Knowledge Management in Software Development

Jahn, Karsten

Publication date:
2012

Document Version
Early version, also known as pre-print

Aalborg University
Department of Computer Science

Ph.D. Thesis

Knowledge Management in Software Development

Author:
Karsten Jahn

Supervisor:
Peter Axel Nielsen

November 30, 2012

Thesis Title: Knowledge Management in Software Development

Author: Karsten Jahn

Supervisor: Professor Peter Axel Nielsen

Associated Paper: Karsten Jahn and Peter Axel Nielsen: "A Vertical Approach to Knowledge Management: Codification and Personalization in Software Processes". International Journal of Human Capital and Information Technology Professionals, Volume 2, Issue 2, Pages 26-36, 2011.

This thesis has been submitted for assessment in partial fulfilment of the PhD degree. The thesis is in part based on a published scientific paper, which is listed above. As part of the assessment, a co-author statement has been made available to the assessment committee and is also available at the faculty.

Preface

"The mere existence of knowledge somewhere in the organization is of little benefit; it becomes a valuable cooperate asset only if it is accessible, and its value increases with the level of accessibility." (Davenport and Prusak, 1998, p. 18)

Abstract

Software development is a very knowledge-intense discipline. People often work in project teams not only to bundle the powers, but also to enable easier sharing of knowledge, because the acquisition of knowledge always involves spending resources. If there is a way to utilize internally available knowledge, the company gains a competitive advantage out of it. Knowledge management is the systematic approach to enable people to share what they know. However, there is no general solution. In order to be successful, a knowledge management system always has to be customized to the environment of each case. The involved people as well as the company's organization are of high importance.

As a part of the EU-founded FP7 project "KiWi – Knowledge in a Wiki", my studies deal with the design of a knowledge management system for a software development company, whose analysis showed a number of knowledge management problems, grouped to four problems regarding *isolated islands of knowledge* and three problems regarding the *inadequate bridging of knowledge*. The analysis led to a distinction of two organizational layers within the case company: Management and development. Each of the layers follows a different strategy to share the knowledge. The management layer follows a codified strategy and the development layer a personalization strategy. These differences are the reason for several knowledge management problems, which influence the whole company.

Based on that understanding, I propose *four design ideas* on which the knowledge management system is based upon. First, the layers are separated and each is being supported with its own knowledge management strategy. Second, the layers have to be connected. That includes the people in the layers, the strategies and the realizing knowledge management systems. Third, the personalization strategy in the development layer is supported by a wiki. Fourth, the codification strategy in the management layer is supported by an enterprise system. All four design ideas form the foundation for a prototype of a large knowledge management system, composed of three sub-systems: The *KiWi platform*, a *Data Exchange Agent* and a *Project Management Application*.

This thesis presents a design study, with an analysis of a case company, the design of a knowledge management prototype, and its evaluation through the case company. I followed the *action design research* methodology, organized in iterations and focused not only on the IT artefact, but also its environment within the company.

My research contributes to different aspects of the knowledge management theory. I elaborate on the codification and personalization knowledge management strategy by presenting how to adapt more than one strategy in an organization. Further, I connect the knowledge management strategies to the knowledge bases and show how specific knowledge bases have advantages in the different strategies.

Additionally, this thesis provides the implementation and evaluation of the realization of a larger prototype dealing with the organizational knowledge management processes.

Keywords: knowledge management, knowledge management strategy, knowledge management system, software development, wiki, enterprise system, action design research

Resumé

Softwareudvikling er en meget videns-intens disciplin. Folk arbejder ofte i projektgrupper eller teams, ikke kun for at samle kræfterne, men også for at muliggøre deling af viden, da erhvervelse af viden altid indebærer brug af ressourcer. Hvis der er en måde at udnytte internt tilgængelige viden, får virksomheden en konkurrencemæssig fordel ud af det. 'Knowledge management' (vidensledelse) er den systematiske tilgang til at give folk mulighed for at dele hvad de ved. Der er imidlertid ingen generel løsning. For at blive successfuld, skal et knowledge management system altid tilpasses til miljøet i hvert enkelt tilfælde. De involverede personer, samt virksomhedens organisation, er af stor betydning.

Mine studier beskæftiger sig med design af et knowledge management system til et software udviklingsselskab. Analysen af det viste en række af knowledge management problemer, grupperet i fire problemer vedrørende isolerede øer af viden, og tre problemer i forbindelse med utilstrækkelig vidensbro. Analysen førte til en sondring mellem to organisatoriske lag i denne virksomhed: Ledelse og udvikling. Hvert af lagene følger en foskellig strategi til at dele viden. Ledelseslaget følger en kodificering strategi og udviklingslaget en personorienteret strategi. Disse forskelle er årsagen til flere vidensdelingsproblemer, som har indflydelse på hele virksomheden.

Baseret på denne forståelse, foreslår jeg *fire designideer* som knowledge management systemet er baseret på. For det første, adskilles lagene, og hvert bliver understøttet med sin egen knowledge management strategy. For det andet, skal lagene forbindes. Det omfatter menneskerne i lagene, strategierne og de realiserede knowledge management systemer. For det tredje, er personorienteret strategien i udviklingslaget understøttet af en wiki. For det fjerde er kodificering strategien i ledelseslaget understøttet af et enterprise system. Alle fire designideer at danner grundlag for en prototype af et stort knowledge management system, sammensat af tre subsystemer: *KiWi platformen*, en *Data Exchange Agent* og en *Project Management Application*.

Denne afhandling præsenterer et design studie, med en analyse af et case selskab, udformningen af en knowledge management prototype, og dens evaluering af case selskabet. Jeg fulgte en metode som hedder *design action research*, der er organiseret i iterationer, og som ikke kun fokuserer på IT-artefaktet, men også dens miljøi virksomheden.

Min forskning bidrager til forskellige aspekter af knowledge management teori. Jeg uddyber på kodificering og personorienteret knowledge management strategien ved at præsentere hvordan vi kan tilpasse mere end én strategi i en organisation. Desuden, forbinder jeg knowledge management strategier til en knowledge base og viser, hvordan specifikke knowledge baser har fordele i de forskellige strategier.

Desuden fremsætter denne afhandling gennemførelse og evaluering af realisering af en større prototype som beskæftiger sig med de organisatoriske knowledge management processer.

Acknowledgements

Being provided with the opportunity to study the topic that interests you for several years, in a very professional environment and surrounded by very skilled people is a great luxury. I want to express my appreciation of all the support and inspiration that I experienced during my PhD studies.

First of all, I want to thank Peter Axel Nielsen for believing in me. His continuous support and constructive critiques in many, many fruitful discussions made him a very valuable supervisor to me.

At Aalborg University I met an environment staffed with a number of talented people. The S+I group was at all times a great source of support and stimulation. I am grateful for the fruitful discussions, the valuable feedback and not forgetting the nice times we have had together. Thanks to all of you: Ivan Aaen, Anders Bruun, Peter Dolog, Fred Durão, Lise Tordrup Hermansen, Janne Jul Jensen, Jesper Kjeldskov, Fulvio Lizano Madriz, Andreas Munk-Madsen, Kenneth Møller Porto Nielsen, Jeni Paay, John Stouby Persson, Dimitris Raptis, Jeremy Rose, Mikael B. Skov, Henrik Sørensen, Jan Stage and Gitte Tjørnehøj.

Outside our group, I want to mention everyone else at the Department of Computer Science, especially the secretaries. Whomever I had business with was always very supportive and nice to me. Thank you all.

My research would not have been possible without the support of the FP7 project "KiWi". All the project partners helped me a great deal in discussing, shaping and realizing the ideas described in this thesis. Thanks across Europe: Andreas Blumauer, François Bry, Julia Eder, Norbert Eisinger, Szaby Grünwald, Inka Havlova, Jana Herwig, Josef Holy, Petr Knoth, Jakub Kotowski, Thomas Kurz, John Pereira, Mihai Radulescu, Peter Reiser, Matthias Samwald, Sebastian Schaffert, Thomas Schandl, Marek Schmidt, Rolf Sint, Pavel Smrž, Henry Story, Stephanie Stroka, Klara Weiand and Michael Zach.

A special role in my research played Logica. The people involved into the KiWi project were available for me at any time, engaging in discussions and providing feedback on many levels. Thanks to Daniel Grolin, Keld Pedersen and Søren Rieck. I also want to thank the developers of Logica that helped realizing the systems, Filip S. Adamsen, Bertrand Dechoux and Mogens Kraus.

During these years that I spend working on this thesis, I had the pleasure of getting to know a huge variety of impressive individuals, people with many different backgrounds. This also goes to all the wonderful people I know, that joined my quest every once in a while. Thanks for inspiring me, for engaging in fertile conversations with me and for providing unconditional moral support.

Further, I want to thank all those marvellous people that helped making Aalborg a home to me. We shared many moments to remember, in the AaB stadium, the Studenterhuset, Jomfru-Ane-Gade or wherever. This addresses a variety of lovely people, close friends for live, thank you all. However, three guys peak out: Hernan, Ricardo and Ubaldo – You are family!

Special thanks go to Wolfram Conen, who encouraged me to apply for the PhD position at Aalborg University.

Finally, I want to thank my family for putting up with me. Bernd, Brigitte, Claas, Georg, Marcus, Milo, Petra und Wibke... Danke für alles!

Karsten Jahn,
August 2012

Contents

List of Figures

List of Tables

1 Introduction

This first chapter explains the background of this PhD study and provides initial explanations to it. I describe my personal background (section 1.1) and how it led to this area of concern (section 1.2). Accordingly, I formulate my research question (section 1.3) and summarize the thesis' structure (section 1.4).

1.1 Personal Motivation

After graduating in media informatics, I was employed as an IT consultant at Valtech[1] in Germany. My working area involved mainly quality assurance, system administration and software development. However, asides from my project related tasks, I learned that the internal communication is a vital aspect of the everyday work in a software development company.

Through my everyday life I learned what knowledge management means for a company and that it is really represented through communication in different forms. This observation was not just made at Valtech itself, but also at the different companies I was involved with as a consultant. Knowledge management helps to spread specific information and background knowledge as well as to locate experts. Often the consultants asked each other for specific skills or experiences with specific technologies. Once the desired knowledge was found, the consultants could either be introduced into the project directly and thus made billable, or support each other internally. But the communication also works the other way around. Valtech organized seminars for all consultants on

[1] http://www.valtech.de/

a regular basis. These provided the opportunity for presentations of different kinds. The consultants often presented their work results, but they also talked about technologies and other activities regarding the organization of project work. As a result these meetings raised an awareness of expertise within the company.

After the presentations a timeslot was booked for all participants for regular interaction among each other. As many of them are involved at different customers or in different projects, this opportunity supports the general communication and exchange of ideas. Aside from these seminars, different other meetings were held on a regular basis. These were usually organized by different groups of people, formed according to interests.

In addition to the personal interaction, several IT systems were utilized to support the communication among the employees. Valtech's intranet consists of different systems to exchange information in multiple ways. One example for that is a wiki, which is a communication platform that allows users to work on shared sources collaboratively. Valtech makes use of its wiki as a documentation platform, allowing all consultants to view and edit the results immediately. Consultants document progress in projects and thus inform others about updates, as everyone can follow the development.

After working with the wiki for a while, I understood its value and explored its possibilities. A wiki's ability to support knowledge sharing amazed me. I helped to advance the use of the wiki. This includes technical improvements as well as new ways of using it. Eventually, I was considered a wiki expert. Based on the good experiences internally, Valtech was able to convince others to adapt this practice. Some customers bought a wiki and set up support, others needed guidance for the operation of a wiki. Either way, we organized introduction sessions and small workshops where my colleagues and I explained the utilization of wikis. Personally, I was strongly involved in this. I was part of many of these seminars, wrote a white paper about wikis in an enterprise context and held a presentation about that at an industrial conference (OOP2008).

During the three years as an IT consultant, I observed the necessity and ability of knowledge management in different software development companies. I was additionally able to recognize, comprehend and experience the power of a wiki as a knowledge management system. As an employee of Valtech I experienced knowledge management mainly approached through personal interaction with the support of a wiki. But time and possibilities for studying these fields in detail was too limited in a company that had to focus on its business. Due to my curiosity I found the way to a university again. I wanted to learn more about knowledge management approaches. I wanted to investigate the utilization of knowledge management systems.

1.2 Area of Concern

Software development is a rather recent engineering discipline that grew to become the widely spread profession that it is today. People all around the globe are grouped to create software. This is done in very different circumstances and for very different purposes. There are loosely coupled open-source developers that work free of charge as well as companies with hundreds of employees. However different they might be established, all of these organizations have a

lot in common: In each of them the most important resource is the knowledge held by the developers.

In every software development company the developers have to understand the systems and programming languages they work with. It is further important for them to know the company's guidelines and processes. Accordingly, software development is a very knowledge-intense activity. For being able to organize the knowledge beneficially, knowledge management becomes a vital task (Bjørnson and Dingsøyr, 2008). The final product is software and every step leading there is or can be realized through a computer. Hence, literally everything is digital. This aspect makes it particularly interesting to study computer-based knowledge management in software development.

In the mid-1980s the rising amount of available information and the increasing complexity of software development initiated an awareness of a need for knowledge management. This triggered researchers as well as professionals from the industry to analyse what knowledge management might be and how to support it (Rus and Lindvall, 2002).

In "The Knowledge-Creating Company" (Nonaka and Takeuchi, 1995) the authors analyse the learning and innovation process in Japanese companies. They show that knowledge creation is the result of actively processing knowledge. Especially the knowledge that is difficult to express can best be shared through demonstration and practice.

Knowledge management can therefore be best described as the approach to handle the know-how of a company. It is done to use the already existing knowledge more efficiently and to maximize the gain of it. Hence, knowledge can be a competitive advantage (Davenport and Prusak, 1998). Different technological solutions were created to facilitate the management of knowledge. These were computer programs that support the handling of the internal know-how, which were later labelled as knowledge management systems.

Much research addressed knowledge management in the recent decades. A variety of different approaches was designed, analysed and documented. It shows that knowledge management is generally structured into two aspects: Collecting the experience or knowledge of employees and learning from that collection. Only a successful combination of both aspects results in effective knowledge management.

Knowledge management can be approached in various ways. The so called experience factory (Basili, 1996) asks employees to codify their experiences and store it in a centralized database. Others can later access this in similar situations in order to learn from it. But not all knowledge management approaches cover details about the whole practice in detail. Most of them focus on different aspects, like the recording of experiences through post-mortems or processes that control the access to these reports.

As knowledge management is a complex field with different requirements for different companies, the approaches differ from one another. The company itself is of high importance. A knowledge management approach has to reflect the company's strategic orientation in order to be successful (Hansen et al., 1999). And further, it is the company's knowledge management approach that has to fit to the company, not the other way around (Davenport and Prusak, 1998). However, there are no general solutions and every knowledge management has to be customized, which should involve the whole company, including its employees, processes, culture, etc.

Knowledge management systems, the IT support for a knowledge management approach, also show a high variance, because different knowledge management approaches need different kinds of support (Rus and Lindvall, 2002). Some focus on document management, while others focus on the communication between its users. The range of systems reaches from file servers, over groupware to wikis. There are also very specialized solutions, which combine particular aspects from different systems. Finding or creating a fitting knowledge management system is one of the complexities in knowledge management.

In software development knowledge management is emphasized, because the work is very abstract and knowledge intense (Bjørnson and Dingsøyr, 2008). Every developer constantly shares their knowledge with others, extends the own knowledge and applies gained knowledge. They are additionally challenged to work in teams, where each member has different experiences and expertise. Knowledge management can thus make their work easier or more effective, especially regarding key problems, like gathering domain knowledge, decreasing the learning curve for new team members or facilitation of different technologies.

Recently, improvement of knowledge management in software development has taken two forms: Agile development and research on knowledge management systems. On one hand, agile software development is a methodology that emphasizes the necessity and value of interactions among individuals and values it higher than processes or tools[2]. The approach involves the grouping of people to engage the knowledge sharing (Larman, 2003). On the other hand, research on knowledge management systems mainly deals with the support of knowledge management through IT systems (Rus and Lindvall, 2002). The focus in this thesis is on the knowledge management systems in software development.

1.3 Research Question

Knowledge management improves the utilization of a company's internal knowledge, in order to be more efficient and beneficial. This is a general statement and counts for companies in virtually every field. However, my studies focus on knowledge management in software development.

The landscape for software development companies is defined by various different obstacles. It is fairly common that customers expect a document-driven development. This acts like an insurance for level-headed work, the people involved have to follow detailed process descriptions. This includes, that they have to report intermediate and final results in specific documents to higher management levels in the company or even to the customers.

Many projects assigned by governments in Europe (e.g., in defence or financial sectors) must be realized only by companies that fulfil the requirements of higher levels of the capability maturity model (CMM). And even if the customers do not oblige a certified maturity standard, it is often considered as a competitive advantage in the market. Accordingly, companies fulfil these standards to be more competitive.

With the aim of increasing the efficiency of the development, companies define their business processes. These can be realized differently. Either way, the goal is to describe the structure of a work procedure. Business processes define the tasks of different employees and how they have to fulfil them, in form

[2]The Agile Manifesto: http://www.agilemanifesto.org/

of a process description. Additionally, a business process contains guidance for the lines of communication. When applying such a business process, it has to be clear whom and in which way the employees have to contact after finishing a task or in case of problems with it.

Business processes are also related to the hierarchy of the company. Flat hierarchies are widely spread in software development companies these days. This involves that the management structure is reduced to a minimum in order to accelerate the decision process. The idea is, to equip the developers with the possibility to take responsibility in corporate decisions.

The communication in flat hierarchies becomes more important, as it is not covered by hierarchical responsibilities anymore. Instead, it is addressed through teams in the company, to tackle the workload. Grouping the employees improves the communication between people that work on the same tasks.

Software development gains complexity due to the fact that the work material evolves. Most applied programming languages, frameworks, tools or utilities are under constant development themselves, so that new versions are available every now and then. Sometimes, whole new technologies enter the market or the company decides to facilitate different technologies. Every change requires the developers to get familiar with the new environment.

Knowledge management could address many of these aspects. It could assist developers communicating outside their team. It could improve the decision making process in the whole company by including or applying knowledge that already exists. It could provide support for developers in learning about their field. It could help the creation and maintenance of business processes. It could support the whole documentation activities. But to achieve any of these goals, it has to be carefully designed.

In this thesis I explore the challenges of system-based knowledge management in software development companies, which leads to my research question:

> ***Research Question:*** *How can IT systems support knowledge management in software development?*

The research question has two objectives. First, it aims at gaining an understanding of what it is that knowledge management needs, with a focus on software development companies. Second, it limits the scope to system-based approaches. The goal is to design a knowledge management system that is able to support the knowledge management in software development. This includes not only a knowledge management system, but also a fitting workflow, to make the system valuable within software development.

My research is part of the EU-founded FP7 project “KiWi – Knowledge in a Wiki”.

1.4 How to read this Dissertation

This thesis’ structure follows a straight line of argumentation (figure 1). After this introduction (chapter 1) I present the findings of other research (chapters 2 and 3). Then I explain the different elements of my research (chapters 4 - 8). And finally, I oppose the two and show the impact of my work (chapters 9 and 10).

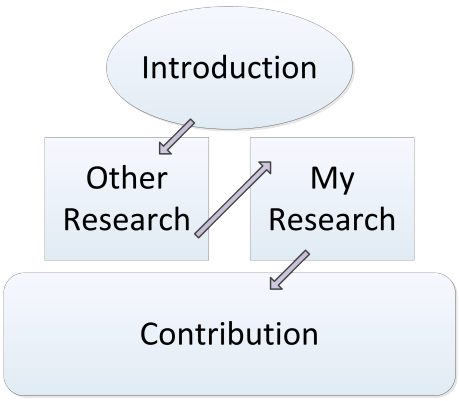


Figure 1: Structure of this Thesis

This means in detail that the related research is presented in (chapter 2), followed by my definition of the terms I use and an introduction to the theories I utilize (chapter 3). Afterwards, I describe the project setting, in which my studies take place (chapter 4), and the applied research methodology (chapter 5). Then I explain the analysis of the case company's problem (chapter 6). Subsequently, I describe the design of a prototype accordingly (chapter 7) and its evaluation (chapter 8). I close this thesis with a discussion of my research (chapter 9) and a conclusion of my studies (chapter 10).

The thesis itself deals with knowledge management. Hence, it contains a variety of concepts that begin with the term *knowledge* (e.g., knowledge management, knowledge model, knowledge sharing, knowledge management system, knowledge management strategy...). All of these are rather long words. However, to not confuse the reader, I abstain from using abbreviations. I always spell out the full term of these concepts in order to avoid missunderstandings. Granted, this thesis is not abbreviation-free, but the few exceptions applied are standard abbreviations for distinct terms (e.g., ADR, ES...). I explain all of them in the text.

Footnotes within this thesis are only used to provide the internet address to a tool or technology. This is intended to support the reader, in the case of desired additional information. All relevant information is included in the text.

2 Related Research

Knowledge management in software development has been approached by many researchers from a variety of different angles. This chapter contains my structured literature study, I analyse the published results and contributions, related to my research in this field in one way or the other.

For my literature study I needed to find research that tries to answer the same or a similar research question as I have (see section 1.3). The search for publications within this area was mainly realized through Google Scholar[3]. I used it to search for the terms "Software Development" or "Systems Development" in combination with "Knowledge Management". The list of publications was then extended with publications that were referenced to, within the same area. This process led to a large number of publications from many different outlets. I then filtered the collection of publications for relevance. That means that a publication has its focus on knowledge management in software development and in particular on IT support. Articles with a purely technical focus and no consideration for support of knowledge management or software development were excluded. After this selection process, my literature study is based on more than one hundred books, journal articles and conference proceedings in total.

This chapter describes the findings of the related research for knowledge management in software development organized in two main parts, general and applied. However, I begin with providing a bit of necessary background on knowledge and knowledge management (section 2.1). Then I explain the relevance of knowledge management in software development and the reasons for

[3] http://scholar.google.com/

knowledge management systems (section 2.2). Afterwards, I describe actual examples of knowledge management in software development (section 2.3), including case studies and specific approaches. Followed by an overview of considerations and difficulties regarding the IT support in knowledge management (section 2.4). The chapter is then concluded by a short summary (section 2.5).

2.1 Knowledge Creation and Transfer

Before I describe actual knowledge management approaches, the theories behind the publications that are covered by this literature study need to be explained. I therefore take a little detour into the general theory of knowledge management. In this section I describe the related research regarding knowledge as such and the necessary activities in knowledge management.

Note that I elaborate on the related theories and terminology in detail below (chapter 3). This section's purpose is to provide an understanding to the reader of what knowledge management means.

2.1.1 Tacit & Explicit Knowledge

When trying to make people share their experiences, it is important to be aware of the distinction between different kinds of knowledge. Polanyi defines tacit knowledge and explains it as "we can know more than we can tell" (Polanyi, 1966, p. 4). He describes that not everything that we know is conscious. Riding a bike, for instance, might be an easy task for many people, yet, most of us fail to explain how to do so. Tacit knowledge is described as the knowledge that is hard to formalize and as a result difficult to express. Explicit knowledge is the opposite of tacit knowledge and thus easy to codify or already codified.

Nonaka based his work on the knowledge states tacit and explicit (Nonaka and Takeuchi, 1995; Nonaka, 1994, 1991). He understands knowledge creation as the conversion of knowledge between these states and distinguishes between four modes of knowledge creation: Socialization, externalization, combination and internalization (figure 2). It is often referred to as the SECI model, based on the first letters of the modes.

Nonaka states that "the key to acquiring tacit knowledge is experience" (Nonaka, 1994). A focus in terms of knowledge management lies on the two conversion patters that involve both states, as they seem complementary, i.e., from explicit to tacit knowledge (internalization) and from tacit to explicit knowledge (externalization). Nonaka's model of knowledge conversion however consists of a combination of all four modes. The interchange from one to the other, through all four modes in an iterative manner, increases the knowledge of a company on different ontological dimensions. Nonaka and Takeuchi (1995) call this a *knowledge-creating company*.

The principles of the knowledge-creating company are used as a theory on which many researchers of knowledge management in software development build upon. One study elucidates the use of the SECI model in the semiconductor equipment industry (Moriya and Benton, 2008). Another one is related to software development and explains socialization and externalization for every step in a software life cycle (Wu, 2010). Some contributions use Nonaka's findings also and extend into different directions. For instance, researchers investi-

Figure 2: Modes of Knowledge Creation, from (Nonaka, 1994)

gate on possible learning strategies for three different actors (workers, managers, and human resource developers) along different project organizations (Poell and van der Krogt, 2001). Others utilized the modes of knowledge creation to create a knowledge transfer model (Liyanage et al., 2009).

Critique on Nonaka's Theories

The SECI model has been widely discussed in the different research communities that it concerns. Walsham (2005) explains that Nonaka's view considers knowledge as an object only, being transferred from one individual to another, while knowledge is much more than that. He points out that the influence of the context and the communication as such is being left out by Nonaka. Many researchers have a similar standpoint and disagree with Nonaka's SECI model.

Another point of the knowledge-creating company that is critizised by other researchers relates to the contextual constraints, which have to be handled with care as they are deep-seated in the Japanese culture and could not be simply taken over without adjustments (Glisby and Holden, 2003).

However, when managing knowledge in software development, it is important to not just focus on the knowledge that is explicit already or easily expressive, but to take care of the tacit dimension as well. Many approaches for knowledge management get back to the concepts created by Nonaka for that.

2.1.2 Knowledge Management Processes

Alavi and Leidner (2001) describe the knowledge management processes of a company, based on what they call knowledge system, the individuals and groups that share their knowledge in a company (figure 3). In this knowledge system the authors elaborate on the different activities related to knowledge management.

This whole *web of knowledge management activities* is constructed on top of the modes of knowledge creation as outlined by Nonaka (1994). On an individual level, every person has tacit and explicit knowledge. The knowledge is being transferred back and forth through the modes externalization and internalization within the individual or through combination and socialization between individuals.

However, similarly to tacit and explicit knowledge of individuals, every group of individuals (e.g., team members) has two types of memory: Episodic and semantic. A group's *semantic memory* represents the available explicated knowledge, for example a document on a file server. The explicit knowledge of an

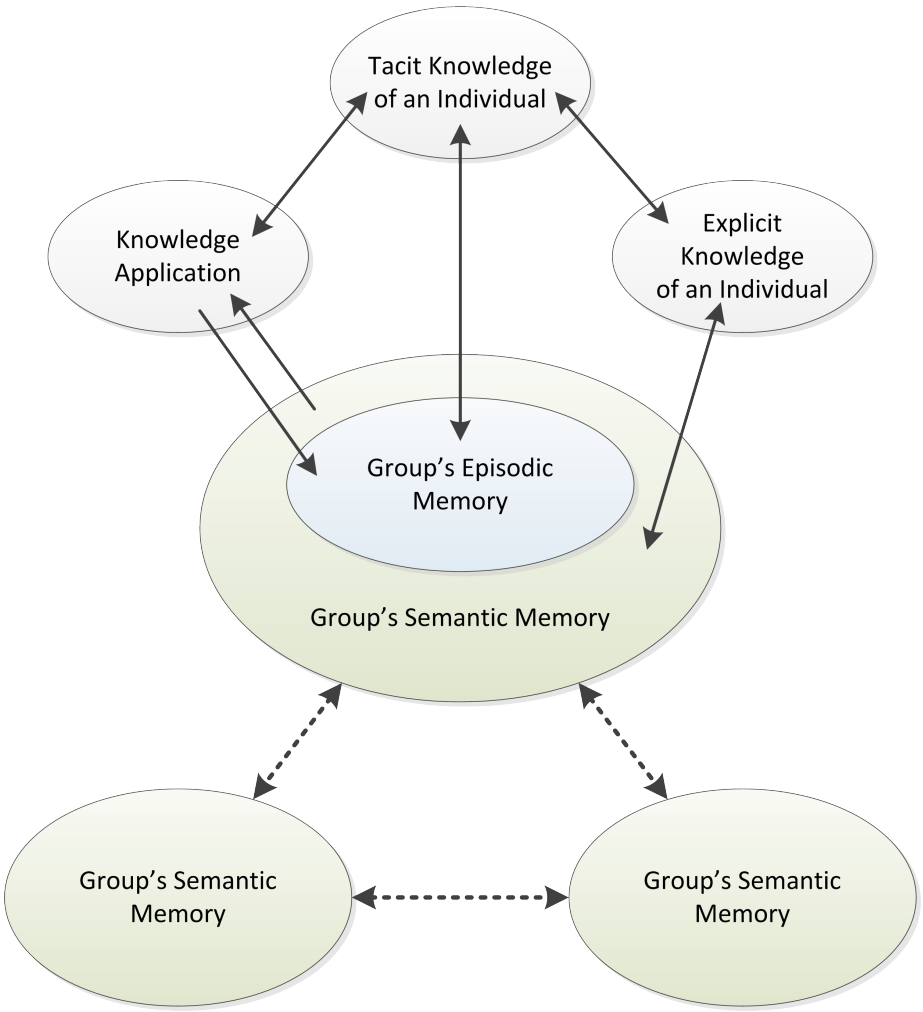


Figure 3: Knowledge Transfer among Individuals in a Group, based on (Alavi and Leidner, 2001)

individual can be made available for the rest of the group by transferring it to the semantic memory of the group. Also, an individual can increase its explicit knowledge by accessing the group's semantic memory. For this learning from the group's semantic memory, the group's episodic memory is a critical necessity. An *episodic memory* represents the collection of shared experiences of the group. Every individual contributes with parts of their tacit knowledge to it.

Beyond the interaction of people is the utilization of knowledge. The knowledge application is always based on an individual's tacit knowledge. At the same time, when applying knowledge, the individual learns from that, which feeds back to the tacit knowledge of the individual. Additionally, the application of knowledge can also be based on the semantic memory of a group directly, which feeds back to the group's episodic memory.

This system of knowledge sharing among individuals in a group occurs in

different areas of a company. Each of these groups then shares their knowledge, via a group-dialogue.

2.2 Relevance of Knowledge Management in Software Development

As software development is a very abstract engineering discipline, knowledge management is an important issue. Here, I explain that relevance. When developing software, a high degree of coordination (Kraut and Streeter, 1995) and management (Sommerville, 2001; Pressman, 2000) become vital tasks. Because the focus is to solve specific problems, the organization of software projects often differs enormously from one to the other (Mockus et al., 2002). Sveiby (1997) points out that most companies face similar problems when it comes to administrating the own intellectual capital. He explains that employees are usually highly educated and qualified professionals whose everyday job is using their competence to develop software. Their major resource is their knowledge; therefore, they are called knowledge workers. And because knowledge is such an important asset in these companies, so called knowledge organizations, the knowledge management becomes a crucial activity. This counts for software development in particular. Hence, knowledge management for software development companies is a wide field with a variety of different approaches (Aurum et al., 2008).

Rus and Lindvall (2002) describe three aspects of software development to be supported by knowledge management: Core software engineering activities, product & project memory and learning & improvement. The core activities of software engineering contain the management of documents or competences as well as software re-use. With product and project memory the authors refer to the evolution of software, e.g. with the help of systems for version control, change management or design documentation. Finally, the learning and improvement includes a recording of results and experiences. The reason is to learn from that and improve future decisions or activities. The desire to improve in these three areas of concern is the motivation for knowledge management in software development.

To conduct knowledge management successfully, many different approaches are possible and documented. Liebowitz and Megbolugbe (2003) propose a framework for implementing knowledge management, which combines an activity cycle of knowledge management levels and the resulting knowledge objects. The different knowledge management levels are conceptualization, reflection, acting and review. Each of these lead to the four knowledge objects: Goals, risks, constraints or measures. The diversity of dimensions illustrates the complexity of knowledge management.

Information systems that are applied in order to manage a company's knowledge or to support the managing of a company's knowledge are referred to as knowledge management systems. Alavi and Leidner (2001) conducted a literature review and illustrate six perspectives on knowledge with their implications (see table 1). They identify the differences in perception of knowledge and describe the influence of a perspective on knowledge management and the knowledge management system.

Perspectives		Implications for Knowledge Management (KM)	Implications for Knowledge Management Systems (KMS)
Knowledge vis-à-vis data and information	Data is facts, raw numbers. Information is processed/ interpreted data. Knowledge is personalized information.	KM focuses on exposing individuals to potentially useful information and facilitating assimilation of information.	KMS will not appear radically different from existing IS, but will be extended toward helping in user assimilation of information.
State of mind	Knowledge is the state of knowing and understanding.	KM involves enhancing individual's learning and understanding through provision of information.	Role of IT is to provide access to sources of knowledge rather than knowledge itself.
Object	Knowledge is an object to be stored and manipulated.	Key KM issue is building and managing knowledge stocks.	Role of IT involves gathering, storing, and transferring knowledge.
Process	Knowledge is a process of applying expertise.	KM focus is on knowledge flows and the process of creation, sharing, and distributing knowledge.	Role of IT is to provide link among sources of knowledge to create wider breadth and depth of knowledge flows.
Access to information	Knowledge is a condition of access to information.	KM focus is organized access to and retrieval of content.	Role of IT is to provide effective search and retrieval mechanisms for locating relevant information.
Capability	Knowledge is the potential to influence action.	KM is about building core competencies and understanding strategic know-how.	Role of IT is to enhance intellectual capital by supporting development of individual and organizational competencies.

Table 1: Knowledge Perspectives and their Implications, from (Alavi and Leidner, 2001)

The perspective on knowledge differs from case to case. Rus and Lindvall (2002) or Nonaka (1994) for example, as mentioned above, apply the object perspective in their research. They understand knowledge as the intellectual capital of a company and suggest systems that support the gathering and sharing of knowledge. On the other hand, the frameworks presented by Liebowitz and Megbolugbe (2003), for example, relate to the access to information perspective. They focus more on finding the needed knowledge and sharing it.

To analyse knowledge management systems, a framework is structured into seven components: Knowledge goals, acquisition, processing, preservation, distribution, utilization and validation (Althoff et al., 2000b). Each of these components contains a different set of attributes, to value the tool accordingly. The outcome is a detailed analysis which helps understanding the system and evaluates its benefit.

Over the time many different knowledge management systems, with different approaches were created and applied. The variety is huge (Rus and Lindvall, 2002; Andrade et al., 2003), it reaches from sophisticated and specialized enterprise systems (Davenport, 1998) over groupware applications (Falbo et al., 2004) to light knowledge tools (Hosbond and Nielsen, 2008).

2.3 Approaches to Knowledge Management in Software Development

Knowledge management is a complex operation (Walz et al., 1993) with the goal to improve the performance of software development significantly (Tiwana, 2004). I categorize specific approaches into the collection of experience (section 2.3.1) and the learning to improve (section 2.3.2). It is difficult to draw a line between these categories, as one cannot exist without the other. Collecting experience does not make sense, if it is not used for anything. Also, it is difficult to improve, when no experience has been collected to learn from.

Note that the reason for this categorization is to structure the contributions found in the literature study within this thesis, according to the main focus. The idea is not to strictly separate the aspects or to make a general distinction, as most approaches cover both of them anyway.

2.3.1 Collecting Experience

The work of any software engineer can be understood as gaining experience. It is quite common for a developer to research a certain topic, e.g., a technology or a work flow, in order to use it. Then by applying the gathered insights, experience is built. There are different approaches to collect this experience, in order to learn from it later. I call this the *recording knowledge phase*.

Post-Mortems in Software Development

The post-mortem analysis or post-mortem review (short: post-mortem) is a study of a project or project phase, after its termination. It is a well-known instrument in software project management and can also be called lessons-learned or retrospective analysis, depending on the applied methodology. Though there

are differences in the processes, all of these approaches aim to collect the experiences of the past and reflect on it, with the intention to learn and therefore improve over time.

Researchers argue that conducting post-mortems helps employees to articulate what was gained as experiences, hence the conversion from tacit to explicit knowledge (Desouza et al., 2005). As a collective learning activity post-mortems are conducted with the majority of the participants during a project or project phase. The focus here lies on reflection, in order to learn from the past, and not on evaluating the work results.

There are basically two different ways of documenting a post-mortem analysis: Creating stories or reports (Desouza et al., 2005). Stories are explanations of the occurrences and progressions, in form of articles. As each story is unique, one story can be very different from another, they can follow a similar general structure though. Reports in contrast follow a very strict structure, they focus on the facts. All reports are very much alike, which makes them easier to comprehend and faster to create.

Case studies show that each type has advantages and disadvantages. On one hand, stories leave space for different interpretations by every individual. Reports present the plain facts and do not leave space for interpretations, but it makes them more difficult to comprehend. Also the possibility of misunderstanding can never be completely eliminated. On the other hand, stories deliver a fuller picture to people who want to understand the occurrences. As they are usually more detailed, the information it contains is much broader. The homogenous nature of reports does not allow that level of detail. And even though it is more easily searchable automatically, the readers of a report will have more difficulties remembering the specifics compared to stories (Desouza et al., 2005).

Combining both approaches is a possibility through the preparation of stories followed by the creation of reports out of these (Schalken et al., 2006). This gains the benefits of both approaches, but it is also very time consuming and therefore expensive.

Pedersen (2005) pointed out 31 barriers to post-mortems, organized in the four categories “designing the organizational context”, “focusing the effort and collecting the data”, “analysing and interpreting the data” and “sharing and exploiting the resulting knowledge”. Case studies show that post-mortems are hardly conducted in software companies, despite the fact that managers agree on their advantages (Kasi et al., 2008). Especially when a project was a financial loss, the additional costs of post-mortems are often avoided. But especially here the impact can be much higher and similar problems can be avoided in the future. Further, post-mortems are more successful in companies that have a sufficient learning capability developed.

Experience Base in Software Development

The post-mortem is an instrument to capture the experience of software development projects after their execution. The gained insights are then filed into what we call experience base or knowledge base. But there are different approaches to build and use these. Experiences are gained during the process of a project and some methods take the continuing work into consideration. The *experience factory* as outlined by Basili (1989) is a concept that helps software

development companies to collect experience and use previous experiences to improve the performance of the current work. The approach separates the activities of software developers into related to the project work and related to the experience base. Hence, the work for a project and the recoding of the gained experience are different tasks, that are clearly distinguished from one another. This independence allows multiple forms of project organizations throughout a company access its one experience factory, as it is not related to specific processes of a certain unit.

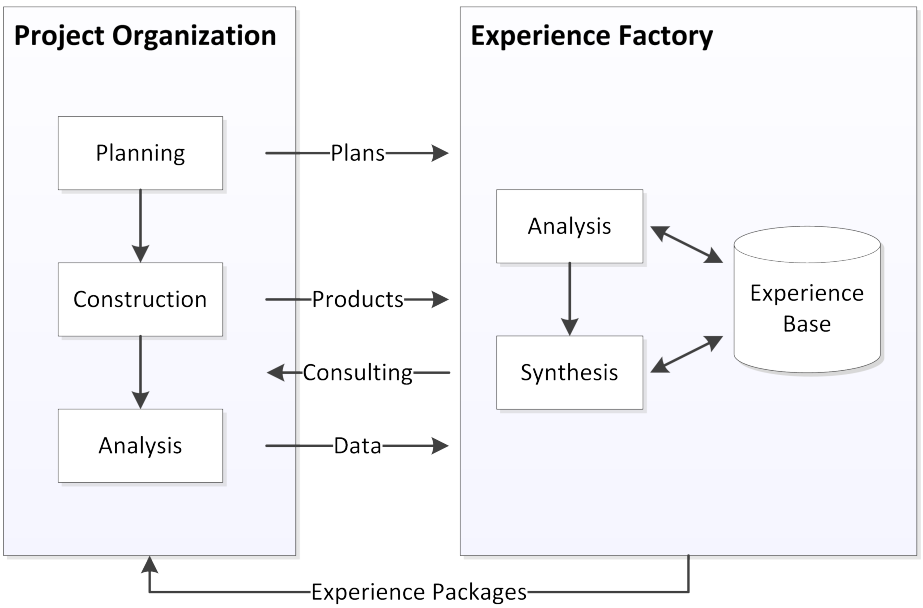


Figure 4: Experience Factory and Project Organization, from (Basili and Caldiera, 1991)

Figure 4 depicts the concept of the experience factory. It analyses and synthesises all kinds of experience that can be externalized and made available, including the outcome of conducted post-mortems, but also different forms of project results and documentation, like plans, products and data. This externalized software project experience is then bundled into a standardized form, so called experience packages, which makes the explicit knowledge easier to access. Developers can thus access these experience packages and learn from them. At the end of every learning process, the developer has to feed the gained insights back into the experience base, to make it richer (Basili, 1989; Basili and Caldiera, 1991; Basili, 1993; Basili and Green, 1994; Basili and Caldiera, 1995; Basili, 1995).

The experience factory draws on the two basic activities in knowledge management: collecting experience and learning to improve. It transforms experience into packages, which is the required step for the following improvement paradigm. This paradigm includes an incremental learning methodology that applies mathematical formalization and organizational institutionalization with the target to support software projects (Basili, 1989; Basili and Caldiera, 1991). The focus of the experience factory however is not on the learning, but on the

creation of an experience base.

An extension of the experience base is an experience management system that allows different methods of packaging experience, that help the user to identify the valuable experiences (Lindvall et al., 2001). An alternative approach to the experience factory organizes the experience packages in a dual approach, combining goal-oriented and similarity-based retrieval (von Wangenheim et al., 2000). To be goal-oriented the experience factory has to structure the experiences according to retrieval goals, so that different re-use scenarios can easily be supported. Similarity-based retrieval builds up on the assumption that similar situations can be solved with similar solutions and that future problems can be analogue to current ones. The system then helps retrieving the best fitting results from the experience base for the user. These approaches are generally also known as need-based and case-based. Need-based, as it is driven by requests for input (Komi-Sirviö et al., 2002), and case-based, as it records the experiences of certain cases (Henninger, 1997).

The experience factory is a widely known concept and implemented in many different organizations. One of them is the Software Engineering Laboratory at the NASA Goddard Space Flight Center, which documents a dramatic increase of re-use across different projects (Basili and Caldiera, 1995). A different study presents an architecture for a "software engineering experience environment" and implements main parts of that into an "intelligent retrieval and storage system" following the principles of the experience factory (Althoff et al., 2000a). This system is instantiated in three projects and users of this system can store artefacts or their contexts or search for these.

Even though the experience factory is well-defined, it is not the only form to manage an experience base. However, comparing different approaches of computerized software experience bases shows that even though the methods differ, the main challenges remain the same (Conradi and Dingsøyr, 2000). For instance, one major key to success, for any form of experience management, is the commitment of the people inside the company, regardless of the hierarchy. Connected to this issue is the feedback to the users, as it is a method to support people involvement. It can increase the motivation to contribute, when one sees benefits. Another challenge in experience management is that it is important to assess the costs and benefits regularly. The authors here recommend an overhead of 1-2%.

2.3.2 Learning to Improve

Most of the studies described in the previous section focus on the recording and management of experience. However, the gathering of experience is useless, if the company does not learn from it in order to improve practice. I call this learning the *applying knowledge phase*.

As software developing projects are usually organized in processes, it makes sense that learning and improvement play a role in these. Processes are defined to establish a context for technical methods and work products to aim at quality goals or other targets (Pressman, 2000; Sommerville, 2001). Striving for a way to improve the operational quality, researchers and practitioners defined a field called software process improvement (SPI). Here, I provide an introduction to the principles of SPI followed by descriptions of actual approaches for learning in software development.

Software Process Improvement

SPI can be approached in three different ways: Evolution, norm or commitment (Aaen et al., 2001). The evolutional approach changes the existing processes incrementally, based on the experiences gained in previous executions. Adopting the existing standards is classified as norm approach and active support of the senior management with attention and resources is called the commitment approach. The underlying idea of SPI is always to learn from own practices and improve evolutionary (Basili and Green, 1994). The IDEAL model (figure 5), for example, divides improvement actions into five phases: Initiating, diagnosing, establishing, acting and learning (McFeeley, 1996). The different activities are hereby based on the principles of Nonaka's knowledge-creating company (Mathiassen et al., 2002).

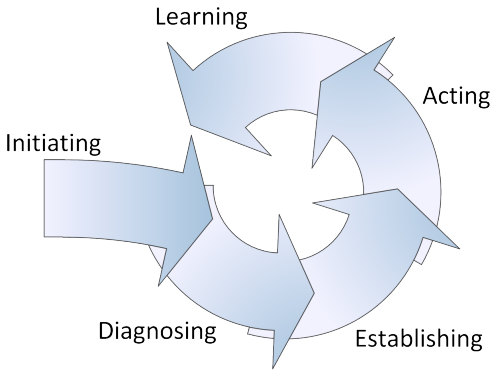


Figure 5: The IDEAL Model, from (McFeeley, 1996)

The SPI literature can be categorized into prescriptive, descriptive or reflective (Hansen et al., 2004). Reviewers noticed a domination of the capability maturity model, CMM (Humphrey, 1989), in the field and that publications are rather prescriptive and not reflective. The approaches for SPI on the particular cases differ clearly; many studies that compare SPI approaches in different settings show this (Basili and Green, 1994; Arent and Nørbjerg, 2000; Iversen et al., 1999). It appears that small and medium sized companies are the preferred target of research, as these more often face changing environments and therefore have to match the changing circumstances with updated processes to stay successful (Ward et al., 2001). Kautz (1998, 1999) delineates four critical factors for successful SPI: A tailored approach, functioning networking, external assistance and external financial support. Others describe a framework which distinguishes between the layered concepts for the company's standard, process and practice (Müller et al., 2008). Here, one feeds into the other through directed communication lines, where each layer informs the adjacent.

The specific SPI differs from case to case. A fundamental element however is always the sharing of knowledge or experience and the intention to learn from that. Mathiassen and Pourkomeylian (2003) explain that the knowledge management has to be a part of the SPI activities for successful software development. They describe a unit that is responsible for the SPI initiatives of a case company and also for the knowledge management. This unit is responsible

to develop a strategy, provide supporting systems and control whether the plan is followed.

Improvement Approaches in Software Development

Learning from shared experiences can be very beneficial in software development and is therefore a crucial aspect within knowledge management. As projects and companies are never identical, the degree of knowledge management affiliation to the processes varies as well. In the experience factory, for example, the process is a significant driver of the whole method. In process-oriented knowledge management the knowledge management is no longer a minor task in a process, but operated systematically (Jablonski et al., 2001; Auerbach and Hauser, 2009).

One study investigated the learning of project managers, utilizing a tool for process description (Kjærgaard et al., 2010). This so called handbook is continuously enhanced. When managers follow the defined processes, they leave comments according to their experiences. As a result, the reality, in form of experience descriptions, is at the same place as the plain process description. Project managers are then able to use these reports of recent implementations to adapt the process.

Other case studies describe the combination of a knowledge management system with a workflow management tool (Kwan and Balasubramanian, 2003; Auerbach and Hauser, 2009; Ramesh and Tiwana, 1999; Falbo et al., 2004). Similar to the handbook approach, the idea is to collect the experience about the process execution and the actual process description in the same place.

Also the scope and method of knowledge management varies from one case to the other. One study for example examines the knowledge sharing during a review process of software architectures (Sherman et al., 2010). Here, the use of collaboration systems enhances the traditional process. As a result the plain study of possible architectures is extended by the company's experience base. Hence, the applied and experience-based input adds to general theories of architectures.

Another field study describes the process of a knowledge management system that is implemented in a software company (Wei et al., 2002). This process intends employees to search a system for support, if they have a problem. In case no applicable solution can be found, the employees can describe the problem and leave a request for support in the system. Experts then find these questions and contact the person directly. Once the problem is solved, the whole communication flow is published in the system and therefore adds, in a case-based manner, to the knowledge base.

But also the processes of learning can have different impact. Different models are described: Direct-impact and mediated-impact (Ravichandran and Rai, 2003). In the direct-impact model the software development is directly benefitting from the employees that gained the original experience. The mediated-impact model is in contrast to that more restrictive. It allows access only to the experience base. However, this can also include requests to employees, what knowledge shall be added to the experience base. Though it might be misunderstood as a restriction or disadvantage here, it is basically nothing more than a different strategy, like leveraging an experience base. Either structural model can improve the software development.

Improving development practice can also be achieved by analysing knowledge flows through a mapping technique (Kautz and Hansen, 2008). The resulting map supports the understanding of the knowledge flows in a company. It provides a census and broader view and can therefore be valuable.

Knowledge Networks

When talking about sharing knowledge, one, of course, must not forget about the most obvious way: Through direct and personal communication. Instead of building and consulting an experience base, like in the examples above, here people contact the expert directly.

The direct communication is driven by the social network that everybody has, a connection of one's co-workers and contacts. However, not all people in a company are equally well connected. And in addition, the quality of these connections differs. Knowledge is shared along the connections of a social network solely; it is therefore also called knowledge network (Nielsen and Tjørnehøj, 2009). Information flow within a network of direct contacts is of different quality. However, it decreases severely when indirect contacts are involved. This holds for the individuals and on group or unit level.

Beyond the opportunity for its members to exchange ideas and experiences, social networks have the strategic potential to lead to new sources of value and of knowledge (McKeen and Smith, 2007). The network effect can thus be very important (Bansler and Havn, 2002, 2004).

The problem with knowledge networks is that their maintenance is easily underestimated. And time invested into maintenance cannot be spent on project-related tasks (Hansen et al., 2002).

2.4 IT Support in Knowledge Management

The research presented in this section applies the different approaches to collect experiences and learn from them, which are portrayed above. However, the scope is extended. This section deals not only with the knowledge management as such, but also with IT systems to support it, the so called knowledge management systems.

This section describes aspects of knowledge management that are important to mind, in order to be successful. Additionally, I explain that the introduction of knowledge management systems is complex and not free of disappointments. Hence, to achieve an effective approach, it is important to analyse reported results of different approaches and follow guidelines to reach positive outcomes.

2.4.1 The People Perspective

The introduction and utilization of knowledge management systems in companies is reported by many different researchers that illustrate problems and achievements. An example is a study of a knowledge management system's implementation in a multi-national pharmaceutical company (Bansler and Havn, 2001, 2002, 2003, 2004). After the system failed, five reasons were spotted: Time pressure, missing incentives, low acceptance for bragging, the preference of personal networks and poor quality of the contributions. These reasons illustrate that the system was not accepted by its users. The authors noticed a direct

connection to the network effect, which implies that the knowledge management system becomes more useful, the more people use it. In other words, the quality and the support for the implementation process of knowledge management improves, if more people are involved.

That example shows that it is important to emphasize the role of the employees for the success of a knowledge management approach within a company. This is seconded by many other studies, like a case study that implemented knowledge sharing processes within a company, which were supported by IT (Kautz and Thaysen, 2001). The authors found that in many cases too much focus is set on the IT, while the communication and collaboration between the people is more important and reaches results of higher quality. Other research agrees with that and illustrates that the technical component is not the most important aspect in knowledge management generally (Davenport and Prusak, 1998; McDermott, 1999; Conradi and Dingsøyr, 2000).

Instead of focusing on the IT, knowledge management systems should support the knowledge related processes (Fehér and Gábor, 2006). These again have to take the whole company into account: The technological perspective, the personal situation and the company's culture. Focusing on just one compromises the balance, which is important for a successful approach.

McDermott (1999) states that knowledge sharing is leveraged through existing communities, which should be developed as an activity of the knowledge management. He suggests supporting communities in a company, with technology and resources. These are not only sharing the knowledge within the community, but also act as experts throughout the company.

These findings demonstrate that a major issue of many knowledge management approaches is the focus on the IT. The users of the system should be stressed instead, as well as their involvement into the knowledge management approach. A knowledge management approach is likely to run into problems, if the communication between employees is not being supported well enough to share their knowledge. Knowledge management cannot solely consist of the implementation of an IT solution, but has to follow an approach that takes the whole company into account. The people within a company have to be an essential aspect. They should be given with the opportunity to share their knowledge within the company.

