Master Thesis
Software Engineering
Thesis no: MSE-2004:28
August 2004

Identifying and Analyzing Knowledge Management Aspects of Practices in Open Source Software Development

Michal Przemyslaw Rudzki, Fredrik Jonson

School of Engineering
Blekinge Institute of Technology
Box 520
SE - 372 25 Ronneby
Sweden

This thesis is submitted to the School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full time studies.

Contact Information

Authors:

Michal Przemyslaw Rudzki
erace at erace.pl
P.O. Box 715, 42-200 Czestochowa, Poland

Fredrik Jonson
fredrik at jonson.org
Karlskronagatan 23a, SE - 372 37 Ronneby, Sweden

University advisor:

Conny Johansson
Department of Software Engineering and Computer Science

School of Engineering
Blekinge Institute of Technology
Box 520
SE - 372 25 Ronneby
Sweden

Abstract

In this thesis we explore how knowledge management is performed in open source projects. Open source projects are often perceived as informal, even unmanaged. Still, they appear to manage knowledge acquisition and sharing sufficiently well to successfully develop software in such a distributed environment as the Internet. This thesis aims to explore that apparent contradiction, and thus complement the currently limited research in this field.

The thesis consists of a literature study of knowledge management theory and open source development, resulting in the analysis of open source practices from a knowledge management perspective.

Currently the field of knowledge management maintains several, partially opposing doctrines. Apart from the business aspect, two main schools of thought are present. The commodity school approaches knowledge as a universal truth, an object that can be separated from the knower. The community school emphasises knowledge as something internal to the human mind, but which can be shared as experiences between people. In the analysis presented, we have applied an analysis method which considers both the commodity and the community perspectives.

The analysis is based on previous research studies of open source, and open source practices, and is furthered by a cursory case study using examples from a selected set of open source projects.

Our conclusions are that knowledge management indeed is present in open source projects, and that it is supported by an ecology like interaction of project practices.

Keywords: knowledge management, open source development, software engineering, codification, personalization.

Abbreviations

ASF	Apache Software Foundation
CMS	Content Management System
CVS	Concurrent Versions System
DIKW	Data Information Knowledge Wisdom
FAQ	Frequently Asked Question
FTP	File Transfer Protocol
GCC	GNU Compiler Collection
GNU	GNU is Not Unix
HTML	HyperText Markup Language
IRC	Internet Relay Chat
IT	Information Technology
KM	Knowledge Management
KMS	Knowledge Management System
LDP	Linux Documentation Project
OSS	Open Source Software
OSD	Open Source Development
RSS	Really Simple Syndication
SE	Software Engineering
WYSIWYG	What You See Is What You Get
XML	Extensible Markup Language

Contents

The major problem with intellectual capital is that it has legs and walks home every day.
– Ioana Rus and Mikael Lindvall

Chapter 1

Introduction

Both open source software development and knowledge management have been subject to substantial growth during the previous decade. The attention towards knowledge management started in the beginning of 1990's (Hansen, Nohria & Tierney 1999) about the same time as Linus Torvalds released the source code of the kernel of the operating system Linux to the Internet community. This boosted the growth of the already existing open source movement (Raymond 2000).

It is challenging to define *knowledge management*, however, most agree that it is concerned with the *collection and dissemination of knowledge to the benefit of an organization and its individuals* (Lueg 2001). In the year 2001, eighty percent of the largest global corporations had knowledge management projects (Lawton 2001). Similarly Linux impacts the society; for the year 2004 the estimated number of computers running Linux as operating system is estimated to be 18 millions (Alvestrand 2004, IBM 2004).

Open source development is typically a collaborative effort in which programmers improve upon the code and share their changes with the community (Webopedia 2004). All practices performed in open source have evolved naturally. It was the need to do things in the most effective way that drove this evolution. Having the only possibility of communication limited to the Internet, without face-to-face communication to be able to work, it was necessary to create appropriate tools and practices. Unlike in traditional software development, where the schedules are very tight and managers are stressed by deadlines and budgets, in open source community nobody is in a hurry. Members of this society share the results of their work voluntarily and at undefined times. Software is released at the time when project owners are satisfied with its maturity and stability (Godfrey & Qiang 2000). Having time and driven by necessity, they keep improving the way they work individually and in groups by creation of new tools, optimization of communication schemes and processes. In addition, as Eric Raymond, one of the fathers of open source, recognized in his book *The Cathedral and the Bazaar* (Raymond 2000), *"Every good work of software starts by scratching a developer's personal itch"*.

Over the years, the nature of projects delivering software has been studied. The findings show that software creation is a knowledge-intensive process (Basili, Caldiera & Rombach 2002). Obtaining knowledge about new technologies and product domain, sharing knowledge about local policies and practices, realizing who knows what, are all activities that are carried out during almost each software project (Rus & Lindvall 2002).

An interesting question arises how the evolutionary approach of open source development has catered for knowledge management. It is reasonable to think that knowledge management has evolved, just as any other managerial aspects has. The difference between knowledge management and other project practices, is that knowledge management often takes place implicitly, as a consequence of other more technical practices. There are only a few studies that explicitly consider knowledge management in open source projects. With this thesis we hope to complement those.

1.1 Motivation

Below we present more detailed reasons for which we believe an assessment of open source development from knowledge management perspective is an interesting and worthy task. First, we give reasons for which open source project and people working on it should be interested in conscious knowledge management. Second, we try to give some basic reasons for which commercial software vendors should consider the subject.

1.1.1 Open source developers

The nature of open source, which will be discussed in more details in chapter 5, is that the individuals working on a project change during the project's lifetime. They voluntarily join and leave at unpredictable times. The people willing to contribute have to learn about many different aspects that are unique to each project. Unlike in companies developing software on site, where developers have a chance to meet together, talk face-to-face and share their knowledge, in open source projects it is necessary to grasp every possible aspect of the projects in an explicit from. The need to obtain knowledge about local policies and practices which is typical for all software engineering projects (Rus & Lindvall 2002) naturally also have to occur in open source development.

Unlike in the defined, traditional development with clearly defined parts, roles and schedules, in open source programmers tend to work on each others code in a nondeterministic way. This requires them to constantly learn and understand the already implemented designs, mechanisms, algorithms, etc. Identifying the people that have developed a given part of application for consultation and future reference (Rus & Lindvall 2002), making this knowledge explicit becomes important.

Another reason is related to the intended competitiveness. The competition with proprietary software vendors is a widely recognized fact (Dougherty 2001). On the other hand, research has shown that the competitiveness of a project can be improved by reducing the amount of work which has to be redone when people leave a project, with the assistance of a conscious knowledge management process (Bollinger 1999).

