are measurable by performing assessments (i.e., through tests, questionnaires). The learning objective network is transformed into the Learning Component Framework. The next step is the embodiment of this framework by Learning Components and Elements.

Before the embodiment can take place, each learning object is refined in a learning activity tree (i.e., similar to the SCORM activity tree that could be derived from the SCORM content packages). The tree serves as a help structure for navigation. Each activity tree consists of learning activities that enable the learner to reach the related learning objective (e.g., reading, thinking about a question posed, removing a real code defect). In contrast to the learning objective network, this structure can only be created during run-time, i.e., after a notification message has been sent to the pedagogical agent. This structure depends on the following factors: the learning objective, the addressed domain and topic, the available content, and the context. A default structure for each learning objective/topic category pair is defined in the instructional design metamodel. The two latter factors influence the adaptation of these structures during run-time, e.g., learning activities have to be abandoned if suitable content is not available or if the context allows transferring learning activities directly into the working environment instead of keeping it within the Learning Space environment. The embodiment creates Learning Components from Learning Elements by using the domain ontology. Afterwards, each learning activity of the activity tree is extended with a reference to Learning Components created.

The Learning Components are presented to the learner by means of Wiki pages. Wiki pages are used within Riki because of their advantages of relating pages - other technologies could also be used instead of Wiki pages. Each Learning Objective represents one Wiki page, respectively one learning activity tree. They are shown by sections and paragraphs using the Wiki syntax. The difference to a standard Wiki page is that only parts of the Wiki page can be changed by the learner. These parts usually consist of specific knowledge such as project, customer, people, or product knowledge (see Figure 2 for the knowledge types) or assessment parts (e.g., answering questions etc.). General learning content cannot be changed (e.g., the definition of ‘Software Quality’ or explanation of ‘Polymorphism’). Only one Wiki page is generated by the agent at a time. The activity tree sometimes offers alternative navigation options to proceed to the next learning objective, respectively to another Wiki page (see link of *LO3* Figure 9). When such a link is chosen, the agent creates the corresponding Wiki page by first resolving the references to Learning Components and second, by generating navigation options by means of links within the Wiki page or external navigation portlets on the web site.

By default, sequencing of learning activities on a Wiki page is done hierarchically (see top of Figure 9). Beside the navigation options between learning objectives, additional navigation options could also be offered within the same Wiki page, i.e., between Learning Components. Within the SCORM Sequencing and Navigation model, navigation requests are processed based on a kind of learner model, i.e., data about, for example, answered multiple choice questions that is stored. This data influences which navigation options are available or whether they are not. In the Riki context,

assessments play a minor role. Only data from the software development task and changing learner preferences will be used to adapt the navigation options.

In summary, a Learning Space consists of several Wiki pages with links forming a hypermedia network. Agent technology allows us to adapt the Learning Space dynamically during run-time, e.g., while the user is browsing through the space and working on his task. Important triggers are forwarded by the task agent to the pedagogical information agent, who adapts the content selecting, sequencing, and navigation of the Learning Space. Observing certain tasks performed by a software engineer and the demand-driven creation of Learning Spaces ensures close integration of the learning process and the working task, and enables the provision of a situated learning environment for the user where he/she can construct his/her individual knowledge.

4.3 Riki: The Reuse-oriented Wiki

In RISE we developed a platform and a methodology for the management of experiences in SE organizations, which is integrated smoothly into the infrastructure. In this section, we will elaborate the architecture and technology of a plugin-based, reuse-oriented Wiki (Riki). This technical system sets the stage for the knowledge-based development of software systems strengthened via the support of social relationships as well as competence development and knowledge sharing in and between projects. As depicted in Figure 10, three domains or spaces are connected to the core system.

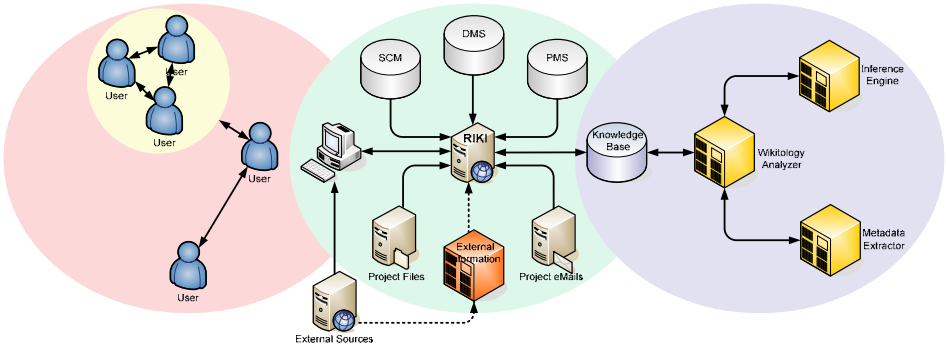

Figure 10. Infrastructure around a Riki system

The social-technical system of RISE into which the Riki is embedded consists of the *social space* on the left side, the *technical space* symbolized in the middle, and the *logical space* on the right. The social space encompasses all users of the Riki from the target group(s). They are all accessible via the personal pages and references such as (co-) author or inspector (i.e., know-who entries). In the technical space, all hard- and software systems are collected. The Riki is statically or dynamically (i.e., online) integrated into the existing infrastructure and previously collected information. The integration of information includes internal sources such as Software Configuration Management (SCM) systems, Defect Management Systems (DMS), Project Management Systems (PMS), as well as project files and email messages. For example, SCM systems can be used to extract information about persons who have changed the source code over the different configurations. If a newbie (at least for this part of the system) needs more information about this subsystem, the newbie can directly access

these persons and knowledge about it. Furthermore, information from external sources such as technology specific mailing groups or search engines such as Google are dynamically integrated.