A knowledge management system, however, strongly influences the knowledge management approach. It can leverage or diminish the knowledge sharing within the company. Therefore, it is very important that knowledge management and the supporting IT system fit together (Swan et al., 1999a). Additionally, the people that are affected by the knowledge management are sometimes heavily demanded to fulfil knowledge management issues, which are difficult to follow. Especially when it comes to tacit knowledge, the requests of managers are often too high. In some companies there is a high pressure on the people, forcing them to stoically follow the knowledge management approach, which is counterproductive and limits the knowledge management approach's success.

A successful knowledge management approach therefore has a clear focus on the people involved. As described above (see section 3.1.1), knowledge is always individual and the people cannot be forced to share it. Hence, a knowledge management system should support the users in sharing their knowledge following the way they prefer and the way that suits the company's strategy.

2.4.2 The Company Perspective

Apart from the involvement of the people there are also other things to consider when designing a knowledge management approach. A literature review delineates 13 aspects that influence the sharing of tacit knowledge (Joia and Lemos, 2010). These contain plenty of different facets. Generally, employees need a common language, mutual trust among each other and the company has to provide time for the knowledge management task for every individual. The company itself impacts the sharing of tacit knowledge through the internal hierarchy, a reward system, the knowledge management strategy and the training for the people. Also, the knowledge management system influences everything, by itself and through the media that it uses. The remainding aspects focus on the people in the company: Their personal network, their understanding of knowledge sharing, the way they question each other and generally the way they cooperate. All these aspects should be considered in knowledge management, as the sharing of tacit knowledge is an important factor.

The long list of aspects that influence the management of tacit knowledge only visualizes that knowledge management is a complex task. It is one out of three problems that many knowledge management implementations suffer from (Desouza, 2003): First, it is utopic to think all experiences can be externalized. Second, software engineers often do not want to be seen as experts. Their future assignments might then be stronger affected by their experiences than they want to. And third, even a sophisticated experience base cannot contain the answers to all questions. Alternatives have to be available for optional use, be it in form of personal peers or different accessible knowledge sources.

The number of possible difficulties is very high, but that stresses the importance to treat a knowledge management approach as a work in progress, not a final result (Desouza, 2003). All these aspects have to be controlled and maintained continuously. A company is obliged to stay flexible and evaluate the possible problems in order to react and solve issues. The knowledge management strategy then would have to be adjusted where it makes sense.

Another important aspect of knowledge management is to balance the effort between exploring and exploiting the available knowledge (Mathiassen and Pedersen, 2005). Providing time to the people for exploring their knowledge and thus experimenting with the available insights and searching for new ones is very important. Exploiting the knowledge is much more focused on increasing the performance, by lowering the variance. This is where expertise is created and strengthened, by knowing details about a potentially best solution. In many cases the exploitation is not at all or just barely considered. But a plain exploration-based knowledge management does not result in developed skills and competences, like learning through exploitation does.

Much research agrees with this dilemma and calls it a fundamental problem in IT-driven knowledge management approaches (Swan et al., 1999b). This issue focuses on the processes of a knowledge management strategy. Processes regarding the knowledge management should be carefully designed, considering the people to interact, the company's culture and the environment.

Fundamental problems in IT-driven knowledge management approaches are that tacit knowledge is not easily codified (as explained above in the people perspective) and the supply-and-demand misunderstanding (Swan et al., 1999b). The latter bases on the general assumption "build it and they will come", which

does not apply when it comes to IT solutions. Dixon (2000) calls this one of the biggest myths in knowledge management. The mere existence of any tool does not make people use it. And the fact that a knowledge base is filled with insights and experience does not guarantee that its content will be applied. The system should help encouraging people sharing their knowledge and use it as a base of communication to overcome irregularities between the supplied and demanded knowledge. People should share their knowledge through it, but also learn and search through the system.

From the technical perspective of knowledge management, a major issue is that in many cases simply the wrong tools are used to support the employees (McDermott, 1999). To avoid this issue, there are many different frameworks to analyse knowledge management systems, Liebowitz and Megbolugbe (2003) present some of these. The authors explain that frameworks like theirs provide the knowledge managers with an important lever to improve the knowledge sharing among people inside the companies. These frameworks help to find out whether one system fits to the company's needs or not and provides help for knowledge sharing activities.

One of these frameworks assess the knowledge management systems according to seven components: Knowledge goals, acquisition, processing, preservation, distribution, utilization and validation (Althoff et al., 2000b). Knowledge management approaches and their IT support can be evaluated according to these components. This helps the designers to compare different approaches and to find the best fitting one according to the company's needs. These frameworks can be used not only for choosing the fitting knowledge management strategy or knowledge management system, but also as a controlling mechanism. This can help the managers to investigate, whether or not the objectives of the knowledge management are still met or changes become necessary.

2.5 Summary

Knowledge management in software development is a research field with many different objectives and approaches. This chapter provided a study of related literature and introduced the reader to the field of knowledge management in software development. I which show that knowledge management is a complex venture. The portrayed research illustrates that there is no correct or generally best solution, but several ways to approach the collection of experience and to learn from it. Many aspects and perspectives are involved, that have to be considered. Additionally, it is important to see, that a knowledge management approach is never finished, it has to be maintained continuously in order to stay beneficial.

I described a variety of specific approaches to knowledge management in software development. The different contributions show that every knowledge management approach can be divided into four basic categories (Meehan and Richardson, 2002):

- **Create.** How is the knowledge recorded (e.g. through post-mortems)?
- **Store.** How and where are the gained insights saved?
- **Share.** How and when should these experience bases be consulted?

- **Leverage.** How should employees access the included information?

These categories are grouped into two phases. First, the *knowledge recording* phase with the first two categories (create and store), which focuses on building up a knowledge base. Second, the *knowledge adaption* phase with the latter two categories (share and leverage), which makes use of the knowledge base in order to improve practice. In the part of this chapter that deals with the knowledge recording (section 2.3.1), post-mortems are explained and the use of an experience base, with a focus on the experience factory. The knowledge adaption part (section 2.3.2) then describes different learning approaches, which are closely related to the research field of software process improvement. Additionally, with knowledge networks an alternate approach of knowledge management was presented, which almost entirely skips the knowledge recording.

Further, the research shows that the employees of the company play an important role within knowledge management. Their requests should not be ignored, as it is on them to share what they know. Additionally, the company has a major interest in creating a knowledge management approach that fulfils the standards and strategies. The knowledge management approach can only be successful, when a suitable balance regarding the people and the company perspective can be maintained. The company's culture and the knowledge management method have to fit:

> *"When you need to transfer knowledge, the method must always suit the culture." (Davenport and Prusak, 1998, p. 93)*

The related research shows that knowledge management in software development is a complex task, which combines a variety of different aspects regarding the treatment of knowledge. The presented findings and considerations should influence the design of any knowledge management approach. It is important for the implementation of any knowledge management approach to not only provide appropriate IT support, but also to define fitting processes.

3 Theory

The aim of this chapter is to provide the relevant reference theory for my studies in the field of knowledge management in software development. I begin with the definitions of the terms knowledge and knowledge management as well as how I apply them in this thesis (section 3.1). Then, I describe the selected core theories regarding different knowledge management strategies (section 3.2) and IT support in knowledge management (section 3.3). With these I build theoretical foundation of my research.

3.1 Definition of Terms

In the previous chapter, I describe different approaches to knowledge management in the related research. This section defines my understanding of the terms knowledge and knowledge management.

3.1.1 Knowledge

The debates on what it is, that we call knowledge, lead back to the ancient Greece (Platon, 1986). An entire philosophical discipline was formed, called epistemology. It is concerned with three questions regarding knowledge: What is it? How is it acquired? And how do we know what we know?

Philosophers all around the globe dedicated their lives to find answers to these questions, to which a precise answer will probably never be found. However, most of them can possibly agree on a rough statement like this: “Knowl-

edge arises in experience. It emerges from reflection. It develops through inference." (Audi, 2010)

But more than a clear definition of the term knowledge itself, I strive for an understanding of how to utilize it. In my field of interest this leads to what is often referred to as the *pyramid of knowledge* (Davenport and Prusak, 1998; Rowley, 2007). It does not provide a definition, instead, it is a delimitation of the terms data, information and knowledge, with one building on top of the other (figure 6).

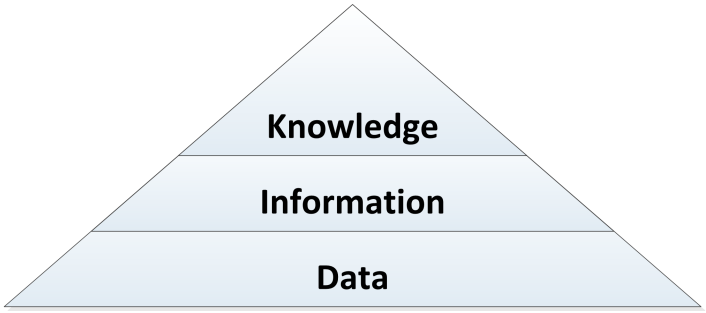


Figure 6: Pyramid of Knowledge, based on (Rowley, 2007).

Pyramid of Knowledge

The general understandings of data, information and knowledge are widely agreed upon, but the detailed definitions vary. Zins (2007) for example documented 130 definitions of these terms, where the differences mostly root in the perspective and target of the research.

The following represents my understandings, based on the literature. I generally agree with Ackoff (1989) and Davenport and Prusak (1998).

Data is raw. It represents symbols that exist, without further significance, and in any form, usable or not. Data is also the product of observation (Ackoff, 1989). Note that data has no meaning, not even to itself, it is pure. Data also denotes objective and discrete facts or statements about events, without relation to anything else (Bellinger et al., 1997; Davenport and Prusak, 1998).

In IT data can, for example, take the form of a field in a spread sheet. Further, it is a material characteristic that data is comparable.

Information is data with a meaning (Ackoff, 1989). This meaning is given through a relational connection and can be useful, but that is not a necessity. The difference between data and information is not structural, but functional. Information as a message, which a sender provides to inform a receiver (Davenport and Prusak, 1998, p. 3). The purpose of the message is to impact the receiver's insight or outlook through data.

In IT information is realized through relational databases, for example. Here the relations between single fields of data provide the meaning.

Knowledge is processed information (Davenport and Prusak, 1998). The acquisition of knowledge is generally called learning (Ackoff, 1989). It is always a personal insight that cannot be separated from the individual. It is further based on accessing and interpreting information. However, this is a very simplified view and focuses on the role of this concept in the pyramid of knowledge. Not a definition, but a characterization, which I can agree with, is this:

> *Knowledge is a fluid mix of framed experience, values, contextual information and expert insight that provides a framework for evaluating and incorporating new experience and information. It originates and is applied in the mind of knowers. In organisations, it often becomes embedded not only in documents or repositories but also in organisational routines, process, practices, and norms. (Davenport and Prusak, 1998, p. 5)*

I explain knowledge management in more detail in the next section (3.1.2); however, the pyramid of knowledge makes it obvious that the conversion between the different levels plays a role there. Knowledge is always subjective and cannot be shared or transferred directly.

Above (see section 2.1.1), I describe the knowledge creation according to Nonaka (1994), which involves different states of knowledge. An individual has to externalize the gained knowledge. This can then be accessed by another individual or group in order to create their own knowledge through internalization (e.g., learning, practicing, organizing or judging). The externalization means to express what one knows and the internalization is to make sense or to learn from what others expressed. These knowledge creating activities contain the conversion between knowledge and information and vice versa. So, what Nonaka calls externalized knowledge is according to the definition above information. Needless to state that knowledge cannot be transferred to a computer-based system either. Explicit, formalized and codified knowledge always becomes information (Grundstein, 2009).

The conversion from data to information or the other way around is mainly a matter of providing a meaning or detaching it. The meaning is commonly achieved through context, purpose or relevance of the data (Davenport and Prusak, 1998, p. 4). Continuing the picture of a message, it is always the receiver that judges the value of it. Only the receiver can decide whether the message truly informs or contains just a meaningless set of data.

Sharing knowledge of one individual with another contains always the conversion from knowledge to information or data (depending on the medium) and vice versa. One person has to express their knowledge in order for another one to create knowledge from that. While this process is discussed above (section 2.3.1), it is important to understand the transition between the layers in the pyramid of knowledge. This is the prerequisite for sharing what one knows.

Perspective of Knowledge

In section 2.2 I explain the perspectives of knowledge (see table 1 on page 12), by Alavi and Leidner (2001). These perspectives describe the view of knowledge and the implications to the knowledge management approach and a knowledge management system. In the literature many contributions do not state clearly

what perspective of knowledge they apply. I chose to put a focus on *Access to Information*: "organizational Knowledge must be organized to facilitate access to and retrieval of content." (Alavi and Leidner, 2001, p. 110)

This perspective implies that knowledge cannot be accessed directly. As I describe in the definition of knowledge above, knowledge is always residing within individuals. Externalizing knowledge equals a transformation of knowledge to information. Therefore the perspective is not concerned with knowledge itself, but with the conditions of accessing information. I understand accessing information as a way to communicate directly, but also as a way to make documents available.

This choice is reflected in my reseach throughout this thesis. This includes the ideas that knowledge management helps providing access to information (details in section 3.2) and a knowledge management system as the leading tool to actually access information (details in section 3.1.3).

I want to have a look at some other perspectives that were implied earlier:

State of Mind. This perspective represents the state of mind that we call knowing or understanding. It reflects my definition of knowledge.

Process. This perspective focuses on adopting expertise. It is an isolated view on what I labeled the applying knowledge phase in the related research (see section 2.3.2). An example is SPI, where knowledge is utilized in a the process to improve (McFeeley, 1996).

Object. This perspective understands knowledge as a thing to be exchanged and used. Large parts of what I labeled the recording knowledge phase in the related research (see section 2.3.1) have this view. An example is the experience factory by Basili (1989), where knowledge is stored and can be found in specific knowledge bases.

One could argue that these three perspectives are inherited in the "Access to Information" perspective. It can be understood as an extension to the "Object" perspective, emphasizing "the accessibility of the knowledge objects" (Alavi and Leidner, 2001, p. 110). Also, processes play an important role in all organizational communication, just as the "Process" perspective suggests. And finally, the "State of Mind" is covered, due to the necessity to convert knowledge to information and vice versa. I understand the "Access to Information" perspective not as a blend of perspectives, but as the organization of knowledge "to facilitate access to and retrieval of content" (Alavi and Leidner, 2001, p. 110).

However, I recognize the adjacencies of the different perspectives. Although my studies focus on the "Access to Information", I want to explore the differences and their utilization in a larger knowledge management approach.

3.1.2 Knowledge Management

The main goal of knowledge management is to obtain a competitive advantage and increase the internal performance (von Krogh, 1998). Knowledge management attempts to enable the employees sharing what they know, in order to benefit from other people's expertise and improve quality of work results. Different possibilities of knowledge management are presented above (see section

2.3), where I explain a variety of knowledge management approaches that follow different strategies. I show that they all have in common to implement four different activities (section 2.5): creating, storing, sharing and leveraging knowledge (Meehan and Richardson, 2002).

Unlike many other resources in a company, knowledge actually cannot be managed. However, knowledge managers can influence the knowledge creation processes positively and thus enable sharing (von Krogh, 1998). Knowledge is always subjective and individual, as I state above, and people cannot be forced to share what they know. Knowledge sharing fully depends on the individuals and is always voluntary. However, a knowledge management approach makes an effort to enable sharing.

There are three different aspects to knowledge management: Organizational, technological and individual (Rus and Lindvall, 2002). The organizational aspect of knowledge management provides a culture that encourages sharing among employees or defines processes that sustain it (von Krogh, 1998). The technological aspect describes IT solutions that assist and support people sharing what they know (Davenport, 1998, p. 18). The individual aspect focuses on the people, that need a certain freedom for effective knowledge sharing (von Krogh, 1998). But these three different aspect must not be seen isolated from each other. Based on the related literature we can see that knowledge management has to address all three aspects, in order to be successful (see section 2.4).

In the literature the two terms knowledge management and knowledge sharing are often used interchangeable. Compared to the four knowledge management activities creating, storing, sharing and leveraging (Meehan and Richardson, 2002), sharing is just one part of the whole. Yet, many sources do not distinguish between sharing and managing knowledge. Others use the terms knowledge sharing, knowledge exchange and knowledge transfer synonymously. Some even emphasize this and call knowledge transfer a key activity in knowledge management (Sveiby, 1997). It appears that the choice of terms and their usage varies from researcher to researcher, which again is disparaged by other ones (Walsham, 2005).

This confusion is based on our language (in this case: English), which treats knowledge as something one can have, i.e., own (Davenport, 1998). No matter whether I share, exchange or transfer my knowledge, all of these terms imply that something is being taken away from me. But that is not the case, as I explain above (section 3.1.1). When we are being literal, all of these terms are confusing. A solution could be to abandon them, but better ones are hard to find and it would still confuse many readers as the term knowledge is part of our common language in everyday usage.

I acknowledge the diversity. Throughout this thesis I use the terms knowledge management, knowledge sharing and knowledge transfer as they are established in current usage:

> *Knowledge management enables the knowledge sharing between people, where one person transfers their knowledge to another one.*

This statement reflects the common understanding of the terms. Common expressions like “employees share their knowledge” cover the full process,which

ends with one person learning something. However, when I describe details regarding the knowledge management system, I utilize the pyramid of knowledge.

The related research (chapter 2) shows that knowledge management is a complex task with a variety of possibilities. Research shows that knowledge management activities often represent the communication between two parties that create a shared understanding among each other and thus create knowledge (Kautz and Kjærgaard, 2008; Kjærgaard et al., 2010). It is an important condition that people have access to the provided knowledge, be it externalized or in form of direct contact to individuals. This matches the perspective of knowledge that I chose to apply above, the "Access of Information".

3.1.3 Knowledge Management System

This section briefly defines the term knowledge management system. A distinction and description of different types can be found in section 3.3 below.

In this thesis, I call an IT solution that supports the knowledge management a knowledge management system. However, the term knowledge management system does not include the processes of how to use this system. It represents exclusively software.

In combination with the definition of the term knowledge management it is important to clarify, that a knowledge management system is only a tool with the aim to support the knowledge management, and not its focus. Davenport and Prusak (1998, p. 4) describe this with an analogy: The medium itself is never the message; the medium can influence the message, but the delivery is always more important than the vehicle.

The ability of a system to support knowledge management is limited (Rus and Lindvall, 2002). Any knowledge management system is able to store information only (Grundstein, 2009), as knowledge cannot be separated from the individual. The support in knowledge management is therefore restricted to the storing and retrieving of information. Additionally, a knowledge management system can aid the communication between peers, which in a broader sense, and in combination to the retrieval abilities, can be argued as supporting the activity of knowledge sharing.

I acknowledge that the term "knowledge management system" itself is in principle very misleading. According to my definitions above, a system can only deal with explicated knowledge and thus information. Correctly, it should therefore be called an "information management system". However, in the literature the term "knowledge management system" is widely accepted, using a different one would only result in general confusion.

3.2 Knowledge Management Strategies

The variety of approaches to knowledge management in companies is big. In the previous chapter (section 2.3) I introduce some of these and show the differences. A main characteristic of knowledge management is the focus. Some approaches focus on collecting, in the sense of post mortems or the experience factory, and others have a stronger focus on learning, like the creation of networks or the branch of software process improvement. This variety of approaches illustrates that the ultimate knowledge management strategy does not exist. Instead, a

company has to tailor a strategy that fits, depending on its own needs and conditions.

Hansen, Nohria, and Tierney (1999) analysed different consulting companies according to their knowledge management strategy and state that it should reflect the company's competitive strategy in order to be successful. Two fundamentally different strategies are defined: Codification and personalization.

My research goals involve the creation of a knowledge management system and at the same time the literature study shows that it is important to focus on people. The theory of knowledge management strategies by Hansen et al. (1999) is relevant, because it explains strategies with a focus on people (personalization) and on systems (codification). Hence, I select this theory as part of my theoretical foundation.

I explain both knowledge management strategies below, based on (Hansen et al., 1999).

3.2.1 Codification

The goal of codification is the re-use of knowledge (Hansen et al., 1999, p. 110). The underlying idea is to extract the knowledge from people and store it somehow. This approach is especially used by process-driven companies, which focus on documentation. Here, employees are supposed to fill out forms and create reports about on-going work or intermediate results. The company's processes intend to codify the gained knowledge, to constantly build up a knowledge base with formalized content about specific tasks or problems. This knowledge base should then be accessed, when similar problems occur in future projects. The target is to learn from the past and approach current problems in similar ways as in the past (similarity-based). Hansen et al. (1999) call this approach the *codified knowledge management strategy* or codification.

To utilize codified knowledge two aspects are of high importance. First, it requires a precisely described problem domain. Without that, finding a fitting report according to the current needs can get more complicated. Second, pursuing the strict formalisms is a vital task. The reported results can be difficult to find, if they do not follow the intended structure sufficiently. This is supported by fine-grained and highly specific input fields. Employees have to understand where they can find the desired information or enter their experiences (people-to-documents).

The codification strategy's goal is to provide scaffolds that lead to standardized reports (Hansen et al., 1999, p. 108). These then are collected in a knowledge base. This standardization makes it easy to search for the documentation of previous results, which, if similar enough, can help in the current situation.

Hansen et al. (1999) explain that a competitive strategy, which would apply the codified knowledge management strategy, typically aims to provide high quality solutions that are reliable and specialized in a certain field (Hansen et al., 1999, p. 110). A company that produces high-end solutions to a specific problem for different customers is an example for that.

The codification process is often executed by higher organizational hierarchies. Creating reports is in many cases related to responsibilities and act like a communication channel between the different hierarchy levels. These reports are commonly used as a controlling tool in crucial and complicated tasks such

as project management, where previous reports act as guidance. So the reports follow two targets at the same time, to communicate between organizational hierarchies and to build up a knowledge base.

Supporting Knowledge Management System

A knowledge management system in the codified knowledge management strategy has to take care of managing this form of documentation and providing help for its users to find the desired information. Hansen et al. (1999, p. 109) specifically state that heavy investment in IT is essential as "the goal is to connect people with reusable codified knowledge".

Codified knowledge, e.g., stored in form of reports, has to be comparable and computable to fulfil its purpose. This is achieved by following a consistent structure, which makes spread sheet or database applications typical implementations. So called enterprise systems are applications that also fit into this category, they commonly handle data in forms and support a company-wide analysis; typical for codification (Davenport, 1998). Today, most companies have an enterprise system applied for operational support. Employees are able to insert and access information in a formalized way. Additionally, many enterprise systems have features that verify the data inputs or automatically create an analysis based on previously entered data.

The experience factory by Basili (1989) is a good example for a knowledge management system supporting codification (see page 14). Separated from the actual problem solving, the gained experiences are packaged by employees. These experience packages are then stored in a form of repository. To make the experience re-usable, the packaging has to fulfil certain standards. It has to be assured that each package contains all necessary input, in order to provide assistance for the experience's consumers. This completeness contains aspects like a problem definition, a generalization, an analysis and the description of the execution. Potentially helpful experience packages can then be located by filtering the available ones for the needed input. The strict formalization here supports the finding, as every user of such a knowledge management system has an awareness of where to expect what kind of information. Searching is further simplified through the uniform coding of the packages and its contained information.

A knowledge management system that supports codification has to follow strict patterns, in both the way it is used and the way it takes care of the involved knowledge. All employees are active users and access the system when the company's processes expect them to. Guidelines define how to codify knowledge in order to simplify the access.

3.2.2 Personalization

The focus of personalization is on people and their direct communication among each other (Hansen et al., 1999, p. 108). Especially in companies, that follow flat organizational structures, the internal communication is important. Encouraging the employees to exchange ideas and experiences is the main principle here. Thus, the employees continuously build up and improve their social network within the company, which they utilize to localize desired knowledge or experts

in the case of need (goal-oriented). Hansen et al. (1999) call this approach the *personalized knowledge management strategy* or personalization.

A company, which is following a personalization strategy, typically tries to support creative and individual approaches to unique tasks (Hansen et al., 1999, p. 110). It faces only very special problems, embracing the difference of each project and customer in order to provide a specialized solution, where different levels and areas of expertise are important. Therefore, the knowledge management is more focussed on connecting the employees (person to person). This is often supported by an open company culture that aids personal communication and provides circumstances to share knowledge (e.g., in form of meetings).

A competitive strategy that applies the personalized knowledge management strategy typically concentrates on customized solutions of high complexity and quality (Hansen et al., 1999, p. 111). It is very common for this kind of companies to have different customers in different domains. Hansen et al. (1999) point out that the corporate network is then used to find people with expertise (e.g., knowledge, experience, interest, etc.), who then share their knowledge. The result is that specialists work on solutions and share their knowledge, which increases the company-wide expertise.

Supporting Knowledge Management System

A knowledge management system in personalization has the purpose to organize direct communication. Hansen et al. (1999, p. 109) state that moderate investment into IT is sufficient, as "the goal is to facilitate conversations and the exchange of tacit knowledge". The authors explain further that "Knowledge is shared not only face-to-face but also over the telephone, by e-mail, and via video-conferences" (Hansen et al., 1999, p. 108). It becomes clear that the focus is on direct communication between two parties, the knowledge-owner and the knowledge-seeker.

The major task in this strategy is the establishment of networks that help spotting a knowing person, who could help solving a problem. The communication itself is considered a minor aspect, technologically. This could be solved by simple e-mail, instant messaging or other peer-to-peer solutions (McKeen and Smith, 2007). But IT can support the networking as well. The internet is heavily used to connect to people and establish and deepen the contacts nowadays. Good examples for that are LinkedIn[4] or Facebook[5]. Both connect people and provide different features to communicate, however, Facebook focuses on personal life and LinkedIn focuses on business contacts. Either system allows you to insert personal information, like in a curriculum vitae, after registration. Then you can connect to other people, see their information and contact them directly. These systems are thus implementations of the personalization strategy, as people build networks and can contact the person of interest.

Systems like these are called social web or web 2.0 and rose in the first decade of the 21st century. Before that, Hansen et al. (1999) published their article in 1999, they could therefore not foresee the impact on enterprises and their knowledge management approaches. Different forms of social web applications provide different possibilities (Dolog et al., 2009d). Not all of them have a focus

[4] http://www.linkedin.com/
[5] http://www.facebook.com/

on the networking aspect, like LinkedIn or Facebook. Others emphasize the communication, like wikis.

Contrary to the statements by Hansen et al. (1999), a higher investment in knowledge management systems can improve the communication between employees. Social software was not considered in their article. Its influence on communication increased in the recent decade and the use of tools in communication changed a great deal through systems of the social web (Dolog et al., 2009d).

3.2.3 Combination of Strategies

Researchers show in different examples and case studies, that the coexistence of both strategies in an equal share is counterproductive and would do harm. A company should find the one that suits its needs best. Hansen et al. (1999, p. 112) argue though, that choosing a primary strategy, supported by the other one can be a successful blend. They suggest realizing that in an 80-20 split: 80% of one, the main knowledge management strategy, and 20% of the other, the supporting knowledge management strategy. A 50-50 share is counterproductive as none of the characteristics of a company is suitably supported or emphasized.

While Hansen et al. (1999, p. 114) suggest one corporate strategy to be followed throughout the whole company, a study by Mathiassen and Pourkomeylian (2003, p. 72) took a different route. They dealt with a unit in a company, which provides seven different services. The authors assigned four of these services to follow a codification strategy and the other three of them to follow a personalization strategy. They successfully showed that differentiation in different areas within the same company is feasible and makes sense, because the different services followed different aims and an overall strategy for all of them would diminish some of the services.

Finally, it has to be mentioned, that there is not one strategy generally better than the other. One might fit the company better than the other or the employees of a specific company might relate better to one than the other. A general statement however cannot be made, it differs from case to case (Hansen et al., 1999, p. 115). Researchers particularly state that formalized knowledge is equally relevant as other types of knowledge (Mathiassen and Pedersen, 2005). Choosing one over the other is an important choice and has to be made wisely as a wrong decision can confuse employees or lower the quality of results (Hansen et al., 1999, p. 113).

3.3 Knowledge Management Systems

Hansen et al. (1999) write about general and operational strategies for knowledge management in a company. These strategies aim at the communication between employees, the knowledge management system support is not a focus of their article. However, this support is explicitly stated in my research question (see section 1.3). Therefore, knowledge management systems are an important part of my studies and a theoretical foundation is necessary. This section deals with different approaches for knowledge management systems and what I chose to base this thesis on.

Note that there is not necessarily a direct relation between strategies and systems. In fact, it is possible to use the same tool for different strategies; the same knowledge management system could be used in either strategy. Choosing a system depends on the knowledge management approach. The variety of knowledge management systems is huge, and the approaches follow wide variations (Andrade et al., 2003). Rus and Lindvall (2002) provide an overview by presenting different tools and their use in software development. Frameworks help finding a system that fits to the knowledge management approach (Hahn and Subramani, 2000).

3.3.1 Knowledge Bases

The concept of a stock for codified knowledge is called a knowledge base or knowledge repository (Davenport and Prusak, 1998, p. 146). I show examples for that in the literature study, like the experience base (see page 14), which can be consulted by employees to learn from other people's codified knowledge. Storing the codified knowledge is a key element of any knowledge management system (figure 7). The system itself however is more than just a storage manager. It assists users externalizing their knowledge and provides the functionalities to access codified knowledge (Davenport and Prusak, 1998; Davenport, 1998; Davenport et al., 1998).

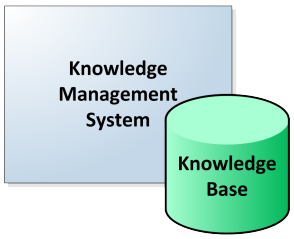


Figure 7: Knowledge Management System has a Knowledge Base

Different types of knowledge bases exist. As I show below, Davenport and Prusak (1998) distinguish between structured and informal knowledge bases. This distinction is particularly valuable in the context of different knowledge management strategies. I therefore chose this as part of the theoretical foundation of my research.

Knowledge management systems can basically vary in two ways: In features provided by the knowledge management system and in the handling of knowledge in the knowledge base. Both influence the knowledge management system significantly.

The features of a knowledge management system are mostly of importance regarding the comforting level, including usability aspects, and the range of functions. For example, the different providers of wikis as knowledge management systems result in huge differences of features (Wagner, 2004). All wikis share a general feature set, but the implementation of some functionalities varies from one to the other. The same counts for basically every classification of systems, e.g. the different groupware applications (Rus and Lindvall, 2002; Andrade et al., 2003).

A much stronger impact on the utilization of a knowledge management system has the underlying knowledge base, as it defines both the type of information to be included and how it is stored. Despite the name, a knowledge base can only store externalized knowledge and thus information, as shown above (see section 3.1.1), however, this can be achieved in many different ways.

Davenport and Prusak (1998, p. 146) define three different knowledge bases: external, structured internal and informal internal. These are distinguished according to the formalization of the included information, which relates to the purpose (table 2). The formalization is classified as either structured or informal. With this classification the authors want to illustrate the differences of information representation. Structured information is for example represented by tables or forms, whereas informal information is a simple text, like an article or letter.

Knowledge Base	**Information**	**Content**
External	Any	Reports, articles, analysis and background input about the market, competitors and partners.
Structured Internal	Structured	Different kinds of materials for documentation and marketing, both research and product oriented.
Informal Internal	Informal	Documentation of discussions and communication filled with experiences and know-how.

Table 2: Knowledge Bases, based on (Davenport and Prusak, 1998)

The external knowledge base contains what people know or knew about the competitive environment. Information regarding the market and competitors can be found here, as well as related research or other third party input. The wide range of information is reflected in different types of information to deal with. It includes reports and other forms of codified knowledge, but not exclusively. Moreover, the variance in source and relevance is an indicator that a consistent structure is difficult to maintain.

However, as my studies focus on the management of the company's internal knowledge, the external knowledge base is out of scope.

Davenport and Prusak (1998) distinguish the internal knowledge base further between structured and informal. The *structured internal knowledge base* contains the company's formal information, where the contents are document-based and mostly related to products or methods of the company. Everything included is considered as codified knowledge. The structure makes it easy to find, compare and apply the information. Examples for inserted information are presentations or reports, as well as any kind of background information regarding a product, tactics or users. All these are created to be re-used by colleagues in similar situations and reduce the consumption of resources by avoiding to do the same work again.

The *informal internal knowledge base* on the other hand covers that part of the knowledge, which is not formalized, i.e., plain text. This loose communication is the best medium to distribute tacit knowledge. Polanyi (1966)

defined tacit knowledge as the knowledge that is not easy to express (see also section 2.1.1). A lack of formalization addresses the possibility to externalize tacit knowledge more easily and to store it in this knowledge base.

Approaches to share the knowledge of the people led to the implementation of community based electronic forms of discussions. Over the time these systems pile up a history of discussions and textual communications, which can be used like an archive. The topics here are of various kinds, including insights from project experiences. The aim is to accelerate and simplify the work for colleagues by providing expertise in text form. Hence, internal informal knowledge bases are actually communication tools, because they:

- provide support during cooperation
- help finding people with expertise
- make asynchronous communication possible

Both internal knowledge bases are worth considering for knowledge management of any company. The advantages or disadvantages depend on the context of their utilization.

3.3.2 Wikis as Knowledge Management Systems

Walsham (2005) advocates that knowledge cannot be transferred, but merely communicated within a context, much research explains that the social nature of knowledge is lost through the processing of information (Pentland, 1995). The social web (also known as Web2.0) acknowledges this flaw and supports the communication between people, while providing the context (Dolog et al., 2009d). This kind of applications promotes collective intelligence, by enabling and supporting the collective work of communities (Jimenez and Barradas, 2010; Wheeler et al., 2008). Hence, the social web reflects the natural knowledge management in software, which is mostly lightweight, encouraging and non-restricting.

An example for a social web application that are candidates for knowledge management systems is the wiki (Gonzalez-Reinhart, 2005).

Since Ward Cunningham developed the *WikiWikiWeb* ("wiki wiki" is Hawaiian for "very quick") in the mid-1990s (Leuf and Cunningham, 2001), wiki systems (short: Wikis) have been increasingly used as communication platforms for collaborative work, particularly in software development (Wagner, 2004). A wiki is a web application, where users can create, read and edit content in a simplistic manner (Leuf and Cunningham, 2001; Wagner, 2004). The content is stored in form of pages, which then can be interlinked, like in hypertext. While users have the power to create and change the content, the system keeps track of edits. Therefore, users have not to worry that old content might get lost, in contrary, it encourages them to simply update pages. Everybody can see the history of pages at every time and jump back to older versions (illustrated in figure 8). Another key functionality in wikis is the search functionality, which helps finding content of interest (Leuf and Cunningham, 2001; Wagner, 2004).

As wikis are not limited regarding domain or context, they are very flexible. In addition, it is the core principle, that wikis are very easy to use and support simple collaboration on text. This flexibility and simplicity is part of the involved philosophy to use these systems: *The Wiki Way* (Leuf and Cunningham,

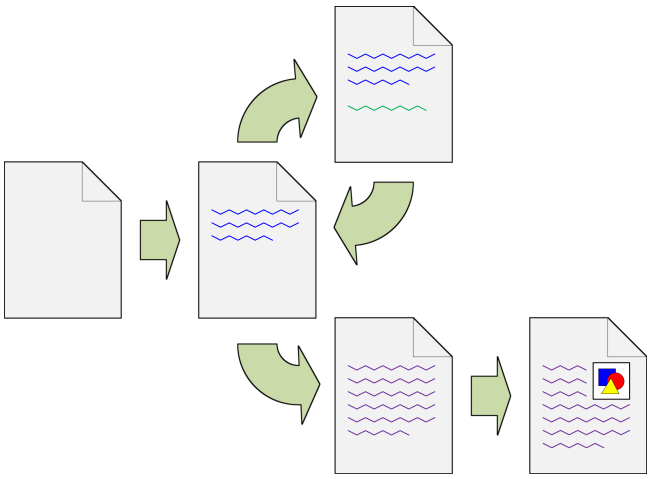


Figure 8: Workflow in a Wiki

2001). It relates to a culture of no worries regarding editing or creating content and supports collaboration. The popularity of the wiki way made scholars and practitioners consider wikis as knowledge management systems, especially in software development (Louridas, 2006).

The strong focus on collective work makes wikis a reasonable choice for knowledge management systems (Gonzalez-Reinhart, 2005) and many companies utilize wikis in their intranet. However, the wiki way is a significantly different approach to knowledge management compared to traditional approaches or systems (Hasan and Pfaff, 2007). Utilizing a wiki requires a unique approach to knowledge management, with a variety of practices (Kussmaul and Jack, 2009) and patterns (Mader, 2007), which are important to consider in order to be valuable (Raman, 2006).

To argue that a wiki is a knowledge management system, I want to briefly discuss its impact on the three aspects of knowledge management, stated by Rus and Lindvall (2002), which I describe above (see section 2.2). Wikis support the core software engineering activities by assisting not only in direct group communication, but the collaboration actually results in widely available documentation. This substitutes single documents on file servers, but the authors also display their competence to readers. Another consequence of the collaboration is the product and project memory. The evolving contents document the projects progress and experiences made through its realization. Learning and improvement is covered as every employee is able to check for previous achievements and build up on that.

Recently, wikis as knowledge management systems in organizations were addressed by research to show the benefits. This includes different case studies (Chau and Maurer, 2005; Raman, 2006) and a survey among corporate wiki users, which confirmed the sustainability of wikis (Majchrzak et al., 2006). Studies show that wikis help to easily exchange ideas and to support the communication process in companies (Raman, 2006; Sousa et al., 2010).

Sousa et al. (2010) analysed the contribution of wikis to the knowledge creation in a company, based on the SECI model (see section 2.1.1). They state

that the emphasis of the applied processes is on internalization, because wikis are mostly accessed by users that seek for information.

Hasan and Pfaff (2006, 2007) point out that corporate wikis have the potential to accumulate the collective experience of a company to a shared knowledge repository, but at the same time abandon knowledge authority. They show that wikis are only successful as knowledge management systems, if they are utilized democratically and not under the monopolistic control of a few.

The simplicity and flexibility of wikis provide many possibilities as a knowledge management system, which is highly appreciated by software engineers (Kim and Yan, 2010). Additionally, it allows technological enhancements to wikis, to improve the usefulness for specific tasks (de Almeida Ferreira and da Silva, 2009; Uenalan et al., 2008; Schaffert, 2006).

4

KiWi Project

In this chapter I explain the environment in which my studies were conducted. I begin with a description the project's general organization (section 4.1). My studies deal with one case, which focussed on the creation of a prototype (section 4.2). This mainly consists of the cooperation of two participating parties: Logica, the case company, and Aalborg University. The whole project is rather large and complex, with a strong focus on technology. That and the vision of the prototype conclude this chapter (section 4.3).

4.1 Project Organization

The project *KiWi – Knowledge in a Wiki*[6] was funded by the European Commission as part of the FP7 framework. Its defined objective was to study the possibilities of a semantic wiki in knowledge management (Schaffert et al., 2009). The aim was to design, implement and evaluate a prototype that addresses this objective. To successfully do that, two distinct cases were utilized. These represent business cases in collaboration with industrial partners. One of them was realized with Sun Microsystems (Prague, Czech Republic) and the Semantic Web Company (Vienna, Austria). The other one, which is the one my studies took place in, involved Logica (Aalborg, Denmark). These business cases were labelled as use cases in the original planning, even though the term use case has a different meaning in the literature (Mathiassen et al., 2000). To avoid confu-

[6] http://www.kiwi-project.eu/

sion here, I will use the term business case or simply case for the organization of the KiWi project.

The scholars of the project came from three universities: Aalborg University (AaU; Aalborg, Denmark), Brno University of Technology (BUT; Brno, Czech Republic) and Ludwig-Maximilians-Universität München (LMU; Munich, Germany). Their tasks included specific research and development tasks, according to their field of interest. The development of specialized solutions was called enabling technologies within the project.

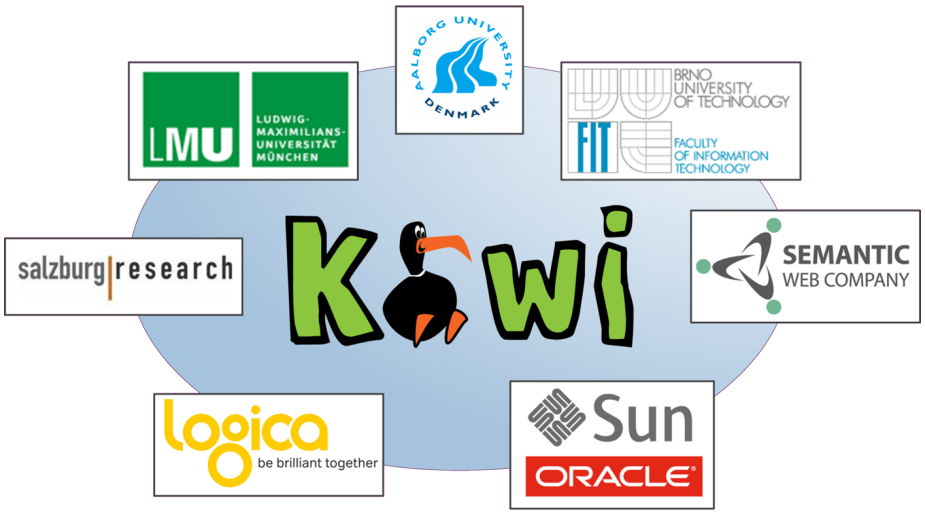


Figure 9: The KiWi Project's Participants

The remaining member of the seven party consortium of the KiWi project (figure 9) was Salzburg Research (Salzburg, Austria). People from there were responsible for the project organization and management as well as the development of the core functionality.

4.1.1 Work Packages & Deliverables

The whole project was divided into several work packages, with different responsibilities for the participants. Each work package contained a list of tasks or deliverables. The work packages were structured into the following classifications:

1. Management and Assessment
2. Enabling Technologies
3. Core System
4. Component Implementation & Integration
5. Requirements (Business Cases)
6. Application Building (Business Cases)
7. Evaluation and Testing (Business Cases)

8. Dissemination & Exploitation

9. Demonstration

Not every project member was involved into each work package. Also, the work packages were not necessarily sequential, some were organized in parallel. Some other work packages were barely related at all. Only the work packages 5 to 7 were really building up on each other, as they concerned the tasks for the cases (details in section 4.2 below).

To finalize a task within a work package, a concluding delivery had to be made. This delivery had to be sent to the coordinating partner (Salzburg Research), who handed it over to the European Commission. The type of a deliverable depended on the task. Some were reports, others the actual software or the work results in different forms.

For the creation of the reports an internal review process was applied. Every document was reviewed by one to three project members that were not directly involved in the work package's tasks. After the internal approvement then, it was sent to the European Commission. Table 3 lists the created deliverables that were reports, all of them are publicly available[7]. The little star (★) next to the title marks documents which were related to my studies and where I was a contributing author.

4.1.2 Scheduling

The project was running from March 2008 to February 2011. In this time range the different tasks from all work packages were scheduled. However, the organization of work within the different work packages was left open for the participants. Cooperating partners could thus organize themselves, within the boundaries of the project plan.

For a good communication among the team members regular meetings were arranged. The entire consortium met three times a year to present and discuss results. These meetings were also used to settle agreements and plan further steps.

Asides from these regular meetings that involved representatives from every participant, several meetings were arranged in smaller circles. They had the focus on specific work packages and only included the people that were directly affected. Additional meetings, which depended on work packages, were organized by the involved people. All kinds of meetings were scheduled among the participants.

4.2 The Logica Case

The whole project contained two business cases, which were identified through case companies. These case companies were project partners, cooperating in the development of a prototype. They were also the target environment to apply the prototype and evaluate it.

I was part of the case that involved Logica as a case company. Internally it was therefore labelled *Logica Case*, even though the name according to the

[7] http://kiwi-project.eu/index.php/publications/33-deliverables/

No.	Title	Reference
D2.1	Reasoning & Querying – State of the Art	(Bry et al., 2008)
D2.2	Reasoning & Querying – Concept and Model	(Bry and Weiand, 2009)
D2.3	Reason Maintenance – State of the Art	(Bry and Kotowski, 2008)
D2.4	Reason Maintenance – Concept and Model	(Bry and Kotowski, 2009)
D2.5	Information Extraction – State of the Art	(Knoth et al., 2008)
D2.6	Information Extraction – Concept and Model	(Schmidt and Smrž, 2009)
D2.7	Personalisation – State of the Art	(Durão et al., 2008)
D2.8	Personalisation – Concept and Model	(Durão and Dolog, 2009b)
D3.1	Architecture Revision	(Schaffert et al., 2008c)
D5.1	State of the Art Software Knowledge Management	(Samwald, 2008)
D5.2	Requirements Software Knowledge Management	(Holy et al., 2008)
D5.3	State of the Art Project Knowledge Management	(Nielsen and Dolog, 2008)
D5.4	Requirements Project Knowledge Management ★	(Nielsen et al., 2008)
D6.1	KIWI Knowledge Model for Sun CEQ Use Case	(Dolog et al., 2009a)
D6.2	Application Building Report Sun Expert Manager Use Case	(Holy et al., 2010)
D6.3	Knowledge Model: Project Knowledge Management ★	(Dolog et al., 2009b)
D6.4	Implementation: Project Knowledge Management ★	(Dolog et al., 2009c)
D7.2	Test Plan: Logica Use Case ★	(Grolin et al., 2010a)
D7.4	Evaluation and Testing Project Knowledge Management ★	(Grolin et al., 2010b)
D8.5	KiWi Vision	(Schaffert et al., 2008b)

Table 3: Deliverables of the KiWi Project

original planning was the *Project Knowledge Management Use Case*. It was targeted on knowledge management as a project management activity.

The original planning followed a waterfall manner: Beginning with the collection of the requirements, followed by the building of the application and concluded with an evaluation (figure 10). Each period here stands for a work product and ends with the delivery of a report summarizing the results. Intermediate results were communicated in additional deliverables. The work products and every deliverable were scheduled in a detailed work plan, which was accessible for every project participant.

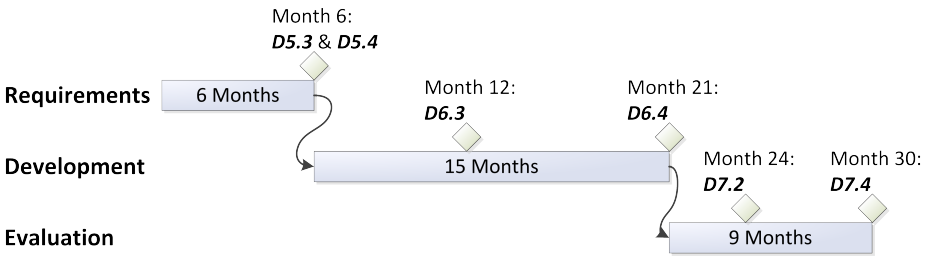


Figure 10: Time Schedule for the Project Knowledge Management Use Case

The project plan contained three work packages for the Logica case: Requirements, development and evaluation. As explained above this organization follows the waterfall paradigm, each work package builds on top of the preceding one. But a closer look at the implementation period, which mainly covered the application building work package, reveals that it actually contains two distinct assignments: Creating a knowledge model (D6.3) and developing a prototype (D6.4). Again the one builds on top of the other, the knowledge model is part of the prototype. These two assignments are sequential with milestones and fixed delivery dates (figure 10). Due to the fact that the KiWi platform is a semantic wiki, a properly developed knowledge model is of high importance. To emphasize its significance, this aspect of the development period became a project phase on its own.

The Logica case was therefore internally understood as being divided into four phases: The requirements specification, the creation of a knowledge model, the development of a prototype and the evaluation of that prototype (figure 11). All four phases were realized mainly in close cooperation with the project partners Logica and the Aalborg University. Below I describe the case company.

Note that one aspect of the Logica cases' work was to utilize the core system and the enabling technologies. This necessitates collaboration with the other partners. They however were hardly involved in the conceptual work of the case and therefore not further mentioned here.

4.2.1 Case Company: Logica

Logica Denmark[8] is a company that provides IT and business solutions with more than 40.000 employees worldwide. About 800 IT and software specialists work in five different cities in Denmark. Software is developed in projects of dif-

[8] http://www.logica.dk/

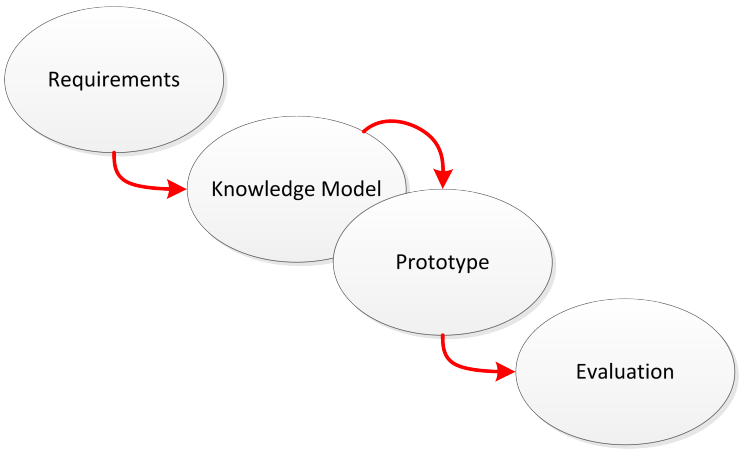


Figure 11: Project Phases for the Logica Case

ferent areas (e.g., banking, governmental agencies, and the educational sector). Most of the development projects are realized in Java.

A project in Logica represents a work unit and is as such staffed with a set of people. Many projects share the administrative matters and different parts of the managing level, but the developers are usually involved into one project at a time. Further, Logica does not have a universal IT infrastructure that is binding for every project. The setup of each project can differ, as the highest priority is solving a customer's problem.

The driver for Logica's participation in the KiWi project is the company's organization in projects. It leaves high potential for improvement through knowledge management, as the inter-project communication is difficult to maintain.

The knowledge management problem of Logica is being analysed and discussed below (chapter 6). Logica's involvement focused on the role of a case company: Helping to understand the problem, defining the requirements, discussing intermediate-results and evaluating prototypes.

4.3 KiWi: A Knowledge Management System

The KiWi project had a clear objective: The development of a knowledge management system: *The KiWi Platform*. It was therefore a project that focused on technical aspects, which are very complex. But the KiWi platform also followed a vision, which was the driving force in this project (Schaffert et al., 2008b).

This so-called KiWi vision describes that social software became an important aspect of communication among many people world-wide. People are connected to networks and exchange ideas and thoughts about multiple different things. The KiWi project's approach to knowledge management makes use of that.

One aim of the KiWi project is, according to the KiWi vision, that users should be able to share their knowledge in a simpler way than most knowledge management systems allow. To achieve that, the user has to be the central element, from the beginning of the project. The system shall support the users

in finding what they need and help them in sharing what might be beneficial for others. The user plays a crucial role in knowledge management generally, as explained above (see section 2.4.1). If the users do not work with the system, the whole approach is likely to fail. Therefore the knowledge management systems have to represent additional value to the users. This could be achieved through efficiency or advanced features. The KiWi project aims for both.

The KiWi vision explains that high efficiency is approached through simplicity. Accordingly, a tool's processes can increase the complexity of simple tasks. The wiki way seems like a solution to that, with its simplicity and flexibility. That is the reason for the KiWi project to build a platform on top of the wiki technology. As explained above (see section 3.3.2), a wiki is a collaboration platform that allows users to cooperate on text pages. The KiWi vision describes that systems of this kind are popular because of their simplicity: Any user, who wants to edit a page, can simply click a button. Desired content changes are easily committed, but the system's history allows users to roll-back the changes. These simple steps also support the collaboration among colleagues, the wiki way (Leuf and Cunningham, 2001).

The KiWi platform's goal was to follow the wiki way. Users should have to be able to collaboratively edit pages. This strong focus on the social web and the wiki way as the underlying philosophy is the basic idea of the KiWi project and as thus the one of the whole approach to knowledge management. To realize a system that follows this philosophy, different technological aspects were necessary.