1.1.2 Commercial perspective

Looking from the perspective of commercial software vendors there are a number of factors that make managing knowledge in open source development an issue worth exploring. The software development companies want to provide customers with solutions employing certain widely used technologies. If they do not have in-house developed components they have basically two options: either obtain those parts – usually closed source – from another software vendor or develop the whole solution from scratch using their own resources, which is very often expensive. Only Linux kernel itself, version 2.6.0, in December 2003 had close to six million lines of code. The biggest Open Source software development website – SourceForge – currently hosts almost eighty thousand projects (Sourceforge 2004). Having a such great amount of already developed open source code available, companies use it as components when building their applications (Hissam & Weinstock 2001). One spectacular and recent example of such use of open source software is the *Science Activity Planner* project, which is a tool for NASA's *Mars Exploration Rover Mission*. It uses eight open source components for critical mission's operations tasks (Norris 2004). For this practice to be possible, the code that is the subject to reuse must be accompanied by a sufficient amount of knowledge. By encouraging knowledge management activities like learning about available and appropriate technologies, functionality and quality of a given solution, recognizing experts in the area can be easier and more effective.

Another commercial aspect is related to the tendency among big software vendors to outsource part of their closed source application to open source community. One such aproach is Progressive Open Source, which was developed by Hewlett Packard (Dinkelacker, Garg, Miller & Nelson 2002). It defines three layers in which application is developed and source code is shared: inner source, controlled source and open source. The first one refers to the corporate environment, second to partners and third to completely open Internet environment. IBM, Hewlett Packard, SUN Microsystems, BEA and others put this idea to use. For example Apple is using FreeBSD operating system as the foundation for their Mac OS X, focusing on delivery of hardware and friendly user interface. Real Networks, created Helix Community which has the purpose of delivering open, multi-format platform for digital media creation, delivery and playback. This development method makes the issue of collecting and sharing the knowledge in open source community even more important.

1.2 Who is the intended audience

In view of our own experiences, which was the motivation for choosing this topic, we consider our intended audience to be software developers and project managers, that are either already working or will soon start to work with open source projects, and want to enhance their knowledge about how to perform open source project and how to approach knowledge management in open source projects.

Ideally, the knowledge of an open source project is easily identifiable, attainable and supportable, so that the project is attractive to developers – both those already acquainted with the project, and newcomers. Within this thesis we try to describe how to take one step towards that goal. At the same time, we want to remind our readers that this thesis is not an effort to cover all bases, but only describes one aspect of open source projects.

1.3 Roadmap

This thesis is organised as follows:

Chapter 1: Introduction. This chapter. Here we introduced the domains of knowledge management and open source development, presented the reasons why we consider the combination of the two interesting, and discussed our motivations for this thesis.

Chapter 2: Thesis Goals and Methodology. In the second chapter, we describe our goals, and the questions we hope to find answer for while working on this thesis. We also describe the research methods which have been used in this thesis.

Chapter 3: Knowledge Management. Chapter three contains the general overview of knowledge management. Theories described here, are used later for the evaluation of the open source development practices.

Chapter 4: Managing Software Engineering Knowledge. Contains a short summary of the current approaches to knowledge management in the field of software engineering.

Chapter 5: Open Source Development. In this chapter we describe the origins and principles of open source development, and what makes it unique in comparison with other development methods.

Chapter 6: Analysis of Practices. Based on the theories found in the literature study, we analyse open source practices from two knowledge management perspectives, the commodity and the community perspectives.

Chapter 7: Discussion and Conclusions. Here, we discuss and conclude the overall knowledge gained in the thesis and suggest possible directions of further research.

1.4 Summary

This chapter describes the reasons why it is desirable to make connections between disciplines of open source development and knowledge management.

People contributing to open source projects, constantly have to satisfy their knowledge related needs typical to software development such as learning about the project's policies and practices, localizing experts or understanding the work of previous contributors. Moreover, the use of software developed as open source has become a common practice for commercial software vendors. They also try to use the open source community to develop some parts of their projects. For those practices to be possible, they also need to have in depth knowledge about developed software.

It is reasonable to assume that in open source development, which has evolved naturally, the ways for satisfying knowledge related needs have also evolved naturally. Putting knowledge management labels on the practices commonly performed in that type of development, might give means of understanding and eventually improving the ways of dealing with knowledge in open source projects. This can be beneficial for both open source community and commercial software vendors.

No problem can be solved from the
same consciousness that created it.
– Albert Einstein

Chapter 2

Goals and Methodology

In this chapter the goals of the thesis and the research methodology are described. We begin by describing the research goals, and the assumptions upon which we based our research questions. Further we discuss the research methods, a literature study and qualitative research in the form of an analysis and cursory case study. Finally we touch upon possible weaknesses in our research approach.

2.1 Thesis goals

The primary goal of our research is to enhance the understanding of the mechanisms present in open source development related to knowledge acquisition and sharing.

Open source software have gained acceptance among computer system users, and it is apparent that the practices that are performed in open source contribute substantially to its growing success (Bollinger 1999). As the practices have evolved naturally, similarily the needs related to knowledge access and distribution has been satisfied naturally. The open source communities created ways to share knowledge about the projects, probably without thinking primarily about knowledge management. Therefore, in general we want to:

1. Identify open source practices related to knowledge acquisition and sharing.
2. Analyse the practices from a knowledge management perspective.

We hope, that our results will assist the understanding of the mechanisms involved, so this knowledge could be reused in traditional software development.

In addition we want to offer the open source community a view on knowledge management, and insight into knowledge management aspects of open source practices.

2.1.1 Assumptions

We hope to find indications of the following assumptions, which stem from our personal observations:

- Open source development is not a fixed and predefined set of development practices, but rather a varying combination of common ones.

- As in a ecology, these practices emerge, interact and disappear, and that this variation and fluctuation changes the project.
- Each practice have different capabilities, that can be used to increase the knowledge managing support of the practice.

2.1.2 Questions

With this thesis we want to address the following questions:

1. How is knowledge management understood, and is it possible to analyse project practices from that perspective?
2. What are the knowledge management aspects of traditional software development and which ones are relevant to open source?
3. What are the common knowledge related practices in open source projects?
4. What are the knowledge management aspects of practices in open source projects?
5. How does open source development differ from other development models in the context of knowledge management?

Through finding answers to these questions we hope to gain enough knowledge to satisfy our primary goal.

2.2 Methodology

Before we describe the methodology which we have used in our research, it is necessary to briefly explain the areas which we are working with *i.e. the open source development model* and *knowledge management*. Their properties are the reason which made us choose the approach which we have pursued.

There is relatively little research published about open source. It seems to have caught the attention of researchers very recently, most of the available studies have been published after the years 2000. Nowadays, open source is being studied more extensively, but still the topic is only sporadically covered, which might be due to the fact that it is evolving rapidly.