4.3.1 The Riki Architecture

The general plugin concept of the RISE system is a service that is invoked via the web-service interface. The web-service approach was chosen to provide the added functionality independent of the actual implementation. The result of the web-service request is delivered text or simple html. The result is then integrated into the GUI via a content plugin, the templating mechanism of the Wiki, or as a pop-up window.

The RISE system has three different types of plugins that are differentiated based on their integration: Plugins that change the workflow of the underlying Wiki, plugins whose results are integrated into a single point on the GUI, and plugins whose results are integrated into the content.

This integration is supported by plugin interfaces that are already provided by some Wikis: First, Wikis might offer a plugin interface to add dynamic content to pages. The RISE system uses this interface by implementing a general plugin to request web-services and to present the result. Second, Wikis might provide an XML-RPC interface to read and write content of a page. The RISE system uses this interface to standardize the access to Wiki content independent of the actual Wiki used.

The basic knowledge flow of the Riki is shown in Figure 11. It depicts the *elicitation subsystem* where information from the community of users (or experts) is injected into the system as a first pool of knowledge in the knowledge base. This knowledge as well as the created ontology and approach for creating Learning Space are used in the *augmentation subsystem* to construct the Learning Spaces and augmented knowledge (e.g., a knowledge component about a project with context-specific information such as similar projects) that are presented to the user. The *integration subsystem* provides additional knowledge and information from the existing infrastructure.

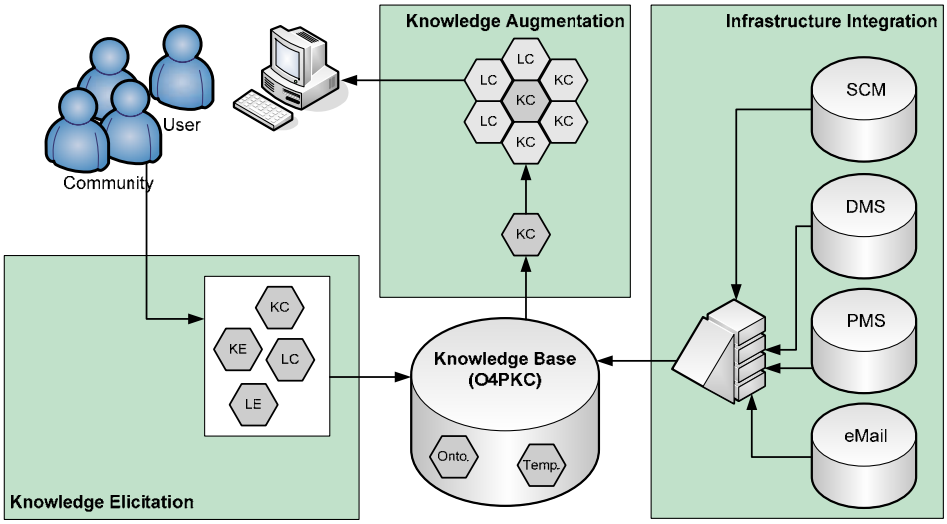

Figure 11. Basic Architecture of the Riki system (*KC: Knowledge Component, LC: Learning Component, KE: Knowledge Element, LE: Learning Element, Onto: Ontology, Temp: Template*)

5 RISE in Retrospective

In this section, we will raise requirements and lessons learned from the development and application of a Riki in the context of two SMEs that one should keep in mind when building such a system.

5.1 Lessons Learned and Implications for Developing a Riki

During the RISE project, while building two KM systems for SMEs in the software industry, we developed the Riki technology and RIME methodology. From the development and evaluation, we extracted several lessons learned that are stated in the following:

- The KM system should be *tailored* to the characteristics and needs of the organization, its projects, and the target group(s). The process where knowledge is generated, how it is recorded, and who reuses it should be considered. In a medium to large organization (i.e., more than 100 persons), the knowledge should be far more general and self-sufficient than in a small organization (e.g., with 5 persons). The larger the organization, the more probable it is that the knowledge is used by persons with total different context, from another culture, or without the opportunity to ask the author.
- The KM system should be *integrated* into the existing tool landscape and socio-technical infrastructure of the organization. It should represent either integrated relevant information from other systems as a single point of consistent knowledge (SPOCK) or offer links to these other systems. Yet, another system that is used to store information will only be used at the start and then vanish over time as nobody cares to store information in multiple systems.
- The knowledge within the knowledge base should be *goal-oriented*. The author should have a clear organizational, project-specific, or personal goal in mind and identify the problem, potential solutions, and side-effects. A simple observational experience that does not inform the reader about a specific problem or solution is of low interest and use in a KM system.
- The knowledge within the knowledge base should allow *individual structuring* by the users. Everybody structures his knowledge individually based on his personal world view (in German, “weltbild”), which needs to be mapped to the general ontology as implemented in the KM system. This can be realized either by giving all the rights to insert personal links or by mapping between personal, project-specific, departmental, organizational, or community-specific ontologies.
- The knowledge in a knowledge base should be versioned and managed using *authorized groups*. For example, knowledge about marketing processes should only be modifiable by marketing personnel, while knowledge about the interface of two departments (e.g., travel application) should be modifiable by all those involved.
- The plugin-based design and development of a KM system allows defining development tasks for separate subgroups of the project. This enables the independent implementation of functionality and improves the further extension of the system. In contrast to a monolithic system architecture, this furthermore allows the independent testing of plugins, thereby reducing development risks.

- By abstracting from the actual implementation and by implementing plugins independently from other plugins, this architecture allows using different configurations of plugins for tailored KM systems. This is particularly important when functionality is provided by different partners that should work independently even after the project.