4.3.1 The Semantic Web

The semantic web was defined as a basic technology in the project's vision (Schaffert et al., 2008b). Berners-Lee et al. (2001) outlined the semantic web as a new form of the internet. They criticised that information on web pages is only apprehensible by human beings. The result was an envisioned scenario, where the data can be accessed and processed by computers directly (Berners-Lee et al., 2001). Based on this vision, much research and development has been done (Shadbolt et al., 2006; Feigenbaum et al., 2007). A strong focus is the development of technologies to store the data on the web.

The underlying concept is radically different from traditional approaches, i.e., the basic ideas behind relational databases. The semantic web follows the open world assumption, which does not claim to be complete. It expects that things exist, which are not represented in the data, yet. The closed world assumption, being the opposite, has a fully defined set of data. That means that something does not exist, if it is not in the data. To illustrate, the KiWi vision mentions the example of a white raven. The closed view states that there are no records about a white raven, therefore it does not exist. The open view instead states that we have no knowledge about a white raven, but that does not necessarily mean it does not exist.

This open world assumption results in the expectation that new input will be added eventually and that the data is at all times likely to be incomplete. Technically, this is realized through different standards and languages. Two commonly used ones are applied in the KiWi project: Resource Description

Framework (RDF[9]) and Web Ontology Language (OWL[10]). RDF is a set of specifications and OWL a family of languages. Both are under the wings of the World Wide Web Consortium (W3C[11]) and both are providing the possibilities to describe data.

Data in the semantic web is defined through statements about a resource using simple expressions like "a person has the name Paul" (figure 12). The data is actually described using a subject for the resource (denoting "a person"), a predicate for a condition of the subject (denoting "has the name") and an object for the target of the predicate (denoting "Paul"). This notion of three parts in RDF is called a triple.

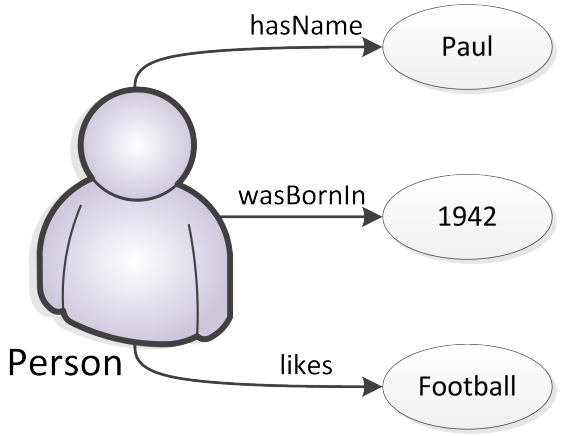


Figure 12: Data Representation in the Semantic Web

In the semantic web lingo, when it comes to realization, the resource is defined by a *type* (commonly used is also the term *class*), the predicate is called a *property*, targeting at either plain data or another resource. Data, that is structured like this, i.e., the data within in the semantic web, is called semantic data.

A set of RDF statements can be used to describe a data structure in a labelled, directed multi-graph (figure 12). But RDF can not only be used to describe data; it can also define a general data structure. Classes can be defined with properties and relations to other types. Such a data structure in the semantic web is called *knowledge model* or *ontology* (figure 13).

The technical realization in the semantic web is a complex field. Many different standards and specifications exist. RDF itself does not specify a syntax. Instead, it defines guidelines, concepts and features to describe data. To put this description into a language, XML[12] is one commonly used form. As RDF is rather limited, when it comes to the creation of more complex knowledge models, other technologies were built on top of that. The knowledge models for the semantic web, which the KiWi platform is supposed to operate, are authored in OWL.

[9] http://www.w3.org/TR/rdf-primer/
[10] http://www.w3.org/TR/owl-ref/
[11] http://www.w3.org/
[12] Extensible Mark-up Language: http://www.w3.org/TR/xml/

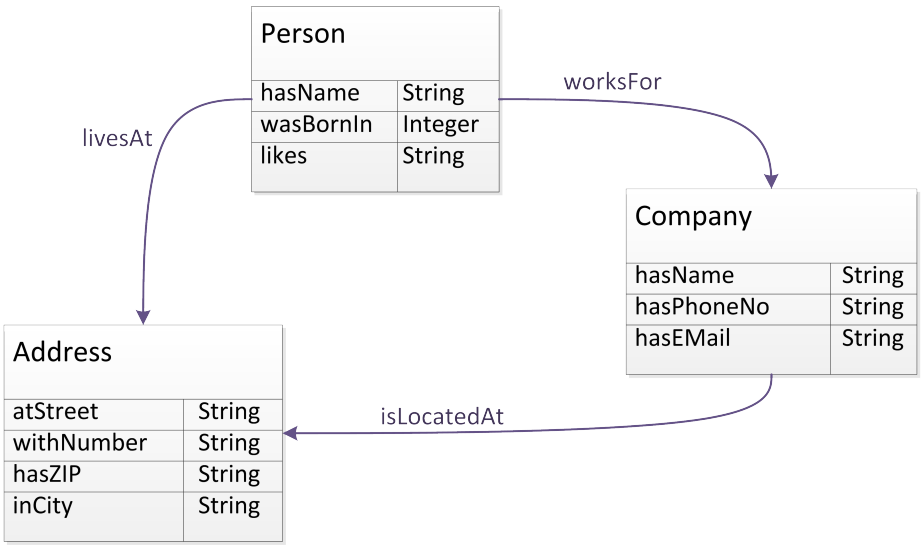


Figure 13: Ontology or Knowledge Model for the Semantic Web

Endorsed by the W3C, OWL is a family of languages that utilizes RDF in XML to define ontologies. But unlike data schemas for relational databases, a semantic data structure is not strict. In contrary, it can be compared to a temporary agreement, made to be extended or improved over time. The whole system is built to embrace evolving ontologies (Berners-Lee et al., 2001; Shadbolt et al., 2006; Feigenbaum et al., 2007).

RDF is not only used to describe data structures in general, but also to provide the actual data. In the semantic web, the goal is to provide information that is not only understandable by human beings, who can read a text, but also by machines. The RDF approach to this vision understands a web address, represented by its Unified Resource Identifier (URI) as a resource. Each resource then can have two documents, one containing the textual content for humans and one containing the semantic data, i.e., the RDF formatted data (figure 14).

An extension of RDF however, allows the combination of textual content and semantic data into the same document. RDFa[13] (RDF in Attributes) enables the use of RDF within the attributes of an XHTML[14] document. This way passages from the textual content can be directly assigned to properties of the semantic data.

4.3.2 Semantic Wikis

The possibilities of the semantic web are huge, which is why it became a candidate to be utilized in knowledge management. There is a variety of different approaches, which try to use the semantic web to integrate enterprise knowledge (Gu et al., 2006). Some create ontologies to define and organize knowledge at a generic level and to enable querying (Gómez Pérez and Benjamins, 1999;

[13] http://www.w3.org/TR/xhtml-rdfa-primer/
[14] Extensible Hypertext Mark-up Language: http://www.w3.org/TR/xhtml1/

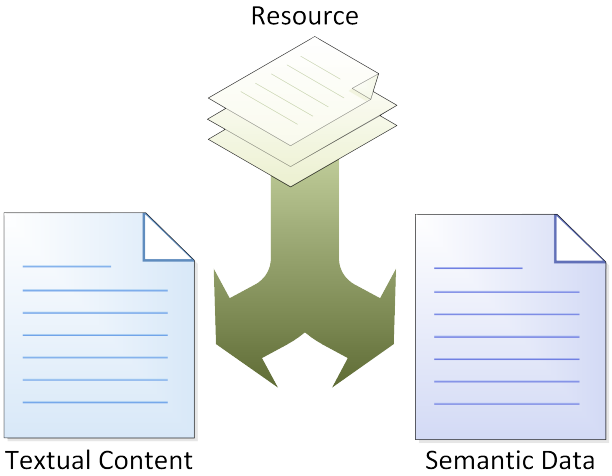


Figure 14: Resources in the Semantic Web

Fernandes et al., 2003) or discuss how ontologies can support knowledge sharing generally (Gruber, 1995).

As wikis are widely utilized for knowledge management activities today (see section 3.3.2), it was recently approached to integrate the semantic web into wiki systems. The KiWi vision (Schaffert et al., 2008b) acknowledges this development and explains that, as users got used to working with wikis, their expectations on this technology rose. The result is called semantic wiki (Schaffert et al., 2008a). It is based on the idea to combine the strengths, wikis have strong support for collaboration, but the content lacks structure, which is provided by the semantic web (Oren et al., 2006). This enhances the technical abilities of handling data within a wiki (Rauschmayer, 2009). Semantic wikis can for example support the re-use of software and knowledge in software engineering (Decker et al., 2005; Shiva and Shala, 2008).

Schaffert (2006) presents a semantic wiki called IkeWiki that follows the basic wiki principles, enhanced with semantic web technologies. Content then can not only be text but also semantic data. A similar approach extends an open source wiki to deal with RDFa (Schmedding et al., 2008). This technological solution allows the user to edit the semantic data directly in form of XML tags through the editor of the main text content. Other technologies require separated files and/or editors. KnowWE is another different semantic wiki that focuses on active problem solving capabilities (Reutelshoefer et al., 2008). Missing links to concepts or relations are derived utilizing the explicit knowledge that is described in a formal syntax. Another approach focuses more on the data access and search functionalities (Oren et al., 2006).

A wiki can be tailored for a task or objective. This focussing or specialization is a quality of wikis in general, which also applies for semantic wikis. The examples above show the variety of focus for different ones.

4.3.3 The KiWi Platform

The prototype imagined in the KiWi vision was supposed to integrate the semantic web with wikis and basic ideas of social software (Schaffert et al., 2008b, 2009). Users were supposed to log on and use it as their knowledge management system. The system works page-based, users have to create pages, edit their contents or comment on them to share knowledge. This system was named KiWi platform or short KiWi.

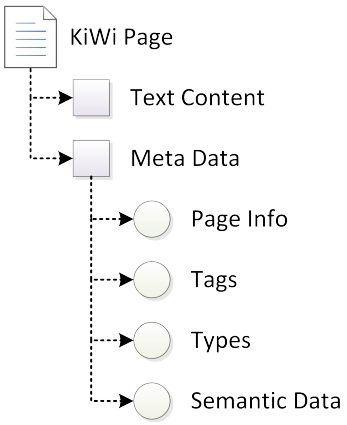


Figure 15: A KiWi Page and its Contents

Pages in the KiWi platform are called *KiWi pages*. Each of these basically consists of two things: Text content and meta data (15).

The *text content* can technically be substituted with all kinds of media that can be displayed in a wiki, e.g., videos or images. Our studies however focused on textual contents. The reasons are simple: Text is the easiest and the most common form of content to create and edit. Additionally, it makes a lot of sense to use text for knowledge sharing, as the people in software development companies are used to creating and editing text. The textual content on a KiWi page therefore symbolizes everything the users write directly through the editor, like with any other common wiki.

The *meta data* contains the additional information to that page, which is partly hidden or marked as specific information. The *page info*, to begin with, is automatically attached to any page. It contains information about the date of creation, the authors, etc. When editing, the author can add *tags* to a page. A tag is a free form term that does not follow a defined structure. If the page deals with the profile of a person for example, tags could be "personal profile" and "employee". Users can add tags to pages and thus use catchwords to illustrate the page's content. The user can also assign *types* from the ontology to a page. With these types a user defines the class of a page. A profile page can for example be assigned to the type "person". Hence, a KiWi page is a resource and can contain *semantic data*. This data can be added and edited manually, supported by the enabling technologies. Semantic data represents the textual content of the page. For the exemplary profile page it could thus be "Person hasName Paul" and the other triples from the example of the semantic web (figure 12).

Note that a KiWi page has more meta data than a regular semantic web URI (cf. figure 14). The semantic data of a regular semantic web resource corresponds to the semantic data in a KiWi page. KiWi pages however have more meta data, based on its background as a wiki. This does not mean that the tags of a KiWi page for example are not handled through semantic data. The KiWi platform however distinguishes between the different types of meta data and utilizes them differently.

4.3.4 Enabling Technologies

The KiWi vision (Schaffert et al., 2008b) explains that the semantic data provides the additional value, for the system to operate on the user generated content. This shall provide a variety of possibilities, which were investigated and utilized by that part of the KiWi project that is responsible for the enabling technologies. The vision describes that these enabling technologies should help to create a system, which is driven by the idea of a system that follows the wiki way.

The enabling technologies are described as a set of components that enhance the KiWi platform by providing specific functionalities in the vision. They are part of the project to show that enhancing a knowledge management system with these technologies increases its quality. Consequently, the two business cases then utilize the enabling technologies and show how they can be applied in knowledge management and how that can be beneficial for the case companies.

The enabling technologies are:

- Reasoning and Querying
- Reason Maintenance
- Personalization and Adaptation
- Information Extraction.

The development of each of the enabling technologies was performed in close cooperation with the development of the core functionality and with the business cases. A close cooperation with the core development was necessary as the component had to be integrated into the system. So the developers communicated on a rather technical level. The relationship to the cases on the other hand was more on a conceptual level. The involved people tried to investigate possibilities and ways to a successful realization, however, the technological possibilities and the envisioned systems did not provide a complete match. Some requirements had to be declined, as they were not feasible to be realized due to technological limitations. On the other hand, the technology provided features that were not anticipated, but utilized afterwards.

Reasoning and Querying

The features provided by the reasoning and querying component allow both system and users to operate on the semantic data within the KiWi platform. The system contains computable data, this component helps processing it. Developed by researchers from LMU, this component was separated into two aspects: Reasoning and querying. Both operate on the ontologies and on the provided data.

Querying is a search or filtering mechanism to find pages within KiWi utilizing freeform expressions, types and tags. Users can actively use KWQL, the KiWi Query Language, to define commands which are then interpreted to access and query the semantic data of all pages within the KiWi platform (Bry and Weiand, 2010). The commands can be typed in manually, but also by utilizing a graphical user interface. The main focus of the querying is to improve the search functionality, which can interpret queries, if formulated in KWQL. However, templates for the most common queries are also provided. This functionality supports the users in finding what they are looking for. Additionally, KWQL can filter the search results to specific aspects, so that the number of search results is limited and stays manageable.

Reasoning is handled by an engine, which adds implicit types or infers additional ones to pages. An example: A user assigns the type `Java` to a page. The system supports the user by adding the implicit type `ProgrammingLanguage`. The reasoning engine infers this from the ontology, which states that `Java` is a subclass of `ProgrammingLanguage`. The system further adds the type `ObjectOriented`. The reasoning engine infers this from a defined rule, which defines Java as one of the programming languages that are object-oriented.

These two examples are of very different nature, even though both result in adding one type. In the first example, the type is concluded according to the class hierarchy within the ontology. All, or relevant, super-types are added. The second example takes rules into account. A rule is a manually defined order of commands the system has to perform, once the condition is fulfilled or an event occurs (like: *if A, then B*). Here it was a simple rule like: `if the page is of type Java, then add type ObjectOriented.`

The reasoning engine does all this work in the background. It is triggered by the system every time a user saves a page.

Reason Maintenance

The project participants from LMU were also responsible for the reason maintenance. This component has basically two tasks related to the reasoning: Explaining it to the user and triggering it.

As the reasoning is running in the background, users might be confused by automatically added types. To avoid that, the system is obliged to explain the details behind these added types and provide reasons for its behaviour. This makes the system better comprehensible for users.

The reasoning can be started manually, but it is also triggered by different events, e.g., when a page is saved. While the reasoning adds or infers types, the reason maintenance component additionally revises these actions.

An example: A user assigns the type `WorkInProgress` to a page. An existing rule defines that every page of type `WorkInProgress` has to be of type `ToDo`, too. The reasoning engine therefore adds the type `ToDo` to the page. Meanwhile the reason maintenance component creates a message for the user with an explanation, where the new type came from. Once a user deletes the type `WorkInProgress`, the reason maintenance component takes care of deleting the `ToDo` type as well, because the condition is no longer fulfilled. The component also creates a message, why the type was deleted.

However, it is not the reason maintenance component's task to avoid inconsistencies within the system. The KiWi platform is considered as a collaborative platform, which human beings use to interact. While doing so, inconsistent data on a page has to be possible and is actually even an expected state. The users have to have full control over all data. The reason maintenance component is therefore just a helper in the background, which tries to support the user. If a user disagrees, the changes of the component can be undone without difficulties.

Personalization and Adaptation

The personalization and adaptation targeted on matching the view of pages or the system in general to individual preferences and behaviour. The component was developed by people from AaU. They focused on supporting users in finding content that is useful for them (Dolog et al., 2011). To do that, the system provides recommendations based on the context. The system knows what pages the users read, which ones they open, edit or comment on. This information is used to create a profile, which allows the system to predict the interests or focus of users.

An example: George is a developer who creates and edits a lot of pages in the KiWi platform that cover databases and programming languages, as that is his main focus. He writes about the technical realization of different projects, discusses set ups with his colleagues and comments on different approaches. When George searches for a specific project, it is quite likely that he is not interested in the financial establishment. Instead, he is interested in the technical aspects of the project. A list of search results could therefore sort the results according to relevance for him.

This example uses the personalization and adaptation component to sort search results. Other applications are the recommendation of pages that might be interesting to the user. This should happen directly, without the need of a user even searching for it. When viewing any KiWi page, the system displays recommended pages that might be relevant to the user in this context. Another possibility is an overview of pages that generally might be interesting for users.

Information Extraction

The researchers from BUT were responsible for the information extraction component. Its target is the analysis of the text content and creation or suggestion of semantic data and tags based on that.

The manual task of adding any form meta data can scare people off. This information extraction assists users in assigning the semantic data. It analyses the textual content of the page and provides suggestions accordingly. This supports the users in their editing speed and at the same time assures that the pages contain proper semantic data.

Many users do not see the direct need for assigning semantic data all the time. Therefore, they have a tendency to not take it very serious and dismiss the tagging activities. The reasons for that are simple; nobody likes to do things they do not value. But the whole system would suffer from poor tagging. Not properly tagged pages would decrease the ability of processing the content. It is therefore important to have support, which makes this activity a simple task for the users and increases the quality of the semantic data at the same time. The

information extraction component is therefore an important aspect to assure quality of service.

4.4 Summary

This chapter introduced two basic yet very different aspects of the KiWi project, which have a major impact on my research. On the one hand there is the vision of a knowledge management system and on the other hand there was the organization of the project.

The development of the KiWi platform followed a clear vision. The knowledge management system had to utilize the semantic web, which is an approach to make the internet better computable by providing semantic data to web resources. This knowledge management system also had to follow the wiki way, which does not just include the wiki functionalities on text editing, but a general approach to simplistic systems. The KiWi platform further was being enriched by what in this project was realized through enabling technologies, which add features for querying, reason maintenance, personalization and information extraction.

The development of the KiWi platform was planned in several aspects. The business case, which is reflected through the research in this thesis, was arranged in close cooperation with the case company, Logica. This case was organized in a waterfall manner, running through four major phases: Requirement specification, knowledge model creation, prototype development and evaluation.

5 Research Approach

This chapter describes the approach to my research. As I explain in the previous chapter, the project organization was defined prior the beginning of the project, which set the boundaries for my research. The project was planned in the way that the research is conducted in close cooperation to the case company. A prototype for a knowledge management system had to be built, approaching the problems of the case company.

With this setting and in respect to the involved partners the scope for a research approach is roughly outlined already. however, in order to be successful, the detailed organization is very important. The dilemma between practical concerns and research goals in collaborations with arrangements like this has been addressed by much research. Both parties, the industrial and academic, strive for different things and the cooperation can only be successful, if both reach their goals. The scholars therefore are obliged to not only use the gained knowledge for own benefits, but also apply it for the practitioner's good. The goals of the industrial partner have to be reflected in the research goals as well.

In a setting like the given one, Mathiassen (2002) distinguished three kinds of research goals and the activities to reach these (figure 16). These three goals are namely: Gaining an understanding, supporting the industrial activity and improving its results. They make it clear that the scholars's role goes beyond a plain observational activity. Instead, successful research in close cooperation with industrial partners requires interaction.

A methodology that takes these goals into account and fits into the setting of the KiWi project is action design research as outlined by Sein et al. (2011). In the following sections I explain this in more detail (section 5.1), give reasons

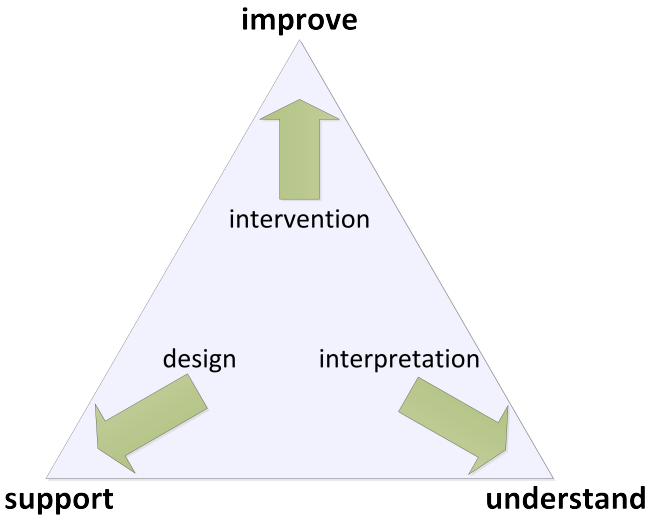


Figure 16: Research Goals and Activities, from (Mathiassen, 2002)

for my choice (section 5.2) and explain how I applied it for my studies (section 5.3).

5.1 Action Design Research

Sein, Henfridsson, Purao, Rossi, and Lindgren (2011) outlined a strategy to perform research, which interacts in close cooperation with industrial partners, called *Action Design Research* (ADR). This section is an introduction of the concepts and processes utilized in ADR.

5.1.1 Prerequisites

ADR is based on a broad consensus within the field of Information Systems research, to approach this kind of research with a dual process. The goal is to make a contribution to both aspects: One aspect is supporting the practitioners in an application domain and the other one is the formation of a theory to be added to the general body of knowledge (figure 17).

Figure 17: Dual Approach in Information Systems Research

IS research always has the goal to gain relevance (Benbasat and Zmud, 1999). The two areas of concern are hereby of equal importance. On one hand, the research's mission is making a theoretical contribution and thus extending the

general body of knowledge. The analysis of the work in an industrial environment, its evaluation and results are supposed to be documented and published, in order to be beneficial for future research. On the other hand, the scholars are interacting with practitioners within an application domain and should support them in their problem solving.

As displayed in figure 16, the cooperation between academic and industrial staff can result in better processes or products. The scholars can contribute to the problem domain by the findings they gained from their studies. This includes knowledge gained from the literature and from the results of the interpretation of the practitioners' current and anticipated problems. Thus, research can support the practitioners in solving these problems (Wieringa, 2009).

In order to gain a better understanding of ADR it makes sense to have a brief look where it came from. ADR is a methodology that is based on design research. The objective of design research is the study of the design process. Doing so, design research aims at getting a better understanding of the design process, in order to improve it (Eekels and Roozenburg, 1991). Many different methodologies build on top of design research. A popular representative of this is *Design Science Research* (DSR) as outlined by Hevner, March, Park, and Ram (2004). Even though both approaches have the common understanding in basic, ADR began as a criticism to DSR.

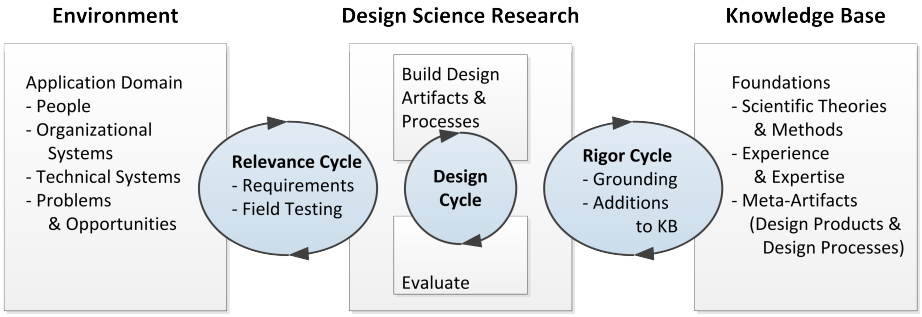


Figure 18: Design Science Research Cycles, from (Hevner, 2007)

DSR follows the dual process as explained above and works iteratively, similarly to ADR. In DSR, an IT artefact is being developed and evaluated, based on an existing problem in the practitioner's environment. Seven guidelines describe this process, which are repeated throughout the project. The whole process is organized in three cycles (figure 18): The relevance cycle, the design cycle and the rigor cycle (Hevner, 2007; Hevner and Chatterjee, 2010). The *relevance cycle* combines the work environment with the research project. It gathers input for requirements and acceptance criteria, in order to create and evaluate the artefact. The *rigor cycle* connects the knowledge base and the research project. Its main target is to ensure research contributions. This is achieved by understanding the state-of-the-art theories and methods. Extensions or new approaches to theories and methods, as well as other findings can thus be communicated, based on an informed grounding. The *design cycle* is internal for the research activities. It iterates more rapidly than the other two circles and combines the creation of artefacts, their evaluation and refinement based on preceded feedback.

ADR agrees with the understanding of the three cycles and their allocation. However, it emphasizes the relevance cycle's influence (Sein et al., 2011, p. 38). On the contrary to DSR, ADR provides a concerted research effort through the explicit guidance for a combination of building, intervention and evaluation (see section 5.1.3).

5.1.2 Ensemble View of IT Artefacts

In both, DSR and ADR, artefacts are created to investigate a problem. The creation of such an artefact and especially its evaluation in cooperation with the practitioners help the scholars to gain new insights about a problem, which can lead to a possible solution. The quality of each artefact is then evaluated and discussed in an iterative process. Based on the reached understanding and findings of the previous evaluation, a new artefact is then created.

Any artefact is built to provide the researchers with an opportunity to analyse it by applying empirical and qualitative methods. The detailed definition of what an artefact is, however, is widely discussed within the field of information systems. In this thesis I do not want to find my own definition, but align to the ones that are used by the authors of the methodologies I utilize.

An IT artefact, in the sense it is used in DSR, is mostly represented in a rather structured form:

> *IT artifacts are broadly defined as constructs (vocabulary and symbols), models (abstractions and representations), methods (algorithms and practices), and instantiations (implemented and prototype systems). (Hevner et al., 2004, p. 77)*

Hevner et al. (2004, p. 77) mention in their article that much research is focused on the instantiations, i.e., the creation of systems. In opposition to that, Orlikowski and Iacono (2000) argue that technology, though it might be a central element, is just one aspect of a dynamic and social process. They explain that any technology is always the result of interaction between people and that technology emerges through design, development and maintenance. The IT artefact is therefore not only dependent of the people working on it, but also the context in which it is developed and applied (Orlikowski and Iacono, 2000). Based on this understanding, the authors suggest to extend the focus from the IT artefact and take the social interaction into account:

> *IT artifacts are designed, constructed, and used by people, they are shaped by the interests, values, and assumptions of a wide variety of communities of developers, investors, users, etc. (Orlikowski and Iacono, 2001, p. 131)*

A plain focus on the technology is therefore too limited. The people using the technology and those that develop it have to be taken into consideration as well. The same counts for the context, in which the technology is applied. Orlikowski and Iacono (2001) coined the term *ensemble view* for this.

The ensemble consists of the IT artefact and the dynamic interactions to people at the same time. The IT artefact here represents the plain technology, as described in DSR. However, it is clearly stated that the technological aspect is just one element of the ensemble. Five premises were created to support the theorization of IT artefacts within an ensemble (Orlikowski and Iacono, 2001):

1. IT artefacts are shaped by the developers and context.
2. IT artefacts are embedded in a specific environment.
3. IT artefacts mostly consist of different components that have to cooperate.
4. IT artefacts undergo various transitions during development.
5. IT artefacts mostly evolve over time, even after the original development finished.

One of the main points of criticism on DSR from ADR is precisely this focus on the IT artefact. Sein et al. (2011, p. 38) explain that the shaping of the design and the deployment of an IT artefact within an organizational context is of high importance in a research project. They therefore apply the ensemble view for ADR. This position widens the perspective in comparison to DSR from a view focused on just the IT artefact itself to the whole context of the IT artefacts; its design, development and application.

Note that the understanding of the IT artefact as such stays the same, similar to what it is described like in DSR. But a plain focus on the IT artefact would be too restrictive and ignore its environment. Therefore the centre of attention within the method is shifted. Instead of plain focus on the IT artefact, the ensemble view takes the dynamic interaction between people and technology into consideration (figure 19).

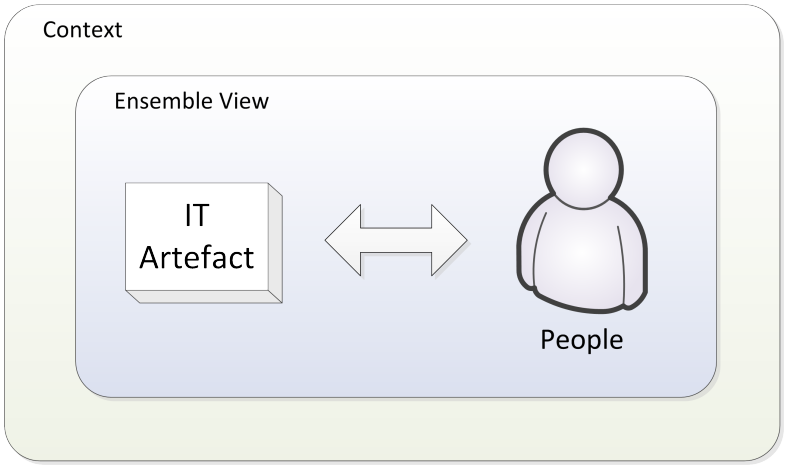


Figure 19: The Ensemble View of an IT Artefact

The shaping of both the design and the deployed artefact is widely influenced by the organizational context, which is what ADR wants to take into account. Working on the ensemble view of IT artefacts therefore contains the interpretation of the situation, the design of an alternative approach and the intervention through it. These are also the actions that lead to the general research goals (Mathiassen et al., 2002) as described above (figure 16 on page 58). Hence, the ensemble view itself is already an attempt to reach the research goals.

DSR acknowledges that the organizational context is of high influence and importance for the research (Hevner et al., 2004, p. 78). However, it does not

provide guidance to deal with these different dimensions. In contrary, ADR stresses this aspect in its procedure, which I explain in the next section. A core concept of the methodology is the utilization of the ensemble view of IT artefacts, in order to take the organizational context into account.

Sein et al. (2011) call the ensemble view of IT artefacts in ADR simply *ensemble artefacts*. Personally, I find this term misleading, as it can be interpreted like a substitute to the IT artefact. Instead, the ensemble view extends the circle of attention (figure 19). The IT artefact is still a central part of the research, yet just one part. Hence, I will use the term ensemble view as coined by Orlikowski and Iacono (2001) in this thesis. The description of IT artefacts and the ensemble view in my studies follows in section 5.3.1 below.

5.1.3 ADR Method

A major critique on design research approaches is the sequencing and therefore the separation of building and evaluating the artefact. Especially in the case of the ensemble view of artefacts the creation is a process of constant alternation between the design and evaluation within the organizational context. Therefore, Sein et al. (2011) combined action research characteristics and design research approach and created ADR.

In design research problems are approached by the creation of artefacts and their evaluation. The reason is to find out whether the problem was properly understood, the design of a solution was appropriate and the result can solve the problem (March and Smith, 1995). DSR, as briefly introduced above, is one approach of a research method that builds upon design research (Hevner et al., 2004).

Equally to design research, action research is a problem solving paradigm. But action research has an emphasis on the intervention by scholars. Introducing change and observing its effects is the core idea.

Action research approaches to solve organizational problems through a combination of intervention and theory generation (Babüroglu and Ravn, 1992). ADR is a design research approach that is strongly influenced by these action research concepts (Sein et al., 2011).

Further, ADR is a method that focuses on building, intervention and evaluation. This is tackled by the utilization of the ensemble view of IT artefacts. Each of these has two goals. On the one side, they should be reflecting the researchers' theoretical objectives and, on the other side, they aim to influence the context (Sein et al., 2011, p. 40). To achieve these goals, ADR is organized in four stages: The problem formulation, building intervention and evaluation of the artefact, reflection and learning, and finally the formalization of learning (figure 20).

These four stages build upon seven principles, which I explain within the description of each stage.

Stage 1: Problem formulation

The perception of a problem triggers people to express the need for research effort. In the first stage the initial research questions are formulated, the initial scope is determined and the participation of practitioners is defined (Sein et al., 2011, p. 40). A research opportunity is then identified, based on existing theories

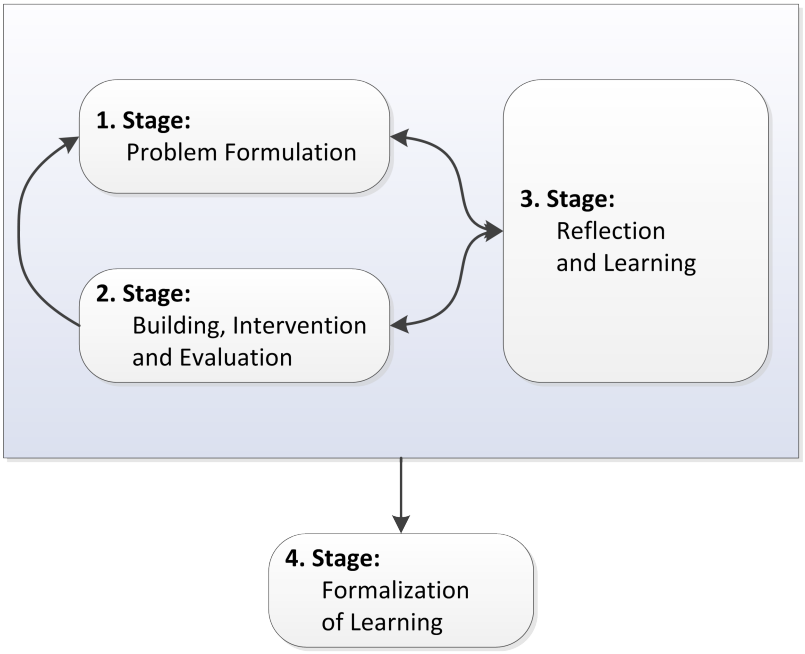


Figure 20: Stages in the ADR Method, from (Sein et al., 2011)

and technologies. The problem formulation further outlines the research efforts and knowledge creation opportunities.

These are the tasks to follow in this first stage (Sein et al., 2011, p. 41):

1. Identify and conceptualize the research opportunity
2. Formulate initial research questions
3. Cast the problem as an instance of a class of problems
4. Identify contributing theoretical bases and prior technology advances
5. Secure long-term organizational commitment
6. Set up roles and responsibilities

Note that task 3 points out, that the problem should be defined as an instance of a class of problems. This classification helps the researchers to generalize and create knowledge that is better applicable in different contexts. Task 5 strives for sustainability. It stresses that the scholars should try to make sure that the results are used in long-term and beyond the research effort.

The whole stage's work is drawn on two principles: Practice-inspired research and theory-ingrained artefact (Sein et al., 2011, p. 40). The *practice inspired research* principle emphasizes that the knowledge-creation opportunities are field problems and not theoretical puzzles. Doing so, scholars are not supposed to act as consultants that solve a specific problem. Instead, the action design researcher should create knowledge, which is applicable to the problem class.

The *theory-ingrained artefact* principle emphasizes that the design of IT artefacts is informed by theories. Structuring the problem, identifying possible solutions and guiding design activities are partly overlapping, but generally the acknowledged ways of using theories (Sein et al., 2011, p. 41). These reflect explanation and prediction theories as well as design and action theories (Gregor, 2006).

Stage 2: Building, Intervention and Evaluation

The second stage utilizes the problem framing and theoretical premises from the first stage. Based on them an initial IT artefact is created. Through following design cycles and organizational use, it is then shaped further. In an iterative process the three phases of a BIE cycle are then held: *Building* of the IT artefact, *intervention* in the company and *evaluation* of the ensemble view (Sein et al., 2011, p. 42). During the execution of the BIE cycle (figure 21) the problem and the artefact are under continuous evaluation.

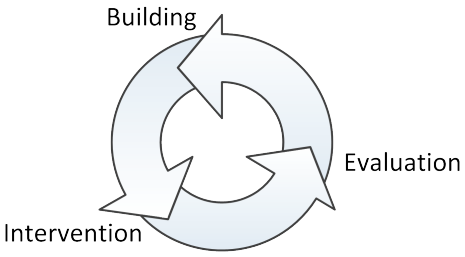


Figure 21: The BIE Cycle

The tasks for this second stage are (Sein et al., 2011, p. 43):

1. Discover initial knowledge-creation target
2. Select or customize BIE form
3. Execute BIE cycle(s)
4. Assess need for additional cycles, repeat

The execution of the BIE cycle, see task 3, is described in a generic schema (figure 22). It shows that initial interactions take place among the scholars only, before the practitioners of the collaboration partners are involved for an alpha version and later end-users get involved for a beta version. These two version synonyms mark a quality state of the artefact, more than a version number. It illustrates the state of development of the IT artefact and also the level of intervention within the company.

Additionally, three principles are involved in this stage: Reciprocal shaping, mutually influential roles, as well as authentic and concurrent evaluation (Sein et al., 2011, p. 43). The *reciprocal shaping* principle emphasizes that the IT artefact and the organizational context influence each other. They are under constant mutual effect.

The *mutually influential roles* principle emphasizes the learning from the other project participants. While the scholars have knowledge in theory and

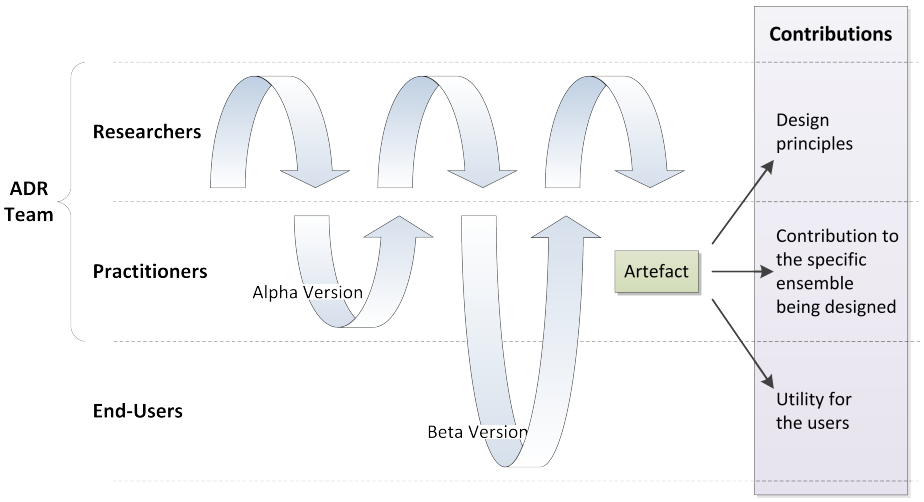


Figure 22: The Generic Schema for IT-Dominant BIE, from (Sein et al., 2011)

technological advances, the practitioners have knowledge in practices. Sharing of knowledge and exchange of ideas is part of the research process.

The *authentic and concurrent evaluation* principle emphasizes that evaluation is not a separated research activity. Instead, it should be interwoven with the design of the IT artefact and the intervention in the company. Evaluation is an on-going activity held in parallel.

Stage 3: Reflection and Learning

The third stage happens in parallel to the first two stages. The problem formulation and the ensemble view of the IT artefact are continuously reflected on (Sein et al., 2011, p. 45). Thus contributions to the body of knowledge can be identified and further, the research process can be adjusted.

The tasks to follow in this third stage are (Sein et al., 2011, p. 44):

1. Reflect on the design and redesign during the project

2. Evaluate adherence to principles

3. Analyse intervention results according to stated goals

This stage draws on the *guided emergence* principle, which emphasizes that the ensemble artefact reflects two aspects (Sein et al., 2011, p. 45). The first is the preliminary design as intended by scholars (see the theory ingrained artefact principle in stage 1). The second aspect is that the artefact is under on-going shaping by the organizational use and the project participants (see the principals reciprocal shaping and mutually influential roles in stage 2), but also as the result of evaluation (see the concurrent evaluation principal in stage 2). These alterations of the ensemble view of the IT artefact are wanted and the ADR team is supposed to embrace them.

Stage 4: Formalization of Learning

The fourth stage is not in direct interaction to the project work, which is described in the first, second and third stage. Instead, it focuses on the formalization of the learning (Sein et al., 2011, p. 45). The project findings should be developed into a general solution concept, which is based on the class of problems as defined in stage one. The goal is to share the findings in a form that makes them applicable for future use. Therefore, addressees are the practitioners and colleagues in the project, but also the research community.

These tasks should be followed in this fourth stage:

1. Abstract the learning into concepts for a class of field problems
2. Share outcomes and assessment with practitioners
3. Articulate outcomes as design principles
4. Articulate learning in light of theories selected
5. Formalize results for dissemination

This stage's work builds upon the principle of *generalized outcomes*. Generalizing findings from a highly situated ADR project can be difficult. However, it is important in order to communicate findings and make them applicable for others. Sein et al. (2011, p. 45) suggest to do this in three levels: First, to generalize the problem instance, then, the solution instance, and finaly, derive design principles from the outcomes.

5.2 Appropriateness of ADR

In the information systems field many researchers employ research methodologies that support the interaction between scholars and practitioners. Especially Europe has a long history and successful tradition in that (Winter, 2008). An overview of the process elements in different disciplines within the field of information systems shows that the evaluation of IT artefacts is always a central element (Peffers et al., 2008). Even though it is very popular to utilize design science research (Hevner et al., 2004; Peffers et al., 2008; Iivari, 2007), I chose to apply action design research (Sein et al., 2011) instead. In this section I explain the reasons for that choice and show that utilizing ADR is appropriate.

Sein et al. (2011) argue that ADR is a legitimate approach within information systems research, as it does not only focus on the technological aspects. They explain that ADR, in comparison to other methodologies, provides more support for the building, intervening and evaluation (Sein et al., 2011, p. 45). The guidance offered by this methodology enables researchers to focus on learning and reflecting. The interaction between the technology and the people, including the client's infrastructure, is focused on and being observed carefully. Knowledge creation based on emergent changes is thus made possible. ADR follows a more holistic approach than other methodologies, when it comes to the design of the IT artefact. ADR understands itself "as a design research methodology that explicitly recognizes the emergence of artefacts at the intersection of IT and organization" (Sein et al., 2011, p. 52).

One of the big advantages compared to other design research methodologies is that the building of the IT artefact and its evaluation is not sequenced and separated in ADR.

To find out whether ADR is suitable for my specific case however, it seems appropriate to examine my research goals and the environment in which my research takes place. In order to do that, I re-visit my research question, which I formulate in section 1.3:

> ***Research Question:*** *How can IT systems support knowledge management in software development?*

I describe in chapter 1 that I want to study the knowledge management challenges with IT support for knowledge management in software development companies. There, I explain my background and my experiences with knowledge management in software development companies. I further portray the landscape of many software development companies. One of the reasons for doing that is to show that I am well aware of the fact, that there is no ultimate solution to knowledge management and IT support for knowledge management in software development. Every company is different and needs a specific approach. The literature study in chapter 2 supports that claim (see section 2.5). The literature also shows that knowledge management approaches, which can easily over emphasize system's features and that is problematic (see section 2.4). Based on this understanding, the research question itself is very general.

Even though my research is situated in a project with very specific and applied targets (see chapter 4), I want to study the problem on a more abstract level as well. My goal is not to only help the case company (section 4.2.1) with their problem in knowledge management (chapter 6). ADR provides guidance in achieving both: Supporting the case company with a specific challenge through addressing a general issue. In stage 1 of the ADR methods the scholars have to analyse the problem and the problem area in order to define a problem class. This generalization assures that the findings of the specific case can be applied in other environments as well.

The research question also stresses the *support* of knowledge management in software development. Mathiassen (2002) depicts support as one of the three research goals (see figure 16 on page 58). The same figure illustrates activities to reach these goals. In the case of support it is design. The author stresses thus the creation of some kind of artefact. This design process is followed to gain knowledge, which then can be used to improve practice (Mathiassen, 2002, p. 327).

The other two research goals are gaining an understanding through interpretation and improving practice through intervention. It can thus be argued, that these three goals together describe the BIE cycle of ADR's second stage. The ensemble view of the IT artefact is constantly evaluated and the design is collaboratively shaped by the involved people. Sein et al. (2011) define ADR as a research methodology that is based on design research and has therefore a native focus on the design. Hence the research question is well addressed through ADR, but this methodology additionally respects the other research goals, namely understanding and improving.

As I describe in chapter 4, the KiWi project was scheduled before it began and before I was enrolled. A research methodology would have to be combinable with the given configuration. The project was organized in three phases with

several different milestones to document the work progression and intermediate results (see figure 10 on page 45). It was also defined that a prototype for a knowledge management system had to be built in close cooperation with people from a case company (see section 4.2).

One of the prerequisites for ADR is the collaboration with practitioners. Scholars and practitioners are partners that work together on achieving a shared goal. Also, they share knowledge among each other about the problem domain and the available theories. Without the combined engagement of scholars and practitioners this methodology would not make any sense.

The other aspect, the pre-defined scheduling of the project, is not equally straight forward to combine with ADR. The project is organized in a waterfall manner, which expects finalized deliverables when reaching specified milestones. Within the first two phases, the requirements specification and the prototype development, ADR can be applied as a small project for each milestone targeting at the specified deliverables. However, the outcome of one milestone feeds into the subsequent one. These deliverables can therefore also be understood as intermediate results in a long term ADR project.

A special case is the final project phase, the prototype evaluation. Here, the project plan intended the intervention of the developed prototype. As the BIE cycles make sure that constant intervention takes place, this phase would be redundant. However, it leaves space to gain deeper knowledge about the ensemble view of the IT artefact and allows further shaping of the design. Additionally, because the original scheduling was separating and sequencing the building from the intervention, ADR thus helps to gain results of higher quality.

Chapters 2 and 3 explain the theory behind knowledge management and related approaches, which show that knowledge management is a complex field. For instance, a plain focus on the technological issues would ignore the perspective of the company and of the people (section 2.4). Both are important aspects of a successful knowledge management approach. A system has to fit the company's goals and respect the users. The KiWi project however has a strong focus on technological aspects. It involved many different and complex technologies. In order to avoid the trap of approaching knowledge management from a plain technical perspective it is important to study more than just the plain prototype. ADR takes all these aspects into account by utilizing the ensemble view of IT artefacts, which emphasizes the impact of the company's context and the participating people.

As a final argument for the appropriateness of applying ADR in this research I would like to stress that ADR has the ability to reach results of both, high quality and relevance. Especially the aspect that ADR emphasises the ensemble view, instead of plainly focusing on the IT artefact, makes a lot of sense. With my background in knowledge management in software development (section 1.1) I understand this as a crucial aspect to succeeding and finding a way to support knowledge management in software development.

Following the arguments provided in this section I conclude that ADR is highly appropriate for my research. The main arguments are that it addresses my research question and fits into the project plan. But limiting the decision to those two reasons would be too easy. The many different aspects add to a summary of good motives that are in favour of action design research.

5.3 Implemented Research Method

The research conducted within the Logica case of the KiWi project followed the ADR method. As explained above (section 4.2), the case's work was organized into four project phases: The requirement analysis, the creation of the knowledge model, the development and the evaluation of the prototype (see figure 11 on page 46). These phases include different milestones, which require the creation of deliverables. The ADR method was applied within each phase. All four stages of the ADR method as explained above (section 5.1.3) were utilized for the shaping of the ensemble view of IT artefact and as guidance for the collaboration.

Note that the project's evaluation phase and the actual evaluation of the prototype are two completely different matters. The IT artefact was under constant or regular evaluation during the development, as mandatory in ADR. However, based on the organization of the project in a waterfall manner, a final phase was scheduled to evaluate the developed system. The final project phase was therefore used to conduct detailed user tests. This was realized through continuing the ADR method through the evaluation phase and emphasizing the evaluation aspect of the BIE cycle. This way the development could solve minor issues and the user tests were able to evaluate the prototype to a broader sense in the context than it was possible during the main development.

In this thesis the different stages and aspects of the ADR method are represented in different chapters. The *problem formulation* (stage 1) is presented in chapter 6, which is my analysis of Logica and its knowledge management problems. Then I present my design and the development of a knowledge management system, in chapter 7. That represents the first aspect of the BIE cycle, the *building* (stage 2). The rest of the BIE cycle, namely *intervention* and *evaluation* (stage 2) is described afterwards, in chapter 8. Here I justify the design according to the situation in Logica and report on user tests. Finally, in chapter 9, I represent the *reflection and learning* (stage 3) of my research. Here I discuss the whole knowledge management approach, its limitations, the implication for practice and my contribution in general. The *formalization of learning* (stage 4) is represented through this thesis and other reports that were created as part of my research.

This section deals with details how ADR was applied and how the outcome was treated. Similar to a case study, the conducted research followed the traditional three areas of research: Design, data collection and data analysis (Dubé and Paré, 2003). Although the research followed a comparable pattern, it is crucial to understand, that this is not a case study. Several conditions for this kind of research are not met. My research is not focused on the case company or on their knowledge management system. Instead, the research is focused on designing a knowledge management system within the case company. It is therefore better framed as a *Design Study*, which matches the design-focus of ADR.

The research of this design study (figure 23) was organized following the guidelines of ADR as described above. During the project work different kinds of data were collected following qualitative approaches. An analysis of the data collection resulted then into reports on one hand side and feedback into the process on the other.

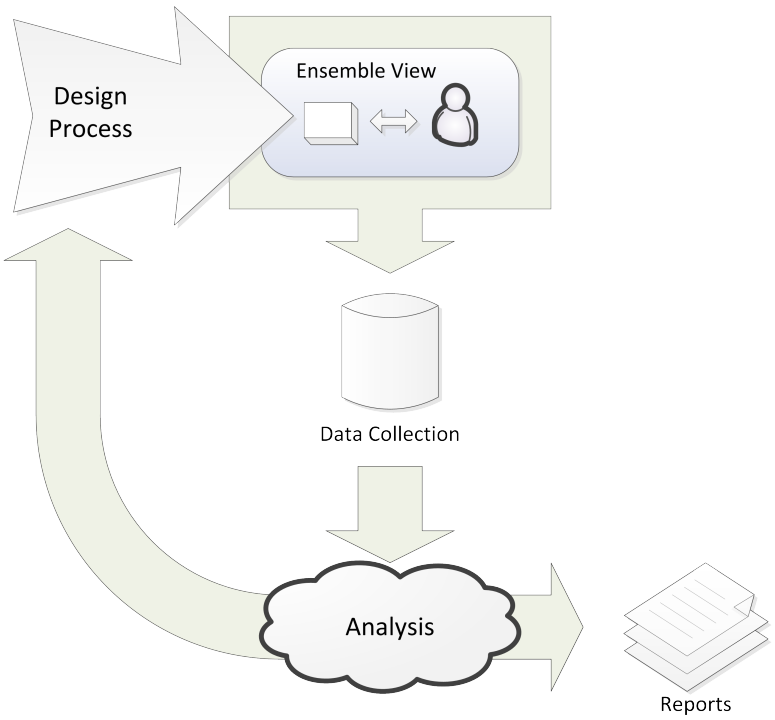


Figure 23: Organization of the Design Study

5.3.1 Data Collection

The data collection in this section does not refer to an activity that follows the project work. In contrary, it is a collection of data, resulting from the project work. Gathering the data is one aspect of the research work by documenting the work process. Based on the different tasks for the scholars, the variety of data in the collection is rather big. Different sources were utilized (figure 24): Data is collected from the IT artefact, i.e., the prototype itself, but also from the ensemble view. Additional sources for the data collection are the design process of the IT artefact within the context of the company and the collaboration with involved people.

Note that even though there are different sources to collect data from, these are not equally used throughout the project. During the requirement analysis the IT artefact for example was of lower significance than it was in the final evaluation phase of the project. However, this was not tracked, nor do I specifically report about it. The different sources feed into an equal data collection which then was utilized as an entire unit for the analysis (figure 24).