Companies that realize that open source can help them to deliver solutions they require, sometimes decide to release some parts of their proprietary software as open source (Dinkelacker et al. 2002). This can at first seem to be a counterintuitive or even self-destructive strategy (Hecker 1999). They do not do it because they understand the mechanisms that are present in open source, but simply because it works, and can be verified by looking at the software that open source delivers (Norris 2004).

Our knowledge about the knowledge management discipline, before starting to work on the thesis, was very limited. We basically knew that knowledge management is a relatively new discipline which is multidimensional – having technical, cognitive, interpersonal, and managerial aspects – and thus very complex.

We have been trying to connect the two areas, and this makes the research even more difficult. Before starting the actual research, we were able to localize only two articles that tied together knowledge management and open source, among which only one has been peer reviewed (Lanzara & Morner 2003).

For the above reasons we realized that it was necessary for us to broaden our understanding of both open source and knowledge management, and through that connect the two disciplines.

2.2.1 Research nature

While deciding on the research approach we tried to understand its nature, to find a suitable methodology. We wanted to review the existing theory and knowledge in the fields of open source and knowledge management, and by joining them together we hoped to be able to assess open source in a new light. Further, as mentioned, there is very little existing knowledge about the areas that we were about to explore. Therefore, the studies that we performed should be classified as *descriptive* and *exploratory*.

2.2.2 Research approach

Since the research is exploratory, the research literature proposed that we should use qualitative methods in the research process (Darke & Shanks 2000). Incidentally, qualitative studies has also been suggested by Thomas Davenport, as the most appropriate when performing research related to knowledge management (Davenport 1999).

As a foundation for the qualitative work, we performed a literature survey. This has, according to Dawson (Dawson 2000), the purpose of identifying, synthesising and analysing the literature relevant to the problem.

A traditional case study is an in depth exploration of a problem or situation, and can be performed directly or by observation or indirectly by studying secondary sources. It has the purpose of examining phenomena in their natural context (Darke & Shanks 2000). We complement out thesis with the analysis, which is based on observation of live projects and secondary sources such as literature, with what we would like to call a cursory case study. With this approach, we aim to integrate examples from real open source projects, with the goal of giving the reader an insight into those project's application of the knowledge management aspects described.

In summary, our research approach follow these steps:

1. A literature survey about knowledge management in general.

   Keywords: knowledge management, knowledge vs information, tacit knowledge, organizational learning, knowledge management strategy, codification, personalization.

2. A literature survey about knowledge management in software engineering.

   Keywords: software engineering knowledge management, experience factory.

3. A literature survey about the open source development model, with the purpose of identifying general characteristics.

   Keywords: open source software, open source development, free software, meritocracy.

4. A synthesis of the literature survey, resulting in a knowledge management characterization scheme .

   Based on the results of our literature survey, we have created a characterization framework, or analysis method, that we used later to characterize open source practices.

5. The identification of open source practices related to knowledge management.

   In this step we have characterized the subject and the context for our case studies.

6. An analysis and cursory case study of the identified practices performed, in the context of the selected projects, with the use of the characterization scheme and support of secondary sources.

   The main focus of the cursory case study was to identify knowledge management related capabilities and mechanisms of the identified practices.

2.2.3 Weaknesses and limitations of the approach

We realize that that the results of the analysis are influenced by our personal background, characteristics and experiences, and therefore depend on our interpretations. However, as we are aware, we aim to describe and motivate the choices we make, as the thesis evolves.

Further, the theoretical base for the characterization scheme is rather general. It gives brief overview rather than precise properties that could be used for actual recommendations. It is likely that a replication of our analysis work could find new aspects, due to the generalized description of the analysis method.

2.3 Summary

In this chapter, the goals and methodology of this research was presented.

The main goal of this thesis was the identification of open source practices related to knowledge acquisition and sharing and their analysis from the perspective of the theories created within knowledge management discipline. For this to be possible, a general theories related to both open source and knowledge management had to be explored.

Therefore, the most critical part of this thesis is the literature survey, covering general knowledge management, knowledge management in software engineering, and the open source development model. Based on the results of the survey, a scheme for characterizing practices in open source development has been derived. After identifying a number of common practices, which to some extent support knowledge acquisition and sharing, these practices were analyzed using the characterization scheme.

In the next chapter, we will present the results of the literature survey on knowledge management in general.

Therefore its name was called Ba'bel, because there the Lord confused the language of all the earth; and from there the Lord scattered them abroad over the face of all the earth.
– Genesis 11:9

Chapter 3 Knowledge Management

This chapter contains a literature overview of general knowledge management theory. It starts with the definition of the basic terms used in texts on the subject. The definitions are followed by the description of two main perspectives proposed by the most influential authors. Those descriptions will later be used to define the framework for the evaluation of the practices performed in open source development.

3.1 Definitions

In order to define the term *knowledge management* (KM), it is necessary to first explore the meaning of the term *knowledge* itself. Before that, the reasons for the recent interest in knowledge management are presented. Section 3.1.2 contains a description of two main tendencies among researches in approaching this task. The *Data-Information-Knowledge-Wisdom* model is furthermore described. It tries to explain the nature of knowledge, by defining boundaries on the continuum from data to wisdom. The model is followed by a definition of knowledge management, describing the scope and presenting two strategies that organizations can use to pursue knowledge management successfully.

3.1.1 Focusing on knowledge

Even though the question of what knowledge is, has been the subject of studies and analysis since ancient times, the attention that the question has been given recently has resulted in the development of many new concepts and ideas (Davenport & Prusak 1998).

The need for paying so much attention to knowledge and knowledge managment is related to the current social end economical trends. Laurence Prusak recognize *globalization*, *ubiquitous computing* and *knowledge-centric view of the firm* as the three main trends (Prusak 2001).

Globalization The complexity and the number of elements – global players, products, and distribution channels, etc. – that must be considered in order to achieve economical success in global trade is unprecedented. In addition, the speed-up of those elements, caused by information technology, compels organizations to ask *what do we know*, *who knows it*, *what do we not know that we should know.* (Prusak 2001).

Ubiquitous computing The ease of access to almost any kind of information at any time in any place increases the value of cognitive skills of individuals. Choosing the right information to process *i.e.* understand, read, internalize, is a great challenge. Even though some source is potentially valuable and relevant to the tasks to perform, following it on a regular basis requires substantial amount of resources. Making sure that only the most relevant information is being processed in order to complete the given tasks is critical in order to achieve efficiency.