5.2 Patterns and Anti-Patterns in Knowledge Management

In the 1990s a new concept was transferred from architecture to computer science, which helped to represent typical and reoccurring patterns of good and bad software architectures. These design patterns (Gamma et al., 1994) and anti-patterns (Brown et al., 1998) were the beginning of the description of many patterns in diverse software phases and products. Today, we have thousands of patterns (Rising, 2000) for topics such as agile software projects (Andrea et al., 2002) or pedagogies (http://www.pedagogicalpatterns.org/) (Abreu, 1997; Fincher & Utting, 2002). Many other patterns are stored in pattern repositories such as the Portland pattern repository (PPR, 2005) or the Hillside pattern library (HPL, 2005), and are being continuously expanded via conferences such as PLOP (Pattern Languages of Programming; see http://hillside.net/conferences/).

While there are similar concepts such as barriers (Riege, 2005; Sun & Scott, 2005) and incentives (Ravindran & Sarkar, 2000) in KM and software reuse (Judicibus, 1996), the identification of patterns seems to be underdeveloped and informal.

We will use the concept of patterns and antipatterns to describe the experience and knowledge we acquired during RISE and several other projects such as indiGo (Rech et al., 2005), V(I)SEK (Feldmann & Pizka, 2003), or ESERNET (Jedlitschka & Ciolkowski, 2004). In software engineering, design patterns are defined as follows:

- **Design pattern**: A design pattern is a general, proven, and beneficial solution to a common, reoccurring problem in software design. Built upon similar experiences, they represent "best-practices" about how to structure or build a software architecture. An example is the façade pattern that recommends encapsulating a complex subsystem and only allows the connection via a single interface (or "façade") class. This enables the easy exchange and modification of the subsystem.

By transferring the concept of patterns to knowledge management, we therefore define knowledge and knowledge management patterns as follows:

- **Knowledge Pattern**: A knowledge pattern is a general, proven, and beneficial solution to a common, reoccurring problem in knowledge design, i.e., the structuring and composition of the knowledge (e.g., on or via Wiki pages) or the ontology defining metadata and potential relationships between Knowledge Components.
- **Knowledge Management Pattern**: A knowledge management pattern is a general, proven, and beneficial solution to a common, reoccurring problem in knowledge management, i.e., the implementation, interconnection, or interface of technical knowledge management systems (e.g., a Wiki system) as well as social methods or systems to foster knowledge elicitation, exchange, or comprehension.

Furthermore, we cluster the patterns into six groups ranging from KM system (Wiki) patterns via content patterns to KM maintenance patterns. We describe several patterns

and anti-patterns from three of these groups. In the following, the format to describe these patterns consists of the pattern name, the description of the problem, the solutions or countermeasures, and causes or consequences. While patterns typically state and emphasize a single solution to multiple problems, antipatterns typically state and emphasize a single problem to multiple solutions.

5.2.1 Knowledge Content Patterns & Antipatterns

These patterns and antipatterns apply to the content of Knowledge Components or Elements and are typically used from the viewpoint of the reader or writer.

Name	***Knowledge Blob Antipattern***
Problem	The description of an experience or Knowledge Component get's larger and larger over time and subsumes more and more information. The search for an arbitrary Knowledge Component will often include the knowledge blob. The knowledge blob can be used for different problems, has multiple solutions or contact data.
Solution / Countermeasures	• Compact knowledge: Summarize and rewrite the knowledge in a shorter form on one page. • Extract Elements: Apply divide & conquer to create several mutually exclusive pages with parts of the original page. • Extract Commonalities: Find elements on other pages with overlapping knowledge and extract this overlapping element from both (or all) pages to a new page.
Causes / Consequences	The KM system makes it easy to find and change (extend) a Knowledge Component; the users are not sensitized to create individual experiences; or there is no maintenance of the knowledge in the KM system.

Name	***Redundant Information Antipattern***
Problem	The description of an experience or Knowledge Component is too long and has information that is either not relevant to the topic, already stored elsewhere, or outdated. The reader has to read more to get little relevant information, which might lead to an abandoned system. Furthermore, the description is longer than one page in the KM system and requires that the user scrolls (and has to interrupt his learning mode).
Solution / Countermeasures	• Compact Knowledge: Summarize and rewrite the knowledge in a shorter form on one page • Offer Templates: Find all Knowledge Components of a specific type and offer a distinct template for every type.
Causes / Consequences	The writer does not really know what to describe in order to produce a simple, short and comprehensive Knowledge Component.

Name	***Unnecessary Breakdown Antipattern***
Problem	Multiple pages are used to describe one topic that are not reusable for other knowledge descriptions, and all have to be read to understand the knowledge. The reader has to read several pages in order to understand the

	knowledge; he interrupts his learning mode and might interrupt the learning activity altogether. Furthermore, a search on the knowledge base might return only a page within this knowledge chain.
Solution / Countermeasures	• Compact knowledge: Summarize and rewrite the knowledge in a shorter form on one page. • See Explicit Start Pattern
Causes / Consequences	The writer does not really know what to describe in order to produce a simple, short and comprehensive Knowledge Component.

5.2.2 Knowledge Maintenance Patterns & Anti-Patterns

These patterns and anti-patterns apply to the maintenance of Knowledge Components or elements and are typically used by the knowledge maintainer or gardener.

Name	***Duplicated Knowledge Anti-Pattern***
Problem	Multiple versions of the same information reside in different locations in the knowledge base. The change of one piece of information causes changes to be made on several pages of different Knowledge Components. If not all replications are changed as well, multiple, slightly different versions might exist in the knowledge base.
Solution / Countermeasures	• Compact Knowledge: Summarize and rewrite the knowledge in a shorter form on one page • Extract Commonalities: Find elements on other pages with overlapping knowledge and extract this overlapping element from both (or all) pages to a new page.
Causes / Consequences	Writers are not aware of or do not care about similar knowledge. Furthermore, either the knowledge base is not cleaned up from time to time, or similar Knowledge Components are not aggregated.