The data was collected through interviews, observations, user tests and the provided access to internal documents. All these provided sources with recorded data during different phases in the project collaboration. This section describes these activities and the data that was collected.

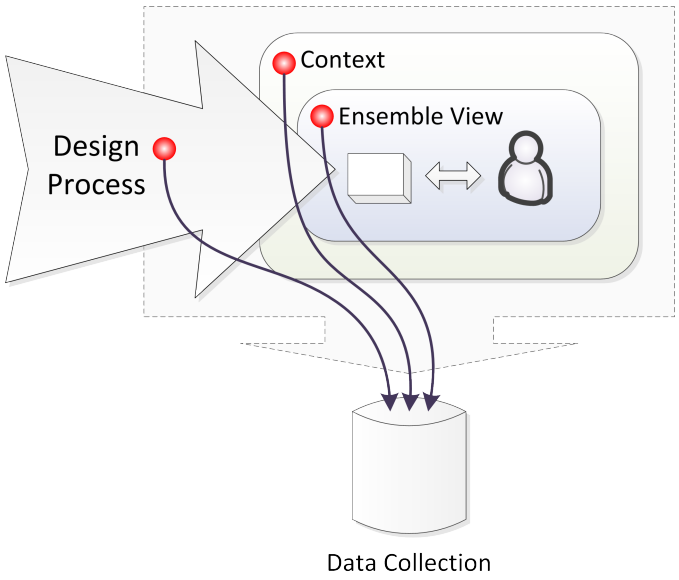


Figure 24: Data Collection and its Sources

Documents

The case company granted access to documents for the scholars, which are used for project management. The intention was to provide an impression and better understanding of the project management tasks and problems during the first project phase, the requirement analysis. These official documents assisted the understanding of the scholars, especially in combination with the other data collection methods (Bryman, 2008; Creswell, 2009).

The documents' nature was exemplary to see the format and understand the structure of communication. For that purpose documents dealing with specification and documentation were made accessible.

Interviews

Different types of interviews were applied throughout the project. One took place in the beginning of the first project phase, during the requirement specification phase. Here, the needs of Logica were explored by visiting development projects and talking to the participants. The other type of interview was part of the evaluation phase and helped to investigate the user test experience. However, every interview conducted as part of my studies followed the seven stages of an interview investigation (Kvale, 1996, p. 88): Thematizing, designing, interviewing, transcribing, analysing, verifying and reporting.

The project visiting interviews had the target to find out how the project organization works and what problems there could be. Two projects were met face-to-face, with two representatives of each project at the same time. A qualitative approach was followed here, as the focus was on investigating a problem (Creswell, 2009, p. 179). This way the employees had the opportunity to explain the current situation and express their needs directly (Bryman, 2008, p. 437).

Both interviews took place in a very early stage of the project in order to explore Logica's knowledge management problems. Even though the two projects address the same domain (municipality administration software) they were very different. Differences were visible in basically every core aspect: The amount of people, the involved technology, the applied infrastructure. These differences were also the reason for choosing them. They provide insights in the company's alterations of the variety of Logica's projects. The focus of the interviews was mainly about the settings in the projects, no technical development details are documented here. These interviews built the foundation for the scholars' understanding of the case company.

The data collected from the meetings was mainly in form of minutes and notes. Internal reports were created afterwards to summarize the interviews and document the statements of the practitioners and the impressions they left.

The user test interviews were part of the evaluation phase (details in chapter 8). After every test run the test user was interviewed in order to explore experiences and opinions about the system under test. These test runs were conducted in three iterations with two test users each. Every test user was interviewed one-on-one immediately after the test run (Creswell, 2009, p. 179). It was therefore directly connected to the objectives and actions of the user test.

The interview itself was semi-structured (Bryman, 2008, p. 438): An interview guide was created that helped to investigate the user's experience and opinion on the test objective. The questions were not fixed or settled so that the interviewer was able to change the order or formulation depending on the interview. In the end however all questions were answered and the aimed insight was achieved.

Each test user was an employee of Logica. They were therefore aware of possible problems and common practices in the company. During the interviews the users were able to express what they liked and what they disliked. The feedback of the users provided valuable insights to the scholars and helped to improve the prototype. All user test interviews were video and audio recorded. Additionally, notes were taken and a report was created.

Observations

During the first project phase, the requirement analysis, different project manager meetings at Logica were observed. These were normal meetings at the case company's offices, which the practitioners hold regularly and scholars were allowed to follow. Hence, they were observations in realistic and natural situations (Järvinen, 2000, p. 147) without active participation by the researcher (Creswell, 2009, p. 181).

Similarly to the project visiting interviews, the observations took place in a very early stage of the project, with the target to explore the knowledge management problem of Logica. A defined focus was not the behaviour of the participants, but the topics and the way agreements were found. During the meetings business necessities were discussed between project managers and the project office in order to evaluate a project's status.

Project management is a very complex task and these meetings helped the scholars to understand how Logica deals with it. The gained insights supported the process of studying the problem in knowledge management for Logica. Three

meetings of this kind were observed. Each of them was audio recorded and transcribed afterwards.

Participant Observations

Throughout the project many different meetings were held with varying participants and targets, like discussing the objectives of the project and evaluating the findings. My involvement during these meetings was in the role of a complete participant, which means that I was not restricted to observations of the other people's interactions only, but was contributing myself (Bryman, 2008, p. 411).

The meetings, that included with me as a complete participant and that are relevant for my studies, can be grouped into three categories: KiWi meetings, Logica case meetings and internal meetings. All of these meetings were documented in different forms, in order to utilize insights in later stages. Field notes were always taken. Each meeting involved participants of different circles, including practitioners and researchers with different backgrounds and experiences. The meetings were then used to discuss the progress and ideas. Hence, every meeting was a pool of input (e.g., opinions, suggestions, critique) from various different sources.

The *KiWi meetings* were organized three times a year at different locations across Europe and involved all project participants. A meeting with larger subsets of the project participants, with a focus beyond the business case relevant for my studies, join this classification. These meetings addressed general issues of the project and were used to communicate intermediate results to the partners. They were thus a simple opportunity to gain feedback from other researchers that are familiar with the project goals.

These meetings were organized in presentations followed by discussions. One person gave an introductory presentation about a topic (e.g., state of development, ideas for improvement or management activities), in order to gain shared knowledge. The topic was then discussed among the participants, which involved the feedback from different areas. The slides of these meetings were shared among the participants. Additionally, minutes document the presentations, the discussions and the taken agreements.

The *Logica case meetings* took place in Aalborg, either at the university or the offices of Logica. Multiple times a year, the issues related to the Logica case, and therefore my studies, were addressed. The participants of these meetings included people from Aalborg University and the case company. Additionally, sometimes also people from other partners were involved, who contributed on specific occasions, even though they were not directly related to the business case.

The relevant project deliveries were prepared within Logica case meetings, as well as the creation of the prototype and its specifications. These meetings were documented in different ways. Meeting minutes were taken every time. Some meetings were additionally voice recorded and if discussions led to drawings on whiteboards, these were photographed afterwards.

The *internal meetings* usually took place at Aalborg University on a regular level, focusing on aspects of the Logica case and the whole project. One aspect was the strategic part of the cooperation with the case company. The other one was the scientific discussion of the planned activities and narrowing down

a focus on what is scientifically more interesting. It was therefore referred to as design meetings, as they dealt with the design of the prototype. All of these meetings are documented in minutes; additional drawings on white boards were photographed.

These three kinds of meetings (KiWi, Logica case and internal) correspond to two of the three cycles as defined by Hevner (2007), which are also referred to within ADR. The case meetings represent the relevance cycle and the internal meetings the rigor cycle. During the Logica case meetings the participants discussed the use of the current ideas and whether it addresses the problems correctly. Prototypes were created and adjusted according to the opinion of the employees of the case company, as they are the experts in the problem domain. In the internal meetings the academic rigor was discussed, approaches were grounded in the literature and articles have been prepared.

User Tests

Evaluation of the ensemble view of the IT artefact is an on-going task and one of the three elements in ADR's BIE cycles. This evaluation took place in different meetings as explained above. Intermediate versions were widely discussed in Logica case meetings, but also in KiWi meetings with project participants involved in the other business case or development of different sub components. A working prototype was then presented and explained in a deliverable that closed the implementation phase of the project (Dolog et al., 2009c). This subsection however describes the data collection during the final project phase regarding the evaluation of the prototype. It was used for intensive shaping and finalizing of the prototype within the organizational context. This evaluation phase had a focus on user tests and on improving the prototype (details in chapter 8).

Just like the rest of the data collection, the user tests were conducted in qualitative studies, because there the "emphasis is placed on the uniqueness of human experiences" (McDavid and Hawthorn, 2006, p. 175). A close engagement between scholars and practitioners (van de Ven, 2007) helps evaluating IT artefacts in their totality, not only according to its functionality. Therefore the user tests were held in three iterations with two test users each, who are employed at the case company.

The risks and uncertainty that a project can face during the evaluation in design research do not differ to those in industry (Carney and Wallnau, 1998). Either strives for the same main goal: Achieving results of high quality. In order to avoid negative impact due to the risks and uncertainties, risk management becomes an important task (Baskerville et al., 2008). Addressing the risks helps to avoid problems and reach the goals. It is thus of high importance that not only the evaluation as such can be finished successfully, but also that the outcome is of a certain quality.

In order to minimize the risks it was decided that the user tests take place in a usability laboratory and not within the case company's environment. The intention was not to examine the usability of the prototype; the focus was on the usefulness only. However, this provided the opportunity to record the test runs in video and audio. This implies that the test system did not operate on real data, but in a synthetic, though real-world-like, test environment, based on realistic data. Therefore, the tests could be set up with a better focus on

the test objectives, because the manipulation of data accordingly was easy and without consequences (Mitchell, 2007, p. 42).

Every user test had the same agenda: A test user obeys a test script, by fulfilling tasks in the prototype, followed by an interview regarding the objectives of the test session. The user tests were moderated (Dumas and Loring, 2008), the moderator guided the users through the tasks and offered support. Hence, questions with the test script or other issues with the system could be solved immediately (Albert et al., 2010, p. 62).

The user tests were executed in three iterations, so that the outcome of one iteration can influence the prototype for the next iteration. The final prototype, and thus the one including the changes from the results of the user tests, is described in this thesis (section 7.2). It contains slight differences to the project deliverable of the development phase (Dolog et al., 2009c).

5.3.2 Data Analysis

The data analysis is the heart of a design study and has the goal to achieve an understanding (Järvinen, 2000, p. 75). During this research the data analysis was conducted in an on-going process, as part of the BIE cycles of ADR. The constant evaluation resulted in a continuously increasing data collection, whose analysis was an important aspect of the shaping process. This was possible by processing the gathered data and feeding back into the design process from the gained insights, which is reflected in the methodology's stage 3 (see section 5.1.3).

Besides feeding back into the design directly there is also the documentation and communication of (intermediate) results. As explained above (section 4.1.1), during this project each project phase had to be concluded with an output in form of reports. In this section I describe the process, how these reports were created as a result of the analysis, for each project phase separately.

Even though this is a design study, the activities follow the standards for a traditional software development building process (Järvinen, 2000, p. 101). It begins with a requirement analysis, followed by the design of both a knowledge model and the prototype, and is then concluded with an evaluation of the prototype. The reason for this is very simple: Despite the different goal between the design and the development of a system, the steps to be taken are similar. It is therefore important to stress that this data analysis focussed on the research aspects of the design study.

The previous section explained the data collection and its sources (section 5.3.1). In a design study like this, the analysis of this data is a crucial aspect of the project work. In the following I describe the analysis in detail, organized according to the project phases. The data analysis is strongly influenced by its qualitative research background and the focus of a design study. Therefore, I describe the steps being taken in prose. This makes it easier to understand the actions that were taken, as well as the reasons for these and their results.

Note that each of the project phases build on top of each other. As shown above (see section 4.2.1), the outcome of one phase is always the input for the following one (see figure 11 on page 46).

Requirement Analysis

The goal of the requirement analysis was to achieve an understanding of Logica's knowledge management problems. To investigate the internal strategies and IT support for knowledge management, project representatives as well as project managers were interviewed and observed. The findings were then, in Logica case meetings, discussed. It was compared whether the gained insights from the interviews represent a realistic view on the company's status. The employees during these meetings represented a different layer of the case company's hierarchy and could provide more input regarding problems and the need for knowledge management. They further provided documents that illustrate the communication within Logica.

In internal meetings, the results of the different interviews and the input from the case meetings were then combined and discussed, in order to form a broader picture. A combination of the views from inside and outside the projects helped to gain the necessary overview. This then was analysed and resulted in a first draft for a document, which sums up the findings by explaining the process management, knowledge flows, knowledge management challenges as well as a scope for the planned knowledge management system.

Through multiple iterations with the KiWi project members of the case companies the draft was then elaborated on and extended with an outline of what the targeted knowledge management system is supposed to do. The system is sketched on the level of use cases and class diagrams. The latter provides an overview of the most significant concepts and their relation to each other.

Once a draft was finished, on which all involved authors could agree, it was forwarded to the reviewers. Two participants of the KiWi project that are not directly involved into the cooperation with this case company provided comments regarding the rigor and quality. These were then considered, objectives were corrected and the final document could be delivered to the KiWi project management.

Type	Content	Form	Reference
Report	Requirements	KiWi Project Deliverable D5.4	(Nielsen et al., 2008)

Table 4: Output of Requirements Specification Phase

I was one out of the two researchers that conducted the interviews with the project representatives and project managers in the beginning of the requirement analysis phase. I participated in all Logica case meetings, which then helped to raise a shared understanding of the involved people. These meetings consisted of shorter presentations of ideas or understandings and discussions among all participants. Based on this gained understanding, I then created a first draft for the deliverable. Large parts of the original content were significantly changed; however, it provided a ground for discussion and collaborative improvement.

Knowledge Model

After submitting the specification of the requirements, the targeted system was further discussed in case meetings. Here, different possibilities were elaborated.

Based on the understanding gained through the requirement analysis a system was envisioned that would support the process work, in a three column layout. After debating it in several meetings, this concept was detailed in a document with use case descriptions. Further development took place, before this concept was discussed in a KiWi meeting, with multiple project members outside this business case.

The feedback from the various discussions was worked into the concept. A new document was created, which contained an adaption of the proposed system to the project's needs and possibilities. This document also extended the class structure of the previous deliverable to a more general knowledge model. It was more detailed and translated into an ontology to be supported by the semantic wiki (see section 4.3).

A description of this knowledge model and its structure was added to the document. The first draft of the deliverable then followed the exact same process as the one during the previous project phase. First, all contributing authors elaborated on it iteratively until they agreed to have reached a final state. Then, the draft was handed over to the reviewers, i.e., project members that are not involved in the documented work. Resulting comments about rigor and quality were appreciated and worked into the draft, which was then finally officially delivered to the project management.

Type	Content	Form	Reference
Report	Knowledge Model	KiWi Project Deliverable D6.3	(Dolog et al., 2009b)
Specification	Ontology	Knowledge Model in OWL	

Table 5: Output of Knowledge Model Phase

The prototypes that were developed during that time already made use of a data structure, which was created by a Logica employee, based on the requirement analysis document. A fellow PhD student, who was also enrolled in the KiWi project, and I translated this data structure from relational tables into an ontology. The deliverable then, a result of shared responsibilities and collaborative writing and editing, compiles the concepts used for the prototypes and the insights gained and discussed through them.

Prototype

The prototype's design began figuratively with the first project phase. Both the requirement specification and the knowledge model add to the design of the prototype. As the development took place in a bigger project with deliverables to the funder, the previous documented insights should be applied. The design therefore had not only to be informed by the previous project phases, but also match the requirements specification and utilize the knowledge model. Also, the design followed a building process that applies the BIE cycles (Järvinen, 2000; Sein et al., 2011).

During this project phase possible designs of the knowledge management systems were further and continuously discussed in various circles and all different meetings. In several iterations an approach evolved from the achieved

findings and dialogues, which addresses the issues of the case company and fits to their environment. For these issues the collected data from the requirement analysis was often consulted and discussed with people from the case company.

A technological proof of concept was then developed in order to show that the envisioned solution is realizable. In short iterations one of these prototypes were discussed in Logica case and internal meetings. The shaping of the design was therefore based on a technologic proof of concept. When the design reached a level of certainty the development of the complete prototype began.

The prototype of the approached knowledge management system was developed through support by programmers of different units in the KiWi project. That happened according to the design as defined within the Logica case. Here, demands on the included technology were defined that advanced the general features of the KiWi platform. These demands were discussed and designed among the developers in regular exchange with the case team. A first working prototype was then presented on all different kinds of meetings with various circles of participants. The resulted feedback was documented and worked into preliminary prototypes.

Finally, the thus collected data was reported in the first draft of the next project deliverable. A user guide was created and the developers documented their implementation themselves. Additionally, the contributing authors wrote about the background, the reasons that led to the design decisions and the final system architecture. They referred their work back to the requirement specification document (Nielsen et al., 2008) and the knowledge model description (Dolog et al., 2009b). Within multiple iterations this document was edited by all contributing authors until they all agreed on having reached a preliminary status. This document was then sent to the internal reviewers. Their feedback was worked into the draft in order to deliver the final version to the project management.

Type	**Content**	**Form**	**Reference**
Report	Implementation of Prototype	KiWi Project Deliverable D6.4	(Dolog et al., 2009c)
Software	Prototype	Knowledge Management System	

Table 6: Output of Prototype Development Phase

The original idea of the three column layout as well as the related use case descriptions were provided by a Logica employee. Long before this phase in the project began, I used them to create concepts for systems and prototypes that apply them. First, the entire development was done by me alone. Later, Logica applied three student developers, who I managed.

The different prototypes, that were initially conceptual, evolved to a prototype of the KiWi systems. In many meetings and in front of varying KiWi project participants, I presented the status of the different prototypes, collected feedback and enhanced the concepts for the further development. My role was therefore much of a coordinator of the developing activities relating the KiWi systems.

The KiWi systems are described below in detail (section 7.3), consisting

of the KiWi platform, the Project Management Application (PMA) and the Data Exchange Agent (DxA). The KiWi platform was developed by the project partners who were responsible for the core and the enabling technologies. The PMA was developed by Logica according to its own needs. And the DxA was developed in close cooperation between a student developer of Logica and me. My role here was again the coordinator; I was responsible for the integration of the different systems.

Large parts of the final deliverable were then written by me, based on additional material. I describe the knowledge management problems of Logica briefly and then explain the idea of the circle of knowledge, which was provided by an employee of Logica. For the remainder of the document, I wrote most of the documentation of the KiWi systems and a user guide. I contacted the developers of the particular features to gather information and compile them to the larger picture. Every involved developer was then able to edit the document before its submission.

Evaluation

The final project phase had the target to evaluate the created prototype. It was used to compare the developed knowledge management systems to the requirement specification and to examine whether it could be beneficial for Logica. As evaluation is an on-going process in ADR, minor changes to the prototype were performed after the delivery of it due to the project's milestone, finishing the previous project phase. These changes however were based on the results of presentations and discussed in Logica case meetings. The major target of the final project phase was the evaluation of the design. The usefulness of the prototype was to be investigated.

The tasks during the evaluation phase were planned carefully, which became a difficult task, because of two major issues. First, the timetables of the necessarily participating people and the period of the actual evaluation showed difficulties regarding the scheduling. And second, the software was not very stable after two years of development, which, at that time, was not fully finished by all project partners. Based on these considerations the evaluation was planned to utilize an iterative approach. Similar to agile software development (Larman, 2003) the idea was to evaluate aspects of the system in different iterations. This provided the possibility to plan the evaluation to certain detail, but keep a particular flexibility at the same time. This has certain advantages, when it comes to optimization and preparation of the system for specific parts of the evaluation and regarding the involvement the available people.

The iterative evaluation was planned by describing the roles of the involved people as well as a timeframe in which the iterations take place. Also, the test cases were defined in form of use cases, borrowed from the literature on requirements engineering (Cockburn, 2000).

Use cases describe a common scenario and the process in which the system should be used. These were carefully created by participants of the business case and further discussed in Logica case meetings. The process qualifies the created report as an artefact and not just as a simple plan for the evaluation. The planning actually included the documentation of detailed process thoughts, i.e. how to use the knowledge management system.

It was planned that during the evaluation these processes then should be followed by employees of the case company. An interview follows the system test, to investigate the opinion of the test person about the system. The test itself was planned to take place in the usability laboratory (Rubin and Chisnell, 2008). Thus, it can be recorded in video and audio.

All these aspects were thoroughly discussed and planned for the evaluation. The results were then described in a document, which after some iterations of editing by the authors was send to other project participants for review. The resulting comments were then worked into the draft and it was delivered to the project management as the official evaluation plan.

After finalizing the plan, the actual evaluation period began. In a case meeting the first objective was chosen, so that the system could be optimized accordingly. Once the required stability was reached, the test runs with the test persons took place. The findings of the user tests were discussed in a Logica case meeting, which resulted in change requests for the software. And finally a report was created describing the evaluation iteration's outcome. This procedure was followed for each of the three iterations.

After finishing the final iteration a document that describes the complete evaluation was created. It included and was based on the iteration reports. The contributing authors edited the document in different iterations, before it was sent to not involved project participants for reviewing. The draft was then modified to include the comments and delivered to the project management as the final delivery, documenting the evaluation of the knowledge management system.

Type	**Content**	**Form**	**Reference**
Report	Evaluation Plan	KiWi Project Deliverable D7.2	(Grolin et al., 2010a)
Report	Evaluation Results	KiWi Project Deliverable D7.4	(Grolin et al., 2010b)
Software	Improved Prototype	Knowledge Management System	

Table 7: Output of Evaluation Phase

One of the researchers of the Logica case had the idea to create a complete feature list of the KiWi systems. I created a draft of this list, documenting all features of the different systems that I have knowledge about. To enable the other project participants to view and extend this list, I created a page in the project-internal wiki with it. I contacted everyone and asked for their participation. Further, I used the feature list, to enhance the use case descriptions, so that every work step additionally lists the involved features. Both, feature list and use case descriptions can be found in the appendix of this thesis.

Based on the problematic situation in the project, a researcher of the Logica case came up with the idea to conduct the evaluation iteratively. I picked up that idea and outlined a systematic evaluation approach, following the patterns of agile software development. I based the evaluation iterations on the use case descriptions created earlier and defined a process as well as a rough time

schedule to be followed. Based on these I created a draft of the deliverable, which was slightly edited by the rest of the involved people before submission.

The evaluation itself was entirely conducted by me. I planned, executed, documented and reported on the user test sessions. Further, I had the idea of an iteration that does not follow the plan, to improve the results by widening the focus of the evaluation. After all test runs, I compiled the reports and created a draft for the deliverable, which again was slightly edited by the rest of the involved people before submission.

The development of the software throughout this phase followed the same lines as described above.

6 Problem Analysis

This chapter describes my analysis of the knowledge management problems in Logica. As such, it represents the first stage, the problem formulation, of the ADR methodology applied in my research (see section 5.1.3).

I first describe the background of Logica (section 6.1), before I provide an overview of the identified problems (section 6.2). Then follows a detailed explanation of the two general problems: Isolated islands of knowledge (section 6.3) and inadequate bridging of knowledge (section 6.4). Finally, I summarize the problems of Logica (section 6.5).

6.1 Background

The focus of my problem analysis is the development department of Logica in Denmark. In fact, my cooperation with Logica focussed on the continental part of Denmark, Jutland. All people involved in my studies worked in Logica offices in Aalborg and/or Århus. I roughly outlined Logica and its organization above (section 4.2.1). In this section, I provide more details to the background of the case company and describe the environment, in which the problems occur. The identified problems are described afterwards. Here, I explain Logica's operational business.

For the sake of illustrating the context and in order to explain the operational business of Logica in Denmark, I extend the focus to Logica worldwide briefly. Note that this is only done to reveal the implications to the local business (i.e., Denmark). The focus of my analysis remains unaffected.

Logica operates globally, providing consultancy for IT and management from offices in many different countries. In this worldwide setting every local business focuses on its own markets. The different situations in these countries lead to a diversity of Logica branches. The different areas of concern (i.e., the markets of the local businesses) decrease the chances of cooperation between the local businesses of different countries. However, some of these local businesses focus on international collaboration. Logica in India, for example, is often included with offshoring projects. On a global scale, Logica thus tries to make use of its international man power in order to gain a competitve advantage.

Logica is a rather new company in Denmark. Just before the beginning of the KiWi project in early 2008, Logica aquired *WM Data*, a Swedish company from the 1970s with many offices throughout entire Scandinavia. Since the acquisition, Logica runs offices in five different cities with about 800 employees in Denmark alone. This includes technical specialists from a variety of fields as well as experienced managers.

After the acquisition all business activities were continued. Logica gradually introduced organisational changes only, in order not to interrupt the day-to-day business. Employees, for example, kept working on the same projects, in the same offices, with the same colleagues. So even though Logica is new in Denmark, the company itself figuratively just changed its name. The company therefore already has a developed self-perception: Providing services and solutions of high quality. This confidence is based on the experience of many years of successful projects. The employees are carefully selected and well educated.

Logica's customers are from many different domains, including the health sector and telecommunications. Clients are for example banks and insurances, but also Danish municipalities. Some customers have collaborated with Logica (and its predecessor WM Data) for many years and the collaboration is ongoing. With every project, Logica investigates the possibilities of follow-up assignments. The goal is to obtain long-term relationships between Logica and the customers; conducting one project after the other. Sustaining collaboration is easier and more cost effective than establishing new ones.

In addition to the divergent backgrounds of customers, the profile of work varies. Sometimes the whole development is done by Logica employees alone and takes place in Logica offices. But sometimes projects have to be developed in the customer's offices. Also, the staffing is not always Logica exclusive. Sometimes cooperation is unavoidable. Projects then involve employees of the customer or even from other contractors by the customer.

The services provided by Logica are organized in projects. Customer requests or assignments are always dealt with in projects. Logica's main business is therefore project work. Projects are considered extremely important and are of high priority within the company. For the lifetime of a project, Logica employees are figuratively working for the customer. Once a project ends, the employees are free to be assigned to other projects.

Every project at Logica reaches a level of independence, its purpose is of high importance. This is the result of the project-orientation in Logica: A project entirely focuses on solving a customer's problem and everything else within Logica is focused on successful projects. The organization of a project holds certain liberties. To some extend projects in Logica therefore become autonomous.

Additionally, the projects themselves differentiate from each other as every

project is specifically tailored to the customer's needs. The time frame and staffing of a project does not follow fixed rules, but varies. While some projects only last a few months, others are running for many years, often including follow-up projects.

Because the project's budged is occasionally very high, Danish and European law requires a public call for bids. And even if it is not a legal necessity, customers usually compare offers. Logica is therefore always in competition with other companies. Any customer-related knowledge can thus be a competitive advantage.

The *project manager* is in control of and responsible for a project. Due to the importance of projects within Logica, a project manager is very powerful. Project managers are responsible for the entire internal organization of a project. They plan and coordinate the work of the project team. Project managers are responsible for reaching the defined goals of their projects. To be prepared for such a responsibility, they are highly educated. Most of them have a masters degree. Further, Logica provides internal trainings and many project managers additionally take external certifications.

The education is important, because a project manager has to take autonomous decisions within the boundaries of a project. The communication between customer and project team, for instance, is strongly influenced by the project managers. The same counts for many different aspects of projects in Logica. Every project manager arranges projects according to its goals, the applied technologies and the involved people.

The company's rules regarding the organization of a project are very strict. Project managers have to report to the management level of Logica. These reports contain the schedules and data about resources, i.e., the financial aspects of a project. The detailled contents and timeframes of such reports are defined in process descriptions. In fact, the whole organization of projects is regulated by process descriptions. A process description is basically a list of tasks to follow, similar to a to-do list. In Logica these process descriptions explicate the company's strategy. They are supposed to make sure that project managers work in the company's interest and with an organized risk management.

In Logica process descriptions are defined by a committee of different people. Such a committee includes (among others) process auditors, project managers and process designers. A creation of a process description is a complex act that involves many people with different backgrounds and intentions. The target of a process description has to be specific enough in order to be used easily. At the same time a process description has to be as general as possible, in order to fit for many different projects. The finalized process descriptions are then stored in Logica's intranet, where every employee can access and utilize them.

To summarize, Logica operates with a strong focus on projects. Project managers are very important within Logica, as they are responsible for the organization and the success of a project. Process designers define the process descriptions, which support and control project managers in their work.

6.2 Overview of Identified Problems

The analysis of the empirical data in the KiWi project (section 5.3) shows that Logica faces several knowledge management problems. Two areas of concern in

Logica negatively affect the internal knowledge management (figure 25). On the one hand, there are issues resulting from the strong focus on project work. The concentration on projects is so high, that they actually isolate the projects from each other. I therefore use the term *isolated islands of knowledge* to describe this area of concern. On the other hand, there are general issues relating Logica's management approach. Problems occur, because the knowledge sharing through documentation does not work as intended. I use the term *inadequate bridging of knowledge* for this area of concern.

The knowledge management problems are hierarchical organized. Both include several problems, I subdivided and numbered them according to areas of concern. These are *information access* (A1), *expert finding* (A2), *sharing support* (A3) and *documentation level* (A4) for the isolated islands of knowledge, and *process complexity* (B1), *feedback circle* (B2) and *connected documentation* (B3) for the inadequate bridging of knowledge (figure 25).

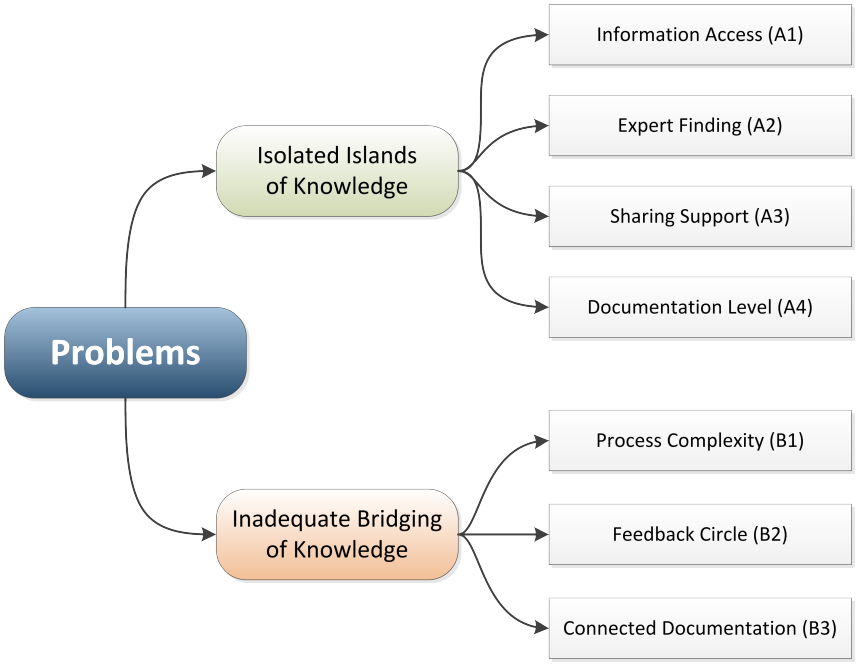


Figure 25: Knowledge Management Problems in Logica

In the following sections I describe each of the found problems separately. I characterize their occurance and effect on Logica. For each problem I created a *diagnostic map*, based on those outlined by Lanzara and Mathiassen (1985). The original approach is designed to help understanding and describing a project situation. I use the diagnostic map to pinpoint and specify problems. The map itself provides an overview, I then describe the background in more details afterwards.

Each diagnostic map comes with four columns. The first column, "Problem", provides the statement of the knowledge management problem in Logica. The second column, "Reasons", provides the background to the problem. Here I explain what the problem emerged from and give reasons for that. The third

column, "Consequences", provides the consequences of the problem. I describe the circumstances that the problem leads to. And finally the fourth column, "Approaches", provides how the problem is already approached within Logica.

6.3 Isolated Islands of Knowledge

As explained in the background description above, Logica is a company with a strong focus on projects. The analysis of the knowledge management problems showed that the projects seclude from each other. This focus on project work results in an isolation of projects from one another. A project can therefore be considered as an island of knowledge. I found four major problems in this area of concern.

6.3.1 Information Access (A1)

Problem	Reasons	Consequences	Approaches
Projects internally encapsulate knowledge, which is created through project work	- Internal collaboration organized autonomously - Lack of connection to other projects	- Knowledge reuse difficult - Decreasing productivity	- Applying tools (File server & Sharepoint)

Table 8: Diagnostic Map for Problem "Information Access"

The analysis of Logica regarding problems with the knowledge management led to a finding concerning the access of information (table 8). During project work in general, employees create knowledge. For example, they gain experiences from utilizing technology or the results of certain work steps. In Logica, this knowledge remains within the boundaries of the project, instead of sharing it with the rest of the company. The information access is the problem: Projects internally encapsulate knowledge, which is created through project work.

> *"We have projects that are almost like departments. They work as if they'd have nothing to do with the rest of the company and knowledge doesn't really get in or out." (Project manager from Logica during a project meeting)*

To explain the reasons for this problem, one has to look at Logica's main focus: Projects. The goals of a project are of so high importance that project managers are figuratively given free rein. My analysis shows that project managers commonly organize projects internally according to the personal experiences and opinions of the project participants. The collaboration strategies and supporting tools are therefore usually chosen according to personal preferences.

To illustrate the reasons for this problem, I provide insights from two different Logica projects, which I will call here *Project Alpha* and *Project Beta*. I analysed both in detail, focusing on the way they organize their project work. Both develop software solutions for Danish municipalities and both use similar technologies for the development.

Each of the projects has their own set up of a versatile issue-tracker called Jira[15]. Project Alpha uses it mainly for bug tracking, test documentation and requirement specification. However, it is not consistently used. A considerable amount of bugs is reported through direct communication instead (i.e., by e-mail or personal talk). Hence, the documentation is not complete and certainly not traceable. Project Beta uses Jira mainly for release management. The project team keeps track of software changes and briefly describes fixes and features in Jira. In combination with the comments in the code, this is the only source of documentation of the software in project Beta.

Both projects work in the same domain with the same technologies and use the same tools for documentation. And yet, the data sets are very different in the two projects. The way Jira is utilized virtually resembles a different tool. The data of project Beta would not make much sense for participants of project Alpha and vice versa. Both projects use the same tool for different tasks.

The overall project planning, regarding resources etc., follows strict guidelines in Logica. For instance, every project manager has to report the details of the project plan. In contrary to that, the internal organization, e.g., the realization of the planning and its communication within the project, does not follow any rules. In project Alpha the project plan is handled through MS Project[16], while in project Beta this is done in an MS Excel[17] spread sheet. This is a tool-choice based on personal preference of the project manager. The difference to the previous case is that here both projects realize similar tasks through different tools.

Direct personal communication is of high importance in both projects. Agreements are partially reached in meetings or direct conversations among people of the project staff. These are not always documented. Additionally, in both, project Alpha and Beta, e-mail is an important factor. In project Alpha e-mail agreements are often not documented elsewhere:

> *"Most important information is somewhere in an e-mail in my Outlook." (Project manager of project Alpha during project interview)*

These examples show that, based on the lack of rules, each project in Logica organizes the internal collaboration autonomously. The documentation of project work differs and in many cases it is even being let slide. Many projects have a low level of documentation. This does usually not influence the project work itself; the strong project internal communication can balance that. But the project participants hardly externalize their knowledge. It is therefore difficult to access for other employees of Logica, there is a lack of connection between projects.

The consequence of the encapsulation of knowledge is that the access to information becomes too difficult. In Logica, employees from other projects can hardly access the knowledge of a project. This also makes the knowledge re-use very difficult. My analysis shows that many projects have to build up this knowledge independently. This is costly, though, and takes time, as it decreases the productivity. By leaving knowledge unused, Logica misses out on a competitive advantage.

[15] http://www.atlassian.com/software/jira/
[16] http://www.microsoft.com/project/
[17] http://office.microsoft.com/excel/

Logica is aware of this grievance. In order to loosen the encapsulation of knowledge two IT systems are available to all employees: A file server and MS SharePoint[18]. The latter is not frequently used by either of the projects from the examples above. Project Alpha uses it to manage links to documents on the file server. Project Beta uses it for internal notes only. However, both are not consistent with either.

The file server is a network drive, which holds a folder and a fixed amount of space for every project. Strict management of the access rights increases the difficulties with the file server. A member of project Beta explained, that some of the developers were not granted the access rights until several months into the project work. The file server was abandoned, because parts of the project participants were not able to reach it. Project Alpha created a folder structure on the file server and stores different documents there. These documents usually relate to the communication with the customer, in form of contracts, reports and the like.

Hence, the approaches to alleviate the information access fail for the same reasons that caused the encapsulation of knowledge in the first place. First, the utilization of the systems can be organized autonomously. Logica does not determine any rules. Second, the level of documentation in these tools is very low. From the perspective of sharing knowledge they are of little value.

6.3.2 Expert Finding (A2)

Problem	Reasons	Consequences	Approaches
Finding experts within the company is difficult	- No accessible repository of expertise - Expertise of colleagues is often unknown	- Expertise has to be built autonomously - External experts have to be hired	- Project supervisors - Project managers interact

Table 9: Diagnostic Map for Problem "Expert Finding"

The analysis of Logica regarding problems with the knowledge management led to a finding concerning the pinpointing of experts (table 9). Experts are generally people with knowledge in a specific field. Logica needs experts to solve specific problems in project work. Projects often have to be realized with cutting edge technology, but also outdated technologies are required occasionally. The project then needs employees with expertise in this certain technology. Localizing the expertise among the employees is therefore a necessity, especially for the staffing of a project. The problem is to find experts within the company.

> *"It is too difficult to find all the relevant information when solving a specific task, sometimes people don't even know that relevant knowledge exists." (Project manager from Logica in a requirement analysis document)*

The problem of finding experts among the employees has a simple reason: Lack of overview. Logica has no system to provide awareness about its em-

[18] http://sharepoint.microsoft.com/

ployees' knowledge. The expertise of employees is therefore discovered by their colleagues in direct interaction only. Expertise of colleagues is often unknown, unless the employees worked together directly before. The project organization in Logica, however, restricts these circles of interaction. Logica's employees usually do not know the knowledge of others they have not directly worked with. This makes finding experts difficult. Often experts cannot be found, even though they exist in the company.

In order to explain the consequences, I describe the problem in expert finding in more detail. My analysis shows that experts are needed in different scenarios, of which I illustrate two. First scenario, projects have the need for an expert as a permanent member. The expert is then the knowledgeable person in the project. A common approach is to share this expertise through training on the job with other project participants. This way more employees gain expertise by learning from the experts.

The second scenario for the need of experts is that the expert is only needed for a short period. An expert often acts like an internal consultant. The tasks in a project then are very different from each other. One task is to solve a specific problem within the project. This usually happens, when most of the project work can be done by the project participants themselves. Sometimes a project faces a very specific problem, which the project participants hardly can solve themselves. An expert then is utilized to solve this problem. The other task for experts that are assigned to projects for a short period is to train the other project participants. Here the expert is not always directly involved into the project work. The trainings are sometimes also organized in form of general workshops.

In Logica, experts are employees that help solving specific tasks in projects. But if no experts can be found, the tasks are not solved. This leads to two options. First, the project participants have to build up the knowledge on their own. This can be very time consuming and lowers the quality of the project work. Second, an external expert has to be hired. External experts work like the internal ones, but they are expensive. Hence, either of the two influences the project.

To make the expert finding easier, Logica implemented two different lines of interaction throughout the company. First, project managers are communicating among each other. There are different meetings where project managers exchange experiences. The goal is to raise the awareness about experts in the company. Second, every project is being supervised. Every so called project supervisor is mentoring a number of different projects. This helps overseeing a vast number of people and their expertise. But even project supervisors have a limited view, although they are mentoring several projects. Despite these two approaches, Logica's employees claim that experts cannot always be found.

6.3.3 Sharing Support (A3)

The analysis of Logica regarding problems with the knowledge management led to a finding concerning the support of sharing (table 10). Sharing the knowledge among employees, which can be considered as helping each other, is often referred to as a culture of sharing. Logica has a well working culture of sharing, within the boundaries of projects. However, on the company level the opposite

Problem	Reasons	Consequences	Approaches
Employees do not share their knowledge with people from other projects	- Sharing is not valued high in company - No policy on sharing - No rules how to use tools	- Productivity gain remains unused - Knowledge sharing is considered a waste of time	- Applying tools (Forum & Mailing lists)

Table 10: Diagnostic Map for Problem "Sharing Support"

is the case. And that is a problem: Employees do not share their knowledge with people from other projects.

> *"Many people try out new stuff, but they never share their knowledge. We had trouble with Jboss server. It took my people too long to install and use it for certain issues. They tried to google their problems and asked colleagues they knew from previous projects. But it did not help. In other projects they even do not know whom to ask. We are always reinventing the wheel." (Project manager of project Beta during project interview)*

This problem, the lack of sharing, has two reasons. The first reason is that sharing is not valued high in Logica; the focus is on projects. My different interview partners explained that project participants are open to share among each other. The employees within a project are keen to communicate and share their knowledge. And yet, on the company level, the keenness to share is rather low. The effort seems too high for employees outside the own project, especially for people that do not know each other directly.

> *"In most cases, sharing with people outside the own project happens only if it means low effort for one self." (Process designer from Logica during an interview)*

The second reason for the lack of sharing is that sharing is not supported by Logica. There is no policy, no guidelines or rules regarding the sharing of knowledge in Logica. This can be observed best when looking at the two different tools that aim at knowledge sharing: A forum and mailing lists.

The *forum* is standard software (Yammer[19]) and accessible for every employee through Logica's intranet. During an interview I was shown how it works: Employees can use it to ask a question or provide details about something to the whole company. Every employee is then able to reply, gather more information in order to find a solution. Thus, it also helps to establish a connection to a colleague with certain expertise.

The idea of such a tool seems fine, but there are several problems with the forum. First, it is hardly used to actively spread information. In most cases employees rely on previously asked questions and the willingness of knowledgeable others to reply. Second, no timeframe is guaranteed. An answer could take a few minutes or hours, but also weeks or months. In worst case the question

[19] http://www.yammer.com/

will not be answered at all. Third, often questions are simply not seen. A question without an answer does not necessarily mean that no one in the company has the knowledge. Most questions are not seen by every employee. Fourth, a knowledgeable person has to be willing to reply a request. Even if employees know the answer to a question, they do not necessarily answer it. Employees are not always keen on sharing their knowledge with people they do not know.

The *mailing lists* are used frequently. Many different of these mailing lists exist, one for each field of interest. Employees can subscribe to them and then participate (actively or passively) in on-going conversations or start new ones. According to people I interviewed, the mailing list for architecture for example contains typically five to ten messages per day. But the value of it was in doubt, because the mailing lists in general suffer from a major problem: Not all posted e-mails are read by all employees. Knowledgeable colleagues do not necessarily read an e-mail targeted at their knowledge. There are various reasons for that. One reason is that the mailing lists involve too many e-mails for a busy person. Another reason is that not everyone participates in the mailing list, so the mail never reaches the knowledgeable person.

Using either of the tools is voluntary. Logica does not have any rules or guidelines for the use of these or other tools to share knowledge. Neither does a policy exist of how to share knowledge in general. The tools are not even company-wide promoted. Many employees in Logica are not aware of the existence or how to use the two tools. This limits the value of the tools.

As sharing is not supported adequately, many opportunities are missed out where employees could help each other. This influences the general quality of all project's work. My analysis shows that, based on the reasons given above, the quality level in forum and mailing lists is not always high. I was told that often questions stay without responses. Some people in Logica claim that the attention to the tools decrease and that these are partly considered a waste of time. But also, a possible gain of productivity remains unused.

Logica approaches the problem that knowledge is not being shared through the application of these two tools. But the missing policy and guidelines for their utilization decreases their value. They barely improve the lack of knowledge sharing.

6.3.4 Documentation Level (A4)

Problem	Reasons	Consequences	Approaches
Documentation of project work is often inconsistent or incomplete	- Documentation is different in each project - Documentation of communication is costly and time-intense	- Difficulties to learn from documentation - Documentation not sufficient to increase productivity	- Nothing

Table 11: Diagnostic Map for Problem "Documentation Level"

The analysis of Logica regarding problems with the knowledge management led to a finding concerning the level of documentation (table 11). In general,

documentation has to be of certain quality in order to learn from it. To enable re-use of knowledge in Logica, every project has to provide documentation of decent quality. This is, however, often not the case. Most projects do not externalize the knowledge about the project work sufficiently, in order for employees from other projects being able to learn from it. The analysis shows that the documentation is usually not detailed enough, incomplete and/or inconsistent, which is a problem.

> *"Knowledge is currently not sufficiently explicated and codified and in such a way which is suitable for other projects to read." (Project manager from Logica in a requirement analysis document)*

Logica's strong focus on project work is also the reason that the documentation level is too low. Logica provides the liberty to every project, to organize itself autonomously and does not oblige standards for the documentation and communication within projects. There are no suggestions regarding the general practice or tool selection. The level of documentation, and therefore the externalization of knowledge, depends on every project separately. The documentation is thus different in each project.

To illustrate, why the ability of organizing the documentation in a project freely influences the quality of documentation, I again provide insights from a Logica project (Project Alpha from section 6.3.1). The project manager explained that knowledge is well shared within the project; the participants generally have the same knowledge. This is achieved through close communication. The project team uses instant messengers (MSN Messenger[20]) and e-mail, but more importantly, they meet on the fly and have personal discussions. Yet, none of these conversations (regardless which channel or how many people are involved) is documented. Many decisions or bug reports are lost in casual communication. The only persistent documentation within project Alpha is done in Jira, however, not consistently. Not every aspect of personal interaction is included here, neither are many bugs or tests.

The documentation provided by projects is often too loose and disconnected to be meaningful. Crucial information is missing in many cases. Most projects do not document their communication and project work sufficiently, because it is costly and time-intense. Logica's employees do not have the time and motivation to actively document the project work or the internal communication.

> *"Too much time is used manually moving and transforming information around different media and programs, and between structured (e.g. risk lists in spread sheets) and unstructured (e.g. emails) formats." (Project manager from Logica in a requirement analysis document)*

The level of documentation influences the re-use-ability. In Logica this level is very low, i.e., the externalized knowledge in form of easily accessible data is not sufficient in terms of quality and quantity. Inconsistent and incomplete documentation results in difficulties to learn from it. My analysis shows that most projects are not able to re-use the externalized knowledge of other projects in Logica, because of these reasons.

[20] http://explore.live.com/messenger/

6.4 Inadequate Bridging of Knowledge

The problems described above as part of the project isolation show a strong relation among each other. In fact, all four of them are consequences of the strong project-orientation in Logica. In this section, the interconnection between the problems is much weaker; some of them are barely related. The inadequate bridging of knowledge works more as an umbrella term here. Each problem illustrates an approach to bridge the islands of knowledge. They present grievances or defects within Logica regarding knowledge management. I found three major problems in this area of concern.

6.4.1 Process Complexity (B1)

Problem	Reasons	Consequences	Approaches
Process descriptions are too complex and numerous	- Studying all process descriptions takes too long - No support for project managers	- Project managers ignore process descriptions - Increases risk potential for projects	- Nothing

Table 12: Diagnostic Map for Problem "Process Complexity"

The analysis of Logica regarding problems with the knowledge management led to a finding concerning the complexity of processes (table 12). In general, a process description resembles practices, which are based on the experiences of previous project work. Following such a process description helps to assure a certain level of stability and security for every project. In Logica it is mandatory for project managers to follow process descriptions. Project managers have to know these process descriptions and choose the fitting one for their projects. The problem is that process descriptions are too complex and numerous.

> *"We face the problem of ignorance. There might actually be a standard procedure or standard process or standard way of doing the job, which likely works, but somehow the people that do this job don't really know about it. Maybe they never heard about it, maybe they have forgotten, maybe something else... And it's not always easy to find out, whether there are some standard tools or processes for that." (Project manager from Logica during a project meeting)*

Before explaining the problem with process descriptions, I want to provide more details about the background. In Logica, process descriptions are obligatory to follow for three reasons. First, they provide certain stability and optimization in the organization. Although following a process description does not guarantee high quality, a process description can help to increase it. Second, the process descriptions are intended to support project managers. They should help project managers to organize their projects in a Logica approved way. The third reason for the process descriptions is the conformity. A high

grade of similarity in the project organization across the company simplifies the auditing and controlling tasks.

My analysis shows that Logica has the process descriptions for good reasons, but also that the project managers are confused and overwhelmed by their amount and complexity:

> *"It's also a matter of complexity and information overload. There are so many good things you could do, there are so many things you could consider, there's so much information... You more or less just give up and just do it the way that you think is good." (Project manager from Logica during a project meeting)*

The complexity of process descriptions becomes a problem, because project managers are often not able to choose the best fitting one. This is based on two aspects. First, Logica has no instance to support project managers in their choice of process descriptions. A project manager is figuratively left alone with that decision. There is actually no support for project managers to deal with the process complexity. Second, my analysis shows that a project manager has to spend days, if not weeks, in order to study all process descriptions at Logica. This is very time consuming and a too demanding task, especially in the initial phase of a project. It simply takes too long to study all process descriptions.

The complexity in finding the right process description has the consequence that it is too difficult for project managers to do so. It is figuratively impossible for them to make an informed decision on which process descriptions to choose. According to my analysis, it is of rare occasion that project managers find the correct process descriptions and apply them correctly. Instead, project managers start ignoring the process descriptions and work according to the best of their knowledge and belief.

In Logica most projects violate the company's rules by not following the process descriptions. However, that might not even be based on a conscious decision, the complexity of process descriptions makes it too difficult for project managers to find the right one. The downside is, that not following the defined processes, potentially contains risks for the project, which the project manager might not be prepared for.

6.4.2 Feedback Circle (B2)

Problem	Reasons	Consequences	Approaches
Communication between process designers and process executors is difficult to establish	- No easy/direct channel for feedback - Finding the correct person to provide feedback to is difficult	- Process descriptions are being ignored - Process descriptions are not being improved	- Nothing

Table 13: Diagnostic Map for Problem "Feedback Circle"

The analysis of Logica regarding problems with the knowledge management led to a finding concerning the feedback circle between the process designers and

the process executors (table 13). Generally, process executors should feed their experiences back with specific process descriptions, in order to enable process designers to improve these process descriptions. In Logica this feedback circle is not closed, feedback is difficult to provide. The problem is that communication between process designers and process executors is difficult to establish.

> *"Official documents (e.g., process descriptions, employee guides) exist and are accessible for every employee. However, changing them is not a simple task. To modify one of these, the responsible manager would have to be contacted. But it is not always easy to find out who the responsible manager is. This expands the feedback circle. In many cases the effort for employees to criticize on documents is too high, so that comments of users are not included." (Project manager from Logica in a requirement analysis document)*

To understand the problem with the extensive feedback circle, one has to understand why feedback on process descriptions is important. Logica has employees creating process descriptions, in order to make the knowledge of previous project work re-use-able. The focus is here on the process related knowledge, not the technical aspects. In order to cover the variety of projects, these process descriptions have to be of general nature. However, not all process descriptions can easily be applied.

In Logica, project managers have often difficulties to simply apply a process description. There are two reasons for these difficulties. First, the process descriptions are too general. Project managers then have to find a way to implement the general process in the specific project. Second, the process description has certain weaknesses. The process description can be imprecise or conflict with other obligations. Even if project managers are able to apply a specific process description, they often have comments or suggestions to it. In each of these cases a project manager should provide feedback to the process designers. This would raise the awareness, that a process description needs to be improved. Providing feedback, however, is a problem in Logica.

The problem with a confusing feedback circle is caused by the lack of a contact line of communication. In Logica, no easy or direct channel between process designers and process executors exists. And even further, it is very difficult for employees to find the correct person to provide feedback to.