Knowledge-centric view of the firm According to Sidney Winter, firms can be seen as *organizations that know how to do thing*s (Prusak 2001, cited in). The companies are more often recognized as collection of capabilities, limited in effectiveness by its cognitive and social skills. The main component of coordinated capabilities is knowledge. It is the knowledge specific to the firm that makes it an unique organization.

Nowadays knowledge is of concern, not only to companies that base their profitability in products of intellect, like for example consulting or pharmacy, but also many others (Davenport & Prusak 1998). Along with land, labor and capital knowledge has become a organizational asset, and a primary resource for individuals as well as the economy in general (Desouza & Davenport 2003, Drucker 1992, Stewart 1997). For this reason knowlege is sometimes referred to as *the intellectual capital* (Stewart 1997, Sveiby 1998) or *an intellectual assets* (Hansen, Nohria & Tierney 1999).

Managers have come to realize that it is necessary for the company to have more than a casual or unconscious approach to their corporate intellectual capital (Davenport & Prusak 1998). The practices, like information management, the quality movement and human resource management, slowly directs focus towards the issue of managing knowledge (Prusak 2001).

3.1.2 Defining knowledge – different approaches

The trends that we mentioned in the previous section have fused creativity and research, which has resulted in a truly great amount of literature about knowledge managment. Unfortunately, there is no common understanding of the field (Wiig 1999). Even the discussion about such basic concepts as definition of knowledge has not been settled. This is probably due to the fact that philosophical analysis of knowledge – epistemology – heavily depends on the intellectual and religious background of the culture in which the related concepts have been studied (Nonaka & Takeuchi 1995).

Even though Peter Drucker suggests (Drucker 1992) that in the new society of organizations, in which knowledge is the primary resource for all individuals and economies, the distinction between Western and other histories will fade away, we still can see and experience effects of the cultural and historical differences.

Swan *et al.* (Swan, Newell, Scarbrough & Hislop 1999) recognizes two main perspectives taken by researchers when looking on knowledge: *the commodity view* (cognitive) and *the community view* (constructionistic) (Stenmark 2002, Swan et al. 1999). Those views can easily be aligned with the approach Western and Japanese authors have towards knowledge.

In the commodity view, knowledge is considered as an universal truth that can be separated from the knower. Westerners tend to view it as something explicit, not so difficult to process with computers (Nonaka, Umemoto & Senoo 1996). It is equal to the objectively defined artifacts such as concepts and facts, that can be handled in the discrete units. The primary objective of knowledge management in this view is to codify, capture and transfer knowledge through networks. Swan *et al.* (Swan et al. 1999) suggests that technology is critical factor when determining the success in this way of thinking. Thomas Davenport, who is one of the most influential authors in the domain, suggests to approach knowledge as an element in a value chain, that should be formally managed with the help of models, schemas, analysis and skills that ensure the replicability of the know-how (Godbout 1996).

On the other hand, supporters of the community view argue, that it is impossible to define knowledge universally. Japanese see it as something tacit, that is context-specific, personal and not so easy to communicate to others (Nonaka, Umemoto & Senoo 1996). It can only be constructed socially in the activities and interactions of individuals, and must be based on experiences (Stenmark 2002). Knowledge can be shared and made sense of through active networking within or between groups (Swan et al. 1999). In the community view, the role of knowledge managment is to encourage knowledge sharing through networking where the dominant factors of success are trust and collaboration. *The Knowledge-creating Company* by Ikujiro Nonaka, and Hirotaka Takeuchi (Nonaka & Takeuchi 1995), is the most famous work which represents the Japanese view on the knowledge related concepts. In the analysis the authors suggest that social activities are critical and irreplaceable elements supporting the learning organization.

A more detailed description of the commodity view and the community view follow in the sections 3.2 and 3.3 respectively.

3.1.3 On data, information and knowledge

In the information age, sometimes also called postindustrial (Castells 1996), with the computer being a critical and most influential factor, the ability for information processing with the help of information technology (IT) have dramatically increased. Technology allow companies to organize and process a great amount of data fast and inexpensively. This has created a gap between what technology provides, and what organizations and users actually needs (Eisner 2002).

Companies realize that the big amount of information stored in the form of documents, exchanged over networks does not necessary contribute to the competitive advantage. It has been recognized that this information has to be processed by human to be of any value (Miller 1999, Sveiby 1994). The need for an adequate theory, that will explain information's nature, its relation to knowledge, and recognize it's value and influence, have emerged (Eisner 2002). In response to this need, researchers distinguish between data, information and knowledge, and define strict boundaries between those elements.

3.1.3.1 DIKW hierarchy

Currently the idea of knowledge is explained by comparing it with the concepts of data, information and wisdom, in a model called *The Data, Information, Knowledge, and Wisdom Hierarchy* (DIKW). In general, researchers agree that there are differences between the above elements, however, depending on the researcher, the terms are differently described and interpreted (Stenmark 2002, Wiig 1999).

A summary have been written by Dick Stenmark. The table (see table 3.1) that he synthesized, captures views of the most influential knowledge management authors on the DIKW related definitions (Stenmark 2001).

Wisdom, which is the top level element in the DIKW hierarchy is very seldom discussed and therefore omitted in Stenmark's compilation. Davenport *et al.* (Davenport & Prusak 1998) state that companies have enough difficulty in distinguishing between data, information and knowledge and for practical purposes omit the discussion of wisdom. In general wisdom is understood as the element that allows one to use knowledge to establish and achieve goals (Bierly, Kessler & Christensen 2000).

Even though the definitions above are presented in almost every paper on knowledge management, the authors rarely use them, and when they do, they often use them interchangeably (Stenmark 2002). Authorities in the domain recognize the DIKW-related definitions as source of much confusion, and that the terms should be used carefully (Davenport & Prusak 1998, Tuomi 1999).

3.1.3.2 Knowledge types

Von Krogh *et al.* have gathered the ideas of a number of authors that were using different categories of knowledge. They were able to distinguish between tacit, embodied, encoded, embrained and embedded knowledge. These categories are heavily dependent on the *object of the knowledge development (biotechnology, mathematics or linguistics)* (Venzin, von Krogh & Roos 1998) and obviously there is no universally accepted categorization scheme. The two concepts of *tacit* and *explicit* knowledge are probably the most widely recognized and used.

Author	Data	Information	Knowledge
(Wiig 1993)	-	Facts organized to describe a situation or condition	Truths, beliefs, perspectives, judgments, know-how and methodologies
(Nonaka & Takeuchi 1995)	-	A flow of meaningful messages	Commitments and beliefs created from these messages
(Spek & Spijkervet 1997)	Not yet interpreted symbols	Data with meaning	The ability to assign meaning
(Davenport 1997)	Simple observations	Data with relevance and purpose	Valuable information from the human mind
(Davenport & Prusak 1998)	A set of discrete facts	A message meant to change the receiver s perception	Experience, values, insights, and contextual information
(Quigley & Debons 2000)	Text that does not answer questions to a particular problem	Text that answers the questions who, when, what, or where	Text that answers the questions why or how
(Choo, Detlor & Turnbull 2000)	Facts and messages	Data vested with meaning	Justified, true beliefs

Table 3.1: Definitions of data, information, and knowledge.