Name	***Dead Knowledge Antipattern***
Problem	Knowledge is considered useless, is not reused anymore by the users, and wastes space in the knowledge base or computational power (e.g., in search algorithms).
Solution / Countermeasures	• Fuse knowledge: Find a similar and “non-dead” Knowledge Component and integrate the remaining useful information (i.e., combine, compact, or rewrite their descriptions). • Forget knowledge: Remove the knowledge from the knowledge base (maybe after an inspection by possibly interested parties).
Causes / Consequences	The knowledge is outdated, too specific, or too general.

Name	***Undead Knowledge Antipattern***
Problem	Knowledge is not used anymore by the system and undiscoverable by the users. While it might be useful to the users it, can not be reused anymore and wastes space or computational power (e.g., in search algorithms).
Solution /	• Reintegrate Knowledge: Reintegrate the component in the

Countermeasures	search index or an applicable navigational structure. • Fuse Knowledge: Find a similar and "non-dead" Knowledge Component and fuse them together (i.e., combine, compact, or rewrite their descriptions). • Forget Knowledge: Remove the knowledge from the knowledge base (maybe after an inspection by possibly interested parties).
Causes / Consequences	The knowledge is not linked anymore and does not show up in any navigational structures or search results.

6 Conclusion

We have shown that reuse in software engineering needs support in order to work in agile software organizations. Poor documentation and management of knowledge, experiences, decisions, or architectural information accompanies the development of software with agile methods in distributed software organizations. The Wiki technology promises a lightweight solution to capture, organize, and distribute knowledge that emerges and is needed fast in agile software organizations.

The RISE framework sketches our approach for agile reuse and tackles several problems in traditional KM and agile software organizations. Semi-automatic indexing of pages improves the retrieval and enables the semi-automatic creation and evolution of ontologies from Wikis (i.e., Wikitologies). The cooperative adaptation of knowledge to community needs, and the didactic augmentation of the content and interface are targeted to improve the usability of lightweight KM applications in agile environments.

As a basis, we are using Wikis as repositories with fast and liberal access to deposit, mature, and reuse experiences made in agile projects. Our next step is the design and implementation of additional functionality to Wikis with a first version targeted for 2006. In the context of our project, we pursue the following research questions:

- *Are free structures of knowledge and hierarchies more accepted by the users than fixed structures?* A long-term goal would be the development of dynamic or individual structures based on personal arrangement of documents.
- *Does the extraction of information from existing sources (e.g., versioning, defect tracking, etc.) improve the integration and interrelation of knowledge*? This will improve the access to experts and knowledge-carriers and will facilitate the build-up of goal-oriented face-to-face communication.
- *Does the didactical augmentation of knowledge improve the understandability and applicability of the knowledge, compared to conventional, non-enriched knowledge descriptions?* A long-term goal is to improve the mechanism for creating Learning Spaces by considering different instructional designs tailored to software engineers.
- *Is Wiki-based management of knowledge better accepted by the users than classical knowledge management applications in agile processes?* Classical knowledge management applications need a well-defined process to be integrated. Wikis – in particular, if enhanced by ontologies – might provide a solution for agile and hence less structured processes.

By using a plugin-based architecture, the Riki system represents a flexible and expandable infrastructure to support reuse in software organizations of different shapes and sizes. The main variability mechanism in this infrastructure is realized by using the plugins as independent services. Currently, this includes only functional aspects of the

developed system, but we plan to adapt this idea to knowledge inside the Riki system to improve the reusability of knowledge across software development organizations. By bundling content and functionality, additional complexity is introduced concerning the variability mechanisms needed to establish this product line. This interrelation between content and functionality is subject of the research area of knowledge product lines. The RISE project will provide a first step into this research area.

7 Future Trends

This section discusses future and emerging trends and provides insights about the future of knowledge transfer and reuse in software engineering. A system for knowledge reuse and transfer such as the Riki might not only be used in software engineering but also in other domains.

- *OSS reuse - from code to content*: Currently, most of the reuse within OSS is focused on software code. However, with Wikipedia and WikiCommons, there is a growing amount of content (such as music, videos, or research results) available under an Open Source style license. This content will help to overcome the initial seeding problem observed in current reuse systems.
- *Bi-directional openness*: Through open standards and APIs (Web 2.0, Semantic Web), future reuse systems can rely on content and functionality already available to the public. For example, code search engines like Koders.com or Krugle.com can be integrated in reuse systems to provide reusable (code) artifacts from outside the organization.
- *Pro-active suggesting instead of searching*: Future reuse systems will make even more use of the context of a user than depicted in Riki. Based on the metadata and their underlying ontologies, inference is done to support the user in generating more high quality knowledge. In particular, showing similar and relevant content reduces redundancy because people are aware of available content and do not create the content from scratch. Hence, the content is subject to continuous evolution.
- *Amount of guidance during learning*: Riki provides a first significant step towards the integration of e-Learning and knowledge management. Nevertheless, several problems remain to be solved and addressed. Riki intends for users to learn at their own pace and decide which content is suitable. Self-directed learning requires that the system provides a certain amount of guidance and support during learning. For example, experts need a different kind and amount of guidance than novice people.
- *Support for situated learning*: These approaches developed mainly at the end of the 1980s emphasize that a human's tasks always depend on the situation they are performed in, i.e., they are influenced by the characteristics and relationships of the context (Brown et al., 1989). Because of the relation between cognition and context, knowledge and the cognitive activities meant to create, adapt, and restructure the knowledge cannot be seen as isolated psychological products – they all depend on the situation in which they take place. This means that the Riki has to gather more context information in order to tailor the Learning Space for situated learning.

Further barriers to integrating learning management and knowledge management were identified during the LOKMOL2005 workshop (Ras et al., 2005). Examples are the lack

of interactivity, lack of dynamic adaptation of content, or adequate presentation of content.