This problem has the consequence that feedback is in many cases simply not provided. Taking the extra effort to provide feedback is often too complex for most employees. My analysis shows that the lack of feedback has two main consequences. First, the process descriptions are not being improved. The process designers, who do not receive feedback, conclude that the process description works fine, because nobody comments on it. Second, the process executers get fed up by the process description and ignore it. Instead of following a process description the employees sometimes organize their work autonomously. Employees then violate the company's policy. Hence, the confusing feedback circle lowers the usefulness of process descriptions.

Problem	Reasons	Consequences	Approaches
Documentation provided by projects is not connected	- Documentation is different in each project - No company-wide documentation policy or tool support for documentation	- Possibility to re-use is lowered - Awareness of knowledge in the company is low	- Project supervisors

Table 14: Diagnostic Map for Problem "Connected Documentation"

6.4.3 Connected Documentation (B3)

The analysis of Logica regarding problems with the knowledge management led to a finding concerning localizing the documentation of the different projects (table 14). It is common in Logica that different projects have similar goals or tasks. They work in the same domain, with the same technology or the like. My analysis shows that projects constantly create all different kinds of documentation of the gained bits and pieces of experiences, which could be re-used by other projects. But to enable this re-use, a connection between the different parts of documentation has to be established. A connection allows employees to find the information of interest. But that is a problem, because the documentation provided by projects is not connected.

> *"It is rarely possible in the current technology to find knowledge items, which are not written directly into a particular document with a particular focus or referred to explicitly by the authoring manager." (Project manager from Logica in a requirement analysis document)*

This problem, the lack of connection among the documentation of projects, is rooted in the autonomous organization of each project. As explained above, every project is granted the liberty to organize its documentation independently. My analysis shows that many different types of documentation exist. The documentation is different in each project. In Logica, every project develops its own lingo and its own documentation routine. Often, even within one project, these are followed inconsistently, which makes it difficult to automatically find the desired information.

An employee that reads a document can relate to most of the topics included and understand the objections. But this is a very complex task, as employees would have to open every document manually. That makes the information difficult to find. Logica could prevent this manual process through company-wide documentation policies or tool support, but neither exists.

The difficulty to find documentation has two consequences. First, it lowers the possibility to re-use knowledge. I learned that even useful information remains unused, because it cannot be found. Second, the awareness of knowledge in the company is low. A project manager of Logica explained to me that the difficulty to find documentation results in a decreasing understanding of the work of other projects. Employees then sometimes do not even consider

searching for other projects documentation, because they have no idea where to start.

To increase the connection of documentation between projects, a project supervisor monitors a variety of projects in Logica. Project supervisors support the projects in finding the recorded experience of other projects. They are able to connect the documentation between projects they worked with. However, this is still a manual task and the project supervisors have a limited view. Despite the help of project supervisors, employees claim that documentation often cannot be found.

6.5 Summary of Identified Problems

Seven different problems were discovered in Logica (table 15), divided into two groups. However, the connection among the problems within each group varies. The problems that are part of the isolated islands of knowledge (Numbered with the prefix *A* in table 15) show a strong relation between each other. They are all aspects of the same big problem, if you will. The connection between problems described as inadequate bridging of knowledge (Numbered with the prefix *B* in table 15) is much looser. Here, the umbrella term shows that each problem is more general and affected by challenges in Logica's organization. These problems are also much more distinct from each other, compared to the project isolation ones. Each focusses on slightly different aspects within the organization of Logica and shows where knowledge management is being prevented or discouraged.

No.	Section	Problem
A1	6.3.1	Projects internally encapsulate knowledge, which is created through project work
A2	6.3.2	Finding experts that are already in the company is difficult
A3	6.3.3	Employees do not share their knowledge with people from other projects
A4	6.3.4	Documentation of project work is often inconsistent or incomplete
B1	6.4.1	Process descriptions are too complex and numerous
B2	6.4.2	Communication between process designers and process executors is difficult to establish
B3	6.4.3	Documentation provided by projects is not connected

Table 15: Logica's Knowledge Management Problems

The distinction between different problems is not always easy. Sometimes different problems intersect or affect another, despite the hierarchical organization and visualization (figure 25 on page 86). Interconnections between problems occur. For example, the lack of connection of the available documentation (see problem B3 in table 15) is a general problem of Logica. However, it can also be understood as a result of the knowledge encapsulation within projects (A1) and the insufficient explication of project knowledge (A4).

Even though the distinction between the different problems can be difficult, because they are related, this is of rather low relevance. Each of the stated findings is a problem on its own. Looking at one at a time, as done above, shows the challenges of Logica regarding the knowledge management. The solution to a single one of them would decrease these knowledge management issues.

> *"We have a lot of people, who know a little bit of this and a little bit of that. And we have to combine what all these people know." (Project manager at Logica during a project meeting)*

My analysis shows that the knowledge sharing between projects in Logica is very low. While the implications and perspectives of this are multifaceted (hence the seven problems), the reasons can be traced back to two simple aspects. First, there is the missing support by the company. A tool could make a change by simplifying the sharing aspects. Second, most problems discussed above emerge from the independent organization of projects. My analysis shows that in many cases, this liberty is misused as a reason, to ignore what might be good for the company. Logica also suffers from the attitude of its employees.

> *"A lot of our project managers have the opinion: It's my project, it is unique." (Project manager from Logica during a project meeting)*

Many project managers think that their projects are very special. This is based on a level of ignorance, but also out of unawareness. In many cases project managers simply do not know about other projects and how they could relate. Be it ignorance or unawareness, both have the same consequence. Either way, knowledge is not being re-used, but built up over and over again. It is expensive for Logica that projects do not utilize the knowledge of other projects within the company or do not provide their knowledge to other projects within the company. Therefore, Logica wastes available resources and misses out on achieving a competitive advantage.

7 Building

The previous chapter covered the problem analysis, where I described Logica's knowledge management problems. That corresponds to the first stage of ADR as outlined by Sein et al. (2011). In ADR the development of the system then happens in the second stage: Building, intervention and evaluation (BIE, see section 5.1.3). Here, the problems are being approached through the constant iteration of these BIE circles. The software is constantly developed, applied and evaluated. The final knowledge management system is therefore the product of an evolution through several iterations. This chapter deals with the first aspect of this circle, the building.

I describe the underlying ideas of the knowledge management system in detail (section 7.1), followed by the overall design of a system to approach the knowledge management problems (section 7.2). Afterwards, the functional design of this system (section 7.3) and the technical design (section 7.4) are explained. Finally I provide a workflow, the process description, which explains the intended use of the envisioned system (section 7.5).

As explained above, ADR follows a strong iterative approach (see section 5.1). The BIE circles are continuously executed to improve the system and elaborate on it over time. The work of this thesis is based on more than two and a half years of iterating through these circles in the KiWi project. These iterations were of varying length and depth. Early iterations focused on conceptual aspects or proof of concepts. Later the development of the system was expedited. In general, ideas were rarely invented from scratch. Instead, ideas mostly developed over time, after discussing an aspect and examining it in the system through different versions. Sometimes ideas were thought of being good, but once they

made it to the system the evaluation showed that they were not. This led to different actions, some ideas had to be improved, others were abandoned again.

ADR is a strongly problem-oriented process. It focuses on solving a problem, by constant evaluation and shaping of an IT artefact. The iteration of BIE circles results in high quality of the work process, the final system and the research outcome.

The downside of an iterative approach like this, however, is that it is difficult to track. The constant changes of the software and its requirements are not easy to explain afterwards. One has to describe step by step of such a process, which results in long and tiresome reports. A chronological description of the real process could be very detailed but often readers would find it confusing and difficult to comprehend.

In order to tackle Logica's knowledge management problems a knowledge management system was created. In this chapter I explain this system in detail. I describe the design process and clarify reasons and decisions. However, to communicate all these different aspects to the reader in the easiest way in this thesis, I step back from the detailed documentation of the iterative process. Sticking to the iterative description would not help the reader to comprehend the system and its creation. Instead, I chose to improve the readability by describing the whole process as if it was linear.

Parnas and Clements (1985) explain that it pays to fake a rational design process. By presenting the system as if its design followed such a rational and systematic way increases the readability. Every aspect reported in this case is relevant for the final system. Intermediate results or dead-ends, despite being an important aspect of the actual iterative design process, are left out of the description. The outcome is a coherent documentation, which explains the design of the final system and the underlying ideas.

I document the system's design and development as if it was the result of distinct consecutive phases and as if it precisely followed a rational and systematic way of software development. The sections in this chapter build upon each other and each provides more details to the previous one. The actual iterative process remains unexposed.

7.1 Underlying Ideas

I utilize the knowledge management strategies by Hansen, Nohria, and Tierney (1999) as my design theory. I discuss these strategies in more depth above (see section 3.2). The authors explain that a strategy has to be followed, in order to successfully organize the knowledge management within a company. Two different *knowledge management strategies* are described: Personalization and codification. *Personalization* focuses on the direct and personal communication among employees. They share their knowledge directly from one to the other. *Codification* focuses on the externalization of knowledge, by all employees. Others then are able to learn from the externalized sources.

Hansen et al. (1999, p. 112) recommend an 80-20 split between the knowledge management strategies (see section 3.2.3). A company should focus on one strategy and apply the other one for support (figure 26).

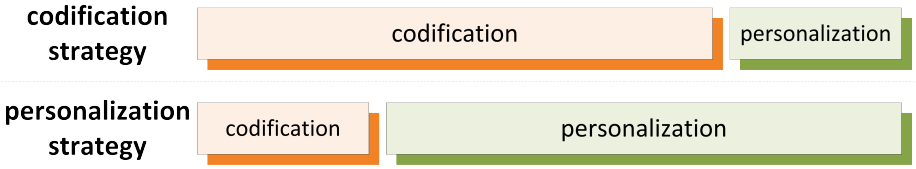


Figure 26: Knowledge Management Strategies and their Shares

7.1.1 Strategies and Layers

One way to look at my problem analysis is that there are different knowledge management strategies applied within Logica in parallel. Hence, I distinguish between the *development layer* and the *management layer*.

With *development layer* I refer to those parts of Logica that are involved in the actual project work. This involves literally all project participants, including the project manager.

The communication among the participants of every project works very well. The employees communicate directly and through many different channels (e.g., personal talk, e-mail, and messenger). Also, they often meet unscheduled, without much organizational overhead. Their meetings are usually arranged on the fly. In many cases employees meet spontaneously and discuss project issues. The constant and direct communication is sharing the personal knowledge among the employees. This direct communication is what Hansen et al. (1999) describe as personalization.

My analysis shows that Logica lacks rules or guidelines for the internal organization of projects (see section 6.1). The effect is that project managers organize their projects autonomously. As described above, several knowledge management problems of Logica (see section 6.5) are influenced by this.

The focus of project work is on achieving the project goals. Projects can be organized autonomously, without direct communication lines to other projects. The effect is that the project-internal communication is much more direct and simpler. The problems A1, A4 and B3 (cf. table 15 on page 98) show that an effect of this is the neglection of documentation. These problems illustrate that the codification in projects is very low; the developers share their knowledge through personalization instead. The project work therefore follows a personalization strategy regarding its internal organization.

With *management layer* I refer to the entire controlling instance of Logica, i.e. those parts that are responsible for the management of the whole company. This involves a variety of different aspects of the company, like the strategic planning, process design or project monitoring. Project managers are also part of this layer.

The communication within the management layer is mostly document-driven: Project managers create reports to provide details about the project's status and process designers create process descriptions. The reports as well as the process descriptions are explicated project knowledge, which are being processed and shared for others to learn from.

This document-driven form of communication is called codification (Hansen et al., 1999). My analysis shows that codification of knowledge as reports and documentation is widely used throughout Logica (see section 6.1) and can be

considered as the company's general knowledge management strategy. Guidelines define the codification approach and the internal hierarchy is aligned to it. However, this is also the reason for several knowledge management problems of Logica, as described above (see section 6.5). Especially the problems A2, B1 and B2 show that the communication in the management layer is not working properly. This has a negative impact on projects and the knowledge re-use approached through the documentation is much lower than aimed. The level of personal communication in the management layer is rather low. Even though meetings are performed regularly, the knowledge sharing is almost entirely based on documents.

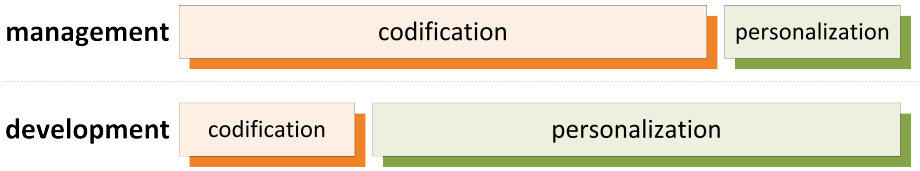


Figure 27: Logica's Organizational Layers with their Knowledge Management Strategies

From this analysis one can see that different knowledge management strategies are dominant in different layers of Logica (figure 27). This leads to the first design idea.

> ***Design Idea 1:*** *Supporting the two organizational layers in Logica with different knowledge management strategies: The management layer follows a codification strategy and the development layer the personalization.*

The idea is that Logica's knowledge management problems shall be approached by taking the division of knowledge management strategies into respect. According to Hansen et al. (1999) a company should pick a primary knowledge management strategy and use the other one to support it, in a split of 20 – 80. Design idea 1 however embodies that Logica follows two different strategies in different layers of the company (Jahn and Nielsen, 2010, 2011).

Additionally, it can be observed that the support of the secondary strategy in either of the layers is very low. As explained above, the management layer has figuratively no personalization and the low level of codification in the development leads to several problems.

7.1.2 Strategies and Problems

In order to defend the first design idea, I compare it to the problems I found through the analysis (section 6.5). I review every problem separately, and discuss its relation to the design idea 1. This includes that I point out which layer and which strategy the problem concerns. Afterwards, I provide a summary for this (see table 16).

Problem A1: Information Access

The problem is that the information access of Logica's projects is difficult, because each project can be organized autonomously and the level of documen-

tation is rather low (see section 6.3.1). However, the knowledge sharing works within the projects on a highly personal basis. Employees outside the own project are barely considered and the codification of knowledge hardly plays a role. Therefore, this problem concerns the personalization part within the development layer. The personalization works within projects only, not between them. It is therefore limited, as it does not work across the entire development layer.

Problem A2: Expert Finding

The problem is that finding experts within Logica is difficult (see section 6.3.2). The knowledge about skills of colleagues is shared through direct contact only. Employees only know about the expertise of those colleagues, whom they worked with. Therefore, this problem concerns the personalization within the development layer.

Logica introduced project supervisors and meetings of project managers to support the knowledge sharing about the expertise of employees. Therefore, this problem also concerns the personalization of the management layer.

Problem A3: Sharing Support

The problem is that knowledge is not being shared through the application of two provided tools (see section 6.3.3). As these tools are voluntary and no guidelines support them, They are hardly in use by employees to codify their knowledge. Therefore, this problem concerns the codification part of the development layer.

Also, as explained above, many employees of Logica are not open to share their knowledge with colleagues they are not directly involved with. Sharing is not part of the company's culture. Therefore, this problem also concerns the personalization of the development layer.

Problem A4: Documentation Level

The problem is that the codified knowledge in form of easily accessible data is not sufficient, regarding the quality and quantity (see section 6.3.4). The project participants focus on the project work and do not consider the re-use possibility. The result is incomplete and inconsistent documentation, which others can hardly gain insights from. Therefore, this problem concerns the codification of the development layer.

Problem B1: Process Complexity

The problem is that the high complexity in finding the right process description has the consequence that it becomes too difficult for project managers (see section 6.4.1). Project managers have to study all process descriptions for a long period of time, which is hardly possible in the project work. Therefore, this problem concerns the codification of the management layer.

Problem B2: Feedback Circle

The problem is that no easy or direct channel between process designers and process executors exists (see section 6.4.2). The line of communication is difficult, which shows that the problem concerns the personalization within the management layer.

Additionally, a lack of feedback decreases the possibility to improve the process descriptions. These process descriptions often remain untouched. Therefore, this problem concerns also the codification of the management layer.

Problem B3: Connected Documentation

The problem is that the lack of connection among the documentation of projects makes documentation within Logica difficult to find (see section 6.4.3). This concerns the documentation of the whole company, and therefore the codification in both layers. The gaps in the documentation affect not only one layer of the company. Employees are not able to find it on the management layer and to use it on the development layer.

Summary

Mapping the knowledge management problems, found in the analysis, to the design idea 1 shows a large overlap. All problems are concerned with strategies of the different layers. Table 16 visualizes that the isolated islands of knowledge (problems A1 – A4) are mostly in the development layer, while the inadequate bridging of knowledge (problems B1 – B3) take mostly place in the management layer.

	Personalization	**Codification**
Management Layer	A2, B2	B1, B2, B3
Development Layer	A1, A2, A3	A3, A4, B3

Table 16: Mapping Logica's Knowledge Management Problems to the Organizational Layers and Knowledge Management Strategies

The problem structure matches the first design idea of different knowledge management strategies in two organizational layers. Whether or not the problems can be solved through a system that follows this design, however, is the objective of the evaluation.

7.1.3 Improving the Situation

I just showed that Logica's knowledge management problems are addressed by the knowledge management strategies. Here, I briefly discuss how the first design idea can improve the situation and solve the problems.

My analysis shows that the participants of a project show close cooperation and extensive personal communication. Between different projects, however, knowledge is barely shared. The analysis found several problems with that concern (A1, A2, A3, A4 and B3). The design intention is to strengthen and

thus improve the codification of the project work within the development layer. I can see the potential for this to support the knowledge sharing between projects. An increased codification in form of documentation can support the re-usability of codified knowledge by employees in other projects.

Regarding the management layer the approach is different. The knowledge sharing here does not work well. My analysis shows that problems emerge from it (B1, B2 and B3). The implemented codification strategy seems to be too complex. Employees state that the process descriptions are too numerous (see section 6.4.1) and difficult to improve (see section 6.4.2). The result, as I describe in the problem analysis, is that employees undermine the codification strategy. Personalization is important, supporting it as the secondary strategy is the design intention. This can improve the codification. When a project manager can contact the designer of a specific process description directly, misunderstandings or problems can be sorted out immediately. The project manager can then continue using the process description without problems.

7.1.4 Connecting the Layers

After pointing out the two layers within Logica, it is reasonable to ask how they connect. The knowledge needs to be shared vertically as well, i.e. across both layers. In Logica this is the responsibility of the project managers, who is part of both layers.

Every project manager is responsible for the organization of their projects. Project managers have to plan and coordinate the project work. This involves tight communication with the project participants. As Logica holds their project managers responsible for the project work, they always have to know the current status of the project. Also, the project managers have to report the project's status and findings to the management layer in regular meetings. This is part of Logica's general controlling mechanism, in order to avoid bigger problems for the company based on bad management. By reporting the status and progress of the project, a project manager figuratively takes the knowledge from the development layer and brings it to the management layer.

This also works the other way around. Project managers apply the process description and follow guidelines on how to manage projects. By re-using codified knowledge, a project manager actually takes the knowledge from the management layer and brings it into the development layer.

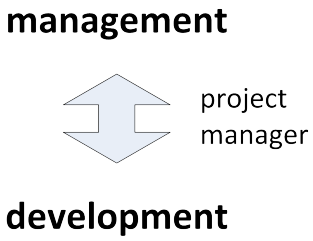


Figure 28: Knowledge Sharing between the Organizational Layers

The project manager is therefore the only instance that is responsible for knowledge sharing between the two organizational layers (figure 28). It is the

project manager's responsibility to share knowledge (e.g., reports, descriptions) from the management layer to the development layer and vice versa. The project manager is therefore an important and crucial role in Logica. The second design idea is supporting the project managers in their tasks.

> ***Design Idea 2:*** *Connecting the two organizational layers in order to establish and support knowledge sharing between them.*

To improve the knowledge management in Logica, the idea is to connect the two layers (figure 29). Each layer should focus on its own knowledge management strategy (design idea 1).

The system to be designed has to ensure the knowledge sharing between those layers, i.e. strategies. It is a deliberate decision to improve the sharing knowledge between the organizational layers, because it helps to tackle all of the problems, to certain extend.

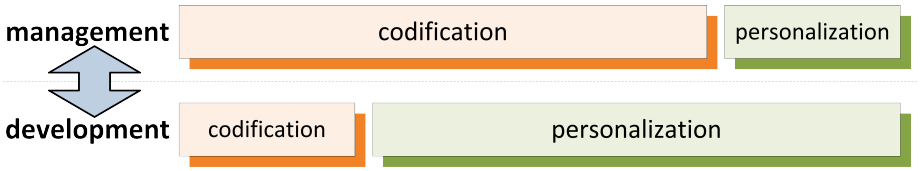


Figure 29: Different Strategies in different Layers of the Organization, from (Jahn and Nielsen, 2011)

Hansen et al. (1999) do not mention an approach that involves different strategies within the same company, neither the connection between different strategies. However, based on the design ideas 1 and 2 the system to be designed has to follow a vertical approach through the different organizational layers (Jahn and Nielsen, 2010, 2011). This can be followed in three different parts:

- Support the codification strategy in the management layer
- Support the personalization strategy in the development layer
- Support the knowledge sharing between the two layers

A knowledge management system has to take these three aspects into account, in order to improve the knowledge management for Logica. Both layers and the communication between them have to be considered.

7.2 Overall Design

I base my deliberations regarding the support of the knowledge management in Logica on the theory of *knowledge management systems* by Davenport and Prusak (1998), which I discuss in more depth above (section 3.3). These systems mainly focus on storing data in one way or the other into a knowledge base. We distinguish between three different knowledge bases (see table 2 on page 36):

- The *external knowledge base* contains input on aspects outside the company, it is therefore not of interest for this study.

- The *structured internal knowledge base* stores formal information of the company.
- The *informal internal knowledge base* represents documentation of communication.

7.2.1 Supporting the Layers

The differed knowledge bases are utilized in supporting the different layers, according to the first design idea. In the following I describe the approaches for a knowledge management system for Logica, which look at each organizational layer separately.

Development Layer

I just described that the development layer is following a personalization strategy. To support the personalization, different systems are available (see section 3.2.2). For this project the design decision has been taken to use a wiki. This is a reasonable choice, as I showed that wikis are collaborative tools, which support the communication on sources (see section 3.3.2).

> ***Design Idea 3:*** *Utilizing a wiki in order to support the personalization strategy within the development layer.*

Wikis contain what Davenport and Prusak (1998) call an informal internal knowledge base. People use wikis to collaborate and to communicate on shared sources. The content is mostly unstructured information in regular text. Knowledge management systems of this kind work like an archive of textual communication.

However, users have to invest manual work, because specific data is not always easy to obtain. Logica's employees can use the wiki to share and discuss topics of interest. They can also use the wiki to collect information about a certain topic from many different people throughout the company. Hence, wikis increase the potential for knowledge sharing from one project to the other. Participants of other projects can simply access the documentation.

Hansen et al. (1999) pointed out that a knowledge management strategy needs support by the opposite one. A wiki supports the personalization, as it is a collaboration tool. It helps to bring people together to cooperate. However, it also strengthens the codification strategy to support the personalization. The whole collaboration is being stored. This also includes the externalized knowledge in the documentation of work process. Hence, wikis embody the codification support of a personalization strategy.

Management Layer

Above, I described that the management layer follows the codification strategy. This can be realized through a variety of tools, which are mostly used under the term *Enterprise System* (ES). An ES supports a company with different aspects of the management work (e.g., data analytics or business processes) (Davenport, 1998; Hitt et al., 2002; Umble et al., 2003).

This however, is very general. There is a variety of tools for these tasks. Logica has a very specific need for an ES that supports the process management and the project management, a so called *process* or *project management system* (PMS). Every project manager in Logica works with and every project is managed through a PMS. This is an essential aspect of the work in Logica. The main focus of such a PMS is to collect, manage, analyse and communicate data from the project work. For Logica such a PMS is absolutely essential. Even though different ones are in use, any knowledge management approach has to take a PMS into account.

> ***Design Idea 4:*** *Utilizing a project management system in order to support the codification strategy within the management layer.*

In most cases, an ES contains what Davenport and Prusak (1998) call a structured internal knowledge base. The general principle of a PMS, to be more specific, is the collection of structured information. Data is stored in forms and the user can view (and, depending on the context, also edit) every field separately. Hence, the documentation through a PMS follows a strict formalization and every data set is the structured codification of knowledge. This makes the contents easy to process for computers. Data sets can easily be compared and calculated. A PMS focuses on bringing the users and the documents together and therefore supports the codification only, but not the personalization.

7.2.2 Connection between the Layers

I explained above that the project manager is a crucial role in Logica. It is the project manager's responsibility to share knowledge between the development and the management layer. The project manager is therefore the connection between the two layers. Design idea 2 projects this role into the knowledge management system. The systems that support each layer have to be connected.

The connection becomes even more important through the utilization of design ideas 3 and 4. Exchanging data between the two different knowledge bases, that are being created for each layer, is a difficult task, which can be very time consuming and complex. The project managers have to be supported in order to achieve a working system.

The knowledge management system to be created has to support the entire company. A plain support of the layers would not benefit Logica. In contrary, the job of the project managers would become more difficult. This raises the danger that both layers become isolated from one another. Connecting the layers therefore also means bringing the different parts of the company closer together. Knowledge sharing between the two layers can improve the general knowledge management within Logica and thus provide a competitive advantage.

This connection between the layers is a challenge on the technical level. In contrary to the knowledge management strategies, which focus on the collaboration of people, this aspect has to be solved when developing the system. Despite the more detail is this aspect already covered by design idea 2, a new one is therefore unnecessary.

7.2.3 A Heterogeneous Knowledge Management System

To approach the analysed knowledge management problems (chapter 6), in the sections above I describe four design ideas for the knowledge management system, which are the fundamental tenets of the design.

Design Idea 1: Supporting the two organizational layers in Logica with different knowledge management strategies: The management layer follows a codification strategy and the development layer the personalization.

Design Idea 2: Connecting the two organizational layers in order to establish and support sharing knowledge between them.

Design Idea 3: Utilizing a wiki in order to support the personalization strategy within the development layer.

Design Idea 4: Utilizing a project management system in order to support the codification strategy within the management layer.

In order to improve the knowledge management for Logica, a knowledge management system is supposed to be created, based on these four design ideas. But especially the design ideas 3 and 4 indicate two different tools to be utilized (table 17) in order to support the different knowledge management strategies (design idea 1). These have to be connected (design idea 2) to create one major system.

Organizational Layer	**Knowledge Management Strategy**	**Type of Knowledge Management System**	**IT**
Management	Codification	Structured internal knowledge base	PMS
Development	Personalization	Informal internal knowledge base	Wiki

Table 17: Horizontal Approach to Knowledge Management

The utilization of the theories regarding the knowledge management strategies (Hansen et al., 1999) and knowledge management systems (Davenport and Prusak, 1998) led to a knowledge management approach with a vertical system for Logica.

Different aspects of the company were taken into consideration. Both layers have to be supported separately, the management layer through a PMS and the development layer through a wiki. Additionally, the sharing of knowledge between both layers hasbe established easily. Connecting the two different systems improves the knowledge management for Logica for two reasons. First, it supports the work of a project manager. Instead of collecting the project knowledge and inserting it manually into an ES the project manager is simply able to re-use the created project knowledge. Second, connecting the layers supports the overall knowledge management strategy. This improves Logica's knowledge management as a whole, because it is more unified than the separation among the layers before.

7.3 Functional Design

To approach the different layers separately, following the four design ideas, two distinct systems were designed, one for each layer. One additional system is responsible to connect these two. The prototype of a knowledge management system created for these studies consists therefore of three systems: The *KiWi platform*, the *Data Exchange Agent* and the *Project Management Application*. In combination all three are called the *KiWi systems* (figure 30).

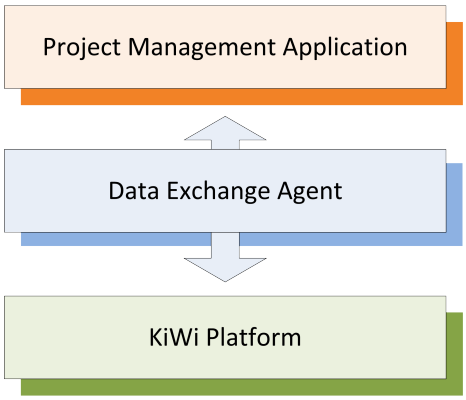


Figure 30: The KiWi Systems

Note that the KiWi systems are not entirely developed by myself. They are instead the outcome of the project participants' cooperation between the different project partners of the KiWi project. Instead of coding, my personal focus was on the design of the systems and on conducting the BIE circles.

A list of all features of the KiWi systems can be found in appendix B.

7.3.1 KiWi Platform

According to design idea 3, the system to support the personalization in the development layer is a wiki. Specifically, the system created is the KiWi platform (or short: KiWi), which is the main outcome of the KiWi project (see section 4.3). It is a semantic wiki with additional features to improve the knowledge sharing abilities for software engineers (Schaffert et al., 2009). As mentioned above, these features are the enabling technologies, namely personalization, information extraction, reasoning and querying (section 4.3.4).

With KiWi the users are provided a system that allows them to interact on text. The project participants can use it to share thoughts and discuss different aspects of their work. They can also use it to collect feedback on ideas. More generally, KiWi is documenting the project work by storing the different aspects and its relations to one another. This helps employees from other projects to recapitulate what happened and to learn from it.

KiWi is divided into several applications like an admin area and the actual wiki (see the big white links in the header in figure 31). The admin area contains all those settings an administrator is allowed to take hands on. The wiki part contains all the pages, which users can read and edit. There is also a dashboard, which can be considered as the starting point of the KiWi platform. When users

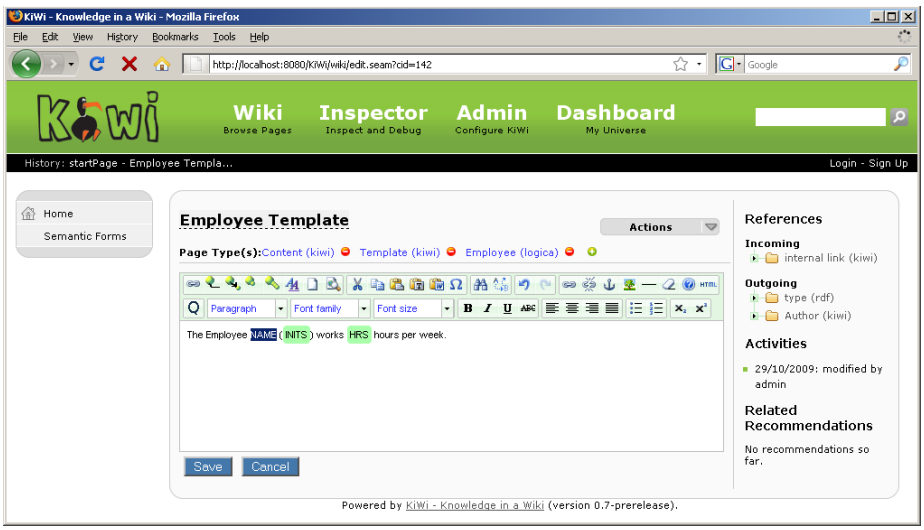


Figure 31: The KiWi Platform, Edit Mode (Screenshot)

log in, they can find an overview of other people's activities or the personal recommendations here.

Every KiWi page contains textual information, just like any regular wiki page. However, in addition to that, it also has semantic data and free form tags attached. The semantic data contains input about the page (e.g., date of creation or author), but users can further assign types from available ontologies and define the properties.

There are two modes of KiWi: Edit (see figure 31) and view (see figure 32). Toggling between these two modes works through the actions drop-down-list on top of each page. The edit mode allows users to edit the contents and semantic values of a KiWi page (figure 31). The different buttons in the editor help the users to perform the different actions for the content of a page. Further, users can assign tags and comments to a KiWi page. In view mode, a user can read the contents of a KiWi page (figure 32) and leave comments.

Both modes of the interface contain a right column, which includes the references of and to the page currently viewed and a recommendation section (see right side of the screenshots, figures 31 and 32). The references show the connections to other pages. That means links that go to or come from the page currently viewed at. Also, it references the authors and types.

Further down, the recommendations are presented to the user. These are provided by the personalization component of the enabling technologies of the KiWi project (see section 4.3.4). KiWi here displays links to pages that could be of interest for the user. To calculate these recommendations, KiWi analyses the user's behaviour. When the user shows interest in certain topics by adding tags or editing pages, the system suggests further pages of similar topics (Durão and Dolog, 2009a,c).

The KiWi platform contains many more features than the ones described above. However, these are those that are the most influential ones for my studies.

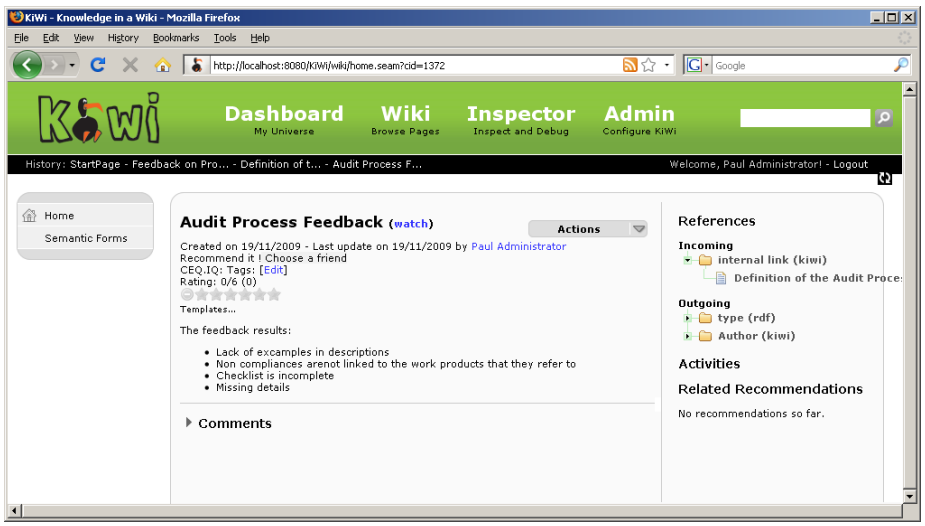


Figure 32: The KiWi Platform, View Mode (Screenshot)

7.3.2 Project Management Application

According to design idea 4, the management layer shall be supported in its codification strategy by a project or process management system (PMS). The system created for this task is the Project Management Application (PMA), a system for enterprise resource planning and process management. It covers the basic functionality that is needed for management issues of software projects. The PMA is exemplary for any possible PMS, customized for Logica's needs. The analysis showed that different systems are being utilized in different projects. Therefore, the PMA was developed as a standalone web application.

A project manager uses the PMA to store data about a project. To do that, the application allows users to view, insert and edit data of different types (figure 33). These are grouped in the main navigation, on the top row, into three different categories: Project, organization and process (see the links in the header in figure 33).

When the user then chooses one type, the system presents an overview page. It contains a list of all entities of this type, as well as a mask to filter the list. Every entity is displayed as an entry in a table, with key data and always with two buttons to view and edit. Clicking on view opens the detail screen for the chosen entity, including all relations and data fields. This view has an edit button with the same function as the one on the overview page: It leads to a screen to edit the chosen entity. Here the user can change the provided data and save it or delete the whole entity.

The entry overview page also contains a button that allows adding a new entity. The "enter new entity" mask is almost the same as the "edit entry" one. References to other entities are always managed through tabs on the bottom of the page that provides the detailed view. Here, the user can view the related entities or choose other ones.

Similarly to other PMS, the PMA is commonly accessed by the project manager and a small circle of people inside the company only. Mostly just one

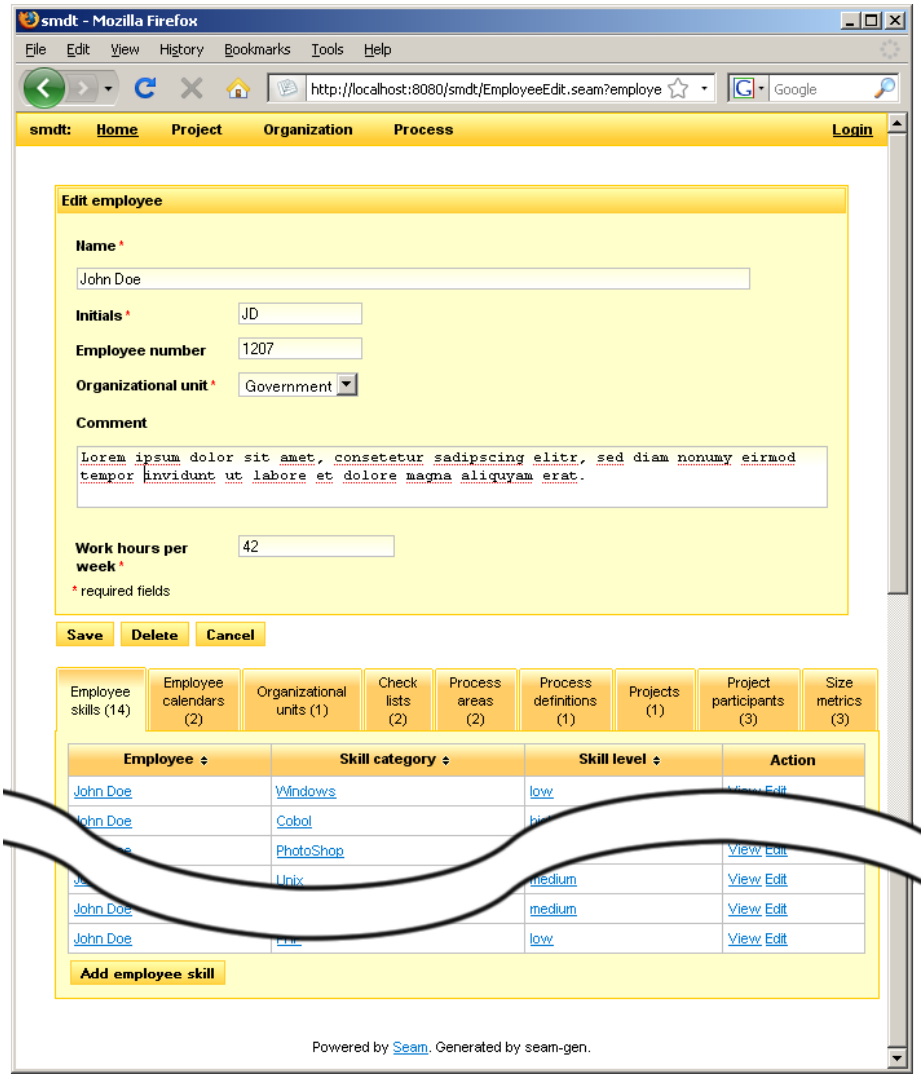


Figure 33: The Project Management Application (Screenshot)

or two persons per team have access to it, not the whole project team. The PMA's main use is the administration of planning and controlling related data as well as the communication of these to the higher management levels.

7.3.3 Shared Knowledge Model

The realization of design idea 2, i.e., the connection of KiWi and PMA, involves the synchronization between these systems. Therefore the data within these systems is important and their approach to store it.

Table: Persons

Name	Birth	Interest
John	1940	Politics
Paul	1942	Football

Figure 34: Relational Database Table

The PMA follows a more traditional approach by utilizing a relational database (figure 34). Typically for an ES, the data is stored in strictly defined tables with several fields. Every entity is represented by one row in a table. In contrary to that, KiWi has a knowledge base realized through technologies of the semantic web (see section 4.3.1). Data is thus stored in form of triplets in a triplet store (figure 35). The types described in the ontology are instantiated to objects and the data fields are connected through properties.

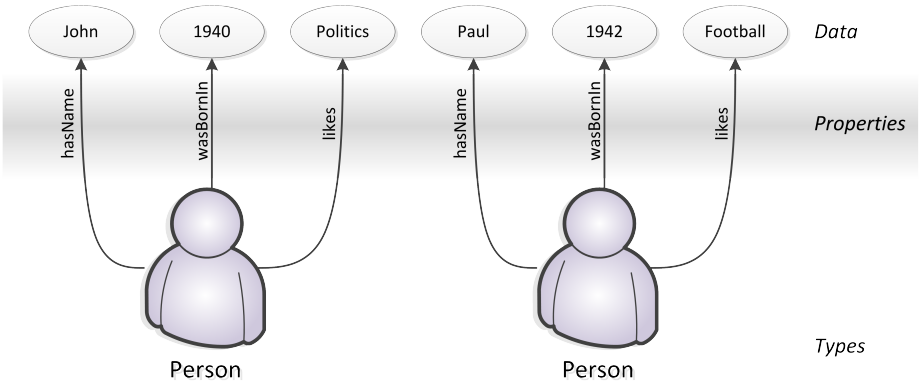


Figure 35: Semantic Web Triplets

Design idea 2 states that the combination and exchange of data is an important aspect of the system. This however becomes a complex task due to the fact, that both systems are based on different technologies. To achieve the connection of the systems a shared knowledge model was created (Dolog et al., 2009b). The idea is that the same data fields can be represented in both systems (figure 36). The PMA was created on top of it, exemplary for any other ES. Its structured internal knowledge base in form of a relational database is equal to the shared knowledge model.

Additionally, an ontology was constructed based on the shared knowledge model. This ontology can then be loaded into KiWi. The informal internal knowledge base of KiWi can thus contain data as defined through the shared knowledge model.

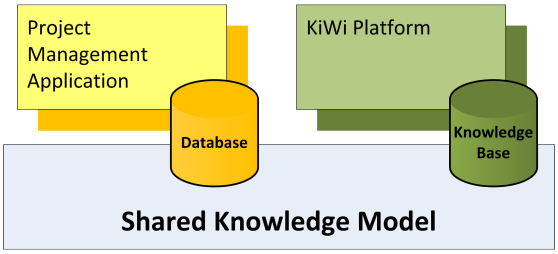


Figure 36: Shared Knowledge Model between the PMA and KiWi

The shared knowledge model covers valuable attributes and data fields of an application for project management. It is designed to support different fields: Designing processes, planning & monitoring projects, managing requirements, configuration management and quality assurance. This is achieved through a complex data structure that includes entities provided for all contingencies. The whole shared knowledge model contains more than 80 entities, accessible in both systems. Each of these entities has several data fields and relations to other entities.

The entity `ProjectPlan`, for example, has fields like a version number or text fields that contain a description of the organization, scope and included resources of a project. But it further contains a variety of relationships, like the project participants, risks and goals (figure 37).

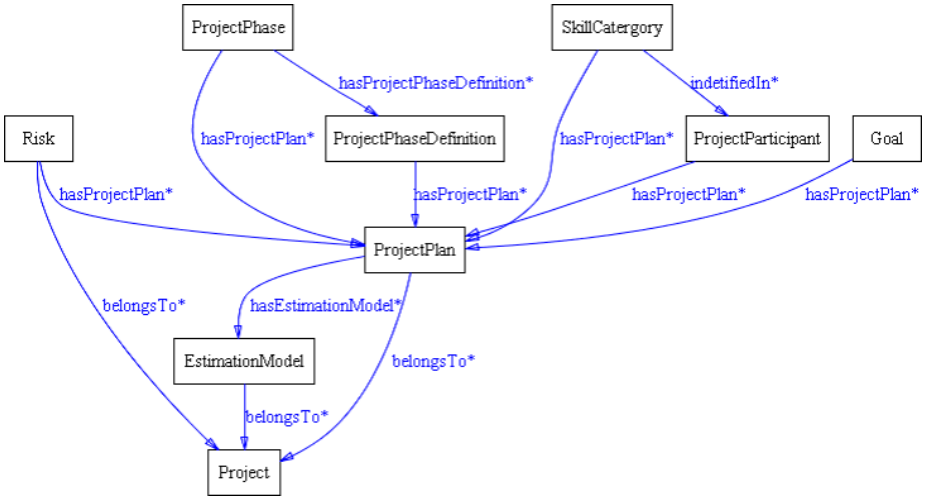


Figure 37: The Type Project Plan and its Conceptual Relationships (RDF Diagram), from (Dolog et al., 2009b)

More details regarding the knowledge model can be found in the deliverable D6.3 of the KiWi Project (Dolog et al., 2009b). Additionally, a few more details

and a link to the actual knowledge model are attached to this thesis and can be found in appendix A.

7.3.4 Data Exchange Agent

To support the exchange of data between the two systems for the management and development layer, the second design idea suggests connecting these. I showed above, that this connection supports the work of the project managers. Therefore, the Data Exchange Agent (DxA) was created. It is a middleware between the PMA and KiWi with the objective to transfer data from one system to the other. A simple interface allows the user to choose information of the project management application for publishing into the KiWi platform as well as updating previously published data. The update works in both ways, the user can decide whether to update in KiWi or the PMA.

The process of exchanging data is established through the use of the shared knowledge model. However, the DxA does not depend on specific hard coded interfaces on either of the systems.

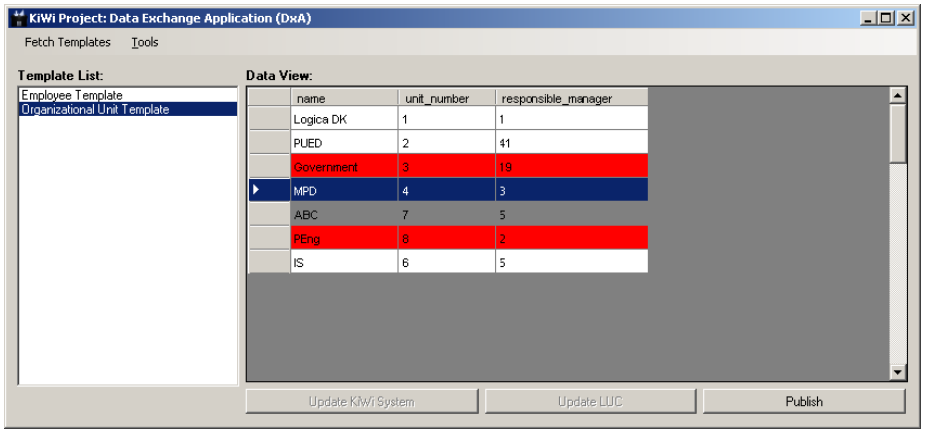


Figure 38: The Data Exchange Agent (Screenshot)

When a user starts the DxA a list of available templates is loaded, which is then shown on the left side of the application (see “Template List” on the left in figure 38). These templates represent the entities that are available for import in the KiWi platform. The user can choose one of these and the DxA displays the data sets that would be regarded by the exchange mechanisms on the right side of the application (see “Data View” on the right in figure 38).

As a next step the user picks an entity and decides to start the import process toward KiWi (the initial step always has to be the publishing of data from PMA into KiWi) or to update the data set in either, KiWi or the PMA. The system supports the user by highlighting the three possible states in background colours of the table rows:

White: New. The entity has not been published, yet, and exists in the PMA only. Only possible operation is to publish.

Grey: In synch. The entity has been published before and the data in the PMA and in KiWi are identical. No operation possible.

Red: Out of synch. The entity has been published before and at least one of the data fields are not equal. Publishing operations are possible, either to KiWi or the PMA.

These three states support the user to exchange the data between different systems. The only user of the DxA is the project manager, who is responsible in updating the critical information in the ES, but at the same time benefits from the collaboration in the wiki. This is the user that decides which data from the PMA is published to the KiWi platform. It is also the user that decides which data will be updated in the PMA based on the results in KiWi.

7.4 Technical Design

This section provides insights about technical details of the KiWi systems. The realization and technical backgrounds are explained in order to provide more depth on the design. I explain the realization of the systems, the shared knowledge model, the data exchange and the handling of the templates.

7.4.1 Templates

Moving data from the PMA to KiWi is initialized by the DxA. This process relies on templates within KiWi. Only the entities of those types can be imported, for which a corresponding template exists. The DxA then takes the template for the specific entity and substitutes the place holders by the real data.

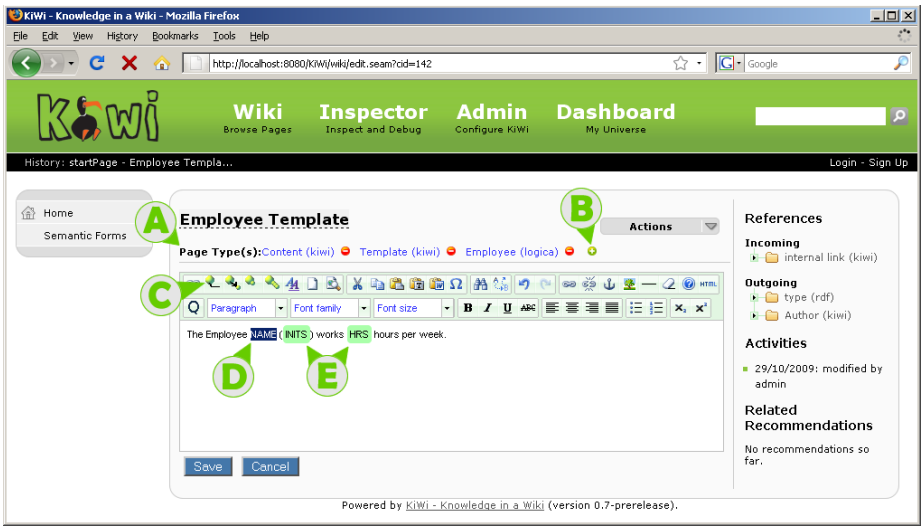


Figure 39: The KiWi platform: Editing a Template (Screenshot)

A template is basically a regular KiWi page, which is marked as a template. This is realized through page types (A in figure 39 or zoomed in in figure 40). These page types assign a type or class from the accessible ontologies to a KiWi page (add a type by clicking the add button, B). In this example a template is created, whose instances should hold the data of employees. Hence, the types of

this page are `Template` from the KiWi ontology and `Employee` from the Logica ontology. The page also has the type `Content` assigned, which is the case for any KiWi page.

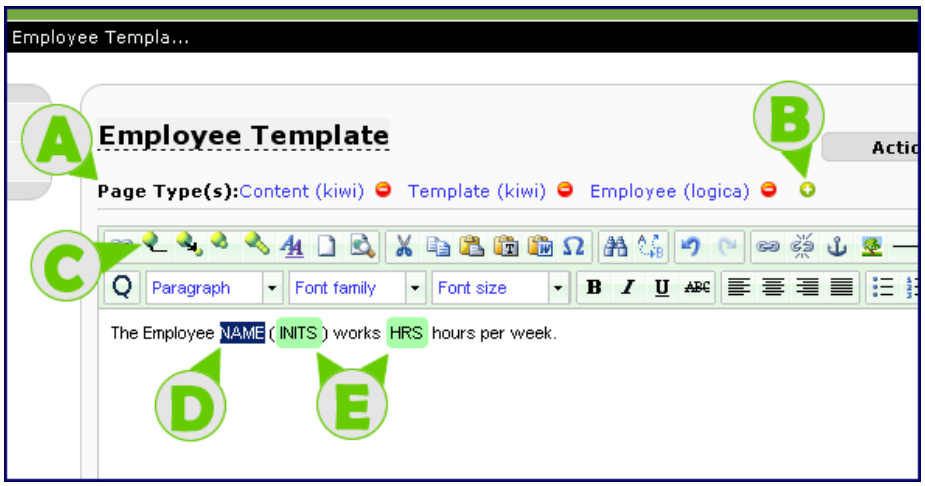


Figure 40: The KiWi platform: Editing a Template, Detailed View (Screenshot)

The text editor of a KiWi page then provides tools for wiring text parts to the properties that are part of the chosen classes. To do so a piece of text has to be written and selected (D) before choosing the fitting tool to assign a specific property of the ontologies classes (C). As a result the editor will highlight the text parts within the editor (E). After editing the text and assigning parts of it to the properties as intended, the page can be saved just like any other page.

When viewing it, the page looks like any other regular page with plain text. The data assigned to properties however can be edited directly through a meta data editor. Further, the assignment of the type `Template` allows the DxA to identify this page as a template and to use it to publish data.

Note that the page keeps the assignment of properties after using a template to publish data. This makes it possible to update the data later and use the other features of KiWi that utilize the semantic data. Further, all instances of a template are linked to it through internal references, which improves the browsing.