The concept of tacit knowledge has been developed by Michael Polanyi, who recognized that *individuals know more that they can say* (Polanyi 1966). His understanding of knowledge was, that it is about the action and the process of knowing (Sveiby 1997). Basically, tacit means something not easily visible and expressible (Nonaka & Takeuchi 1995), and is heavily influenced by the individuals background, past experiences, ideals, values and emotions (Nonaka 1994).

Polanyi recognized that knowledge to some extent can be articulated with words, made *explicit* through language and focused for reflection (Sveiby 1997). In this context, knowledge is something that is formal and systematic; easily transferable, codified into documents (Nonaka 1994), and processed in the same manner as information. Explicit knowledge is a written explanation of how to perform a certain task that a person with the *appropriet background* would be able to perform (Brooking 1999). Nonaka *et al.* (Nonaka 1994) states, that it is a common practice to use the term *information* interchangeably with the term *explicit* knowledge.

3.1.4 Knowledge management

As difficult it is to define knowledge, it is also tricky to define knowledge management. Progress in research in economics, sociology, psychology and philosophy has shaped the view upon knowledge-related developments. Depending on the author's background *i.e.* culture, fields of study or job interests, the approach to the subject differs. It is influenced by a wide variety of practices performed in organizations, thus making it a meta-practice.

Being a such complex entity, the best way to define knowledge management, is by looking on its objectives. On the most general level, it has been recognized that having a conscious management of knowledge, help organizations to maintan their current level of success, and by creation of new knowledge, encourage future advancement (von Krogh, Ichijo & Nonaka 2000). This can be achieved by enhancing exploitation[1] and exploration[2] of knowledge (Levinthal & March 1993). To accomplish this advancement, organization should try to:

- improve organizational learning capabilities (Swan et al. 1999)
- "know what they know" (Davenport & Prusak 1998)

[1]Capturing, transferring and deploying existing knowledge; replication (Teece 2002).
[2]Through sharing of the existent, creation of the new knowledge.

- make sure that the right people have the right knowledge at the right time (Handzic 2003)

Managing, means *conducting and supervising something* (Merriam-Webster 2004). Naturally it must be conscious and related to the practices. Therefore, knowledge is managed when an organization is taking managerial actions, which satisfy their knowledge-related objectives.

3.1.5 Correctness of the term "knowledge management"

Some authors promoting the community view argue, that the term *knowledge management* is improper. Since, in their understanding, knowledge itself can not exist outside the human being, and is *what the people know*, it is impossible to manage (von Krogh, Ichijo & Nonaka 2000, Miller 1999). Since the knowledge can only be tacit, the action of making it explicit automatically turns it into information.

However, those authors represent a minority, and unfortunately the term knowledge management has been used interchangeably with information management a very long time, thus gaining wide acceptance. It is very unlikely that the term will be redefined to something more general, acceptable by all interested parties.

3.1.6 Strategies in knowledge management

Before applying knowledge management supporting practices, a knowledge management strategy should be chosen. Hansen *et al.* recognize two strategies that an organization can choose when introducing knowledge management (Hansen, Nohria & Tierney 1999). The two proposed strategies are the result of studies performed within a number of companies in different industries (Davenport & Prusak 1998). Both approaches can be aligned with the commodity and community views.

The codification strategy centers around the computer. In this approach all knowledge is codified and stored in data repositories. Easy access to the stored knowledge is granted to anyone in the company.

In contrast, *the personalization strategy* is pivoted around person-to-person contacts. Individuals who posses the type of knowledge that is necessary for the work of their co-workers are located with the help of computer systems. In this case, the strategy supports knowledge seekers locating knowledge holders (Davenport & Prusak 1998).

Hansen *et al.* claim that an organization have to persue predominantly only one of the strategies, and use the other merely to support the first one (Hansen, Nohria & Tierney 1999, p. 107). Choosing the wrong dominant strategy or performing both in parallel, can undermine the organization. The choice of the dominant is not arbitrary:

> ... it depends on the way the company serves its clients, the economics of business, and the people it hires.

Codification is preferred in organizations where knowledge assets can be easily reused many times. It requires heavy investments in IT – mainly computer systems, which allows codification, storage, dissemination, and reuse of knowledge.

Personalization is a choice in organizations heavily relaying on experts, delivering customized solutions to unique problems. Only a moderate investment in IT have to be done, to facilitate conversations through linking people, and allowing them to share their tacit knowledge.

3.2 The commodity view

The following section contains a presentation of the properties and capabilities of knowledge, while treated as an universal truth that can be easily separated from the knower into an explicit form, and generally become codified. People who decide about project guidelines, rules, policies and think about codifying knowledge, should consider the issues presented in the following section. This could eventually guide them to a successfull knowledge management initiative, which supports retention of the created knowledge, and allows it to be reused in the future.

The theories presented below, is a basis for creating one part of the framework used, further on, to evaluate practices performed in open source. The summary contains mostly ideas presented by Davenport and Brooking, complemented by supporting theories from other articles.

3.2.1 Knowledge markets

As any other asset, knowledge assets are the subject to transactions. Similarly to tangible assets, knowledge can be bought or bartered. For this to occur they need to be located. The *knowledge market* is the place where the transactions happen (Davenport & Prusak 1998). They are similar to the traditional markets, with the main difference that the space they operate in is logical rather than physical (Desouza & Yukika 2003). The market place for example may take an electronic shape in the form of an intranet site. Desouza *et al.* (Desouza & Yukika 2003) have even recognized black markets which are created when members of an organization spread knowledge outside of acknowleged parties[3].

There are three kinds of players that use knowledge markets: *sellers*, *buyers* and *brokers* (Desouza & Yukika 2003). The buyer is simply an individual or organization that wants to solve a problem which he is unable to solve by himself, he need to acquire applicable knowledge. The seller is someone in the organization who posses the required knowledge. It is intersting to note that not all knowledgeable employees can be good sellers. They are considered good sellers only when they are able to articulate their knowledge in a way that is understandable to the knowledge buyer. The third type of player, the broker, is the one knowing "who knows what", and who is able to connect buyers with sellers. A librarian, who makes people-to-text and people-to-people connections, is an example of a knowledge broker.

There are three reasons why people want to become sellers. They want to get something back (reciprocity), or they want to be known as experts and by sharing knowledge prove their value to organization (reputation), or they are happy to share what they know since they are excited and passionate about the subject (altruism).