To cope with these problems, we see the need for further research and development in the following directions:

- *Ontology usage and development*: The content of reuse systems needs to be indexed according to currently available ontologies such as SWEBOK, Dublin Core, and FOAF, in order to make it available to inference support. This indexing should be done automatically wherever possible. Based on the experience gained during this indexing, the need for further software ontologies can be derived (e.g., an ontology of software engineering artifacts).
- *Integrated context models*: Besides indexing content according to ontologies, models for describing content information are another issue to be addressed further in the future. Based on context information, knowledge and Learning Space can be tailored to the current situation. Different approaches in software engineering exist for describing domains and context. However, they mostly focus on one context dimension (e.g., organizational context, group context, activity context, project context, product context, individual or process context). Context models have to be developed that integrate the different dimensions in order to tailor the content delivery (e.g., by Learning Space) to the current situation and needs of the software engineer. Standards such as AttentionXML will play a bigger role in context description in the future (see http://developers.technorati.com/wiki/attentionxml).
- *Integrated user models*: In order to provide user tailored content, we need to know about the users' activities, their competence profiles, their learning and working preferences, their roles, and their relationships to other people and teams. The first challenge is to integrate this information in a standardized user model and the second one is to investigate how this information can be gathered automatically during daily work.

8 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. We thank our colleagues Bertin Klein, Christian Höcht, Lars Kilian, Volker Haas, and Ralph Trapphöner as well as Prof. Klaus-Dieter Althoff, Dr. Markus Nick, Ludger van Elst, Heiko Maus, and Dr. Ingeborg Schüssler for their ideas during the first phases of the project.

9 References

Abreu, F. B. E. (1997). Pedagogical patterns: picking up the design patterns approach. Object Expert, UK * vol 2 (March April 1997), no 3, p 37, 41, 3 refs.

Alrubaie, M. (2006). Trac Tags / What are tags, from http://dev.muness.textdriven.com/trac.cgi/wiki/tags

Anderson, J. R. (1993). Rules of the mind. Hillsdale, N.J.: L. Erlbaum Associates.

Anderson, L. W., & Krathwohl, D. R. (2001). A taxonomy for learning, teaching, and assessing : a revision of Bloom's taxonomy of educational objectives (Complete ed.). New York: Longman.

Andrea, J., Meszaros, G., & Smith, S. (2002). Catalog of XP Project 'Smells'. Paper presented at the 3rd International Conference on XP and Agile Processes in Software Engineering (XP 2002), Alghero, Sardinia, Italy.

Aumüller, D. (2005). SHAWN: Structure Helps a Wiki Navigate. Retrieved 29.9.05, 2005, from http://the.navigable.info/2005/aumueller05shawn.pdf

Basili, V. R., Caldiera, G., & Cantone, G. (1992). A reference architecture for the component factory. ACM Transactions on Software Engineering and Methodology, 1(1), 53-80.

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., Costa, P., Lindvall, M., Mendonca, M., Seaman, C., Tesoriero, R., et al. (2002). An experience management system for a software engineering research organization. Paper presented at the Proceedings of the 26th Annual NASA Goddard Software Engineering Workshop, 2001.

Basili, V. R., & Rombach, H. D. (1991). Support for Comprehensive Reuse. Software Engineering Journal, 6(5), 303-316.

Bergmann, R. (2002). Experience management : foundations, development methodology, and Internet-based applications, from http://link.springer-ny.com/link/service/series/0558/tocs/t2432.htmAvailability: Check the catalogs in your library. Libraries worldwide that own item: 164More About This In: Books in Print (Publication status and optional reviews)

http://link.springer-ny.com/link/service/series/0558/tocs/t2432.htm Note: Restricted to Springer LINK subscribers

Berners-Lee, T.Semantic Web Layer Cake. Retrieved 29.09.05, 2005, from http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html

Bloom, B. S. e., Engelhart, M. D., Furst, E. J., Hill, W. H., & Krathwohl, D. R. (1956). Taxonomy of educational objectives; the classification of educational goals (1st ed.). New York: Longmans Green.

Brickley, D., & Guha, R. V. (2004). RDF Vocabulary Description Language 1.0: RDF Schema, from

http://www.w3.org/TR/2004/REC-rdf-schema-20040210/

Brown, J. S., Collins, A., & Duguid, P. (1989). Situated cognition and the culture of learning (No. 481). Champaign, Ill.: University of Illinois at Urbana-Champaign.

Brown, W. J., Malveau, R. C., McCormick, H. W., & Mowbray, T. J. (1998). AntiPatterns: refactoring software, architectures, and projects in crisis. New York: John Wiley & Sons, Inc.

Bruner, J. S. (1973). Beyond the information given; studies in the psychology of knowing (1st]. ed.). New York: Norton.

Collins, A., Brown, J. S., & Newman, S. E. (1989). Cognitive Apprenticeship: teaching the crafts of reading, writing and mathematics. In L. B. Resnick (Ed.), Knowing, Learning and Instruction: Essays in honor of Robert Glaser (pp. 453-494): Hillsdale, NJ: Lawrence Erlbaum Associates.

Consortium, R. P. (2005). RISE Homepage, from http://www.rise-it.info

Cunningham, W. (2005). Wiki Design Principles, from http://c2.com/cgi/wiki?WikiDesignPrinciples

Dahl, I., & Eisenbach, M. (2005). Anwendung: Semantic Wikis. Unpublished Seminal Thesis, Karlsruhe.

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 KMDAP Workshop 2005: Knowledge Management for Distributed Agile Processes, Kaiserslautern, Germany.

Decker, B., Rech, J., Althoff, K.-D., Klotz, A., Leopold, E., & Voss, A. (2005). eParticipative Process Learning - Process-oriented Experience Management and Conflict Solving. Data & Knowledge Engineering, 52(1), 5-31.

Dupuis, R., Bourque, P., & Abran, A. (2003). Swebok Guide: An Overview of Trial Usages in the Field of Education. Proceedings - Frontiers in Education Conference. 3.