7.4.2 Data Exchange

The shared knowledge model makes sure, that data can exist in the two systems. However, to exchange the data from one system to the other it has to be connected. Whether the entity that describes an employee is actually called “Employee”, “Worker” or “Person” is irrelevant. Important is only that it has to exist in both systems, in order to move an employee entity from one system to the other.

I explain above, that the exchange of data between the two systems is handled by the DxA. It is also the DxA that is responsible for connecting the entities in the two systems. This is realized through a matching table. The matching table equates the entities from the PMA to those in KiWi. Hence, it contains

the information that, for example, the field `ID` of a `Worker` table translates to the property `WorkerNo` of an `Employee` type. This information is stored using a simple XML format, which makes it simple to maintain.

The matching table therefore sorts out naming differences between the entities, but also allows a different data structure. It is important that the required data exists, the containing entity is mostly irrelevant. This allows to utilize systems that do not follow the created knowledge model precisely. An ES could be connected to KiWi, even if the internal data structure differs slightly.

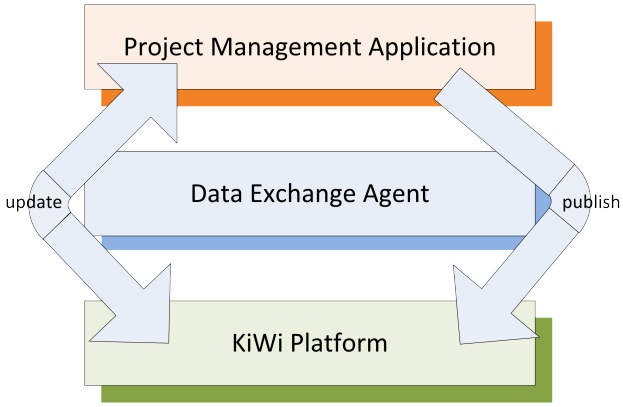


Figure 41: Directions of Data Exchange among the KiWi Systems

The data exchange can then follow different routes (figure 41): There is the initial publishing, which is always from the PMA to KiWi, and the updating, which can go both directions. Following are some details about these.

Initial Publishing

The initial step is always to publish data from the PMA to KiWi. Data sets, as defined in the templates that are not yet published are easily identifiable in the DxA. Once a user chose to publish one, the DxA starts the publishing process. First, the DxA sends a request to KiWi, asking for an instance of the desired template type and provides a name for the page to be created and the data that should be published on it. All pages' names are the template's type plus the ID of the PMA entity.

Then, KiWi searches for the requested template and creates a copy of it, with the name that was provided by the DxA. At this point of the transaction a plain copy of the template exists in KiWi, with a name defined by the DxA. If this transaction is successfully completed, KiWi inserts the data. This data insertion uses the exact same service as during the update from PMA to KiWi (see below).

When the DxA packed the data to be published, it was send in key value pairs. The DxA knows from the matching table how a data field from the relation database is called in the ontology. It can thus create the triplets in RDF, based on the choice of data sets to be published. KiWi's update service receives this RDF and a page's name. The latter makes it possible to identify the target page. KiWi then simply substitutes the available triples with the new ones. If this process finished successfully, KiWi sends a response to the DxA,

which again prompts a message to the user. The user is thus notified about the success and can open the created KiWi page with a simple click on a button within the message window.

Update from PMA to KiWi

Previously published data sets provide the possibility to be updated, if they are not synchronized. The DxA highlights these in red. If the user marks one of them and clicks the "Update in KiWi" button, the DxA collects the data from the PMA and creates an RDF with triplets based on it. As the pages names follow a strict convention, the DxA can send a request to KiWi with the RDF to update the data of this specific page.

KiWi identifies the page and substitutes its data with the data that was sent. It follows the principle of a complete flush here. The old data is deleted and the new data is stored, instead of just overwriting the data. The difference is that data that was set, but empty in the update, is deleted. This supports the users that triggered the process, as they can be sure that only those data sets find their way to the KiWi page, that were chosen. No page can contain legacy data. If the users prefer that, they have to manually roll back parts of the data, as all data changes are stored on a page's history in KiWi.

Once the update was successful KiWi sends a response to the DxA. The user then receives the note in a window, which also contains a button, to open the updated page.

Update from KiWi to PMA

The update mechanism works in both ways. The data can be updated in KiWi as well as in the PMA. The DxA highlights data sets, which are not synchronized. The user can mark these and click the "Update in PMA" button, in order to use the data in KiWi and send it to the PMA. After that command the DxA accesses an interface of KiWi to access the RDF triplets, which belong to the selected entity.

Every entity in KiWi is stored using a convention based on the primary key fields of the data model. The DxA can therefore find the related entity without storing a relation of the two systems. The DxA accesses the RDF data, and inserts it into the database of the PMA.

The data substitution does not follow a complete flush in this direction. A data field that is not set in KiWi will not delete existing data in the PMA. The reason for this behaviour lies in the open world assumption of RDF in general. As explained above (see section 4.3.1), RDF is not expected to be complete. Not existing data does not equal an empty data set, it just shows that the information for this field is not available.

After successfully inserting the data into the PMA, the DXA prompts that in a message to the user.

7.5 Workflow Design

The combination of data from two different knowledge management systems enables various ways to deal with, i.e., the data can 'travel' different routes.

Part of the design of the knowledge management system is the design of its workflow. It is called the *knowledge loop* (figure 42).

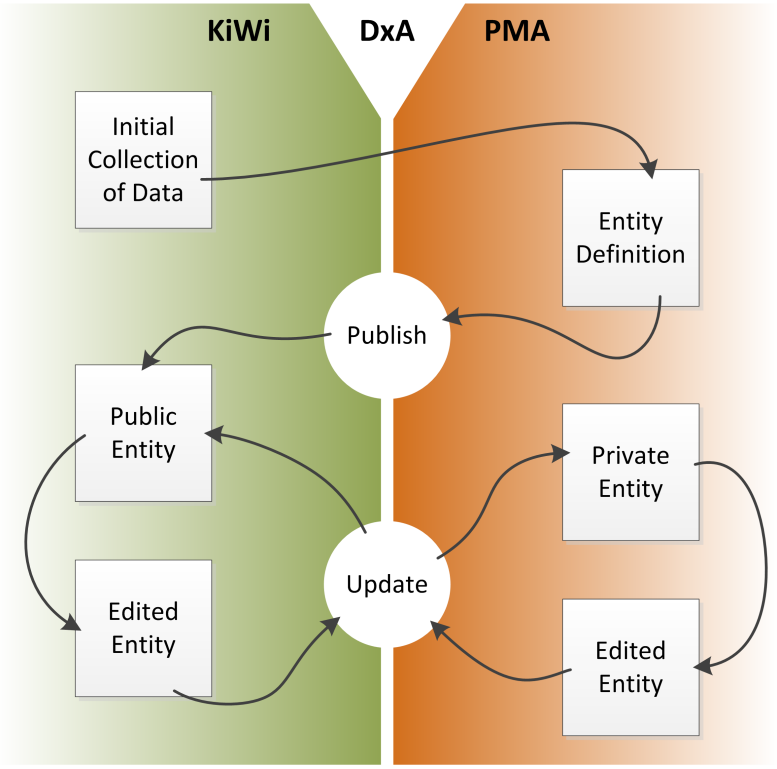


Figure 42: The Route of Information in the KiWi Systems

A very first step of the knowledge loop is the collection of data regarding a specific topic, task or interest. It is advantageous to utilize the wiki functionality here to benefit from collaborative features. This improves the quality of the collected data and gives therefore a higher level to start from.

In the next step, the collected data is then taken to create the entity of interest within the project management application. It can be based on the content provided by the collaboration in the wiki. Once this entity reaches a certain quality, it can be published to KiWi and therefore made available to the authorized people within the company. With this publication, the knowledge loop is entered.

The knowledge loop describes the constant transition of information within and between the two subsystems. When an entity is published from the PMA to KiWi, the connection is never lost. A public accessible entity is barely static. Employees can edit the page or leave comments. The content is then different from the one in the PMA.

Something similar can happen within the PMA; the limited access to smaller circles of people does not mean that the content cannot change. In contrary, as experts are operating on the system, edits are likely to appear. The result however would be the same; the PMA then contains different data than KiWi.

Both aspects can also happen in parallel, i.e., the entities both in KiWi and the PMA change during the same time frame. And though the effect seems more complex than in both cases before, the consequence would not differ: KiWi and the PMA represent different data for the same entity.

The KiWi systems provide functionality to synchronize the two systems again. To do so, an update mechanism can be started. It then takes the data of one system and inserts them to the other. This process is manually triggered, so a responsible manager compares both sources and decides which should mark the new state.

In databases the term "dirty data" is used in comparable cases of data integrity. Data is dirty, when it is not correct or out-dated. Correcting it can therefore also be called "cleaning". In this analogy, the updating mechanism represents the cleaning of the dirty data of an entity within the KiWi systems.

Note that the terms publishing and public in this context are not meant as being made available to the public in general. Instead, it refers to the targeted audience of authorized people within the company. Security issues and rights management however, are not considered in my studies.

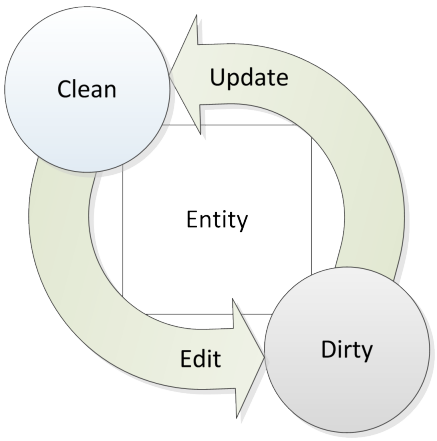


Figure 43: Synchronization Status Circle within the Knowledge Loop

This knowledge loop is the connection of two circles that meet at the point where the data in one of the two systems is being updated with the data from the other one (figure 43). In principle, it is even irrelevant whether a system provides the data for the update or receives it, the important aspect is that both systems contain the similar data again.

The workflow is based on the descriptions in the deliverable document regarding the first prototype (Dolog et al., 2009c). To make it easier to grasp, I explain this process by using an example in the following subsections. I describe the work of George, a process designer. He designs processes and relies on feedback of those that execute them.

7.5.1 Initial Data Collection

George was assigned to design a new auditing process. To do that, he accesses the KiWi platform and creates a new page (figure 44). This page has the goal

to collect information about this process. Another goal is to allow other people to influence the process design.

George therefore writes a brief introduction. He explains the goals of the page, that he appreciates his colleagues providing feedback of any sort and that this should be done within a certain time frame. He then outlines a rough draft of the audit process with the data that he has so far.

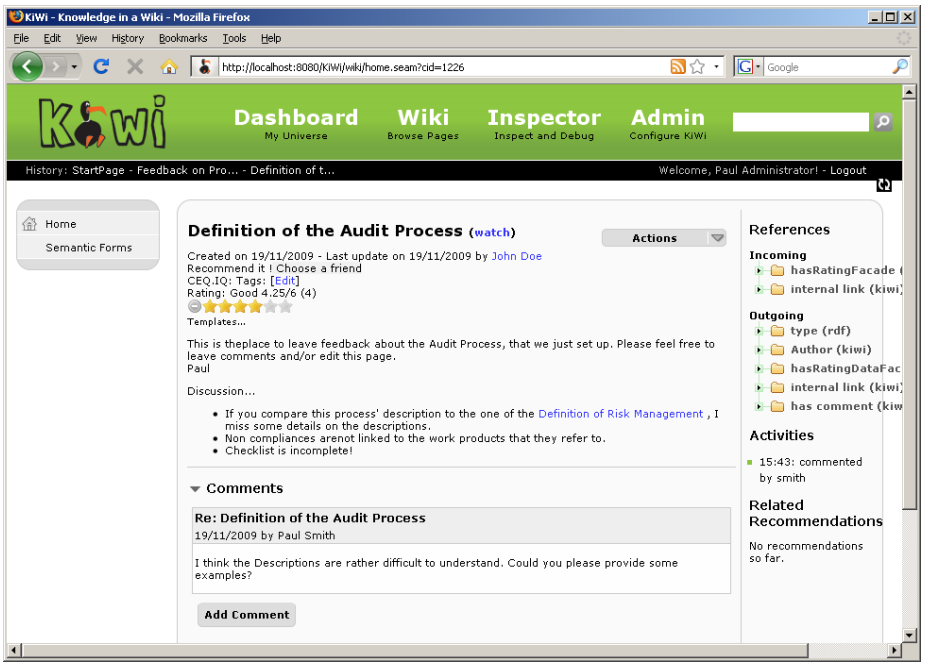


Figure 44: KiWi Page for Discussions (Screenshot)

Only when the page is rather filled with relevant content, George invites colleagues to the page and asks them to contribute or provide feedback. The initial page provides already a sufficient understanding to the other readers, so that their input can be beneficial.

The invited colleagues then edit the page or leave comments. They provide further information or links to additional sources. Also, there could be positive and negative criticism. George is happy about all different forms of contributions and gratefully accepts the feedback.

7.5.2 Entity Definition

After a certain time frame, George analyses the page that he created to discuss the new audit process. He reads all the provided feedback carefully and even contacts some of the authors directly. Further, he follows links, examines the input and revisions of the page that contains his previous outline for the process.

Based on this newly gained understanding he accesses the PMA and creates a new entity of the type “process”, which he then defines as the audit process and provides all the necessary information (figure 45).

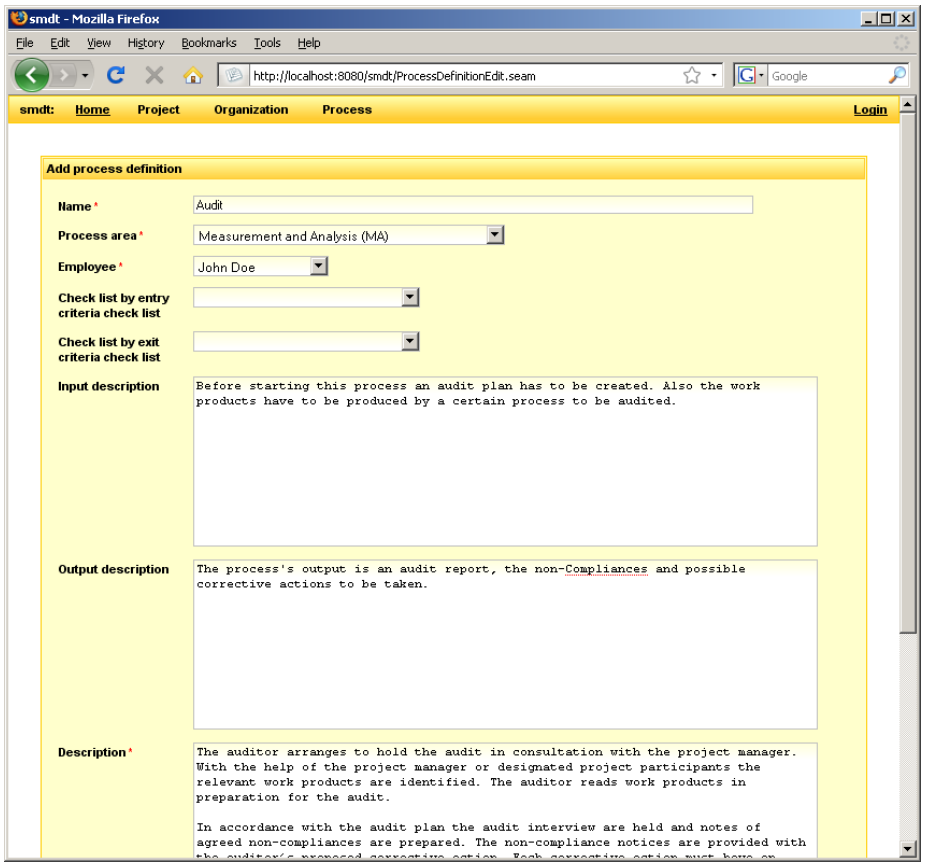


Figure 45: Definition of the Process Entity (Screenshot)

George creates this process based on his own expertise, influenced by the feedback of those that are supposed to follow the process in the future. He is the expert in process design and is aware of associations that others might not be. Therefore, he does not simply blend or compile the results of the discussion page into a process definition, but uses it as additional input.

7.5.3 Publish

After George finished the work at the process definition, he publishes it to the KiWi platform. To do that, he opens the DxA, chooses the specific process entity from the list and clicks on the "publish" button (figure 46). The DxA then starts the publishing mechanism and creates a KiWi page with the information that George provided in the PMA before.

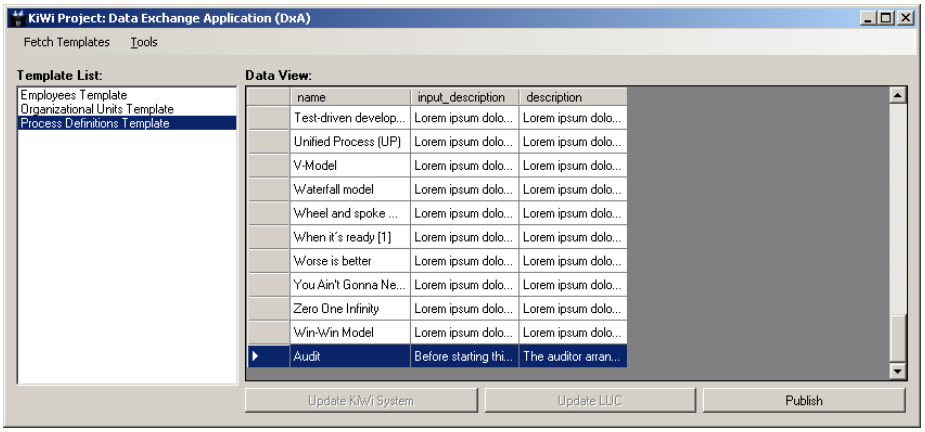


Figure 46: Publishing the First Draft of the Process to the KiWi Platform (Screenshot)

7.5.4 Entity Editing

The published audit process is then accessible to George's colleagues through the KiWi platform (figure 47). He refers to it from the previously created discussion page and again asks for feedback on this draft at any time.

George starts monitoring this page, like all the processes he is responsible for. He then receives notifications whenever people edit or comment on a page he monitors. This helps him to communicate with the people that actually apply the process descriptions he created.

7.5.5 Update

The plain use of the process description then might induce people to provide feedback. However, George might also have to ask users of the process description to do so. In any case, the comments or edits on the process description page in KiWi will eventually be made. Additionally, George might see the need for changes of the process description, which he fulfils into the PMA. Either way, the systems run out of sync.

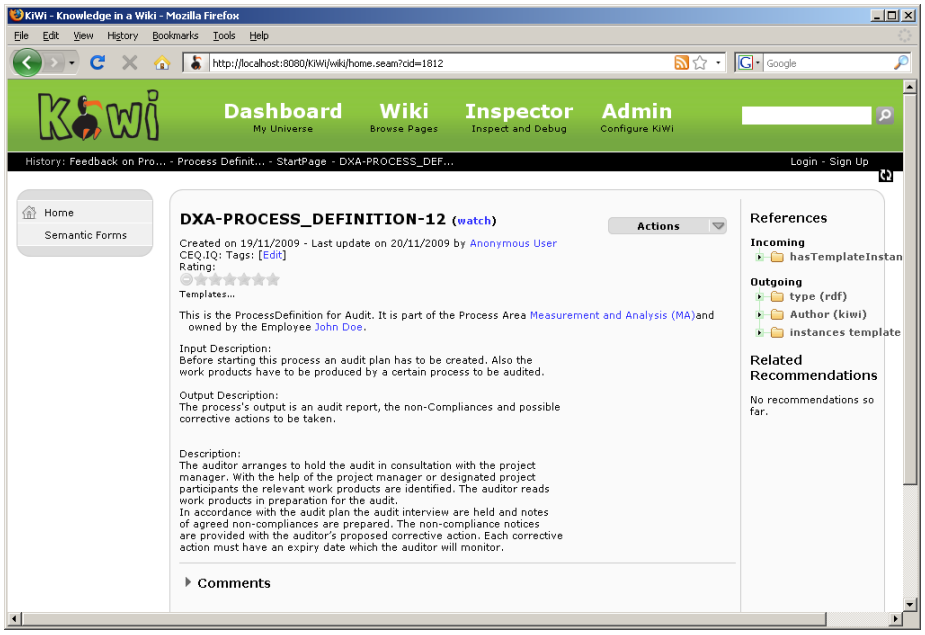


Figure 47: Published Process Definition (Screenshot)

In order to synchronize them again, George has to enter the DxA again. The system visualizes which data sets have changed, so that it is easier for him to find the differences. He then has to compare both systems and make a decision, which dataset should be used. According to his choice George tells the DxA by clicking the specific button to update either the PMA or KiWi. Once the DxA finishes that operation both systems are synchronized and again contain the same data.

8

Intervention & Evaluation

In the previous chapter, I describe the design of the KiWi systems (chapter 7), which are meant to approach the knowledge management problems of Logica (see section 6.5). In this chapter the remaining parts of the BIE circle follow (see section 5.1.3), the intervention and evaluation.

I begin with explaining how the intervention and evaluation worked during my studies (section 8.1). Afterwards, I report on the results of the final evaluation of the KiWi systems (section 8.2).

8.1 Consecutive Intervention & Evaluation

The second stage of an ADR project is basically a constant iteration of the three tasks building, intervention and evaluation, in BIE circles (Sein et al., 2011, p. 42). I explain these circles, the whole method and how I applied it above (see section 5.3). In the previous chapter I describe the building aspect as if it was a linear rational design process (see the introduction of chapter 7). Nevertheless, the building was part of the BIE circles. In fact, there were many short iterations of the BIE circle throughout the whole KiWi project.

Mostly, the intervention and the evaluation were not separated, but actually mixed. This is mainly due to the fact that the created systems were never applied in Logica. Instead prototypes were evaluated by experts with large experience as practitioners.

It was a design decision of the whole KiWi project (see section 4.1) that the prototypes of the KiWi systems were designed, shaped and discussed in close collaboration with personnel of Logica. The KiWi project's participants from

Logica were able to reflect on the project's progress as representatives of the target group. Each of them is experienced in different management activities and has detailed knowledge about the processes of Logica. Their gathered expertise made them to valuable opponents for ideas. Additionally, most of the insights about Logica were gained through their knowledge and experiences.

The collaboration with these highly skilled people during the project work was regarded as the intervention at Logica. These people evaluated the prototypes. The intervention and evaluation became equal aspects of the collaboration, which are difficult to separate.

The BIE circles were executed in visionary, but short iterations. To maintain the flexibility, the participants agreed on a general timeframe and decided on meetings with detailed agendas during the process. The timeframe was defined by the due date of project deliverables. As I state above, the project organization made the creation of these reports obligatory, which were then reviewed by outsiders (see deliverables in section 4.1.1).

The project work included a small circle of people from Logica and Aalborg University mostly; the two parties directly involved into the Logica business case (see section 4.2). Each participant contributed with their expertise. Ideas were discussed and decisions were taken in combination, researchers and practitioners. The other parties of the KiWi project were involved only when it became necessary for the progress or it made sense, due to common goals. However, most of the work was isolated from the others.

The process of work in the Logica case differed. Mainly, the work was organized straight forward. Every meeting had basically three points on its agenda. First, the progress since the previous meeting has to be summed up. This involves the presentations of the work results of the participating people, but also an update of the remaining parties of the KiWi project. Second, the current state is being discussed. This is often fluidly connected to the presentation of the work results. And it often leads directly to the final part of the meeting. Third, the next steps have to be planned. Responsibilities and tasks are being assigned and the next meeting scheduled. The detailed agenda for each meeting was defined by the tasks for the participants.

However, there were many exceptions to this standard procedure, two of which I describe briefly: Workshops and the intense work duet. First, during a period, which led to a detailed requirement analysis, several workshops were planned. All meetings were scheduled up front, with an agenda and tasks for the participants. The results of these tasks were then presented and evaluated in the meetings. Second, later in the process I spend a couple of consecutive days in the office of a KiWi project participant, who is employed by Logica. Together we discussed possible processes and realizations of the KiWi systems. This lead to the knowledge loop (see section 7.5).

The systems and the project work were not only evaluated within the Logica case. The remaining researchers and practitioners of the KiWi project were also able to provide feedback. Several times every year the majority of the KiWi project participants met to present the current results to each other.

The target of the project work changed according to the project phase (see section 4.2) and the BIE circles within. The outcome of an iteration of the BIE circles was not always a new version of the software. Sometimes the circle was all about gaining a shared and detailed understanding between the participating parties. This was the case for the whole requirements analysis, for instance. In

these cases, the result was often presented in one form of report, like the project deliverables or internal manuscripts that document an agreement.

Also, sometimes concepts were discussed in a similar way or in combination with a piece of software that acts like a prototype. This software then has only the specific task to study a concept and its possibility to be realized. The resulting software was then not more than a simple test system and a basis for discussion.

Discussions of concepts in short iterations helped to be more focused on the main goals and to be more efficient. When ideas proved to be not good in early stages and could be discarded easily. Short iterations helped to lower the amount of detours in the system design. Sometimes the proof of concept showed to be pointless, but led to other ideas.

The people from Logica had a clear understanding of the internal knowledge management problems. By explaining and discussing it, the involved people gained a mutual understanding of the situation. Everyone was able to come up with new ideas. The people from Logica were then able to provide feedback to these ideas. Often discussions arose from the ideas and feedback to them. In many cases this led to other ideas.

The interventions and evaluation during the consecutive iterations of the BIE circles can hardly be separated. A combination of both is the result of close cooperation between scholars and practitioners in the Logica case of the KiWi project. In many meetings the different stages were discussed by all participating parties, which led to the evaluation of the prototype. This was constant shaping in the sense of Sein et al. (2011); the approach follows therefore the idea of ADR.

8.2 Final Evaluation

The evaluation of the IT artefact, and therefore of the prototype, was an ongoing process throughout the development (see section 5.1.3). In the final phase of the KiWi project (see section 4.2.1) user tests were conducted, in order to evaluate the entire systems. This section focuses on these user tests and the whole final evaluation of the KiWi systems, i.e., the KiWi platform (KiWi), the project management application (PMA) and the data exchange agent (DxA).

Note that the KiWi systems are a prototype, the intention was to evaluate usefulness and feasibility, not to finalize a product.

Large parts of this section are reflections of what was reported in the KiWi project deliverables D7.2 (Grolin et al., 2010a) and D7.4 (Grolin et al., 2010b).

8.2.1 Organization

The evaluation phase was partitioned into three evaluation iterations. Each iteration had an objective or a theme, which was agreed upon at the beginning of the iteration. Also, every iteration followed the same schedule: The preparation of the system, the user tests and the creation of a report. In the beginning, the test machine had to be optimized and equipped with test data according to the objective. This also provides the opportunity to update the systems based on the feedback of previous test sessions. The user tests were held separately.

Details about this are described below (section 8.2.2). At the end of an iteration a report was created, which summarizes the test results.

The reasons for the organization in evaluation iterations were the identified risks. In industrial projects as well as in design research projects a variety of risks and uncertainty can have a negative impact on the progress (Carney and Wallnau, 1998). A reason for this is that the people are often spread on different locations (Persson et al., 2009). Both parties try to achieve results of high quality and therefore risk management becomes an important task. It helps avoiding the negative impact due to the risks and uncertainties (Baskerville et al., 2008). Addressing possible problems increases the likelyhood to finish the evaluation successfully and with an outcome of a certain quality.

One identified risk was the stability of the software. At the time of the user tests, the KiWi platform was under on-going development by several other project participants. That made it difficult to get a stable running system with all necessary features.

The software also had performance problems. It was not sufficiently stable to be run in a productive environment inside Logica and needed high amounts of maintenance. Therefore a test within the company and larger number of users became impossible. Instead, the decision was taken to have the tests in a usability lab using real-world settings. The scheduling of dedicated user tests allowed a more detailed and thoroughly preparation of the test computer.

Iterations

Three iterations were scheduled. In the beginning of the evaluation phase a plan was created, with respect to the different time schedules of the involved people. It contained three evaluation iterations. Although the iterations were scheduled, they were not planned through in detail. The objective of each iteration was to be set in a kick-off meeting at the beginning of an iteration. Here, the participants reflect on the current system and the previous results to come to a decision. The objective describes the goals of the user tests, it set a focus on certain features to be evaluated.

According to the objective the KiWi systems were prepared. The preparation had many different aspects. First, the systems were improved. Feedback from a previous iteration could be implemented into the KiWi systems. This allowed evaluation of an evolving system, the test users were able to see how their suggestions work in the system. However, this improvement could be time consuming and was therefore limited to smaller aspects of the feedback. Second, a stable version of the KiWi systems had to be prepared. As the software was not always stable, it had to be made sure that the necessary parts of the KiWi systems for the objective were working without major problems. Third, test data had to be included. Before the user tests were able to begin, the system needed to be filled with data, in order to maintain a real-world scenario. Empty systems are not realistic. Further, systems that are based on data collections like the KiWi systems must have a certain level of data included.

Once the systems were prepared, the test users were invited and the user tests conducted. As this is the major part of each evaluation iteration, I describe it in detail below (section 8.2.2).

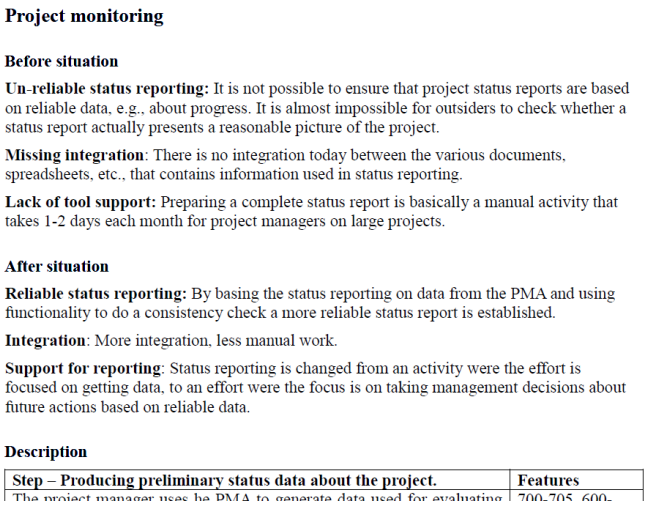
Subsequently, when the user tests were held, I created a report summarizing the organization and the results of the iteration. The reports were agreed upon

by all members of the Logica case. The reports included suggestions or input for the following iteration. After completion of the final evaluation iteration a KiWi deliverable was created, mostly based on the evaluation reports (Grolin et al., 2010b).

Use Cases

The decision on possible objectives for an iteration was mainly based on a pool of use cases. Each of these focuses on different aspects of the KiWi systems. The term use case is borrowed from the literature on requirement engineering (Cockburn, 2000) and describes a general pattern of interaction between a system and its users in the application domain (Mathiassen et al., 2000).

Each use case was carefully designed by the Logica business case. The participants took care to create storylines that reflect realistic procedures of such systems, if they were implemented in Logica. All of them cover aspects of the daily work of project managers or process designers in Logica.



Project monitoring

Before situation

Un-reliable status reporting: It is not possible to ensure that project status reports are based on reliable data, e.g., about progress. It is almost impossible for outsiders to check whether a status report actually presents a reasonable picture of the project.

Missing integration: There is no integration today between the various documents, spreadsheets, etc., that contains information used in status reporting.

Lack of tool support: Preparing a complete status report is basically a manual activity that takes 1-2 days each month for project managers on large projects.

After situation

Reliable status reporting: By basing the status reporting on data from the PMA and using functionality to do a consistency check a more reliable status report is established.

Integration: More integration, less manual work.

Support for reporting: Status reporting is changed from an activity were the effort is focused on getting data, to an effort were the focus is on taking management decisions about future actions based on reliable data.

Description

Step – Producing preliminary status data about the project.	Features
The project manager uses he PMA to generate data used for evaluating	700-705, 600-

Figure 48: Snippet of a Use Case for the User Tests

During the planning and organization of the evaluation iteration four use cases were designed. They deal with project planning, project monitoring, project work and process design. During the process of the evaluation a gap was noticed, the evaluation of information access was not sufficiently covered by the use cases. To make up for that, a fifth use case was created, which only focuses on the access of information.

All five of these use cases can be found in the appendix of this thesis (see appendix C).

The preparation of the use cases also included a mapping of the features of the KiWi systems to the use cases that work with it. In order to do that, a list of features was created. Every developer of the whole KiWi project provided input to the list. The features were ordered according to its affiliation: The core technology or one of the enabling technologies of the KiWi platform. Also, the features of the remaining systems (i.e., DxA and PMA) were regarded.

The full feature list contains 55 features and can also be found in the appendix of this thesis (see appendix B).

The use case then contained three different parts (figure 48): The before picture, the after picture and the description of the tasks. The before picture describes the situation in Logica as it used to be, without the KiWi systems. The after picture describes how the KiWi systems improve the situation of Logica and what aspects are beneficial in the context of the use case.

Step - Analyzing the scope of the project.	**Features**[1]
Preparing: The primary input to the planning process is the requirements and the products that have to be developed. The requirements specification as well as the contract is studied in great detail by the project manager, the business analyst and a technical architect Requirements are entered into the system. Products are identified and a product-breakdown-structure (e.g. sub system a, b, c…) is created. For each product a responsible project participant is chosen.	700 - 709
Publishing: The product-breakdown structure with information about all the products as well as the related requirements are published in the KiWi system.	600 - 603

Figure 49: Snippet of a Use Case Description for the User Tests

The next part is a description of the tasks to follow the use case (figure 49). The description uses an imaginary example, which is explained in detail. It is written in a way that users are able to follow the separate work steps easily. Each of the tasks shows the involved features as well.

For a better understanding I provide two snippets of use cases here (see figures 48 and 49), find the full use cases and the feature list in the appendix (see appendices B and C).

To provide an idea to the reader what the use case looks like, this is a summary of the project planning use case (the full one can be found in the appendix, section C.1):

- **Before situation:** The planning tools and the tools to store the business model are not integrated.
- **After situation:** The KiWi systems provide full integration.
- **Description:**
  - The project manager enters the basic information about the requirements of product into the PMA.
  - The project manager publishes the data to KiWi, using the DxA.
  - The project team can provide input to the requirements by giving feedback.
  - The project manager reviews the data and edits it according to the feedback in KiWi.
  - The project manager updates the data in the PMA with that from KiWi.

The idea behind the use cases is to document a designed work flow with the KiWi system in Logica's environment. Therefore they were used for the evaluation. The use cases were not only approved by Logica employees as being realistic, they were in large parts designed by them. This provides the necessary real-world scenario for user tests in a lab.

Test Data

The use cases describe scenarios of tasks in the management level of Logica. However, the realization of these tasks requires data. To maintain the real-world settings it was a crucial aspect of the evaluation to also have realistic test data.

This turned out to be difficult to achieve, because original data of Logica would have to be approved by the board of Logica. That was mostly not possible, for confidentiality reasons, i.e., non-disclosure agreements with customers. In order to still have realistic data, artificial test data was created in cooperation of Logica employees and me. They were able to design the test data similarly to the data that can be found in Logica.

The only exception was possible for the final evaluation iteration. Use case descriptions for a real developing project of Logica were provided in a text document. I created KiWi pages based on this document.

The structure of these pages was similar to the document. I, as a developer, have created wiki pages similarly in the past. This is a subjective perspective, but as the use case dealt with data access, it is important that the test users do not know all details about the realization in Logica. Also, every developer can organize contents in KiWi differently. Hence, a structure, which the test persons are not familiar with, made the test even more realistic.

8.2.2 User Test Setting

The user tests were conducted in a usability lab (figure 50). Prior to every user test session a dedicated computer was prepared with access to the KiWi systems, which were equipped with test data.

Figure 50: The Usability Lab

For each iteration two test users took part of the evaluation, they had to follow a use case and were interviewed about their impressions and opinions afterwards. Both, the user test and the interview happened in the lab.

In this thesis I call the two test persons John and Paul. Both are very experienced employees in Logica and were therefore able to provide valuable critique on the KiWi systems. Paul is a process designer and John is a project manager. As both were part of the KiWi project from the beginning, they had a good understanding of the objectives and goals of the project, which are

reflected in the user tests. Also, due to their experience in the project, they had a rather good understanding of how to operate the KiWi systems.

The history of the test users in the KiWi project made the preparation for the user tests easier, because no new people had to be trained. Additionally, John was deeply involved into the development of the knowledge model and the PMA. The test persons are to certain extent familiar with the systems under test. That kept the focus for the user tests on the usefulness and not on the usability.

Each user test session was recorded in video and audio. The recording covered the whole user test session, including the interviews after the user tests. A camera filmed the test users and the test moderator during their interactions with each other and the KiWi systems. But not only the involved people were recorded, also the screen as the test person sees it was. The recording contains therefore a screen-in-screen image of the people involved in the user test session and the screen image. Further, a microphone recorded the entire conversation in the lab.

The user tests with the test persons were held separately. Each one followed the same simple agenda:

- Introduction to the objective of the test
- Discussion of the use case
- Introduction to the systems
- Test user follows the use case
- Interview of test user

For the whole user test a test moderator joined the test person. It was the moderator who provided the introductions and who held the concluding interview. Also, the tests were guided by the moderator. He was following the progress of the test persons and supporting them, in case problems occur.

The tests had the target to investigate the usefulness of the systems, not the usability. Therefore, it was part of the moderator's obligation to help the test users in handling the systems and maintaining the focus on evaluating the concepts of the KiWi systems (figure 51 shows a brief discussion between the test user and the moderator during the user test session; note the picture of the screen in the top left corner).

For all user tests, I was the person with the moderator role.

The user tests all followed the same simple agenda from above. Note that the whole test session is videotaped. In the beginning, the moderator welcomes the test person and explains the objectives of this user test session. Both briefly discuss the topic to reach a common understanding of what shall be evaluated. Then, they have a look at the use case, which shall be followed during the user test. The discussion about the objective is being picked up and aligned to the context of the use case.

After the clarification of the user test's purpose, the moderator gives a brief introduction to the test users. As the test users have a certain experience with the KiWi systems, based on their involvement in the project work, this introduction is rather short. Often it contained not much more than an update about recent changes for the test user. Here, the moderator again points out

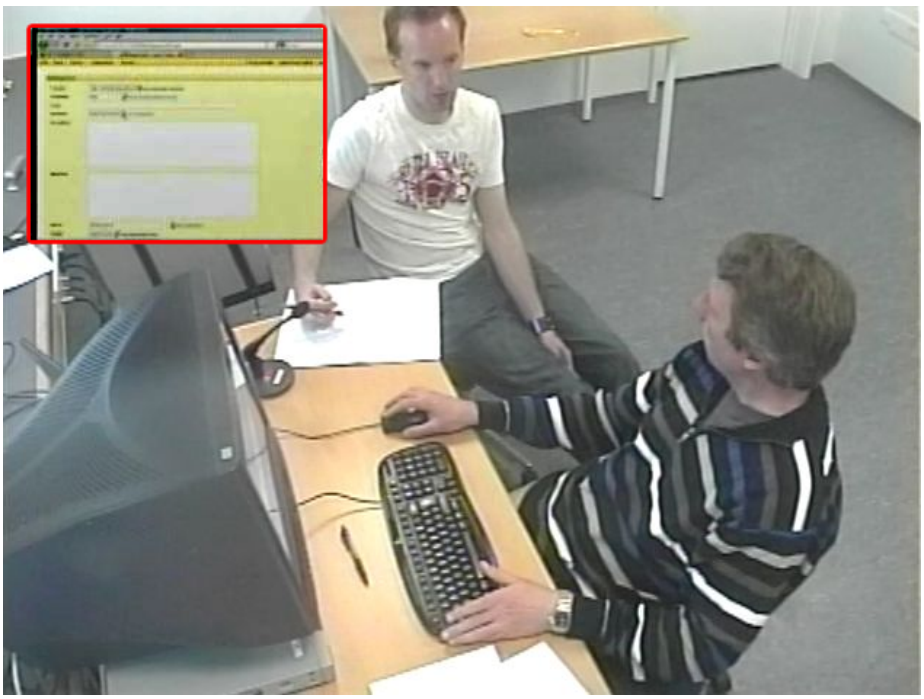


Figure 51: Test Person and Moderator during User Test Session

that the purpose of the user test is not usability tests, but to evaluate the usefulness of features and basic concepts of the KiWi systems.

Then the actual user test begins and the test user follows the use case. The moderator assists in case problems occur, but also asks the test persons to explain what they are doing. Finally, after the test user finished the whole use case, an interview is being held. The moderator prepared an interview guide in advance which helps him to investigate the user's experience and opinion on the test objective. These semi-structured interviews also provided the possibility to change the focus of the discussion according to the events that occurred during the user test or what the test person say. Then the test session is officially concluded with the moderator saying goodbye to the test person.

Later, the moderator sums the result of the whole test set (i.e., both test users) up and writes a test report. At the end of the project's evaluation phase the different reports were combined to create the project delivery (Grolin et al., 2010b).

8.2.3 Results

The previous descriptions show that the final evaluation of the KiWi systems was conducted systematically, driven by use cases. In three iterations two test users evaluated the systems' usefulness according to Logica's requirements. In the following I describe these three iterations and the findings that resulted from the user tests.

As stated above, the goal of the user tests was to evaluate the usefulness of the KiWi system, not the usability.

Iteration A: Project Planning

The first iteration focussed on project planning. The use case (see appendix C.1) makes a project manager collect information about the scope and the risks of a project, to finally estimate and schedule it. The whole process is driven by the project manager, but requires input of the project team. The underlying idea is to show that the connection of two different systems can support the collaboration within the development layer and that project managers can benefit from that.

During the tests, the test users played the role of a project manager, preparing a project. Each test person invented the data of a project on their own during the tests. The collaboration was simulated by the moderator, while the test users left the room.

The conclusion of the first iteration was rather negative. These are the findings:

- Usability of KiWi systems is bad
  - KiWi is not stable enough
  - Test users were not sufficiently trained with KiWi
- Structure of PMA is confusing
- Publishing through DxA is too bulky
  - Related entities should be published together
  - Published pages are difficult to find
- Security issues
  - DxA does not require a log-in
  - Every user of DxA can update all data
- Data changes are difficult to follow
- Collaboration is simple
  - Possibility to provide feedback is valueable

During this user test session the users were not very happy with the KiWi systems. They complained about the usability, even though this was not being evaluated. The complains had basically two reasons. First, the KiWi platform was not very stable and crashed during each user test. Even though there was hardly a loss of data, the test users were interrupted and annoyed by that.

The second reason for the test users' complains was that I, as the test moderator, over-estimated the their knowledge regarding handling of the KiWi platform. Despite being very familiar with the concepts and technologies of KiWi, they both had problems in using it. A better introduction to working with KiWi could have sorted many of these problems out. The lack of knowledge about the use of KiWi, however, amplified the usability issues of its interface. Both

reasons affected the test users' behaviour during the tests. They had difficulties to put into practice what they had in mind.

> *"The usability is so bad, that it is difficult to evaluate even the basic concepts." (Test person Paul during user tests)*

Additionally to the usability issues with KiWi, test user Paul reported that he had difficulties to fully comprehend the PMA. He was confused by the structure of the underlying knowledge model and the organization of entities within it.

Both test users have large experience with using project management systems like this, but unlike John, test user Paul was not involved into the development of the PMA. When following the use case, he was slightly puzzled during the tasks in the PMA. He had difficulties finding the specific entities, as he was not aware of the correct name or how it was organized in the navigation. This is again a problem related to the introduction to the system.

Also, he was sometimes confused by the structure of the data. He expected data as part of an entity, which is located in a related entity only. For example, he was surprised that a `ResponsiblePerson` of a `Project` does not have to be a `ProjectParticipant`. But despite his confusions, he was able to successfully finish the tasks. However, he mentioned that he would have designed the system differently.

Both test users stated that the publishing mechanism through the DxA does not work straight forward. They expressed two major points of critique. First, each entity has to be published manually. There is no functionality that automatically includes related entities. Often entities do not make sense on their own, but only in connection to others. These could then be automatically published as well. For example, an `OrganizationalUnit` should be automatically published when one of its `Employee`s is published.

Second, the test users commented that published entries are difficult to find. A published page has a title like `DXA-EMPLOYEE-3`, which is auto generated, based on the entity's type and its ID in the PMA. An elusive title like that however is not easily traceable for users. Additionally, to browse through the published pages, users have to open the templates and browse through the instances. The test users suggested an overview page of a certain kind, where content could be linked automatically. It would decrease the difficulty to find the entities. Such an overview page could act like a starting point for a project or for users that want to browse the available information. Generally, the test users had major problems finding pages they have previously published.

A broader comment was that the DxA lacks a log-in functionality. Figuratively any user could thus import and update data to and from either system. The test user also commented that every user of the DxA had full access to all data in KiWi and the PMA. The user would thus have the ability to change important data without limitations. He did not like these data security issues.

Test user Paul commented that he had problems figuring out the changes of published data. The DxA showed that a certain data set is different between KiWi and the PMA, but it does not show which field is different. The user has to go to KiWi and manually search the history of a page to find the field with data updates. After trying that once, the test user ignored this step and relied on the correctness of data. He updated the data from KiWi to PMA directly, without checking.

Despite the different problems of the KiWi systems the test users remarked that the collaboration of the project team is well supported in KiWi. Both users understood the simplicity to provide feedback to basically any KiWi page. They mentioned that this is especially useful for process related contents, where the users can comment on process descriptions.

> *"These KiWi systems are not more efficient for project planning than the current solutions in Logica. However, I can see that they [the KiWi systems] are more efficient regarding the collaboration features." (Test person Paul during user tests)*

These user tests showed that the prototype suffered from a variety of problems. The test users had problems relating to the usefulness of the systems, because of the poor usability. For the following user tests this has to be improved and/or the test users have to be better focused on evaluating the features and not their handling.

Iteration B: Process Design

The next iteration focused on process design. The use case (see appendix C.4) involves a project manager asking for feedback for a process description and refines it in two iterations. Similarly to the use case utilized in the previous evaluation iteration, this use case is driven by a project manager, but involves the feedback of other people. The idea is to show the collaboration functionalities of the KiWi systems and how this provides the possibility as well as the benefits of direct feedback between the people designing and implementing processes.

During the tests, the test users played the role of a project manager, working on a process description. Each test person invented a process description on their own during the tests. The collaboration was this time simulated by the test users themselves, to better evaluate the collaboration.

As the first user test results were rather negative, different aspects were improved for the second user tests. First, a more advanced version of the software was used. This approaches mainly the stability of the software. During the first iteration all KiWi systems underwent further development, which led to higher stability. During this iteration's user tests, the software did not crash, but was very slow for test user Paul.

Second, the test users have been trained with the software in more depth. I, as the moderator, took better care of the test users and their preparation to the use case regarding the handling of the KiWi systems. During the user tests, I was also more active to support the test users to put into practice what they have in mind. Additionally, the test users were asked in advance as well as during the test run, to focus more on the evaluation goals and avoid being distracted.

Third, different changes to the software have been made, which the test users commented on in the first iteration. For this iteration the DxA contained a button, which helps its user to jump to specific entities, for example. This helps the user to find the published content more easily.

The overall conclusion of the second iteration was more positive compared to the first one. These are the findings:

- Problems are confirmed
  - DxA is inefficient when publishing related entities
  - Confusing structure of data in PMA
- PMA lacks business logic
- KiWi has flaws
  - History does not work as intended
  - New pages can only be created through a link
  - URLs are not static
  - Reading comments when editing a page is not possible
  - Tag suggestions are not satisfactory
- Concept makes sense
  - Customizing process descriptions is valuable
  - Collaboration is easy

During these tests, the users had fewer problems than before. However, some of them remained. For example, when publishing a process definition and the necessary related entities, both test users commented that the DxA's functionality is too limited. They wished for a feature that allows the project manager to publish also related entities automatically. In the current implementation a user has to manually find all the different related entities and publish them, one after the other. The test users explained that this is very time consuming.

Another problem, which was documented in the first user test by test user Paul already, is the structure of the PMA. Again, he found it confusing to follow and had difficulties to find the entities that he needed. One of his comments was that the `TaskDefinition` and the `ProcessDefinition` look too much alike according to his taste.

The same test person further commented that the PMA lacks business logic. He investigated the system and criticised that it does not actively support the users in any tasks. Not even the validity of data fields was checked, i.e., whether the data has the right format or range.

Both test users mentioned several issues in KiWi. Test user John tried to investigate on the data changes after an update, but the history function of KiWi showed no results. On other pages this was possible, the history of a page was accessible and provided the relevant information.

Further, he missed a connection between the comments to a page and the content of it. To find out, to which version of a page the comment was given the user has to compare the time stamps manually. He valued that as a drawback.

John had difficulties to find out how to create a new page in KiWi. The only way to do so is to create a link to an already existing page. When clicking this link KiWi tells the user that the page does not exist yet, but it can be created now. The test user found this procedure strange and would prefer to have a button provided, which allows the creation of a new page.

The other test user, Paul, tried to copy the URL of a KiWi page and keeping it as a short cut for later. He argued that this could be a common scenario as

people would want to send the link to a KiWi page through e-mail to their colleagues. However, this did not work. KiWi uses dynamic URLs, which are set according to the session. Direct links to a KiWi page, so called permalinks, are provided on every page. But the test user did not find them, which shows that they appear hidden.

During the use case it is one task to work the results of the collaboration into the text of a KiWi page. This collaboration happened to large parts in the comments of the given page. While test user Paul edited the page, he criticized that he cannot see the comments while editing. It would make his task easier to see the comments underneath the editor. The current solution forced him to save the results, view the comments, remember them and edit the text again.

Another task in the use case was to give tags to a KiWi page. KiWi has a component to support the users in doing that. It was developed by one part of the enabling technologies: The information extraction. This component analyses the content of a KiWi page and shows possible tags that are likely to fit to the text. Test user Paul commented that the provided tag suggestions were not helpful at all. He provided tags manually and could not use the suggestions of the system.

Despite the critique on the systems and problems that I just described, both test users were very positive about the KiWi systems in general. They said that the concepts are clear and make sense in a given scenario as the use case.

Test user Paul pointed out that the connection of an enterprise system and a communication tool like in the KiWi systems are very valuable and do not exist, according to his knowledge. Both test users saw the strengths in the collaboration mechanisms of the KiWi platform. They were able to edit, create and comment on pages without any problems. Each stated separately that collaboration has to be as simple as this in order to work in Logica.

They can see the use for the KiWi systems especially in the communication between the process designers and process executers. John explained that the KiWi systems allow project managers to utilize process descriptions more easily. They can take an existing one and customize it according to the requirements of the project.

> *"With this system, it is actually easier to follow the rules than to violate them." (Test person John during user tests)*

These user tests showed a major improvement to the previous ones. The main achievement was that the test users could gain a better focus on the usefulness of the features, with only minor distraction of the usability. The feedback on the prototype was rather positive, most features made sense to the test persons and the systems were considered being valuable.

Iteration C: Information Access

The final iteration had the focus on the information access. The use case (see appendix C.5) includes different angles to find specific information in KiWi. Unlike those use cases utilized previously, this one is not based on a realistic scenario, but it is a list of realistic tasks instead. The idea was to show that KiWi supports users in finding and accessing relevant information. This user test session was conducted entirely using the KiWi platform only.

During the tests, the test users played the role of a project manager, working on a use case description for a project. The user does that by searching for available information and re-using it where possible.

The test data in KiWi was provided by Logica and entered by the moderator before the user tests. To enable the full strength of any recommendation, a large data set is required. The data used for this test was too limited to fully exploit all the features. However, it was enough to see the component work and to get a grasp of the concepts.

The understanding gained in the previous iterations was that the concepts of KiWi make sense and that collaboration is valuable for the project work. However, an important aspect of the features was not yet evaluated: The components for recommendation and information extraction. Provided by a part of the enabling technologies (see section 4.3.4), this set of features is meant to support KiWi users with finding and re-using information. The test procedure was therefore slightly changed from use cases to a task list that covers all features of the recommendation and information extraction.