The most critical element needed to operate a knowledge market is trust. Without it, people will not be willing to share their knowledge. Since knowledge transactions does not depend on enforceable contracts, sellers must be sure that they get credit for their work. Also, if the purpose of sharing knowledge is based on reciprocity, the seller want to be sure that he will get something back. Knowledge markets are mostly credit based, which further emphasize trust as a vital component.

3.2.2 Codification

Since, in the commodity perspective, knowledge is viewed as something explicit, it is treated as an object, which a number of actions can be taken upon. Knowledge can be created, captured, acquired, transformed, transferred, accessed, applied and so on. One of the main objectives of knowledge management in this view is codification, that is, the action of transfering the knowledge from its tacit form, into explicit from, so that it can be stored. The goal of codification is to turn knowledge into forms that could be acquired and used by the people that need it, the buyers. Knowledge gathered in this form can be easily accessed and reused. Codification is considered to be the process of generating infrastructure assets (Brooking 1999).

Among methods that are available for capturing knowledge Brooking recognizes the following: verbal reporting, retrospective reporting, questionnaires, task in context analysis and interviews. There are also a number of systems, sometimes referred to as knowledge repositories, in which knowledge can be stored. Among the most commonly used is paper documents, documents in management systems, groupware software, expert and knowledge based systems (Brooking 1999).

[3]This is undesirable, since it can result in imitation (Teece 2002).

3.2.3 Capabilities

Knowledge repositories should have a number of properties that allow effective *creation*, *reuse* and *maintanace* of knowledge. Certain capabilities are necessary, for successful knowledge related actions to take place.

3.2.3.1 Accuracy and value

Before the codification process can take place it is necessary to consider which properties should be the subject to storage and retrieval (Brooking 1999). Brooking suggests that to recognize the relevant characteristics, is to helpful to try to teach someone else to do what the knowledge holder does.

A more general approach is suggested by Daveport *et al. (Davenport & Prusak 1998).* According to those suggestions, in order to achieve success in codification, a few principles should be kept in mind:

1. It must be decided what business goals the codified knowledge should serve.
2. Managers must identify existing knowledge, in different form, that will help the organization to achieve those goals.
3. The identified knowledge must me evaluated for usefulness and appropriateness for codification.
4. An appropriate medium for codification must be identified.

It is also critical to keep in mind, that in the process of codification, relevance is far more important than completeness (Davenport 1997). Further, in order to be reusable, the codified knowledge has to be well structured and as compact as possible (Vegas, Juristo & Basili 2003).

3.2.3.2 Dynamics

Changing and growing knowledge, as opposed to static information, which can be stored in documents, has to be codified in a way that allows quick and cost effective updates. Recognition whether the knowledge is static or dynamic and the choice of the right storage media, is an important part of the codification strategy (Brooking 1999).

3.2.3.3 Validity

Knowledge that previously has been acquired, while performing a task or solving a problem, can at a later point be relevant again. This is especially true in an organization where innovation is the base of operation. Different actions, like correcting mistakes in a product, improving the design, adding new functionality, and adjusting to new environments, likely require reuse of the knowledge used to generate particular product in the first place.

A good example is the Y2K problem, which was created by the way old computer systems stored time. In many cases it required modification of applications that were written decades before the year 2000. The knowledge of the old systems, programming techniques used in the past, and tools – all those elements had to be revived to fix the Y2K problem.

Using dated information can have serious implications for the safety and security of a system. For the above reasons, it must be possible to decide if codified knowledge is still valid (Brooking 1999).

3.2.3.4 Concurrency

If the number of people in an organisation is great or if they are dispersed into a variety of locations, but require simultaneous access to codified knowledge, the system must support concurrent access (Brooking 1999). Concurrency should be implemented in a way that allow different peers to access and modify stored knowledge, without overlapping with other tasks, or destroy each other's work.

3.2.3.5 Confidentiality, accessibility and roles

In some cases, sensitivity and confidentiality of knowledge is relevant (Brooking 1999, Teece 2002). In organizations where innovation is the source of competitive advantage it is desirable to store related knowledge in a way that guarantees access only to eligible parties.

Different parties also have different needs in terms of access to knowledge (Brooking 1999). Some want to codify and update where others just need to have access to it. In order to assure the integrity, not just anyone should be allowed to perform modifications. Therefore, some kind of role based system which defines different levels of access might be required. In this context, accessibility levels define the amount of control that is given to and individual or group over the knowledge artifacts.

3.2.3.6 Categorization and mapping

To be able to locate the necessary resources easily, knowledge maps can be used (Davenport & Prusak 1998). In general it serves a purpose similar to the yellow pages in a phone book. A knowledge map serves a guiding purpose, pointing to knowledge, rather than being a repository, containing knowlege itself. In addition, codification of the richest tacit knowledge is generally limited to locating someone with this knowledge, in this case a knowledge map allows the buyer to locate the seller. In some cases, knowlege maps take the form of an individual – a knowledge broker.

An example of a knowledge map is a categorization system. Obviously depending on the organization and tasks the categories found in a categorization system is very specific, and therefore varies between organisations.

A knowledge map can also be used to determine how relevant some specific codified knowledge is to a task or a role. For example, a knowlege map can be used to decide what resources should be monitored on a regular basis by employees. A decision which can affect the efficiency factor of an organization (Davenport 1997).

3.2.3.7 Search

Another method for locating relevant knowledge is text searching systems. The knowledge in textual databases can be indexed on the basis of keywords and their occurrences in the text. A good thesaurus thus is essential to most knowledge repositories (Davenport & Prusak 1998).

3.2.3.8 Expert localization

Expert localization is the effort of making it easier to access knowledge (Davenport 1997). To support expert location, one can create an expert network initiative, a network in which it is easy to identify and locate experts within a given area (Rus & Lindvall 2002).

An interesting issue related to expert localization is how an expert is identified. What is the definition of an expert? Brooking recognized that job titles actually mask the people's knowledge (Brooking 1999). Usually, titles do not reflect the actual expertise of the individuals. She has identified three concepts which are supposed to help us understand and identify the expert:

- *Competence* – the ability typically gained as a result of learning or training, for example when attending a course.
- *Proficiency* – a person that have gained competence can complete it, by obtaining more specialized knowledge, for example through reading journals. This creates proficiency, which tends to track competence.
- *Capability* – the ability of an organization to perform a certain task. To gain capability, the organisation must have both competence and proficiency.

3.2.4 Critique of the commodity view

Finally, it is interesting to consider possible weaknesses with knowledge management approaches that are based on the commodity view. As discussed before, knowledge management as a field of study have recieved some critique, especially when information managment software companies relabels their products, calling them knowledge management system instead. This confusion seems to be especially strong in organizations promoting software based on the commodity view.