EclipseWiki. (2005). EclipseWiki Website. Retrieved 6. Oct., 2005, from http://eclipsewiki.sourceforge.net/

Enns, C. Z. (1993). Integrating Separate and Connected Knowing: The Experiential Learning Model. Teaching of Psychology, 20(1), 7-13.

Ericsson, K. A., Krampe, R. T., & Tesch-Romer, C. (1993). The Role of Deliberate Practice in the Acquisition of Expert Performance. Psychological review. 100, no. 3.

Feldmann, R. L., & Pizka, M. (2003, 6 Aug. 2002). An on-line software engineering repository for Germany's SME - an experience report. Paper presented at the Advances in Learning Software Organizations. 4th International Workshop, LSO 2002, Chicago, IL, USA.

Fincher, S., & Utting, I. (2002). Pedagogical patterns: their place in the genre. SIGCSE Bulletin, USA * vol 34 (Sept. 2002), no 3, p 199 202, 18 refs.

Friesen, N. (2001). What are Learning Objects? Interactive Learning Environments, 9(3).

Gagné, R. M., Briggs, L. J., & Wager, W. W. (1988). Principles of instructional design (3rd ed.). Fort Worth: Holt Rinehart and Winston.

Gamma, E., Richard, H., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software (3rd printing ed. Vol. 5): Addison-Wesley.

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.

HPL. (2005). Hillside Pattern Library. Retrieved 10. Oct., 2005, from http://hillside.net/patterns/

Jedlitschka, A., Althoff, K.-D., Decker, B., Hartkopf, S., Nick, M., & Rech, J. (2002). The Fraunhofer IESE Experience Management System. KI, 16(1), 70-73.

Jedlitschka, A., & Ciolkowski, M. (2004, 19-20 Aug. 2004). Towards evidence in software engineering. Paper presented at the International Symposium on Empirical Software Engineering, Redondo Beach, CA, USA.

Jedlitschka, A., & Nick, M. (2003). Software Engineering Knowledge Repositories. Lecture notes in computer science. no. 2765.

John, M., Jugel, M., & Schmidt, S. (2005). Software development documentation – A solution for an unsolved problem?,. Paper presented at the international conference on agility, Otaniemi, Finnland.

Judicibus, D. d. (1996, 8-9 Jan. 1996). Reuse: A cultural change. Paper presented at the Proceedings of the International Workshop on Systematic Reuse: Issues in Initiating and Improving a Reuse Program, Liverpool, UK.

Klusch, M. (2001). Information Agent Technology for the Internet: a Survey. Data & Knowledge Engineering Archive, 36(3), 337-372.

Kolb, D. A. (1984). Experiential learning : experience as the source of learning and development. Englewood Cliffs, N.J.: Prentice-Hall.

Legrand, S., Tyrväinen, P., & Saarikoski, H. (2003). Bridging the Word Disambiguation Gap with the Help of OWL and

Semantic Web Ontologies. Paper presented at the EROLAN 2003, the Semantic Web and Language Technology, Budapest.

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

Manola, F., & Miller, E. (2004). RDF Primer, from http://www.w3.org/TR/2004/REC-rdf-primer-20040210/

Mason, J. (2005). From e-learning to e-knowledge. In M. Rao (Ed.), Knowledge Management Tools and Techniques (Paperback ed., pp. 320-328). London: Elsevier.

Maurer, F. (2002). Supporting Distributed Extreme Programming. Paper presented at the XP Agile Universe.

McIllroy, M. D. (1968, 7th to 11th October 1968). Mass-produced Software Components. Paper presented at the NATO Conference on Software Engineering, Garmisch, Germany.

Merriënboer, J. J. G. v. (1997). Training complex cognitive skills : a Four-Component Instructional Design model for technical training. Englewood Cliffs, N.J.: Educational Technology Publications.

Merrill, M. D. (2000). First principles of instruction. Paper presented at the International conference of the Association for Educational Communications and Technology (AECT), Denver.

Mohan, P., & Daniel, B. (2004). The Learning Objects' Approach: Challenges and Opportunities. Paper presented at the E-Learn 2004, World Conference on E-Learning in Corporate, Government, Healthcare & Higher Education, Washington D.C., USA.

Naur, P., & Randell, B. (1968). Software Engineering: Report of a conference. Garmisch, Germany: sponsored by the NATO Science Committee.

Nick, M. (2005). Experience Maintenance through Closed-Loop Feedback. Unpublished PhD thesis, Technical University of Kaiserslautern, Kaiserslautern.

Nonaka, I., & Takeuchi, H. (1995). The Knowledge-Creating Company. New York: Oxford University Press.

North, K. (2002). Wissensorientierte Unternehmensführung : Wertschöpfung durch Wissen (3., aktualisierte und erw. Aufl. ed.). Wiesbaden: Gabler.

Oezbek, C. (2005). WikiDoc Homepage. Retrieved 28.9.05, 2005, from http://www.inf.fu-berlin.de/~oezbek/

Paech, B., & Kohler, K. (2003). Task-driven Requirements in Object-oriented Development. In J. H. J. C. D. Sampaio do Prado Leite (Ed.), Perspectives on Software Requirements.

Palmer, S. B. (2005). RDFWiki Homepage. Retrieved 29.9.05, 2005, from http://infomesh.net/2001/05/sw/#rdfwik

Platypus. (2005). Platypus Wiki Wiki Website. Retrieved 6. Oct., 2005, from http://platypuswiki.sourceforge.net/

PPR. (2005). Portland Pattern Repository. Retrieved 10. Oct., 2005, from http://c2.com/ppr/, http://en.wikipedia.org/wiki/Portland_Pattern_Repository

Ras, E., Memmel, M., & Weibelzahl, S. (2005). Integration of E-Learning and Knowledge Management - Barriers, Solutions and Future Issues. Paper presented at the Professional Knowledge Management (WM2005).