The conclusion of the final iteration was rather positive. These are the findings:

- Usability of KiWi is poor
- Navigation in KiWi is confusing
- Tagging
  - Users miss guideline
  - Suggestions partly helpful
- Recommendation of content
  - Does not include available data
  - Is valuable for re-use

During this user test session the test users were curious and surprised by the features of KiWi. But, just as in the previous tests, they had problems with its usability. For example, test user Paul had difficulties finding the "save" button after editing a KiWi page.

Closer connected to the actual use case is the fact, that both test users had problems navigating through the recommendations and links of KiWi. There is a section on every page, where the incoming and outgoing links are listed. The test users found this part confusing and hidden.

Also, the navigation through tags felt strange to them. While a tag is a link, the users expected to reach a list of pages that are tagged with this tag. Instead the users found a KiWi page which is the representation of the tag, without any content.

The tagging of a KiWi page delivers valuable data to the system and the recommendation component builds up on that. When tagging, both test users felt insecure about how to tag a page. This is not surprising, as they are not regularly in such a situation. But it showed that users have to be trained regarding the tagging; test user Paul actually requested a guideline for tagging. The test users came up with individual solutions and assumed that homogeneous

tagging would be better. When giving tags, KiWi analyses the content of the current page and shows suggestions for tags to the user accordingly. The test users reflected on these tags and where not always happy with them. This depended on the context, sometimes the detailed tags were appreciated, and other times they were not. Especially for overview pages, the test users preferred to tag rather general, for pages that carry the data, the tagging was very specific.

When evaluating the access of information, based on recommendations and search, the test users stated that it "seems to work" and is "to some extend useful". Paul explained that this would be a big improvement compared to the current systems. However, the test users also commented, that some of the recommendations were too general, which was likely based on the small amount of data. Also, the recommended pages were not always precise hits compared to what they were looking for.

A major critique from both test users was that the KiWi systems do not take all available data into account for the recommendations. Test user John stated, that the current solution is so general and not specific enough, he explained that it could also be used for a cookbook. The domain knowledge, which is part of the system through the ontology is not used. If KiWi would do that, both test users claimed to value the usefulness even higher.

> *"The recommendation component might help to re-use existing solutions. This can make the difference between winning and losing a case." (Test person Paul during user tests)*

These user tests showed that a key aspect of the KiWi systems, the information access, is working well at large. The test users were able to understand and apply the different ways of accessing information through KiWi. Despite different drawbacks, it was considered valuable.

Limitations

The project organization had a strong impact on my work, in particular on the final evaluation phase. This has the effect that, unlike intended in action design research, the evaluation was not conducted in a fully realistic setting.

As I describe above (see section 8.2.1) a set of risks regarding the evaluation were identified, which might have hindered an evaluation. A main risk was the potential lack of stability of the software. The decision to conduct user tests in realistic lab settings instead assured relevant results nevertheless. However, this also limits the results.

The final evaluation of the KiWi systems has three major limitations. First, the software was evaluated in a lab and not in the case company's usual work environment. The tests were run on a specifically prepared computer with realistic data. Second, the evaluation was conducted in a number of short sessions and not in a longitudinal evaluation. The feedback was mainly based on domain expertise, not as much on experience with the systems over a longer time of practicing. Third, the number of people involved in the evaluation is smaller than what was intended. Two domain experts provided feedback as representatives for a larger group of project managers.

These limitations weaken the results of the evaluation. However, they are made out of necessity. The situation in the project had to be taken into consideration and evaluation results had to be achieved. It was a vital aspect of the

KiWi project to gain valuable insights about the KiWi systems from the case company. The opportunity to utilize a usability lab was therefore gratefully accepted.

This obviously impacted the findings to some degree. Because the domain experts were very experienced they provided very detailed feedback. What we do not know is how it would look with a broader group of evaluators with varying experience. The domain experts made an effort imagining the systems in the environment of the case company and in their daily business, so they can provide valuable feedback about the usefulness of the systems, but also regarding possible improvements. We also do not know how a group of evaluators with varying interest into this research or the own company's knowledge management would have responded.

Overall

The evaluation phase was considered a success for the Logica business case. However, the results were not completely positive. Generally, the user tests confirmed the concerns regarding the software. The KiWi systems showed many usability issues, which would have made it impossible to be evaluated within Logica. After these tests it became clear that the KiWi systems would have to undergo large improvements, before it reaches the state of production ready software. However, this was never the goal of the KiWi project (see section 4.3).

The iterative approach to perform such an evaluation proved to be working well. It provided the possibility to implement results during the process. Also, it showed that the systems could be adjusted according to feedback provided by the test users.

Another positive aspect of the iterative approach is that the procedure could be adjusted during the process. The participants noticed that the information access was not evaluated sufficiently and modified the process slightly. Instead of using one of the complete use cases a task list was created that includes the required actions. This increased the quality of the evaluation.

Generally, the test users expressed legitimate criticism toward all three parts of the KiWi systems. Beyond usability issues, they commented on aspects of the systems, which make regular use figuratively impossible.

The user tests were conducted with close guidance of me as a moderator. A lack of this form of support would increase the test users' problems. But despite the problems with the handling of the KiWi systems, the test users understood the functionality and considered it valuable.

From the Logica point of view, the prototype showed a variety of pros and cons. It was valued very positive by the test users, that the information access through the KiWi systems is a big improvement, compared to the situation at Logica. It is also positive, that the communication between the people within the development layer, but also between the two layers, is supported.

Further, the test users valued it positive that the processes are easier to follow with the KiWi systems. However, the test users also criticized the prototype. They strongly disliked the usability of the systems, even though it was not in the scope of the evaluation, they commented on that consistently. Negative comments were also given to the way KiWi treats the provided information. It should make more use of what is available. Additionally, one of the test users regarded the PMA as confusing.

From the KiWi project point of view, the prototype also shows pros and cons. The final evaluation of the KiWi systems raised much feedback, which can be used to improve the systems in future research. The only real negative point is the stability of the systems and their usability. This influenced the evaluation more than anticipated.

Positive however was the evaluation of the usefulness. Due to their experience as professionals, Logica's employees could map the tasks of the use case to their daily routines and see the benefits. They were convinced that a system like this would be beneficial in the evaluated settings (i.e., project planning, process design and information access), as it solves current issues at Logica.

9 Discussion

In this chapter I discuss the findings of my PhD study. I show to what extend and in which way the KiWi systems are a solution to the knowledge management problems of Logica. Further, I explain what other researchers and practitioners can learn from my studies.

To begin with, I discuss the design ideas that lead to the KiWi systems, being my contributions (section 9.1). The limitations of my research are explained afterwards (section 9.2). I conclude this chapter with suggestions for future research (section 9.3).

9.1 Contribution

The KiWi systems, whose design and evaluation was described in the previous chapters, are an answer to my research question. I formulated it in the first chapter (see section 1.3):

> ***Research Question:*** *How can IT systems support knowledge management in software development?*

Being a design study, conducted in close cooperation with a case company, the approach of my research was not to create the perfect knowledge management system. Much research shows that each knowledge management approach has to be customized for its purpose and environment (Davenport and Prusak, 1998; McDermott, 1999; Kautz and Thaysen, 2001; Rus and Lindvall, 2002; Bansler and Havn, 2001). Instead, I analysed Logica for its very specific knowledge management problems. To address these, I utilize four design ideas (see

section 7.2.3). The design of the KiWi systems is based on the design ideas. My contribution to the body of knowledge are these design ideas, how they emerge from the actual knowledge management problems, how they address them and lead to the design of a knowledge management system and how they are realized and evaluated in the KiWi systems.

In the following I explain how each of the design ideas for itself solves some of the identified problems of Logica (see section 6.5). I further describe how they relate to the literature (see chapter 2 and 3). This shows that the findings can be generalized and are not limited to the case company of my studies. I argue where I confirm or disconfirm the literature, for each design idea.

9.1.1 Design Idea 1: Multiple Strategies

The first design idea is presented in section 7.1.1. It has the focus to acknowledge that Logica consists of different layers. It further recognizes the different needs for each layer regarding the knowledge management. The idea is to support both equally.

> ***Design Idea 1:*** *Supporting the two organizational layers in Logica with different knowledge management strategies: The management layer follows a codification strategy and the development layer the personalization.*

The two layer organization of Logica is not directly related to the knowledge management problems. However, it explains their grouping. In the analysis I distinguish between two groups of problems, according to their area of concern (see section 6.2): The isolated islands of knowledge (problems A1, A2, A3 and A4) and the inadequate bridging of knowledge (problems B1, B2 and B3). This division reflects the two layers.

The *isolated islands of knowledge* relate to the development layer. The focus, of the problems grouped together, is on the project work. All of these problems are based on issues regarding the personalization. Communication of too low quality between the projects, and thus within the development layer, is the (very general) reason for all four problems. The first problem (information access, A1) recognizes that a project encapsulates the internal knowledge and makes it difficult to communicate to. The second problem (expert finding, A2) points out that the communication among the people is too low, in order to spread an understanding of personal expertise. The third problem (sharing support, A3) illustrates a general lack of support to communicate and share the knowledge with colleagues across the company. And the final problem (documentation level, A4) shows that the communication of project results is inconsistent and incomplete. Supporting the communication addresses all of these problems. This is covered by the design idea, with a focus to improve the personalization strategy in the development layer.

The *inadequate bridging of knowledge* on the other hand, relates to the management layer. This group of problems has the focus on dealing with the project work, from a more distant perspective. All of them are based on the codification issues, which basically state that the externalization and the re-use of externalized knowledge do not function as intended. The first problem (process complexity, B1) illustrates that the codification strategy is too complex. The second problem (feedback circle, B2) recognizes that communication regarding

the codification is difficult. And the final problem (connected documentation, B3) points out that the codified knowledge is not interconnected. Supporting the codification addresses all of these problems. This is covered by the design idea, with a focus to improve the codification strategy in the management layer.

Looking at the different nature of problems, which is reflected in the grouping, makes it clear, that the design idea 1 aims at a fitting solution to each of the two layers in Logica. It acknowledges the difference between the development and the management, in order to provide an own knowledge management strategy for each.

Hansen et al. (1999) explain that the knowledge management strategy should reflect a company's competitive strategy. They describe the two strategies codification and personalization (see section 3.2) and their drivers, showing that the wrong strategy is counter-productive. The design idea confirms this and applies one strategy per layer. Each layer shows specific needs, which are addressed with a fitting strategy (details in section 7.1.1).

Following the suggestion of Hansen et al. (1999), to pick the best fitting knowledge management strategy according to the existing needs, is the core of this first design idea. It acknowledges the situation of a layer, and choses the strategy that reflects the mode of operation best. The task of the system that is based on the design ideas is then to support the knowledge management strategies, which address the identified knowledge management problems.

Further, Hansen et al. (1999) emphasize that a company should focus on one strategy only. They stress that a mix is counter-productive and suggest a main strategy supported by the secondary strategy in a share of 80-20. However, my analysis shows the different needs in different parts of the company. It would be too limiting to simply choose one strategy and force the entire company to follow it. One of the layers then would have to follow a strategy that does not fit to its mode of operation.

It would be counter-productive for the development layer, if it would have to follow the codification strategy. The development layer would most likely abandon or ignore the strategy, as it can be seen in some of the identified problems. The management layer is organized with a strong focus on documents and documentation; it is figuratively incapable of following the personalization strategy. The whole company operates through a codification.

It would be contradicting to the ideas of Hansen et al. (1999) to overrule one of the layers and impose a knowledge management strategy that does not fit. Instead, the design idea follows the sense of the suggestions by the authors on a more literal way and on a very fine-grained level. Therefore I slice the company into two logical layers, where each one applies the knowledge management strategy that suits best to the dominant needs.

Alavi and Leidner (2001) explain that knowledge is being seen from different perspectives (see table 1 on page 12). This perspective on knowledge influences the knowledge management approach as well as the knowledge management system. During my studies it became clear that the perspectives cannot be separated from one another completely, but strongly depend on the context. For instance, when users need information about something specific they are driven by the condition of access to information, and hence the access to information perspective. But, once the users found the desired information, their perspective on knowledge changes to the process of applying the expertise. Additionally, in

a knowledge management system the object perspective is usually applied, as this is the common way to deal with things in IT.

All perspectives on knowledge, as pointed out by Alavi and Leidner (2001), are represented in a knowledge management approach like the one presented in my studies. However, the underlying design ideas set a focus. Especially this first design idea delineates an emphasis on the perspectives. I see it as a confirmation of the original idea regarding perspectives on knowledge, and an extension when related to the knowledge management strategies.

The predominant perspectives on knowledge are different in the two layers. Based on the approach, where the systems play an important role, both layers have a strong understanding of knowledge as an object. This is because the knowledge has to be stored somewhere. The development layer has a focus on access to information and the state of mind. This covers the availability and the retrieval of information, as well as the opportunity to learn from it. Developers have to access the necessary information and make sense out of it, in order to address their problem. The management layer on the other hand has a focus on the process perspective. This represents the attempt of distributing the available information. These different perspectives are not contradicting each other.

Further, every perspective appears in each layer. Yet, the focus is different. They are not evenly distributed, some are emphasized, others are subordinated.

9.1.2 Design Idea 2: Connecting the Layers

The second design idea is presented in section 7.1.4. Its focus is to avoid a big gap between the two layers within Logica. The idea is to connect them.

> ***Design Idea 2:*** *Connecting the two organizational layers in order to establish and support knowledge sharing between them.*

This design idea is based on the understanding that both layers run the risk to isolate themselves of one another, especially when they are separately approached, as described in design idea 1. While the first design idea acknowledges the differences of the layers, this second design idea brings them together and allows interaction between them.

Further, it has to be stressed that this design idea is not entirely represented in the development of the DxA. Though the DxA covers large parts of the connection of the systems, it is not entirely responsible for the connection of the layers. In fact, the design idea is meant in a more abstract manner, beyond what was implemented.

Design idea 2 tackles two of the identified knowledge management problems directly: The feedback circle (B2) and the connected documentation (B3). It approaches the problem B2, which states that the communication between process designers and process executers is difficult to establish (see section 6.4.2). Process designers are part of the management layer and process executers of the development layer. Hence, the problem can be seen in the way, that the communication between people from one layer to people from the other layer is difficult. This is the focus of the design idea: Increasing the communication between the layers by connecting them.

Design idea 2 approaches also problem B3, which states that the documentation, that is provided by projects, is not well connected (see section 6.4.3). Every project creates its own documentation, but this is very isolated from the other projects. It is created in the development layer, but the codification level does not match the needs of the management layer. A project's documentation remains isolated, because it cannot be connected to the documentation of other projects within the management layer. The design idea focuses on connecting the layers, so that the documentation of different projects can be interconnected. In this case the connection of the layers' knowledge management strategies is of high importance. The results from working with the personalization strategy have to be connected in a layer that is using a codification strategy.

These two problems describe different levels of connection. Problem B2 focuses on a connection of people working in different layers of the company. Problem B3 focuses on a connection of the knowledge management strategies in the two layers of the company. This illustrates the complexity of the second design idea.

Rus and Lindvall (2002) explain three important aspects of software development regarding knowledge management: Core software developing activities, the product & project memory and the learning & improvement. Only when the two layers of Logica are connected, all three aspects can be fully supported across the whole company. Two of these aspects are directly addressed by this design idea. The support of the "product and project memory" reassembles problem B3. This aspect focusses not only on the recording of the project memory (i.e., documentation), but also on its traceability. The connection of available documentation allows that.

The design idea also improves the access of information, which influences the support of "learning and improvement". The connected documentation allows users to search the history of different projects with similar backgrounds. The users can then learn from these projects. This is an important aspect of the work, especially within the management layer.

Hence, the aspects of software development, that have to be supported by a knowledge management approach according to Rus and Lindvall (2002), are not equally spread. They cover very different aspects of work within a company. It takes a larger system, like the KiWi systems, to address all of them.

Connecting the two knowledge management strategies also confirms the knowledge creation as outlined by Nonaka (1994). He describes that knowledge is created by cycling through the four modes socialization, externalization, combination and internalization. However, not all modes are covered by each strategy. Socialization for instance is a primary activity of the personalization strategy and externalization is a primary activity of the codification strategy. Hence, the modes of knowledge creation can merely work in a mixture of knowledge management strategies. Following only one strategy would not allow the full circle of knowledge creation; design idea 2 remedies this issue. Therefore, the second design idea links the modes of knowledge creation (Nonaka, 1994) to the knowledge management strategies (Hansen et al., 1999).

Further, the design idea is an extension of the experience factory as outlined by Basili (1989). His approach is following a strong codification strategy. People externalize knowledge and store it into an experience base. Others can then retrieve the formalized knowledge from there. The knowledge that is being externalized is the result of management tasks. This shows that the experience

factory focuses on management work only.

If compared to the situation in Logica, it becomes obvious that the experience factory covers only half of the company and ignores the other half. The experience factory can be seen as the management layer following the codification strategy. However, the development layer or a personalization strategy is not considered. Hence, connecting the two layers and their strategies expands the experience factory.

On the technological level, the connection of two different knowledge management strategies resulted in the connection of two different knowledge bases. The evaluation shows that this approach successfully address the knowledge management problems. An alternative approach was created by Sint et al. (2009). The authors focus on the different representations of data in different parts of the same system, instead of connecting different systems. Their work is another outcome of the KiWi project and addresses the same identified problems from a different angle.

9.1.3 Design Idea 3: Wiki for Personalization

The third design idea was presented in section 7.2.1. It defines the choice of knowledge management system in the development layer.

> ***Design Idea 3:*** *Utilizing a wiki in order to support the personalization strategy within the development layer.*

Wikis are collaboration tools; they support the communication between people. This exchange of explicated knowledge can then be viewed or joined by others. Wikis not only support the sharing of knowledge between the people that are actively communicating through it, but works as a knowledge base at the same time.

For Logica, the evaluation shows that the KiWi platform (or short: KiWi) addresses several of the identified problems regarding the knowledge management. The problem regarding the information access involves that knowledge is encapsulated within projects and people from outside the project can barely access it (A1, see section 6.3.1). The choice of the KiWi platform as the collaboration tool throughout the development layers improves this situation. KiWi stores the communication and documentation centrally. Every user of it, and therefore every employee, is able to browse or search through all available contents. This enables the users to learn from and possibly re-use the content of other projects, which can be a competitive advantage. The evaluation shows that the technological enhancements in KiWi facilitate the information access.

The problem that relates to the difficulties to find people with a certain expertise within Logica (A2, see section 6.3.2). These experts exist; however, they are not easy to spot. Although a wiki is not a direct solution to this problem, it approaches it indirectly. The same counts for KiWi. It does not provide the functionality to manage skills of people, but people that are active and write about their expertise can easily be identified. For example, the employee George is searching for someone that has knowledge about a certain database. In the KiWi he then finds a variety of pages about this database, all written and/or edited by the same person. This person might not even be aware of it, but for George the experience and expertise of this person is highly beneficial. By assisting to access relevant information, KiWi thus helps to identify experts.

The problem that deals with the lack of sharing support (A3, see section 6.3.3) illustrates that employees do not share their knowledge, because they mainly lack incentives or opportunities to do so. KiWi addresses this issue. It is a centralized system, which is utilized by all employees to share their knowledge. Based on the personalization strategy in the development layer, the employees also are impelled to use the KiWi platform for their collaboration. The evaluation shows that KiWi makes it easy for the users to share what they know.

The KiWi platform as a knowledge management system in the personalization strategy also tackles the problem that deals with the documentation level (A4, see section 6.3.4). In Logica the available documentation is often inconsistent or incomplete. However, KiWi provides certain incentives to improve the level of a project's work documentation. One aspect is that everybody in the whole company can easily benefit from the available content. Another aspect is that KiWi, like any other wiki, is a platform for collaboration. The people use it to discuss and document the work steps and results. This results in a living documentation of the project and its progress, traceable for anyone across the company. During the evaluation of the prototype, the expert test users assessed this aspect and were convinced by its value. They stated that the collaboration features could be very valuable for Logica and a huge improvement.

The problem relating to the connected documentation is that the documentation of the different projects is not interconnected (B3, see section 6.4.3). This problem was also approached by the design idea 2 above. The basic connection of documentation in KiWi is very basic and equal to the realization in any other wiki. Having the whole content in a single system alone provides valuable search results. This is already an improvement as the documentation in Logica prior to my studies was scattered across several different systems. However, this is not a close connection, but KiWi provides other possibilities, based on its technological enhancements. A main focus here lies on the semantic web technologies. This allows closer connection of topics through tagging and typing. A page can be tagged, by assigning labels that reflect the content. And a page can be typed as being of a specific class from an ontology. Additionally, the typing can be very fine-grained, single data fields can be assigned to a property of a class. Both, the tagging and typing can connect content directly in KiWi and by improving the search results. The evaluation showed that the use of the information that is available for KiWi is not properly used in the evaluated prototype. However, both test users explained that a better utilization of the available information would make KiWi to a powerful tool and be very valuable.

KiWi tackles the different problems with two aspects: First, it is designed to be a centralized system that is accessed by everyone throughout the whole company. Second, the evaluation shows that it provides valuable support for collaboration and communication in projects, which other projects can access. Both aspects are similar to any regular wiki and can be found in the literature. Hence, the core feature is the accessibility of information. Davenport and Prusak (1998) point out that finding what is needed is the main problem and the accessibility of knowledge is an important aspect of knowledge management. The evaluation showed that KiWi successfully addresses these issues and therefore qualifies as a general knowledge management system.

Much research explains the advantages of wikis as a knowledge management system (Kim and Yan, 2010; Raman, 2006; Sousa et al., 2010). It is being

shown on different examples that the usefulness of wikis regarding knowledge management is high. Sousa et al. (2010) specifically state that wikis are a valuable source for expert finding. Schaffert (2006) provides ideas for a wiki that utilizes semantic web technologies and explains how knowledge management can benefit from them. My studies explore these possibilities. The evaluation of the KiWi systems shows that an enhanced wiki like this (here with additional enabling technologies, see section 4.3.4) is valuable.

The experience factory by Basili (1989) describes a centralized solution. Every employee has access to the experience base, similar to the way Logica employees access KiWi. But the two systems are applied differently. I explained earlier that the experience factory follows a codification strategy, which can be illustrated through the way they are utilized. In KiWi, and for that sake any wiki in general, the people communicate with each other through the system. In the experience factory the people communicate with the system only. In a wiki the connection is directly, people collaborate; they ask questions and receive responses. The experience factory is very indirect. People document their experiences in a system. Others then are able to search the system for generalized solutions, without any form of direct communication.

The design idea suggests wikis as general knowledge management systems for the personalization strategy. Hansen et al. (1999) explain that the goal of IT here is to facilitate conversations. Wikis are in general capable of that, and the evaluation shows that KiWi specifically complies this. People are able to communicate through KiWi by editing a page or using comments on one. But Hansen et al. (1999) further explain that it makes sense to support a strategy with a secondary in an 80-20 split. In this case the secondary knowledge management strategy would be the codification. According to Hansen et al. (1999), the goal of IT here is to connect people and to allow the re-use of codified knowledge. This is also achieved by a wiki.

Granted, not all content in a wiki is either communication or codified knowledge, but both are possible in a wiki. And the KiWi platform is even one step further: Different levels of codification exist. In fact, the levels of strong codification and weak codification (i.e., in communication) are fluid. A page can have everything tagged and assigned to the ontology or only parts of it, and sometimes even nothing. The level of codification in the system depends on its users. Hence, KiWi represents the codification support of the personalization strategy; however, the extend varies.

9.1.4 Design Idea 4: ES for Codification

The fourth design idea was presented in section 7.2.1. It defines the choice of knowledge management system in the development layer.

> ***Design Idea 4:*** *Utilizing a project management system in order to support the codification strategy within the management layer.*

A project management system is an enterprise system with the focus on tasks related to the work of project managers. In this case it also involves process management, which is closely related, as the processes define the tasks of project managers. Both, the process and the project management are part of the management layer. In Logica it is a necessity to use a system like this,

because of the general orientation towards the codification strategy. The project managers use it to organize the project scheduling and for the resource planning. This gathered information is then assembled in a report to be communicated with the management layer. Within the management layer all communication is based on documents, which are created through project management systems. The utilization of these is therefore needed in the day-to-day business.

Additionally, the knowledge management problem is tackled that deals with the connection of available documentation (B3, see section 6.4.3). My analysis shows that every project manager uses their own project management system. Applying the same system (or connecting the content of the several systems) throughout the entire management layer of the company addresses the problem. The isolated applications are avoided by making everybody use the same one.

The evaluation shows that utilizing a project management system in the management layer is valuable. The test users are experts in the field and experienced project and process managers. Despite smaller problems related to the usability, both rated the PMA as valuable. They further explained that it helps to connect the data and make it easier available. The problem regarding the connection of documentation (B3) is therefore successfully addressed according to the test users.

Hansen et al. (1999) describe that the goal of the codification strategy is to codify, store, disseminate and allow the re-use of knowledge through a system. With this design idea I confirm this theory and extend it by suggesting a specific tool for the IT support. I explain and evaluate that an enterprise system that focuses on process or project management can be successfully applied for that. Such a system brings all necessities for process and project managers to codify their work and allow others to re-use or learn from it. The availability of data and its accessibility is important (Davenport and Prusak, 1998).

But not only on the choice of tools to utilize the strategy, also the general utilization of IT is very vaguely defined by Hansen et al. (1999). The authors basically just state that people codify their knowledge into documents. This design idea suggests a stronger focus on data: A system that is based on a structured internal knowledge base as outlined by Davenport and Prusak (1998). Applying a strict tool in form of the PMA is the result. Hence, the design idea connects the theories regarding the knowledge bases and the knowledge management strategies by showing that the codification is well addressed through a structured internal knowledge base. This connection is done regarding the requirements of the management layer in Logica. The communication is entirely based on data, representing details about the project and process management. Additionally, the evaluation of the KiWi systems shows that connecting the strategy and knowledge base is valuable and addresses the approached issues.

The fact that my contribution regarding this design idea is limited to the aspects relating to the codification strategy in one of the organizational layers in the case company is not surprising. This part is very traditional. Basically every company uses enterprise systems and much research has focussed on it for a long time. With this design idea, I respect the necessities, which increases the likelihood that the entire knowledge management approach can be successful.

9.1.5 The KiWi Systems

The resulting prototype of a knowledge management system is more than just the sum of the four design ideas. Three systems (KiWi, DxA and PMA) are combined to create an entire new knowledge management approach, which deals with Logica as a whole, the development layer as well as the management layer.

The combination of all the systems addresses the problem that deals with the process complexity (B1, see section 6.4.1). The vast number of process descriptions and the resulting complexity influences the work in the management layer. The system has to provide support for the user, to find and apply the correct process description. KiWi provides support in finding the fitting process description and through the PMA and the DxA the project manager can easily apply it. The evaluation showed that these aspects work as intended. Further, the test users explained that with such a system it would be more difficult to violate the company's rules than to follow the processes.

In the previous sections, I assign the identified problems to the different design ideas, which address them. However, the design ideas cannot always be seen isolated from one another. Especially in the realization they are based on or require each other. Hence, each of the problems is of course also addressed by the system as a whole. But instead of describing them together, I chose to set the focus on the underlying design ideas. Table 18 provides an overview.

	Problems
Design Idea 1	A & B
Design Idea 2	B2, B3
Design Idea 3	A1, A2, A3, A4, B3
Design Idea 4	B3
KiWi Systems	B1

Table 18: Design Ideas addressing Knowledge Management Problems

The knowledge management approach includes a workflow description (see section 7.5), which focusses on the handling and interaction with the KiWi systems. It provides scenarios in which the usage and examples for applications are explained. My studies do not cover the creation of process descriptions for Logica, although I acknowledge that knowledge management always has to inform the processes of a company. It is further a necessity that the employees of Logica, and thus the potential users, accept the knowledge management systems. The people additionally have to want to work with the systems, otherwise the knowledge management approach will not be successful.

These two aspects are important for a successful knowledge management approach, the considering the company's processes and the involved people. Much research documents this, as I report in the related research chapter (see sections 2.3.2 and 2.4). Both aspects have been valued as being plausible by the experts during the evaluation of the KiWi systems.

Mathiassen and Pourkomeylian (2003) strongly suggest to acknowledge the involved knowledge management strategies and to address them directly. The KiWi systems follow that advice and focus very much on the strategies within

Logica. The differences in the natural approach to their work in the management and development are acknowledged through a distinction into separate layers. This extends also the work of Hansen et al. (1999), who state that every company should focus on one knowledge management strategy only. The knowledge management approach presented in this thesis and therefore also the KiWi systems go one step further and apply a fitting strategy where it is crucial.

My analysis shows that in the company two different layers can be detected, with different work goals. Each of these is provided with a separate knowledge management strategy. The KiWi systems then supports each layer separately, connects both of them and ties them together. This allows to share knowledge across the entire company, not limited to a layer.

The KiWi systems enable the knowledge sharing also on a different level. Alavi and Leidner (2001) show that the organizational knowledge management process includes the sharing of knowledge not only within a group of people, but also between the different groups in a company. For Logica, a project team reflects such a group. The problems related to the isolated islands of knowledge (problems A1, A2, A3 and A4) deal with the lack of knowledge sharing between the teams of different projects. Alavi and Leidner (2001, p. 123) present a theoretical approach for the knowledge sharing between groups (see figure 3 on page 10). The KiWi systems are an implementation of such an idea. Projects are connected through the KiWi systems and the knowledge sharing between projects can easily be established. During the evaluation, the test users explained, that this connection of projects makes the re-use easier and more likely.

Additionally, the KiWi systems extend the idea of the experience factory by Basili (1989). The experience factory has a strict separation between project work and knowledge base. The employees interrupt their project related tasks to create knowledge items, which then will be stored in the knowledge base. In the KiWi systems these two different areas are united. The working platform is also the knowledge base. When people cooperate through KiWi their work and the containing knowledge items are automatically in the knowledge base.

This connection influences the way the knowledge is treated in the knowledge management approach. While the KiWi platform allows active communication as well, the experience factory is limited to passive communication. People have to create content, which eventually might become useful for someone. In KiWi this is more targeted. The passive approach to store information of putative users is also possible.

However, during project work KiWi is supposed to follow the active approach, i.e., for collaboration with a number of colleagues. Communication through the system is in both systems the sharing of knowledge. The mere existence of another approach provides more possibilities. In the end storing becomes sharing in passive mode only. This reflects the direct and mediated impact to learn according to Ravichandran and Rai (2003), the KiWi platform embodies both.

In the related research I show that the IT should not be the focus of a knowledge management approach (Kautz and Thaysen, 2001) and that social aspects are an important factor in knowledge management (see section 2.4.1). My research acknowledges that, through providing IT support for the personalization with the social web. This seems contradicting at first, as it sets a focus on the IT. However, the social web, here represented through the KiWi platform, sets

the focus on the people. And it does this throughout the company, not limited to one layer.

9.2 Limitations

My PhD studies, the approach to knowledge management presented in this thesis, as well as the design study itself have a number of limitations. I am aware of this and want to raise an awareness to the reader with this section.

The role of a scholar in any study involves a variety of decisions, for example during the interpretation of data or the design. Every decision results in a chosen focus and is therefore a limitation of the outcome. Further, it can never be ruled out that these decisions are influenced by personal opinions, views or attitudes. My PhD studies are of interpretative nature and should be recognized as such.

The KiWi project was the foundation of my studies. It was a defined outcome, to create and assess the use of an envisioned system (the KiWi platform) in knowledge management. Other communication platforms than KiWi were never an issue.

Additionally, despite being an active part of the KiWi project, my power was limited. I was able to influence decisions of the project team, but only to certain extend. The different involved parties had their own agendas, the Logica business case was one aspect of the KiWi project's work among many.

Other limitations come with the choice of action design research. The close collaboration of scholars with practitioners, as in my studies, is always a mutual exertion of influence. It has thus to be taken into consideration that the mere engagement influences the outcome of a study significantly.

I discuss above that the case company is not unique, because all problems are based on the literature. However, all companies are different from each other, at least to certain extent. The target company is developing software for different customers in a variety of projects. Even if other companies have a similar organization or focus that does not imply that the KiWi systems would solve their knowledge management issues. My research can be generalized along specific characteristics in an organization. Logica is a big software development company, with a strong view on processes, but acknowledges that not everything can be completely based on these processes. These characteristics lead to knowledge management problems, which are addressed in my studies. Companies with similar characteristics as the case company can benefit from my findings.

Another limitation is the data collection within the design study. The understanding of the problems is defined by the collected data. The access to the case company was limited, due to the arrangements set in the project. Legal issues also influenced the view on the case company's data. Documents were under non-disclosure agreements and the practitioners were not allowed to share them with outsiders. This research therefore relies on the available sources. Different perspectives or further insights can always change the understanding of problems.

The amount of involved people is always a limitation to the outcome in ADR. As it emphasizes the intervention and evaluation during the work process so much, the influence of the people involved in these stages is very strong.

In my studies two to three representatives of the case company were involved, although they are very skilled, knowledgeable and experienced. The engaging of scholars and practitioners in ADR leads to the work results. Therefore, different or more people could have led to other results or directions of the study.

For the user tests in the final evaluation phase only two test persons were involved in three test sessions each, this is a limitation on the outcome of the whole evaluation of the systems. As I describe above (see page 144), this decision was made out of necessity. It was not possible to assign more people or more time for further tests. The stability of the systems as well as the organizational boundaries of the project limited the available test users. A higher variety of test persons might have led to a different view on the usefulness of the KiWi systems.

As shown in many parts throughout this thesis, my research builds upon existing research. The research that I refer to is the result of a literature study. The involved works were taken into account due to conscious decisions on focus. However, they draw from research published in a limited amount of outlets and over a limited period of time. The field of knowledge management is rather big and a lot of research in different fields is involved, which made the focus necessary.

9.3 Future Research

This PhD study provides insights into the design of a knowledge management approach and a knowledge management system for a software development company. These insights have limitations and further scrutiny is needed with the help of different research approaches and further assessment. In particular, more observations and future research are needed to gain more insights about the application of communication tools like wikis in knowledge management and in hybrid forms of the knowledge management strategies.

The KiWi systems comprise of three different systems. During the final evaluation one of the comments by the test users was that this is a lot to deal with. Especially the DxA was criticised. I find it interesting to see whether the three systems could be integrated into one. The design of such an integrated system would be very challenging, as it has to provide the necessary functionality for each distinct layer. Such a system would have to combine both types of knowledge bases into one. And maybe such a system could also advance the mix of the knowledge management strategies. The distinct differences of layers could be blurred and the strategies might be applied on the personal level. As a result, every user can define themselves which way of dealing with data they prefer.

My research was bounded to the application of the KiWi platform. However, there is a variety of other communication platforms in general, and wikis specifically, available. I am curious to see design idea 3 realized through other options. The influence of the different systems on the knowledge sharing can be compared between different communication platforms.

Additionally, to dig a bit deeper, I would like to investigate the use of wikis in the enterprise context more thoroughly. A focus on the development layer helps to find out about the challenges that wikis bring. I expect that only through

a proper integration into the company's processes and people with adequate motivation for sharing with others wikis can be productive and beneficial.

I think the knowledge model utilized in the prototype is of very high quality, but has not been explored satisfactory. If the PMA and KiWi provide more than just input fields for the data models, the entire approach would benefit from that. For example, it would be possible to use the data more holistic in KiWi and not just page based.

Also, the KiWi, though built on top of the semantic web technologies, does not make little use of the possibilities regarding the knowledge model in connection to the wiki philosophy. It is not very transparent to the users, either. The possibilities here are wide, though. The knowledge model could be edited by users directly, which, for obvious reasons, influences other users. Different interactions and their influence on the behaviour of KiWi and the entire knowledge management approach might be tremendous.

These ideas for further research document my broad interest in knowledge management and communication tools.

10 Conclusion

This thesis dealt with the description and documentation of the design, implementation and evaluation of a larger prototype dealing with the organizational knowledge management processes in the setting of a big international research project. Based on the FP7-funded KiWi project, my research had the goal to find a way to support the knowledge management in software development. Hansen et al. (1999) state that a company should pick either a codification or personalization strategy to support knowledge management. I elaborate on their work by proposing how to carry out a different strategy in the different organizational layers of a software development company. This insight is based on research involving the design of a knowledge management approach and system, with people from the software development company. Following the *Action Design Research* methodology as outlined by Sein et al. (2011), my research involves a cyclic organization and ensemble view of IT artefacts. The cyclic organization with constant iteration of building, intervention and evaluation allowed a continuous shaping of the design. The ensemble view of IT artefacts allowed consideration of not only the IT, but also the whole environment.

The analysis of the case company showed a number of knowledge management problems. I differentiate between four problems regarding *isolated islands of knowledge* and three problems regarding the *inadequate bridging of knowledge*. These problems and the remaining insights gained from the analysis lead to the conclusion that I deal with two layers of the case company: A development layer and a management layer. My analysis shows that these two layers have a different style of knowledge sharing. Many of the problems occur because of issues between these layers, a knowledge management system has to consider this.

The design is based on four design ideas, which address the different knowledge management problems.

Design Idea 1: Each of the organizational layers has to be assigned their own knowledge management strategy: Codification in the management layer and personalization in the development layer supports the strengths and overall strategy in each layer.

Design Idea 2: The layers and the strategies have to be connected. As the gap between both layers caused a variety of problems, it is essential for a successful knowledge management approach that this gap is bridged and sharing of knowledge across the borders is feasible.

Design Idea 3: A wiki can support the personalization strategy in the development layer. Wikis have strengths to support collaboration and interaction on shared content, which meet the needs of developers.

Design Idea 4: An enterprise system, focussed on process and project management, can support the codification strategy in the management layer. The case company functions very process-driven and a project management system like this is basically required.

Based on these four design ideas the prototype of a large knowledge management system was created, composed of three sub-systems. The *KiWi Platform* is the supporting system for the development layer. This is a wiki enhanced with different technologies, like a recommendation component or semantic web technologies. The system to support the management layer is the *Project Management Application*. This is a system that focuses on the administration and creation of data. Both systems are connected through the *Data Exchange Agent*. This is an application that allows the project manager to move data from one system to the other. All three systems in combination are called the *KiWi Systems*. A final evaluation by experts showed that the prototype is valuable and could be beneficial for the case company.

My research contributes to different areas in the field of knowledge management theory. My studies mostly deal with the codification and personalization knowledge management strategies of Hansen, Nohria, and Tierney (1999). I elaborate on these and present an approach, which adapts both strategies within the same company by distinguishing between two layers: The management and the development layer. Additionally, I connected these strategies to the knowledge bases. I argued and provided suggestions regarding to the link between a knowledge management strategy and a fitting knowledge base.

Appendices

A

Knowledge Model

The knowledge model is the underlying data structure of the KiWi systems. By sharing this general data model it is possible to exchange data between the two systems. The shared knowledge model is therefore an aspect of the realization that follows the second design idea (see section 7-1-4), to connect the layers.

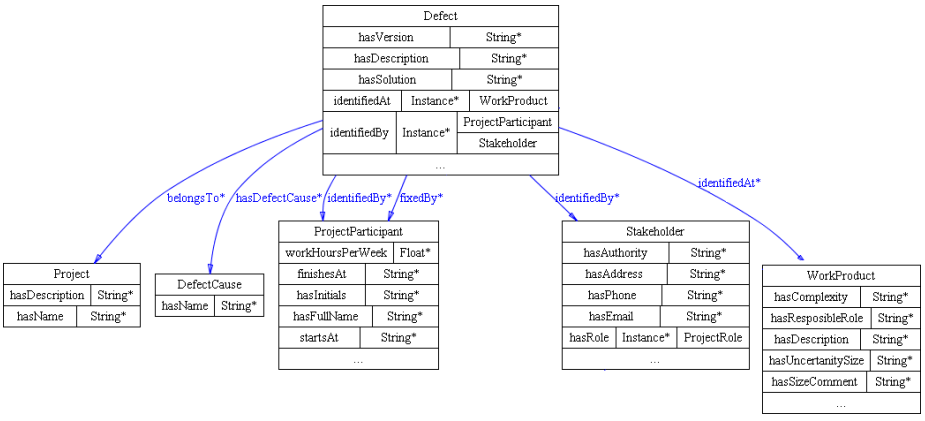


Figure 52: The Type Defect and its Relations (RDF Diagram), from (Dolog et al., 2009b)

This general data model has to be translated into the data structure, suitable for each system (see section 7.3.3). The data between the PMA and KiWi are to be exchanged; hence the knowledge model has to be translated to a relational

database for the PMA and into an OWL ontology for KiWi. The DxA then has a mapping table, which is more like a dictionary that can knows all entries of the knowledge model and how they are represented in the two data models (see section 7.4.2).

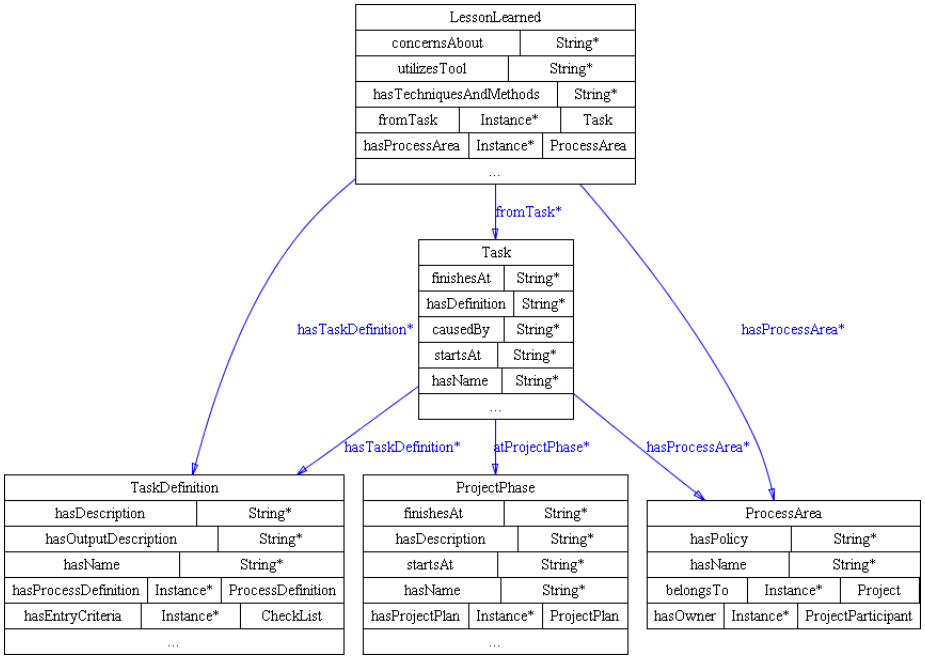


Figure 53: The Type LessonsLearned and its Relations (RDF Diagram), from (Dolog et al., 2009b)

The knowledge model itself contains elements for all plausible aspects of the project work. It describes the relationship between entities. Figure 52 shows an example of the `Defect` entity, figure 53 the `LessonsLearned` entity. You can see the fields it contains and the entities it is related to. Note that for sub-entities the relations are mostly left out. The reason for this is the readability; this does not imply that there are none. All three figures (those two above as well as figure 37 on page 117) are examples for aspects of the knowledge model.

More details regarding the knowledge model can be found in the deliverable D6.3 of the KiWi Project (Dolog et al., 2009b). The OWL representation of the knowledge model is online; as a part of the KiWi open repository. It can be accessed directly under this URL:

`https://svn.salzburgresearch.at/svn/kiwi/KiWi2/trunk/extensions/ontologies/resources/logica/logica.owl`

B

Feature List

This part of the appendix shows the full feature list of the KiWi Systems. It is equal to the one previously published in the KiWi project's deliverable D7.2 (Grolin et al., 2010a). The features are numbered following a simple code to understand the developing party (table 19).

No.	Objective
1xx	Core functionality
2xx	Enabling technology: Reasoning
3xx	Enabling technology: Reason maintenance
4xx	Enabling technology: Information extraction
5xx	Enabling technology: Personalisation
6xx	Data Exchange Application
7xx	Logica application

Table 19: The Numbering System for Features, from (Grolin et al., 2010a)

The complete feature list (table 20) covers the following pages. It was created in close cooperation with all developing participants of the KiWi project.

No.	Name	Usage
100	Login	Click “Login” in the top right corner of the page to jump to the Login Page. Insert details and click “Login”.
101	Create User	Click “Sign Up” in the top right corner of the page or “Register first” in the Login Page to jump to the Sign Up Page. Insert details and click “Store”.
102	Edit Page	Click on “Actions” to activate a drop down menu. Choose “Edit” here to jump to the Edit Page. After editing the content click “Save” to persist.
103	Comment Page	Click “Comments” underneath the page’s content to open the comments view. Then click “Add comment” to open an editor; write comment and click “Add” afterwards to persist.
104	Link to Page	In the editor either type the page’s name in double square brackets [[page]] or click the “create/edit internal link” icon and enter the target page’s name.
105	Create Page	Create a link (No. 104) to a page that does not exist. Navigating to that page opens the edit screen of the new page automatically.
106	Tag Page	Viewing a Page; click “Edit” next to Tag underneath the Headline. This opens a Tag window. You can add new tags or use recommended tags (by clicking the green plus sign) or add weights to previous set tags (by clicking the green plus sign).
107	Search content	Enter the search word(s) into the form on the top right corner of the screen and press enter or click on the magnifying glass. You will be lead to the Search Page and the results are listed.
108	Edit Metadata	Click on “Actions” to activate a drop down menu. Choose “RDF Metadata” here to jump to the Metadata Page. Here you can edit the values by just clicking them or deleting it by clicking on “Delete” at the end of the row.
109	Load Ontology	Enter the Admin area; click “Load Ontologies” from the menu; choose the ontology to load and click “Load Ontology”.
110	Revert Page	Click on “Actions” to activate a drop down menu. Choose “History” here to jump to the Edit Page. Click on the link from the list to revert the page to the specific version number.
111	Compare Page Versions	Click on “Actions” to activate a drop down menu. Choose “History” here to jump to the Edit Page. Choose two or more tickboxes of the versions you want to compare and click “Compare selected revisions” to jump to the comparing page.
112	Add Page Type	In Editing Mode (No. 102) below the headline; users can add types for a page by clicking the green plus icon and choosing the type from the list in the pop up window and then clicking the “Add” button.

Table 20 – *Continued on next page...*

Continuing from previous page.

No.	Name	Usage
113	Delete Page Type	In Editing Mode (No. 102) below the headline; users can delete types for a page by clicking the red minus icon next to the type.
114	Microformat support	special formats like hcalender; hcard are generated out of the rdf data of a contentitem; so these formats can be imported to the os adress book
115	Semantic Forms	Defined in simple HTML that has to be stored in the filesystem. In the editor it then can be applied.
116	View Article in TagIT Extension	If the current wiki-view shows a location-based ContentItem it is possible to click on the TagIT extension
117	Dashboard	The dashboard is used as kind of a social networking extension; where one can define/refine personal information; upload a profile picture; add or remove friends and join/leave groups
200	Reasoning	Either online-reasoning is switched on then nothing is required to do or it is switched off. If it is switched off then go to Admin ? Reasoner and click "Run reasoner".
201	User-specified rules	Change the rules.txt file.
202	Online reasoning	Enable online reasoning in Admin → Reasoner → Online reasoning
203	Inspect triples	Go to Inspector → Base triples or Inspector → Inferred triples.
204	Extended Search	On the normal search results (see 107) click on "kwql" to get to the search results.
300	Triple explanation	Inspector → Explanation Enter a triple id and click "Explain". You can get the triple id from the "Inspect triples" feature.
301	Explanation tooltips	Hover mouse over inferred types (in italics) in Ingoing/Outgoing relations on the right side of the Wiki application.
400	Tag Recommendation	The system provides recommendations for tagging (see No. 106).
401	Create RDFa Property	In Editing Mode (No. 102) mark the text that should become a RDFa Property's Value; click the "Create/edit RDFa property" icon and choose the property from the list in the pop up. Then click "Save" to persist.
402	Edit RDFa Property	In Editing Mode (No. 102) mark the RDFa Property's Value that should be edited; click the "Create/edit RDFa property" icon and choose a different property from the list in the pop up. Then click "Save" to persist.

Table 20 – *Continued on next page...*

Continuing from previous page.

No.	Name	Usage
403	Create RDFa Link	In Editing Mode (No. 102) mark the text that should become an RDFa link; click the "Create/edit RDFa link" icon; choose the property from the list in the pop up and write the name of the page that should become the target of the link.
404	Edit RDFa Link	In Editing Mode (No. 102) mark the RDFa link that you want to edit; click the "Create/edit RDFa link" icon; change the property from the list in the pop up or/and the name of the page that should be the target of the link.
405	Reject Tag Recommendation	The user can reject a suggested tag recommendation; when it does not make sense. The recommendation improves and the user is not bothered with it any more.
406	Tag Active Learning	On a page of a tag; a list of all documents that have this tag as a suggestion is displayed. A user wanting to use this feature would: 1.manually tag a small set of example documents using this tag. 2. Go to the page of the tag. 3. Initialize the classifier by selecting a 'type' of the tag (such as; whether it tags a whole document; or just specific fragments) 4. Go through the suggestions presented and acknowledge or reject any of them. The suggestions are computed using machine learning algorithms based on previously tagged documents/fragment and learn based on the user feedback.
407	RDFa/fragments suggestions.	The user can display/hide suggestions for RDFa datatype properties; object properties or tagged fragments in the text. She can then accept or reject each suggestion separately.
408	Creation and Editing of Fragments	The user can select a piece of text; click on a Fragment button; which creates a fragment. Tags can be added to this fragment. To edit a fragment; one would place the cursor into a fragment and click the Fragment button. User can add or remove tags on the selected fragment.User can also delete a seleted fragment.
500	Recommendations by Tag	The system recommends pages with similar tags.
501	Personalized Search	First of all tick the "personalized search" check box. Then; enter the search word(s) into the form on the top right corner of the search box. You will be lead to the Search Page and the results are listed.

Table 20 – *Continued on next page...*

Continuing from previous page.

No.	Name	Usage
502	Social Recommendation	On the head of current content item; select a friend from your contact list and recommended the current content item. Your friend will be able to see the item when accessing the dashboard; more specifically on the social recommendation widget
503	Tag purpose	On the dashboard; there is a menu option "My Tags". Users are able to define the purpose of their tags such as: qualification (ex: good; bad); conceptualization (ex: book; city) and so forth. Such entries will be utilized as filter for later information retrieval.
600	Get Templates	Click on "Get Templates" Button in UI.
601	Choose Template	Click on Template in List.
602	Choose Entity	Click on one field in the "Data View". Whole row has to be marked.
603	Publish Data	Click on "Publish" button in UI after selecting row of interest.
604	Update Logica Application	Click on "Update LUC" button in UI after selecting row of interest.
605	Update KiWi System	Click on "Update KiWi system" button in UI after selecting row of interest.
606	Open Entity in KiWi	Click on "Open Entity KiWi Page" button in UI after selecting row of interest.
700	Login	Click "Login" in the top right corner of the page to jump to the Login Page. Insert details and click "Login".
701	Show Entities	Navigate to the entity list using the main navigation.
702	Show Details	On the entity's list page; click on entity's name or "View" in list.
703	Filter Entities	On the entity's list page enter the filter words into the mask on top and click "Search".
704	Edit Entity	Click "edit" in the entity list or on the entity's details page.
705	Create Entity	Click "Create Entity" button on the bottom of the entity list page.
706	Delete Entity	Click "edit" in the entity list or on the entity's details page and click the "Delete" button.
707	Add 1:Many Relation	On the Entity Details page underneath the details in tabs the relations are displayed. Click the "Add Entity" button to go to an edit page; insert details of the related entity and "Save".
708	Edit 1:Many Relation	On the Entity's Relation list in the tabs; click "Edit" to jump to the entity's details edit page directly.
709	Delete 1:Many Relation	On the Entity's Relation list in the tabs click "Edit" to jump to the entity's details edit page and click the "Delete" button.