Von Krogh *et al*, also recognize three pitfalls that can be found in the commodity view (von Krogh, Ichijo & Nonaka 2000):

1. *Knowledge management relies on easily detectable, quantifiable information.* The main problem of this pitfall is that it as a common practice to misunderstand, or rather flatten the term of knowledge. Either it is used interchangeably with information or is treated as an information that makes a difference. In reality knowledge is something far more complex which is related to beliefs and tight to action.
2. *Knowledge management is devoted to the manufacture of tools.* Tools which are used incorrectly tend to discourage the real flow of knowledge which occurs in social context and between individuals rather than through the use of repositories.
3. *Knowledge management depends on a knowledge officer.* Usually, the purpose of such officer is to manage intellectual assets by establishing management systems, implementing information-technology platforms, establishing value of intellectual capital, and etc. while those initiatives should be related to creating context encouraging knowledge sharing. From knowledge-controllers, they should become knowledge-activists.

3.3 The community view

The community view is an approach to knowledge management where the emphasis is on human interaction in an organization as the basis for knowledge creation. Knowledge managment issues relevant to this view can be person to person communication, learning by doing, and the dialogue and learning as a tool for knowledge creation.

Von Krogh et al, support this focus on the individual and organizational interaction, and provides three premises which they argue a successful knowledge enabling initiative rely on (von Krogh, Ichijo & Nonaka 2000):

1. *Knowledge is a justified true belief, individual and social, tacit and explicit.* Knowledge, especially tacit knowledge, which relies on experiences, is something more than information. It can not be placed in repositories. Sharing knowledge is a social process that occurs between individuals.
2. *Knowledge depends on your perspective.* Changing the perspective also changes the perception of a issue, and therefore influences the knowledge about it.
3. *Knowledge creation is a craft, not a science.* Approaching knowledge from a scientific perspective, for the puspose of delivering models that would become tools in the hands of managers in not important. Knowledge management is about culture, common language, stories and metaphors.

3.3.1 A model of organizational knowledge creation

Probably the most influential work, focusing on the community and interactions within it as the dominating factor in a knowledge-sensitive organization, has been developed by Ikujiro Nonaka, and Hirotaka Takeuchi (Nonaka 1994, Nonaka & Takeuchi 1995). They have created a framework describing the process of knowledge creation, by which they claim that ideas are created in the minds of individuals, and that an organization as such cannot create knowledge. Instead, the interaction between people plays the central role in developing those ideas. The communication that takes place within a certain network of people, is the necessary element for those knowledge creating interactions to occur.

3.3.1.1 Knowledge conversion modes

Nonaka *et al.* have based their framework on the assumption that there is only two types of knowledge: explicit and tacit (Nonaka 1994). According to Nonaka, tacit knowledge is more valuable, but also requires more effort to share. For knowledge creation to occur, tacit knowledge needs to be mobilized; the process of creating new concepts and ideas, is driven by continuous dialog between the two.

There are four combinations of interactions, during which knowledge is converted from one type to another, and they are called *knowledge conversion modes.* The conversion mode between tacit and tacit is called *socialization*, between tacit and explicit it is called *externalization*, between explicit and explicit it is called *combination*, and between explicit and tacit it is called *internalization.* Each of the four conversion modes can create knowledge independently.

Socialization – tacit to tacit The process of creating tacit knowledge through shared experience is called *socialization.* As a result of this process, mental models and technical skills are created and shared. It is possible to acquire knowledge from others without the use of any language. It usually starts with creating a team or field of interaction. According to Nonaka *et al.* (Nonaka & Takeuchi 1995) trust is a necessary element that must occur in order for socialization to start.

On the job training is an example of a process that reflects the idea of socialization. During training, a trainee observe, imitate and practice the things that trainer is doing, and through this process gain necessary knowledge and experience.

Externalization – tacit to explicit Articulation of tacit knowledge into explicit concepts is called *externalization.* The term *explicit concepts* refers to metaphors, analogies, hypotheses and models. During the process of externalization, while the knowledge holder try to express what they know, promote another aspect of externalization, called reflection. In most cases the expressions used to express knowledge are inadequate, inconsistent and insufficient; which leads to interaction between individuals. This way, externalization is triggered by successive rounds of meaningful dialogue.

Documentation is an example of externalization. Apart from facilitating transfer of explicit knowledge to other people – helping them experience the experiences of others indirectly[4], by the creation of manuals, individuals internalize what they experience, and thus enrich their tacit knowledge. Knowing comes from doing and teaching others (Pfeffer & Sutton 2000).

Combination – explicit to explicit Combining different bodies of explicit knowledge and systemizing concepts into a knowledge system is defined as *combination.* Often, combination takes the form of meetings and conversations, but it can also occur through media, for example over the phone or using a computer network. Sorting, adding, categorizing, and defining context are some examples of actions that lead to new knowledge in combination.

Formal education and training schools are examples of this knowledge conversion mode. Also, computer systems, and repositories can assist the process of combination. It can also be triggered by coordination between team members of different sections in the organization – for example through documentation of existing knowledge.

Internalization – explicit to tacit Embodying explicit knowledge and turning it into tacit form is called *internalization.* Obtaining know-how is an iterative process of trial and error, concepts are articulated and developed until they finally emerge in a concrete form. This experimentation triggers internalization, for example through the process of learning by doing. For explicit knowledge to become tacit, it helps if the knowledge is verbalized or diagrammed in documents, manuals or expressed through oral stories.

[4]*Reading and listening to success stories make people feel the realism and essence of the story* (Nonaka & Takeuchi 1995).