Ras, E., & Weibelzahl, S. (2004). Embedding Experiences in Micro-didactical Arrangements. Paper presented at the 6th International Workshop on Advances in Learning Software Organisations, Banff, Canada.

Ravindran, S., & Sarkar, S. (2000). Incentives and mechanisms for intra-organizational knowledge sharing. Proceedings of 2000 Information Resources Management Association International Conference, Anchorage, AK, USA, 21 24 May 2000 * Hershey, PA, USA: Idea Group Publishing, 2000, p 858.

Rech, J., Decker, B., Althoff, K.-D., Voss, A., Klotz, A., & Leopold, E. (2005). Distributed Participative Knowledge Management: The indiGo System. In R. A. Ajami & M. M. Bear (Eds.), Global Entrepreneurship and Knowledge Management: Local Innovations and Value Creation. Binghamton (New York): Haworth Press.

Rhizome. (2005). Rhizome Website. Retrieved 6. Oct., 2005, from http://rx4rdf.liminalzone.org/

Riege, A. (2005). Three-dozen knowledge-sharing barriers managers must consider. Journal of Knowledge Management, UK * vol 9 (2005), no 3, p 18 35, 80 refs.

Rising, L. (2000). The pattern almanac 2000. Boston: Addison-Wesley.

Robbins, J. (2005). Readyset Requirements Specification Template, from http://readyset.tigris.org/

Robertson, J., & Robertson, S. (2005a). Volere Requirements Specification Template, Version 10.1. Retrieved 10. Oct., 2005, from http://www.volere.co.uk/template.htm

Robertson, S., & Robertson, J. (2005b). Volere Requirements Specification Template, from http://www.volere.co.uk/

Ruhe, G., & Bomarius, F. (1999, June 16-19, 1999). Proceedings of Learning software organizations (LSO): methodology and applications. Paper presented at the 11th International Conference on Software Engineering and Knowledge Engineering, SEKE'99, Kaiserslautern, Germany.

Rus, I., & Lindvall, M. (2002). Knowledge Management - Knowledge Management in Software Engineering - Guest Editors' Introduction. IEEE software. 19, no. 3.

Schank, R. C., Bermann, T. R., & Macperson, K. A. (1999). Learning by Doing. In R. R. C. (Ed.), Instructional Design Theories and Models: A New Paradigm of Instructional Theory (Vol. II, pp. 161-181). Mahwah: NJ: Lawrence Erlbaum Associates.

Schreiber, G., Wielinga, B., Hoog, R. d., Akkermans, H., & Velde, W. V. d. (1994). CommonKADS: A Comprehensive Methodology for KBS Development IEEE Expert: Intelligent Systems and Their Applications 9 (6), 28-37

Secchi, P., Ciaschi, R., & Spence, D. (1999). A Concept for an ESA lessons learned system (No. Tech. Rep. WPP-167). Noordwijk: The Netherlands: ESTEC.

Self, J. (1992). Computational Mathetics: the missing link in Intelligent Tutoring Systems re-search? Directions in Intelligent Tutoring Systems (No. 91).

Semantic Wikis. (2005). Semantic Wiki Overview. Retrieved 6. Oct., 2005, from http://c2.com/cgi/wiki?SemanticWikiWikiWeb

Senge, P. M. (1990). The fifth discipline : the art and practice of the learning organization (1st ed.). New York: Doubleday/Currency.

Simons, C. L., Parmee, I. C., & Coward, P. D. (2003). 35 years on: to what extent has software engineering design achieved its goals? IEE Proceedings Software, 150(6), 337-350.

Smith, M. K., Welty, C., & McGuinness, D. L. (2004). OWL Web Ontology Language Guide, from

http://www.w3.org/TR/2004/REC-owl-guide-20040210/

Software Engineering Body of Knowledge, Iron Man Version. (2004). from http://www.swebok.org/ironman/pdf/Swebok_Ironman_June_23_%202004.pdf

Souzis, A. (2001). Rhizome Position Paper. Retrieved 29.9.05, 2005, from Adam Souzis

Stenmark, D. (2001). The Relationship between Information and Knowledge. Paper presented at the IRIS-24, Ulvik, Norway.

SubWiki. (2005). SubWiki Website. Retrieved 6. Oct., 2005, from http://subwiki.tigris.org/

SUMO Ontology. from http://ontology.teknowledge.com/

Sun, P. Y. T., & Scott, J. L. (2005). An investigation of barriers to knowledge transfer. Journal of Knowledge Management, UK * vol 9 (2005), no 2, p 75 90, 45 refs.

Szyperski, C., & Pfister, C. (1997). Workshop on Component-Oriented Programming, Summary. Paper presented at the ECOOP96.

Tautz, C. (2001). Customizing Software Engineering Experience Management Systems to Organizational Needs. Unpublished PhD thesis, University of Kaiserslautern, Kaiserslautern.

Tennyson, R. D., & Rasch, M. (1988). Linking Cognitive learning theory to instructional prescriptions. Instructional Science, 17, 369-385.

TikiWiki. (2005). TikiWiki Website. Retrieved 6. Oct., 2005, from http://www.tikiwiki.org

Volere. (2005). Requirements Tools, from http://www.volere.co.uk/tools.htm

Völkel, M., Schaffert, S., Kiesel, M., Oren, E., & Decker, B. (2005). Semantic Wiki State of the Art Paper, from http://wiki.ontoworld.org/index.php/Semantic_Wiki_State_of_The_Art_Paper

Weber, R., Aha, D. W., & Becerra-Fernandez, I. (2001). Intelligent lessons learned systems. Expert systems with applications. 20, no. 1.

WikiEngines. (2005). WikiEngines Compilation. Retrieved 6. Oct., 2005, from http://www.c2.com/cgi/wiki?WikiEngines

WikiMatrix. (2006). WikiMatrix - Overview of Wikis.