Table 20 – *Continued on next page...*

Continuing from previous page.

No.	Name	Usage

Table 20: The complete Feature List, from (Grolin et al., 2010a)

C

Use Cases

This part of the appendix shows the use cases of the user tests for the evaluation of the KiWi Systems. It is equal to the one previously published in the KiWi project's deliverables D7.2 (Grolin et al., 2010a) and D7.4 (Grolin et al., 2010b).

	1xx	**2xx**	**3xx**	**4xx**	**5xx**	**6xx**	**7xx**
UC1	100, 102-109, 112-113, 117			401-405	500, 502	600-605	700-709
UC2	102-107					600-604	700-705
UC3	100-109, 112, 117	201, 204		400-404	501-503	600-604	700-705
UC4	102-107	204			501, 502	600-603	700-705
Not Eval.	110, 111, 114-116		300, 301	406-408		606	

Table 21: Use Case Evaluation Coverage, from (Grolin et al., 2010a)

Each of the use cases states the features of the KiWi systems (see appendix B). The coverage of the whole system can be seen in table 21. An exception from this is the fifth use case, as it was created during the process to specifically evaluate the information access (for details see section 8.2).

C.1 Use Case 1: Project Planning

The first use case deals with the activities of a project manager that concern the planning of a project.

Before Picture

Missing integration: There is no integration between planning tools and tool used to store the business model. It is an entirely manual task to produce a project plan that is consistent with the business model. Doing so is a huge task given the size and the complexity of the business model (containing several hundred processes, policies, requirements etc.). Creating the proper sets of plans based on the business model requires 300 – 700 hours during project start up, and the project managers are typically overworked at this particular time because they also have to invest time in contractual issues, forming a team, on customer relations etc... Basically the organization gets trapped in a situation where the body of valuable and relevant knowledge increases as more experiences and best practices are collected and integrated in the business model, actually makes it more difficult for project managers because the amount of knowledge, as well as the complexity, increases.

IT support for planning: Very modest.

Inconsistency: Making sure that plans are consistent (e.g. that there are tasks that deals with identified risks) is manual, and very time consuming task.

After Picture

Integration: The planning tool (PMA) and the business model are integrated, in the sense that the business model can be used directly in the planning task. Support for planning: furthermore the planning is supported by functionality that automate parts of the planning process.

Consistency: There are functionalities that can analyse and check whether a plan is consistent.

Description

The detailled descriptions and the different work steps of the use case are described in table 22.

Work Step	Features
Part 1 – Analysing the scope of the project	
Preparing: The primary input to the planning process is the requirements and the products that have to be developed. The requirements specification as well as the contract is studied in great detail by the project manager, the business analyst and a technical architect Requirements are entered into the system. Products are identified and a product-breakdown-structure (e.g. sub system a, b, c...) is created. For each product a responsible project participant is chosen.	700 - 709

Table 22 – *Continued on next page...*

Continuing from previous page.

Work Step	Features
Publishing: The product-breakdown structure with information about all the products as well as the related requirements are published in KiWi.	600 - 603
Collaborative scope analysis: Project participants analyse, review, comment and add detailed information about the various products and requirements that they are responsible for. They might also enter ideas about how to fulfill the requirements, design ideas etc for later use. During the process they identify issues that needs to be clarified, and they challenge and review information about product size and complexity, since this information is used during the estimation process. When doing so the participants exploit information entered by other projects about how they have dealt with similar requirements and products. Pages containing relevant information are suggested by the system. The participants also use process descriptions and checklists relevant for the tasks, e.g. checklists that highlight important issues to remember when reviewing requirements.	100, 102-107
Consolidation: When finished the information is feed back to the PMA application. Project management establishes an overall overview over the scope and the requirement related issues that needs to be resolved with the customer.	604, 701
Part 2 – Analysing Risks	
Preparing: The project manager creates one or more pages in the system for entering information about risks and invites project participants to take part by entering risks.	100, 104-106
Individual Risk identification: The project participants enter possible risks into the system. They are supported in several ways e.g.: They can easily access risks from other projects, they can use the risk identification process as well as checklists stored in the system, and they can access requirements and product related information from this specific project.	102-104, 107
Risk identification meeting: The project group meets and discusses the identified risks. During the discussion notes are documented on the various pages.	102, 103
Risk identification clean up: The project manager enters the detailed information about the risks (e.g. who's responsible). Some risks are split up and treated as several risks. Other risks identified by different persons are merged into one because they address the same problem. While doing so the project manager also adds information about possible resolution strategies.	700-705
Risk resolution meeting preparation: The project manager publishes the almost finished risk analysis for commenting and reviewing.	600-603
Risk resolution and review meeting: The project group reviews the risk analysis and especially the resolution strategies proposed by the project manager. The resulting changes and comments are documented on the pages containing the risk analysis.	102, 103

Table 22 – *Continued on next page...*

Continuing from previous page.

Work Step	Features
Project plan integration: The project manager imports the resulting changes into the pm application and integrates the risk analysis in the project plan.	604, 700-705
Part 3 – Estimating the project	
Prepare planning: The project manager chooses a lifecycle model in the PMA application, combines the product-breakdown structure and the processes to generate a first version of the estimates. When doing so the estimates are based on productivity metrics as well as the product size information. During this process project participants are appointed as responsible for various activities.	701, 704, 705
Involving participants: The activities and estimates are published in KiWi together with all the other kinds of information. That is: For each activity the project participants can, not only se the estimates, they can also see the product information (e.g. sizing information), they can see the requirements for the products, the can see any risks that are related to the activity, and they can see the metrics that the risks are based upon, and the processes they are supposed to use.	600-603
Commenting on the plan: Project participants get the opportunity to review, comment and adjust the estimates based on their experience. When doing so they can exploit information from similar projects/activities (how was a similar task estimated in a previous project?), various checklists as well as information about products, requirements and risks for this specific project.	102, 103, 105
When the project manager uses the system, she can easily get an overview over changes and comments on various pages, and provide "comments to the comments". Finalizing the estimates: When finished the information is feed back to the pma. The project manager checks the estimates, e.g. by comparing estimates across different but similar activities.	604, 700-705
Part 4 – Scheduling the project	
Making a draft: The project manager prepares a schedule based on the estimates, and dependencies between products and activities, project participant availability and the number of hours they are allocated to the project. When doing so the project manager can chose between making a conservative, realistic, or aggressive schedule.	700-709
Publishing the draft: The schedule is published in KiWi. The schedule is published in a way that makes it easy for project participants to see their personal schedule	600-603

Table 22 – *Continued on next page...*

Continuing from previous page.

Work Step	Features
Reviewing the schedule: The participants (and other stakeholders) can comment on the overall schedule as well as their own personal schedule describing when they are supposed to start and finish activities. When finished the information is feed back to the PMA application. The project manager finalizes the schedule, and checks consistency using the PMA application, the consistency check (e.g. is expected future productivity for remaining activities realistic compared to past productivity?) becomes a part of the status report. A final project plan, schedule etc is published in KiWi.	102, 103, 604, 701-704, 605

Table 22: Use Case 1, from (Grolin et al., 2010a)

C.2 Use Case 2: Project Monitoring

The second use case deals with the monitoring of projects by employees from the management level.

Before Picture

Un-reliable status reporting: It is not possible to ensure that project status reports are based on reliable data e.g. about progress. It is almost impossible for outsiders to check whether a status report actually presents a true picture of the project.

Missing integration: There is no integration today between the various documents, spreadsheets etc that contains information used in status reporting.

Lack of tool support: Preparing a complete status report is basically a manual activity that takes 1-2 days each month for project managers on large projects.

After Picture

Reliable status reporting: By basing the status reporting on data from the PMA application and using functionality to do a consistency check a more reliable status report is established.

Integration: More integration, less manual work

Support for reporting: Status reporting is changed from an activity were the effort is focused on getting data, to an effort were the focus is on taking management decisions about future actions based on reliable data.

Description

The detailled descriptions and the different work steps of the use case are described in table 23.

Work Step	Features
Part 1 – Collecting data about the project	
The PMA generates data used for evaluating the project status. **DxA:** The status information is published in KiWi. **KiWi:** Project participants enter status information regarding the tasks assigned to them and modifiy estimates if necessary. Project participants also enter status information regarding other objects that they are responsible for, e.g. risks and run-resolved issues. **DxA:** When finished the information is feed back to the PMA application.	700-705, 600-603, 102-107, 604
Part 2 – Finalysing and publishing the status reports	
PMA: The project manager prepares a final status report using the application based on the input from the project participants. As part of this step, corrective actions are defined if necessary. These actions might involve that the schedule is modified, tasks being re-allocated etc. **DxA:** The status report as well as a new plan reflecting the needed changes which are published. **KiWi:** Management can read and comment on the status report in the KiWi, and the project participants can see any changes to the project than might affect their work.	704, 705, 600-603, 102-107

Table 23: Use Case 2, from (Grolin et al., 2010a)

C.3 Use Case 3: Development or Project Work

The third use case deals with the general project work of the development layer.

This use case overlaps with the first step of the project planning use case (section C.1). It does, however take a different perspective and focus to uncover a different type of usage.

Before Picture

Clarifications lost or non-distributed: A lot of the information that pass between individual developers and the architects is undocumented because it is requested and received informally. Other developers working on the same work package do not benefit from the clarification.

Missing documentation for formal changes: The need for changes to plans will require justification and without written clarification there is nothing to refer to in formal requests.

Missing Integration: Planning, distribution and management of work packages are carried out in different systems.

Monitoring: If an issue is of interest there is no way to stay in the loop unless all parties discussing it are asked to include the third party.

Difficult to find relevant information: The developers are not aware of issues that arise for other developers and have no way of finding out.

After Picture

Clarifications documented and available: The clarification given to developer is placed on a page that is searchable and viewable by all project participants.

Change information available: Change requests and revision can refer to clarification given.

Integration: The integration allows the architect and project manager to maintain the work packages in one system.

Syndication: The project participants can get notification when new information is added to the page about an issue of interest.

Intelligent information finding: The developer can receive recommendations or actively search for solutions to problems that they face.

Description

The detailled descriptions and the different work steps of the use case are described in table 24.

Work Step	Features
Part 1 – Work package assigned.	
PMA: The PMA application generates the scope and content for each package. The work package is broken down into assignable development tasks. **DxA:** The work package information is published in KiWi. **KiWi:** Developers can pose questions about the designs to the architects to answer. If the question cannot be answered by the architect the question is passed on to the customer. Issues raised by the developers and the answers they provoke may affect estimates and it may cause the customer to make a change request to designs. **DxA:** If feedback causes formal changes to plan or design these are feed back into the PMA system.	700-705, 600-603, 102-107, 604
Part 2 – Update information	
PMA: Updates are made as a consequence of the clarification. **DxA:** The updates are published. **KiWi:** Developers are notified that their work package or tasks have been affected.	704, 705, 600-603, 117

Table 24: Use Case 3, from (Grolin et al., 2010a)

C.4 Use Case 4: Process Design

The fourth use case deals with process design activities.

Before Picture

Disconnected process description and execution: The description of the process and the application (insofar as there is one) that supports it, are disconnected and users do not necessarily consult them.

Inflexible application: The applications are generally not designed for frequent changes to the process it supports.

No collection of innovation ideas: Since descriptions of the process and the execution of the process are maintained different places, users seldom decide to record ideas for change, the application having no place to compile observations.

After Picture

Support process in execution: The process description follows, side-by-side, with the pages on which process execution is carried out.

Process change support: The tool support is flexible enough to handle most change to the process it supports.

Process innovation support: The tool supports the process to gather process innovation ideas and put them up for general consideration.

Description

The detailled descriptions and the different work steps of the use case are described in table 25.

Work Step	Features
Part 1 – Call for change recommendation **KiWi:** The Project Manager creates a page in KiWi where he asks for comments or other feedback regarding the audit process. He then uses the KiWi search and recommendation to find people, that work with this process and invites them to give feedback as well. He then watches the page and the discussion, contributes, when needed, but mostly just reads the comments. To create a page, he sets a link on another page to the one he has in mind (e.g. [[Definition of the Audit Process]]). Clicking on it automatically opens the edit screen for the new page. To invite others, he simply sends the link via e-mail.	100, 105, 102, 103, 107
Part 2 – Refinement **KiWi:** After a while, when he does not expect any further input, the Project Manager refines the discussion's results. He might do that on a dedicated KiWi page which links to the discussion page or directly underneath on the same page. However, he analyses the feedback and creates a list of things to change. Editing a KiWi page works basically like standard text processing tools do. The main functionalities are represented in icons of the edit screen.	105, 102

Table 25 – *Continued on next page...*

Continuing from previous page.

Work Step	Features
Part 3 – Publish draft **KiWi:** With this list the Project Manager creates a first version of the new audit process definition. He does not just copy and paste the refined feedback results but inserts data bits and bullet points according to these. He understands it as a draft version and wants to develop it further at a later point in time. This version is basically a more precise version of the refined feedback results plus general input from his experience as a Project Manager. **PMA:** In order to do so, he opens the Logica Application in his browser and uses the main navigation to open the entity of concern, in this case ProcessDefinition. As he wants to enter a new data set, he clicks on "create new" on top of the page. **DxA:** The Project Manager might publish it directly to KiWi to get feedback on this draft. In this case, he jumps to the last step, then repeats the first step, and updates the data according to the comments.	102, 700, 705
Part 4 – Revision **KiWi:** Of course the data needed for the process definition is not just given through the feedback and the Project Manager's general knowledge. He has to take care about consistency with the process guidelines, resource planning and other related topics. As a process manager it is his job to provide a complete process. But he starts by formulating the bullet points to complete sentences and defining the data he just put roughly before. **DxA:** In the Logica Application he browses to the list of ProcessDefinition using the main navigation and clicks edit behind the entity created in step three. This leads him back to the editing screen where he can change the entries and clicks "Save" afterwards to persist his changes.	704

Table 25 – *Continued on next page...*

Continuing from previous page.

Work Step	Features
Part 5 – Final publication **DxA:** As the Project Manager considers the process to be defined and done, he publishes it. For that he has to open the DxA, chooses the template for process descriptions and then the audit process. By clicking the button "Publish" the data transfer to KiWi starts. After the successful import, he browses to the KiWi page with the new created process definition. The new process is ready for application. The publishing is handled in the DxA, it monitors the published data and checks whether published data has been changed. After starting it, the Project Manager has to ask for the Templates by clicking on "Fetch Templates". The DxA retrieves a list of templates and displays it. The Project Manager then chooses the ProcessDefinition template, which triggers the DxA to provide a list of all data sets the Logica Application contains, which are possible for import. The DxA shows a list of all the data fields that are affected for the import. The Project Manager chooses the row that he created in the Logica Application concerning the Audit Process and clicks on "Publish". After the DxA finished the download successfully, it provides the user the possibility to open the created page directly in a browser.	600, 601, 602, 603, 102

Table 25: Use Case 4, from (Grolin et al., 2010a)

C.5 Use Case 5: Data Access

Unlike the use cases above focuses this one on the evaluation of the information access within KiWi.

Test data, provided by Logica, was inserted into KiWi, and the test users have to find out details about it, without precise knowledge about it. During the test run, the users should act like they are not aware of the fact that KiWi already contains data of interest.

Tasks

The tasks are described in table 26.

Note: The whole evaluation takes part in the KiWi platform only.

Task	Description
Create a new UseCase	The project manager creates a page in KiWi to describe a use case. He puts a link on the project's overview page that leads to the use case. There he writes a short introduction to what this use case will be about and saves the changes. Afterwards he tags the page with freeform tags that represent the content that is already available and supposed to be collected on this page and on subpages. Whenever he finds pages of interest, he leaves comments or adds missing tags. He also links to important pages from his project to the related ones.
Finding through Tags	To find related pages, the project manager clicks on the created tags and checks the pages with the same tags.
Finding through Recommendations	The project manager goes back to the use case page he created and has a look at the KiWi recommendations in the sidebar, whether related pages are suggested.
Finding through Dashboard	As every user has a dashboard, he goes there, and tries to get help through the provided information.
Finding through Search	The project manager uses the search to find and filter the KiWi repository.

Table 26: Use Case 5, from (Grolin et al., 2010b)

Bibliography

Ivan Aaen, Jesper Arent, Lars Mathiassen, and Ojelanki Ngwenyama. A Conceptual MAP of Software Process Improvement. *Scandinavian Journal of Information Systems*, 13:81–101, 2001.

Russell L. Ackoff. From Data to Wisdom. *Journal Of Applied Systems Analysis*, 16:3–9, 1989.

Maryam Alavi and Dorothy E. Leidner. Review: Knowledge Management and Knowledge Management Systems: Conceptual Foundations and Research Issues. *MIS Quarterly*, 25(1):107–136, 2001.

William Albert, Thomas Tullis, and Donna Tedesco. *Beyond the Usability Lab: Conducting Large-Scale Online User Experience Studies*. Morgan Kaufman Publishers, 2010.

Klaus-Dieter Althoff, Frank Bomarius, and Carsten Tautz. Knowledge Management for Building Learning Software Organizations. *Information System Frontiers*, 2:349–367, 2000a.

Klaus-Dieter Althoff, Wolfgang Müller, Markus Nick, and Björn Snoek. KM-PEB: An Online Experience Base on Knowledge Management Technology. In *Knowledge Creation Diffusion Utilization*, pages 335–347, 2000b.

Javier Andrade, Juan Ares, Rafael García, Santiago Rodríguez, Andrés Silva, and Sonia Suárez. Knowledge Management Systems Development: A Roadmap. In *Proceedings of th 7th International Conference on Knowledge-Based Intelligent Information and Engineering Systems (KES)*, pages 1008–1015, 2003.

Jesper Arent and Jacob Nørbjerg. Software Process Improvement as Organizational Knowledge Creation: A Multiple Case Analysis. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, 2000.

Robert Audi. *Epistemology: A Contemporary Introduction to the Theory of Knowledge*. Routledge, 3rd edition, 2010.

Mirko Auerbach and Andreas Hauser. Process Oriented Knowledge Management – IT System and Case Study. In *Proceedings of the International Conference on Knowledge Management and Information Sharing*, 2009.

Aybüke Aurum, Farhad Daneshgar, and James Ward. Investigating Knowledge Management practices in software development organisations An Australian experience. *Information and Software Technology*, 50(6):511–533, May 2008.

Oguz N. Babüroglu and Ib Ravn. Normative Action Research. *Organization Studies*, 13(1):19–34, January 1992.

Jørgen P. Bansler and Erling C. Havn. Exploring the role of network effects in IT implementation: The case of knowledge management systems. In *Information Technology & People*, 2002.

Jørgen P. Bansler and Erling C. Havn. Sharing Best Practices: An Empirical Study of IT-Support for Knowledge Sharing. In *Proceedings of the 9th European Conference on Information Systems*, 2001.

Jørgen P. Bansler and Erling C. Havn. Building community knowledge systems: an empirical study of IT-support for sharing best practices among managers. *Knowledge and Process Management*, 10:156–163, 2003.

Jørgen P. Bansler and Erling C. Havn. Exploring the role of network effects in IT implementation: The case of knowledge repositories. *Information Technology & People*, 17(3):268–285, 2004.

Victor R. Basili. Software Development: A Paradigm for the Future. In *Proceedings of the 13th Annual International Computer Software and Applications Conference (COMPSAC'89)*, 1989.

Victor R. Basili. The Experience Factory and its relationship to other Improvement Paradigms. In *Proceedings of the ESEC'93*, pages 68–83, 1993.

Victor R. Basili. The Experience Factory and Its Relationship to Other Quality Approaches. *Advances in Computers*, 41:65–82, 1995.

Victor R. Basili. The Role of Experimentation in Software Engineering: Past, Current, and Future. In *Proceedings of ICSE-18*, pages 442–449, 1996.

Victor R. Basili and Gianluigi Caldiera. Methodological and Architectural Issues in the Experience Factory. In *Proceedings of the 16th Annual Software Engineering Workshop, NASA/GSFC*, pages 17–46, 1991.

Victor R. Basili and Gianluigi Caldiera. Improve Software Quality by Reusing Knowledge and Experience. *Sloan Management Review*, Fall, 1995.

Victor R. Basili and Scott Green. Software process evolution at the SEL. *IEEE Software*, 11:58–66, 1994.

Richard Baskerville, Jan Pries-Heje, and John R. Venable. Evaluation Risks in Design Science Research: A Framework. In *Proceedings of the 3rd International Conference on Design Science in Information Systems and Technology*, 2008.

Gene Bellinger, Durval Castro, and Anthony Mills. Data, Information, Knowledge, and Wisdom, 1997. URL `http://www.systems-thinking.org/dikw/dikw.htm`.

Izak Benbasat and Robert W. Zmud. Empirical Research in Information Systems: The Practice of Relevance. *MIS Quarterly*, 23(1):3–16, 1999.

Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, May 2001.

Finn Olav Bjørnson and Torgeir Dingsøyr. Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used. *Information and Software Technology*, 50(11):1055–1068, October 2008.

François Bry and Jakub Kotowski. D2.3 Reason Maintenance – State of the Art, 2008. URL `http://kiwi-project.eu/images/stories/deliverables/d2_3.pdf`. KiWi Project Deliverable.

François Bry and Jakub Kotowski. D2.4 Reason Maintenance – Concept and Model, 2009. URL `http://kiwi-project.eu/images/stories/deliverables/d2_4.pdf`. KiWi Project Deliverable.

François Bry and Klara Weiand. D2.2 Reasoning & Querying Concept and Model, 2009. URL `http://kiwi-project.eu/images/stories/deliverables/d2_2.pdf`. KiWi Project Deliverable.

François Bry and Klara Weiand. Flavors of KWQL, a Keyword Query Language for a Semantic Wiki. In *Theory and Practice of Computer Science (SOFSEM 2010)*, pages 247–258, 2010.

François Bry, Klara Weiand, and Tim Furche. D2.1 Reasoning & Querying – State of the Art, 2008. URL `http://kiwi-project.eu/images/stories/deliverables/d2_1.pdf`. KiWi Project Deliverable.

Alan Bryman. *Social Research Methods*. Oxford University Press, 3rd edition, 2008.

David J. Carney and Kurt C. Wallnau. A Basis for Evaluation of Commercial Software. *Information and Software Technology*, 40(14):851–860, December 1998.

Thomas Chau and Frank Maurer. A case study of wiki-based experience repository at a medium-sized software company. In *Proceedings of the 3rd International Conference on Knowledge Capture*, pages 185–186, 2005.

Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley Professional, Reading, Mass., 2000.

Reidar Conradi and Torgeir Dingsøyr. Software Experience Bases: A Consolidated Evaluation and Status Report. *Product Focused Software Process Improvement - Lecture Notes in Computer Science*, 1840:391–406, 2000.

John W. Creswell. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. Sage Publications, 3rd edition, 2009.

Thomas H. Davenport. Putting the Enterprise into the Enterprise System. *Harvard Business Review*, 76(4):121–131, 1998.

Thomas H. Davenport and L. Prusak. *Working Knowledge: How Organizations Manage What They Know.* Harvard Business School Press, Boston, MA, 1998.

Thomas H. Davenport, David W. de Long, and Michael C. Beers. Successful Knowledge Management Projects. *Sloan Management Review*, 39(2):43–57, 1998.

David de Almeida Ferreira and Alberto Manuel Rodrigues da Silva. An Enhanced Wiki for Requirements Engineering. In *Proceedings of the 35th Euromicro Conference on Software Engineering and Advanced Applications*, pages 87–94, 2009.

Björn Decker, Eric Ras, Jörg Rech, Bertin Klein, and Christian Hoecht. Self-organized Reuse of Software Engineering Knowledge Supported by Semantic Wikis. In *Proceedings of the Workshop on Semantic Web Enabled Software Engineering (SWESE) at ISWC*, 2005.

Kevin C. Desouza. Barriers to effective use of knowledge management systems in software engineering. *Communications of the ACM Magazine*, 46(1), 2003.

Kevin C. Desouza, Torgeir Dingsøyr, and Yukika Awazu. Experiences with conducting project postmortems: reports versus stories. *Software Process: Improvement and Practice*, 10(2):203–215, April 2005.

Nancy M. Dixon. *Common Knowledge: How Companies Thrive By Sharing What They Know.* Harvard Business School Press, Boston, 2000.

Peter Dolog, Frederico Durão, Daniel Grolin, Josef Holy, and Thomas Schandl. D6.1 KIWI Knowledge Model for Sun CEQ Use Case, 2009a. URL `http://kiwi-project.eu/images/stories/deliverables/d6_1.pdf`. KiWi Project Deliverable.

Peter Dolog, Frederico Durão, Daniel Grolin, Karsten Jahn, Peter Axel Nielsen, Andreas Munk-Madsen, and Keld Pedersen. D6.3 Knowledge Model: Project Knowledge Management, 2009b. URL `http://kiwi-project.eu/images/stories/deliverables/d6_3.pdf`. KiWi Project Deliverable.

Peter Dolog, Frederico Durão, Karsten Jahn, Keld Pedersen, Marek Schmidt, Rolf Sint, and Stephanie Stroka. D6.4 Implementation: Project Knowledge Management, 2009c. URL `http://kiwi-project.eu/images/stories/deliverables/d6_4.pdf`. KiWi Project Deliverable.

Peter Dolog, Markus Krötzsch, Sebastian Schaffert, and Denny Vrandečić. *Social Web and Knowledge Management*, pages 217–227. Springer Berlin Heidelberg, 2009d.

Peter Dolog, Frederico Durão, Karsten Jahn, Lin Yujian, and Dennis Kjæ rsgaard Peitersen. Recommending Open Linked Data in Creativity Sessions using Web Portals with Collaborative Real Time Environment. *Journal of Universal Computer Science*, 17(12):1690–1709, 2011.

Line Dubé and Guy Paré. Rigor in Information Systems Positivist Case Research: Current Practices, Trends, and Recommendations. *MIS Quarterly*, 27(4):597–636, 2003.

Joseph S. Dumas and Beth A. Loring. *Moderating Usability Tests: Principles and Practices for Interacting*. Morgan Kaufman Publishers, 2008.

Frederico Durão and Peter Dolog. Analysis of Tag-Based Recommendation Performance for a Semantic Wiki. In *Proceedings of 4th Workshop on Semantic Wikis (SemWiki2009) in conjunction with the 6th Annual European Semantic Web Conference (ESWC2009)*, 2009a.

Frederico Durão and Peter Dolog. D2.8 Personalisation – Concept and Model, 2009b. URL `http://kiwi-project.eu/images/stories/deliverables/d2_8.pdf`. KiWi Project Deliverable.

Frederico Durão and Peter Dolog. A personalized tag-based recommendation in social web systems. In *Workshop on Adaptation and Personalization for Web 2.0 (UMAP'09)*, pages 40–49, 2009c.

Frederico Durão, Peter Dolog, and Karsten Jahn. D2.7 Personalisation – State of the Art, 2008. URL `http://kiwi-project.eu/images/stories/deliverables/d2_7.pdf`. KiWi Project Deliverable.

J. Eekels and N. F. M. Roozenburg. A methodological comparison of the structures of scientific research and engineering design: their similarities and differences. *Design Studies*, 12(4):197–203, 1991.

Ricardo A. Falbo, Daniel O. Arantes, and Ana C. C. Natali. Integrating Knowledge Management and Groupware in a Software Development Environment. *Practical Aspects of Knowledge Management*, 3336:94–105, 2004.

Péter Fehér and András Gábor. The role of Knowledge Management Supporters in software development companies. *Software Process: Improvement and Practice*, 11(3):251–260, 2006.

Lee Feigenbaum, Ivan Herman, Tonya Hongsermeier, Eric Neumann, and Susie Stephens. The Semantic Web in Action. *Scientific American*, December 2007.

Abílio Fernandes, Ana Maria de C. Moura, and Fábio Porto. An ontology-based approach for organizing sharing, and querying knowledge objects on the Web. In *Proceedings of the 14th International Workshop on Database and Expert Systems Applications*, pages 604–609, 2003.

Martin Glisby and Nigel Holden. Contextual constraints in knowledge management theory: the cultural embeddedness of Nonaka's knowledge-creating company. *Knowledge and Process Management*, 10(1):29–36, January 2003.

Asunción Gómez Pérez and V. Richard Benjamins. Overview of Knowledge Sharing and Reuse Components: Ontologies and Problem-Solving Methods. In *Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods*, pages 1–15, 1999.

Jennifer Gonzalez-Reinhart. Wiki and the Wiki Way: Beyond a Knowledge Management Solution. *Information Systems Research Center*, pages 1–22, 2005.

Shirley Gregor. The Nature of Theory in Information Systems. *MIS Quarterly*, 30(3):611–642, 2006.

Daniel Grolin, Karsten Jahn, Peter Axel Nielsen, and Keld Pedersen. D7.2 Test Plan: Logica Use Case, 2010a. KiWi Project Deliverable.

Daniel Grolin, Karsten Jahn, Peter Axel Nielsen, and Keld Pedersen. D7.4 Evaluation and Testing Project Knowledge Management, 2010b. URL `http://kiwi-project.eu/images/stories/deliverables/d7-4.pdf`. KiWi Project Deliverable.

Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human Computer Studies*, 43: 907–928, 1995.

Michel Grundstein. Distinguishing knowledge from information: A prerequisite for elaborating KM initiative strategy. In *Proceedings of the 1st Conference on Knowledge Management and Information Sharing*, 2009.

Wendong Gu, Guoping Xia, and Weijia You. Enterprise Knowledge Integration by Semantic Web. In *International Federation for Information Processing*, pages 203–212, 2006.

Jungpil Hahn and Mani R. Subramani. A framework of knowledge management systems: issues and challenges for theory and practice. In *Proceedings of the 21st International Conference on Information Systems (ICIS '00)*, pages 302–312, 2000.

Bo Hansen, Jeremy Rose, and Gitte Tjørnehøj. Prescription, description, reflection: the shape of the software process improvement field. *International Journal of Information Management*, 24(6):457–472, December 2004.

Morten T. Hansen, Nitin Nohria, and Thomas Tierney. What's Your Strategy for Managing Knowledge? *Harward Business Review*, 77(2):106–116, 1999.

Morten T. Hansen, Hall Soldiers, and Field Park. Knowledge Networks: Explaining Effective Knowledge Sharing in Multiunit Companies. *Organization Science*, 13(3):232–248, 2002.

Helen Hasan and Charmaine C. Pfaff. The Wiki: An Environment to Revolutionise Employees' Interaction with Corporate Knowledge. *Proceedings of the Australian Conference on Computer-Human Interaction (OZCHI'06)*, pages 377–380, November 2006.

Helen Hasan and Charmaine C. Pfaff. Democratising Organisational Knowledge: The Potential of the Corporate Wiki. In *Proceedings of the International Conference on Information Systems (ICIS'07)*, pages 1–19, 2007.

Scott Henninger. Case-Based Knowledge Management Tools for Software Development. *Automated Software Engineering*, 4(3):319–340, 1997.

Alan R. Hevner. A Three Cycle View of Design Science Research. *Scandinavian Journal of Information Systems*, 19(2):87–92, 2007.

Alan R. Hevner and Samir Chatterjee. Design Science Research in Information Systems. In *Design Research in Information Systems: Theory and Practice*, chapter 2, pages 9–22. Springer, Boston, MA, 2010.

Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105, 2004.

Lorin M. Hitt, D. J. Wu, and Xiaoge Zhou. Investment in enterprise resource planning: Business impact and productivity measures. *Journal of Management Information Systems*, 19(1):71–98, 2002.

Josef Holy, Peter Reiser, and Jakub Franc. D5.2 Requirements Software Knowledge Management, 2008. URL `http://kiwi-project.eu/images/stories/deliverables/d5_2.pdf`. KiWi Project Deliverable.

Josef Holy, Peter Reiser, and Thomas Schandl. D6.2 Application Building Report Sun Expert Manager Use Case, 2010. URL `http://kiwi-project.eu/images/stories/deliverables/d6_2.pdf`. KiWi Project Deliverable.

Jens Henrik Hosbond and Peter Axel Nielsen. Software Process Improvement using Light Knowledge Tools. In Peter Axel Nielsen and Karlheinz Kautz, editors, *Software Processes & Knowledge*, chapter 7. Software Innovation Publisher, Aalborg, 2008.

Watts S. Humphrey. *Managing the software process*. Addison-Wesley, Reading, 1989.

Juhani Iivari. A Paradigmatic Analysis of Information Systems As a Design Science. *Scandinavian Journal of Information Systems*, 19(2):39–64, 2007.

Jakob H. Iversen, Peter Axel Nielsen, and Jacob Nørbjerg. Situated assessment of problems in software development. *SIGMIS Database*, 30(2):66–81, 1999.

Stefan Jablonski, Stefan Horn, and Michael Schlundt. Process oriented knowledge management. In *Proceedings of the 11th International Workshop on Research Issues in Data Engineering (RIDE 2001)*, pages 77–84. IEEE Comput. Soc, 2001.

Karsten Jahn and Peter Axel Nielsen. Codified vs. Personalized - A Vertical Approach to the Dilemma of the Knowledge Management Strategies. In *17th EuroSPI Conference: Industrial Proceedings*, pages 3.11 – 3.20, 2010.

Karsten Jahn and Peter Axel Nielsen. A Vertical Approach to Knowledge Management: Codification and Personalization in Software Processes. *International Journal of Human Capital and Information Technology Professionals*, 2(2):26–36, 2011.

Pertti Järvinen. *On Research Methods*. Opinpajan Kirja, Tampere, Finland, 2000.

Guillermo Jimenez and Carlos Barradas. Knowledge Management System Based on Web 2.0 Technologies. In Jing Tao Yao, editor, *Web-Based Support Systems*, chapter 13, pages 273–301. Springer London, London, 2010.

Luiz Antonio Joia and Bernardo Lemos. Tacit Knowledge Transfer within Organisations. In *AMCIS 2010 Proceedings*, 2010.

Vijay Kasi, Mark Keil, Lars Mathiassen, and Keld Pedersen. The post mortem paradox: a Delphi study of IT specialist perceptions. *European Journal of Information Systems*, 17(1):62–78, February 2008.

Karlheinz Kautz. Software process improvement in very small enterprises: Does it pay off? *Software Process: Improvement and Practice*, 4(4):209–226, 1998.

Karlheinz Kautz. Making Sense of Measurements for Small Organizations. *IEEE Software*, 16(2):14–20, April 1999.

Karlheinz Kautz and Bo Hansen Hansen. Mapping Knowledge Flows. In Peter Axel Nielsen and Karlheinz Kautz, editors, *Software Processes & Knowledge*, chapter 6, pages 89–102. Software Innovation Publisher, 2008.

Karlheinz Kautz and Annemette Kjærgaard. Knowledge Sharing in Software Development. In Peter Axel Nielsen and Karlheinz Kautz, editors, *Software Processes & Knowledge*, chapter 4, pages 43–68. Software Innovation Publisher, Aalborg, 2008.

Karlheiz Kautz and Kim Thaysen. Knowledge, learning and IT support in a small software company. *Journal of Knowledge Management*, 5(4):349–357, 2001.

Dan J. Kim and T. Andrew Yan. A New Approach for Collaborative Knowledge Management: A Unified Conceptual Model for Collaborative Knowledg Management. In *AMCIS 2010 Proceedings*, 2010.

Annemette Kjærgaard, Peter Axel Nielsen, and Karlheinz Kautz. Making Sense of Software Project Management: A Case of Knowledge Sharing in Software Development. *Scandinavian Journal of Information Systems*, 22(1):3–26, 2010.

Petr Knoth, Marek Schmidt, and Pavel Smrž. D2.5 Information Extraction – State of the Art, 2008. URL `http://kiwi-project.eu/images/stories/deliverables/d2_5.pdf`. KiWi Project Deliverable.

Seija Komi-Sirviö, Annukka Mäntyniemi, and Veikko Seppänen. Toward a practical solution for capturing knowledge for software projects. *IEEE Software*, 19:60–62, 2002.

Robert E. Kraut and Lynn A. Streeter. Coordination in software development. *Communications of the ACM*, 38(3), 1995.

Clif Kussmaul and Roger Jack. Wikis for Knowledge Management: Business Cases, Best Practices, Promises, & Pitfalls. In Miltiadis D Lytras, Ernesto Damiani, and Patricia Ordóñez de Pablos, editors, *Web 2.0: The Business Model*, chapter 9, pages 147–165. Springer US, 2009.

Steinar Kvale. *Interviews: An Introduction to Qualitative Research Interviewing*. Sage Publications, Thousand Oaks, CA, USA, 1996.

M. M. Kwan and P. Balasubramanian. Process-oriented knowledge management: A case study. *Journal of the Operational Research Society*, 54(2): 204–211, 2003.

Giovan Francesco Lanzara and Lars Mathiassen. Mapping situations within a system development project. *Information & Management*, 8(1):3–20, 1985.

Craig Larman. *Agile and Iterative Development: A Manager's Guide*. Addison Wesley, Boston, MA, 2003.

Bo Leuf and Ward Cunningham. *The Wiki Way: Collaboration and Sharing on the Internet*. Addison-Wesley, 2001.

Jay Liebowitz and Isaac Megbolugbe. A set of frameworks to aid the project manager in conceptualizing and implementing knowledge management initiatives. *International Journal of Project Management*, 21(3):189–198, April 2003.

Mikael Lindvall, Michael Frey, Patricia Costa, and Roseanne Tesoriero. Lessons Learned about Structuring and Describing Experience for Three Experience Bases. *LSO2001, Lecture Notes in Computer Science*, 2176:106–119, 2001.

Champika Liyanage, Taha Elhag, Tabarak Ballal, and Qiuping Li. Knowledge communication and translation – A knowledge transfer model. *Journal of Knowledge Management*, 13(3):118–131, 2009.

Panagiotis Louridas. Using Wikis in Software Development. *IEEE Software*, 23 (2):88–91, 2006.

Stewart Mader. *Wikipatterns*. Wiley, 2007.

Ann Majchrzak, Christian Wagner, and Dave Yates. Corporate Wiki Users: Results of a Survey. In *Proceedings of the 2006 international symposium on Wikis (WikiSym'06)*, pages 99–104, 2006.

Salvatore T. March and Gerald F. Smith. Design and natural science research on information technology. *Decision Support Systems*, 15(4):251–266, December 1995.

Lars Mathiassen. Collaborative Practice Research. *Information Technology & People*, 15(4):321–345, 2002.

Lars Mathiassen and Keld Pedersen. The Dynamics of Knowledge in Systems Development Practice. In *Proceedings of the 38th Hawaii International Conference on system Sciences*, pages 1–11, 2005.

Lars Mathiassen and Pouya Pourkomeylian. Managing knowledge in a software organization. *Journal of Knowledge Management*, 7(2):63–80, 2003.

Lars Mathiassen, Andreas Munk-Madsen, Peter Axel Nielsen, and Jan Stage. *Object-Oriented Analysis and Design*. Marko Publisher, Aalborg, 2000.

Lars Mathiassen, Jan Pries-Heje, and Ojelanki Ngwenyama. *Improving Software Organizations: From Principles to Practice*. Addison-Wesley, Upper Saddle River, NJ, USA, 2002.

James C. McDavid and Laura R. L. Hawthorn. *Program Evaluation and Performance Measurement: An Introduction to Practice*. Sage Publications, Inc, 2006.

Richard McDermott. Why information technology inspired but cannot deliver knowledge management. *California Management Review*, 41(4):103, 1999.

Bob McFeeley. *IDEAL: A User's Guide for Software Process Improvement.* Carnegie Mellon University, Pittsburgh, PA, February 1996.

James D. McKeen and Heather A. Smith. Social Networks: Knowledge Management's "Killer App"? *Communications of the Association for Information Systems*, 19(1), 2007.

Bridget Meehan and Ita Richardson. Identification of Software Process Knowledge Management. *Software Process: Improvement and Practice*, 7(2):47–55, 2002.

Peter P. Mitchell. *A Step-by-Step Guide to Usability Testing.* iUniverse, Inc., 2007.

Audris Mockus, Roy T. Fielding, and James D. Herbsleb. Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 11(3):309–346, July 2002.

Tsuyoshi Moriya and Caroline Benton. Knowledge sharing and creation in the semiconductor equipment industry. In *International Symposium on Semiconductor Manufacturing (ISSM)*, Tokyo, Japan, 2008.

Sune Dueholm Müller, Peter Axel Nielsen, and Jacob Nørbjerg. Standards, Processes and Practice. In Peter Axel Nielsen and Karlheinz Kautz, editors, *Software Processes & Knowledge*, chapter 3, pages 29–42. Software Innovation Publisher, Aalborg, 2008.

Peter Axel Nielsen and Peter Dolog. D5.3 State-of-the-Art on Software Project Management Knowledge, 2008. URL `http://kiwi-project.eu/images/stories/deliverables/d5_3.pdf`. KiWi Project Deliverable.

Peter Axel Nielsen and Gitte Tjørnehøj. Social networks in software process improvement. *Journal of Software Maintenance and Evolution: Research and Practice*, 22:33–51, 2009.

Peter Axel Nielsen, Peter Dolog, Daniel Grolin, Karsten Jahn, Andreas Munk-Madsen, Andreas Grauber, François Bry, and Keld Pedersen. D5.4 Requirements Project Knowledge Management, 2008. URL `http://kiwi-project.eu/images/stories/deliverables/d5_4.pdf`. KiWi Project Deliverable.

Ikujiro Nonaka. The Knowledge Creating Company. *Harvard Business Review*, pages 96–104, 1991.

Ikujiro Nonaka. A Dynamic Theory of Organizational Knowledge Creation. *Organization Science*, 5:14–37, 1994.

Ikujiro Nonaka and Hirotaka Takeuchi. *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation.* Oxford University Press, New York, 1995.

Eyal Oren, Max Völkel, John G. Breslin, and Stefan Decker. Semantic Wikis for Personal Knowledge Management. In *Proceedings of the 17th International Conference on Database and Expert Systems Applications*, 2006.

Wanda J. Orlikowski and C. Suzanne Iacono. The Truth Is Not Out There: An Enacted View Of The "Digital Economy". In Erik Brynjolfsson and Brian Kahin, editors, *Understanding the Digital Economy: Data, Tools, and Research*, pages 352–380. MIT Press, Cambridge, MA, 2000.

Wanda J. Orlikowski and C. Suzanne Iacono. Desperately Seeking the IT in IT Research: A Call to Theorizing the IT Artifact. *Information Systems Research*, 12(2):121–134, 2001.

David L. Parnas and Paul C. Clements. A rational design process: How and why to fake it. In *Formal Methods and Software Development*, volume 186 of *Lecture Notes in Computer Science*, pages 80–100. Springer Berlin / Heidelberg, 1985.

Keld Pedersen. *Managing Learning in Systems Development Projects*. Phd thesis, Computer Science Department, Aalborg University, Denmark, 2005.

Ken Peffers, Tuure Tuunanen, Marcus A Rothenberger, and Samir Chatterjee. A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3):45–77, 2008.

Brian T. Pentland. Information Systems and Organizational Learning: The Social Epistemology of Organizational Knowledge Systems. *Accounting, Management and Information Technologies*, 5(1):1–21, January 1995.

John Stouby Persson, Lars Mathiassen, Jesper Boeg, Thomas Stenskrog Madsen, and Flemming Steinson. Managing Risks in Distributed Software Projects: An Integrative Framework. *IEEE Transactions on Engineering Management*, 56(3):508–532, August 2009.

Platon. *Theätet*. Reclam Verlag, 1986.

Rob Poell and Ferd van der Krogt. Learning strategies of workers in the knowledge-creating company. *Human Resource Development International*, 6(3):387–403, 2001.

Michael Polanyi. *The Tacit Dimension*. Routledge & Kegan Paul, London, 1966.

Roger S. Pressman. *Software engineering: A Practitioner's Approach (European Adaption)*. McGraw-Hill International, 5th edition, 2000.

Murali Raman. Wiki Technology as A "Free" Collaborative Tool within an Organizational Setting. *Information Systems Management*, 23:59–66, 2006.

Balasubramaniam Ramesh and Amrit Tiwana. Supporting Collaborative Process Knowledge Management in New Product Development Teams. *Decision Support Systems*, 27(1-2):213–235, November 1999.

Axel Rauschmayer. Next Generation Wikis: What Users Expect; How RDF Helps. In *Proceedings of 3rd Semantic Wiki Workshop at ESWC*, 2009.

T. Ravichandran and Arun Rai. Structural analysis of the impact of knowledge creation and knowledge embedding on software process capability. *Engineering*, 50(3):270–284, 2003.

Jochen Reutelshoefer, Joachim Baumeister, and Frank Puppe. Ad-Hoc Knowledge Engineering with Semantic Knowledge Wikis. In *Proceedings of the 3rd Semantic Wiki Workshop (SemWiki 2008) at the 5th European Semantic Web Conference (ESWC 2008)*, 2008.

Jennifer Rowley. The wisdom hierarchy: representations of the DIKW hierarchy. *Journal of Information Science*, 33(2):163–180, 2007.

Jeffrey Rubin and Dana Chisnell. *Handbook of Usability Testing: How to plan, design and conduct effective tests*. Wiley, 2nd edition, 2008.

Ioana Rus and Mikael Lindvall. Knowledge Management in Software Engineering. *IEEE Software*, 19(3):26–38, June 2002.

Matthias Samwald. D5.1 State of the Art Software Knowledge Management, 2008. URL `http://kiwi-project.eu/images/stories/deliverables/d5_1.pdf`. KiWi Project Deliverable.

Sebastian Schaffert. IkeWiki: A Semantic Wiki for Collaborative Knowledge Management. In *Proceedings of the 15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE '06)*, 2006.

Sebastian Schaffert, François Bry, Joachim Baumeister, and Malte Kiesel. Semantic Wikis. *IEEE Software*, 25(4):8–11, 2008a.

Sebastian Schaffert, François Bry, Peter Dolog, Julia Eder, Szaby Grünwald, Jana Herwig, Jozef Holy, Peter Axel Nielsen, and Pavel Smrž. D8.5 KiWi Vision, 2008b. URL `http://kiwi-project.eu/images/stories/deliverables/d8_5.pdf`. KiWi Project Deliverable.

Sebastian Schaffert, Rolf Sint, Szaby Grünwald, and Stephanie Stroka. D3.1 Architecture Revision, 2008c. URL `http://kiwi-project.eu/images/stories/deliverables/d3_1.pdf`. KiWi Project Deliverable.

Sebastian Schaffert, Julia Eder, Szaby Grünwald, Thomas Kurz, Mihai Radulescu, Rolf Sint, and Stephanie Stroka. KiWi – A Platform for Semantic Social Software. In *Proceedings of the 4th Workshop on Semantic Wikis*. CEUR-WS, 2009.

Joost Schalken, Sjaak Brinkkemper, and Hans van Vliet. A method to draw lessons from project postmortem databases. *Software Process: Improvement and Practice*, 11(1):35–46, January 2006.

Florian Schmedding, Christoph Hanke, and Thomas Hornung. RDF Authoring in Wikis. In *Proceedings of the 3rd Semantic Wiki Workshop (SemWiki 2008) at the 5th European Semantic Web Conference (ESWC 2008)*, 2008.

Marek Schmidt and Pavel Smrž. D2.6 Information Extraction – Concept and Model, 2009. URL `http://kiwi-project.eu/images/stories/deliverables/d2_6.pdf`. KiWi Project Deliverable.

Maung K. Sein, Ola Henfridsson, Sandeep Purao, Matti Rossi, and Rikard Lindgren. Action Design Research. *MIS Quarterly*, 35(1):37–56, 2011.

Nigel Shadbolt, Wendy Hall, and Tim Berners-Lee. The Semantic Web Revisited. *IEEE Intelligent Systems*, 21(3):96–101, May 2006.

Sofia Sherman, Irit Hadar, and Meira Levy. Enhancing Software Architecture Review Process via Knowledge Management. In *AMCIS 2010 Proceedings*, 2010.

Sajjan G. Shiva and Lubna A. Shala. Using Semantic Wikis to Support Software Reuse. *Journal of Software*, 3(4), 2008.

Rolf Sint, Sebastian Schaffert, Stephanie Stroka, and Roland Ferstl. Combining Unstructured, Fully Structured and Semi-Structured Information in Semantic Wikis. In *Fourth Workshop on Semantic Wikis (ESWC2009)*, pages 1–15, 2009.

Ian Sommerville. *Software Engineering*. Addison-Wesley, 6th edition, 2001.

Fernando Sousa, Manuela Aparicio, and Carlos J. Costa. Organizational Wiki as a Knowledge Management Tool. In *Proceedings of the 28th ACM International Conference on Design of Communication (SIGDOC '10)*, 2010.

Karl Erik Sveiby. *The New Organizational Wealth: Managing & Measuring Knowledge-Based Assets*. Berrett-Koehler Publishers, Inc, San Francisco, CA, USA, 1997.

Jacky Swan, Sue Newell, Harry Scarbrough, and Donald Hislop. Knowledge management and innovation: Networks and Networking. *Journal of Knowledge Management*, 3(4):262–275, 1999a.

Jacky Swan, Harry Scarbrough, and John Preston. Knowledge management - The next fad to forget people? In *Proceedings of the 7th European Conference on Information Systems, (ECIS'1999)*, pages 668–678, 1999b.

Amrit Tiwana. An empirical study of the effect of knowledge integration on software development performance. *Information and Software Technology*, 46 (13):899–906, October 2004.

Oezguer Uenalan, Norman Riegel, Sebastian Weber, and Joerg Doerr. Using Enhanced Wiki-based Solutions for Managing Requirements – Solution Concepts to Support Basic Practices of Requirements Engineering. In *Proceedings of the 1st International Workshop on Managing Requirements Knowledge (MARK'08)*, pages 63–67, 2008.

Elisabeth J. Umble, Ronald R. Haft, and M. Michael Umble. Enterprise resource planning: Implementation procedures and critical success factors. *European Journal of Operational Research*, 146(2):241–257, April 2003.

Andrew H. van de Ven. *Engaged Scholarship: A Guide for Organizational and Social Research*. Oxford University Press, Oxford, UK, 2007.

Georg von Krogh. Care in Knowledge Creation. *California Management Review*, 4(3):133–154, 1998.

Christiane Gresse von Wangenheim, Klaus-Dieter Althoff, and Ricardo M. Barcia. Goal-oriented and similarity-based retrieval of software engineering experienceware. *Learning Software Organizations, LNCS*, 1756:118–141, 2000.

Christian Wagner. Wiki: A Technology for conversational Knowledge Management and Group Collaboration. *Communications of the Association for Information Systems*, 13:265–289, 2004.

Geoff Walsham. Knowledge Management Systems: Representation and Communication in Context. *International Journal*, 1(1):6–18, 2005.

Diane B. Walz, Joyce J. Elam, and Bill Curtis. Inside a Software Design Team: Knowledge Acquisition, Sharing, and Integration. *Communications of the ACM*, 36(10), 1993.

Robert P. Ward, Mohamed Ebrahim Fayad, and Mauri Laitinen. Software Process Improvement in the Small. *Communications of the ACM*, 44:105–107, 2001.

Chih-Ping Wei, Paul Jen-hwa Hu, and Hung-Huang Chen. Design and Evaluation of a Knowledge Management System. *IEEE Software*, pages 56–59, June 2002.

Steve Wheeler, Peter Yeomans, and Dawn Wheeler. The good, the bad and the wiki: Evaluating student-generated content for collaborative learning. *British Journal of Educational Technology*, 39(6):987–995, November 2008.

Roel Wieringa. Design science as nested problem solving. In *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology (DESRIST '09)*, 2009.

Robert Winter. Design Science Research in Europe. *European Journal of Information Systems*, 17(5):470–475, October 2008.

ZhiXin Wu. Research on tacit knowledge sharing in the Outsourcing enterprises: A case study of H company in Hangzhou. In *Proceedings of the International Converence on Educational and Information Technology (ICEIT2010)*, pages 270–274, 2010.

Chaim Zins. Conceptual Approaches for Defining Data, Information, and Knowledge. *Journal of the American Society for Information Science and Technology*, 58(4):479–493, 2007.