Wikipedia. (2006). Your first article, from http://en.wikipedia.org/w/index.php?title=Wikipedia:Your_first_article&oldid=41631731

Wooldridge, M., & Jennings, N. R. (1995). Intelligent agents: theory and practice. Knowledge Engineering Review, 10(2), 115-152.

XWiki. (2005). XWiki Website. Retrieved 6. Oct., 2005, from http://www.xwiki.org/xwiki/bin/view/Main/WebHome

Yacci, M. (1999). The Knowledge Warehouse: Reusing Knowledge Components. Performance Improvement Quarterly, 12(3), 132-140.

10 Internet Session: Knowledge and Software Reuse

http://www.sei.cmu.edu/productlines/index.html

Software product lines are a way to describe commonalities and variabilities in a software system that are used in different contexts by different groups of people such as embedded software systems on mobile phones (each with different hardware characteristics and software features).

Interaction:

Survey the information presented at the websites on product-line software engineering (PLSE) and software reuse methods and theory. Prepare a brief presentation on the core concepts and history of software reuse with a focus on PLSE. Alternatively, transfer the ideas behind PLSE to knowledge or learning management and assume that "software = knowledge" or "software = course". How would a "knowledge product line" or "course product line" look like? Are there commonalities or variabilites in KM/LM systems or the knowledge/courses itself?

11 Useful URLs

- RISE: Website of the RISE project, http://www.rise-it.info
- SWEBOK: The software engineering body of knowledge with more information on SE and software reuse (see page 4-4), http://www.swebok.org/
- ICSR-09: The ninth bi-annual international conference on software reuse on June 12-15, 2006 in Torino, Italy, http://softeng.polito.it/ICSR9/
- Lombard-Hill's bibliography: The largest bibliography on literature about software reuse, http://www.lombardhill.com/biblio1.html
- The TOA bibliography: A similar large bibliography, http://www.toa.com/pub/reusebib.htm
- Sverker Janson: Internet survey on Software Agents and Agent-based Systems, http://www.sics.se/isl/abc/survey.html

- A collection of arguments why Wikis work, http://c2.com/cgi/wiki?WhyWikiWorks and http://en.wikipedia.org/wiki/Wikipedia:Our_Replies_to_Our_Critics
- An overview of Wikis with a comparison feature: www.wikimatrix.org
- Different types of knowledge that might be taken into consideration when building a KM system (e.g., for knowledge flow descriptions or templates in a KM system) http://www.knowledge-sharing.com/TypesOfKnowledge.htm
- Pattern in general: Descriptions of patterns with links to patterns in architecture http://en.wikipedia.org/wiki/Patterns
- Software patterns: Starting page with information about patterns in software engineering, http://en.wikipedia.org/wiki/Design_pattern_%28computer_science%29
- Pedagogical patterns for seminars and teaching, http://www.pedagogicalpatterns.org/
- The Hillside Pattern Library, http://hillside.net/
- The Portland pattern repository, http://c2.com/ppr/, http://en.wikipedia.org/wiki/Portland_Pattern_Repository
- Classification and references for patterns based on the book "The pattern Almanac", http://www.smallmemory.com/almanac/
- Hillside Pattern bibliography, http://hillside.net/patterns/papersbibliographys.htm
- Quality overview: The paper "Construction of a systemic quality model for evaluating a software product" gives a nice overview about several software quality models, http://www.lisi.usb.ve/publicaciones/SQJ%2011%203%202003%20Ortega%20Perez%20and%20Rojas.pdf
- ISO 9126: Part 1 of the Software Quality standard with a focus on Quality models, http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=22749
- Dromey's quality model in the paper "A Model for Software Product Quality", http://www.sqi.gu.edu.au/docs/sqi/technical/Model_For_S_W_Prod_Qual.pdf

12 Further Readings

- Steffen Staab, Rudi Studer (Eds.): Handbook on Ontologies. International Handbooks on Information Systems Springer 2004, ISBN 3-540-40834-7
- D. Fensel: Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce. Springer-Verlag, Berlin, 2000.
- Cunningham, Ward und Leuf, Bo: The Wiki Way. Collaboration and Sharing on the Internet. Addison-Wesley (2001). ISBN 020171499X
- H. Mili, A. Mili, S. Yacoub, E. Addy, "Reuse-based Software Engineering", John Wiley & Sons, Inc., 2002
- I. Jacobson, M. Griss, P. Jonsson, "Software Reuse: Architecture, Process and Organization for Business Success," ACM Press, 1997.

- J. Sametinger, "Software Engineering with Reusable Components," Springer-Verlag, ISBN 3-540-62695-6, 1997.
- E. Karlsson, "Software Reuse: A Holistic Approach," John Wiley and Sons Ltd, 1995.
- W. Schaefer, R. Prieto-Diaz, M. Matsumoto, "Software Reusability", Ellis Horwood, 1994.
- T.J. Biggerstaff, A.J. Perlis, "Software Reusability: Volume I Concepts and Models," ACM Press, 1989.
- T.J. Biggerstaff, A.J. Perlis, "Software Reusability: Volume II Applications and Experience," ACM Press, 1989.
- Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995, hardcover, 395 pages, ISBN 0201633612, Design Patterns CD, 1997 ISBN 0201634988
- Rising, L.: The Pattern Almanac 2000. Boston et al. 2000.
- Roock, S., & Lippert, M. (2005). Refactoring in Large Software Projects (Paperback ed.): John Wiley & Sons.
- Kerievsky, J. (2005). Refactoring to patterns. Boston: Addison-Wesley.
- Fowler, M. (1999). Refactoring: Improving the Design of Existing Code (1st ed.): Addison-Wesley.

13 Possible Paper Titles/ Essays

- Commonalities and variabilities of software reuse and knowledge management
- Software patterns for knowledge and knowledge management
- Knowledge management in learning software organizations: Methods and Tools
- An effective knowledge management system
- Knowledge management in software engineering: problems and research directions