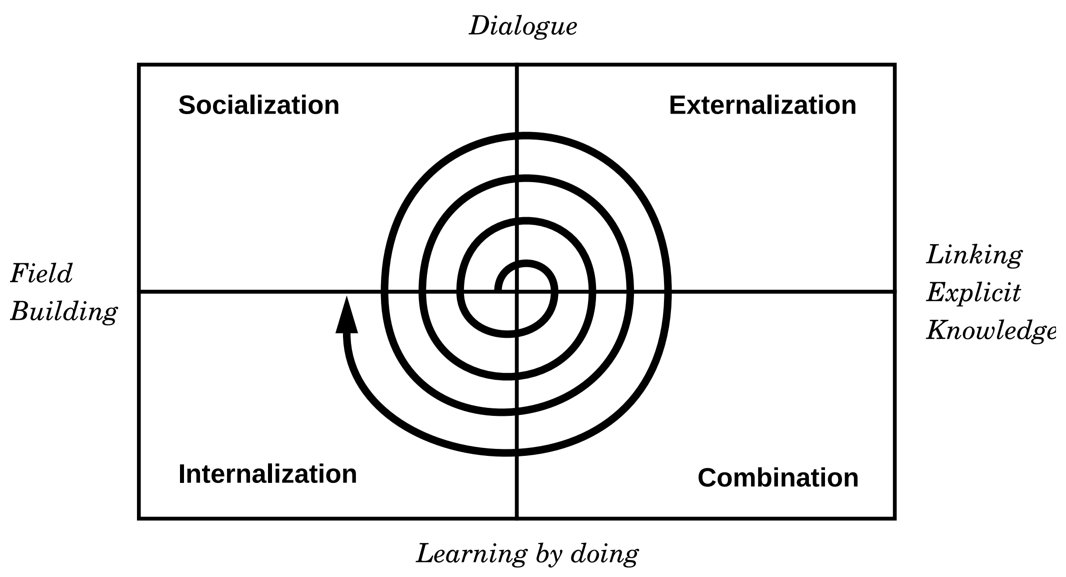


Figure 3.1: The knowledge spiral (Nonaka & Takeuchi 1995).

3.3.2 The knowledge spiral

Even though knowledge is created independently in each of the conversion modes, Nonaka argues that all four need to interact simultaneously in order to create a true dialogue between tacit and explicit knowledge (see Figure 3.1). This is because each of the conversion modes delivers different kinds of knowledge, and each mode has its own limitations. For example, *pure combination can become superficial interpretation of existing knowledge* (Nonaka 1994) when individuals do not have sufficient commitment and neglect the individual interpretation of the knowledge. Further, the ease of sharing knowledge created in pure socialization *may be limited and as a result difficult to apply in fields beyond the specific context in which it was created* (Nonaka 1994). The process of internalization and externalization can fail to shape concepts sufficiently, thus inhibiting further knowledge creation in a wider social context.

3.3.3 Critique of the community view

As in the commodity view, the critisism of the community view is focused on how the knowledge management proponents appear to mistake information for knowledge and vice versa. Especially Nonaka's work has recieved critique on this point, some even claim that Nonaka completely have misunderstood Polany's original definition of tacit knowledge (Wilson 2002). Wilson argues that the tacit knowledge Nonaka refers to in his model would have been better described as *implicit knowledge.* The difference is, that implicit knowledge is knowledge that has not yet been expressed, while tacit knowledge is, and will allways continue to be inexpressible. Wilson continue to argue, that the externalized knowledge Nonaka refers to, is nothing more than plain old information.

The critisism on Nonaka's model, is limited to the definition of words and epistemology. In this thesis we have chosen to continue to use Nonaka's definition of tacit knowledge, even though we to some extent sympathise with the argument of calling it implicit knowledge. The value of Nonaka's model of knowledge creation remains in the emphasis of the different modes of interaction between people, and how that is the basis for knowledge creation.

3.4 Summary

In this chapter, the results of the literature survey of general knowledge management are presented.

The current social and economical trends, globalization, ubiquitous computing and the knowledge-centric view of the firm, has caused knowledge to catch a lot of attention. It is widely recognized, that along with land, labor and capital, knowledge has become a primary asset of most organizations, and a core resource for individuals and for the economy in general. In fact, knowledge is often referred to as intellectual capital or intellectual assets. Because knowledge holds a such high value, it is desirable to have a conscious and defined approach towards managing it. In response to that need, a discipline called knowledge management has arisen.

Differences in intellectual and religious background of the Eastern and Western cultures – in which the concepts of knowledge are studied – has resulted in different approaches towards the field of knowledge management. Westerners tend to see knowledge as a universal truth that can be separated from the knower, something explicit, which is not difficult to process with computers. Therefore, the purpose of knowledge management in the western perspective, called the commodity view, is to codify, capture and transfer knowledge through computer networks. In the eastern perspective, called the community view, knowledge is considered to be tightly coupled with the process of knowing and the experiences of the knower. The community view suggest, that supporting social activities and interactions of individuals, and encouraging knowledge sharing through networking, where trust and collaboration are key factors, should be the goal of knowledge management.

Some concepts, such as the Data Information Knowledge Wisdom model and knowledge categories, seem to be present in both perspectives. However, their definitions are very ambiguous, and are rather a source of confusion than a base for delivering an unified approach towards knowledge management.

In general, in order to obtain as complete picture of the field as possible, it is important to explore the concepts presented in both the commodity and the community view.

In the next chapter, we will present suggestions on how to manage knowledge in software engineering.

We have spend a great deal of time and money bringing water to the horse, but we do not even know if he is thirsty and we have no idea how to get him to drink.

– Pacific Bell manager in *Information Ecologies*

Chapter 4 Managing SE Knowledge

The goal of this chapter is to show the needs for, and approaches towards knowledge management in software engineering as proposed by scholars and practitioners. It contains a brief overview of the most influential and widely quoted literature on the subject. The chapter starts with an identification of general knowledge related needs in software engineering. The needs are followed by a presentation of frameworks dealing with knowledge management in software organizations. The chapter is concluded with a summary of the possible barriers and obstacles to knowledge management initiatives in software engineering.

4.1 Needs in software engineering

The field of software engineering have some interesting knowledge related characteristics; software development is knowledge-intensive (Lindvall & Rus 2003), knowledge within the field is dynamic, evolves with technology, and heavily depends on the creativity of not only individuals but also whole teams (Aurum 2003).

Software organizations are interested in delivering higher quality systems at lower cost (Rus & Lindvall 2002). One of the ways to achieve this is through the reuse of and modification of the same elements (Basili, Caldiera & Rombach 2002). Those elements can be actual pieces of software, but not only. The elements also include characteristics of the environment, processes, resources and defect data (Basili, Caldiera & Rombach 2002). Knowledge of how to improve them is created during production and delivery of each product. It is critical to a software organization to retain such knowledge. In projects, individuals make decisions all the time and need to inform peers about their actions. In small teams informal communication is usually sufficient to share knowledge related to decisions (Rus & Lindvall 2002), but in larger projects that is obviously impossible.

Apart from business needs like *decreasing development time*, *increasing quality* and *support for making decisions*, Rus *et al.* recognized a number of needs that exist in software projects and software engineering that are related to knowledge (Rus & Lindvall 2002). These needs are addressed implicitly in all software projects. The goal of knowledge management in software engineering should be to keep the knowledge created, and reuse it in future development, simply put: *capture*, *share* and *coordinate* (Aurum 2003). These needs are described as follows (Lindvall & Rus 2003, Rus & Lindvall 2002):

- *Acquiring knowledge about new technologies.* In order to stay competitive, developers have to keep in pace with recent developments in computer science (Desouza 2003).

- *Accessing domain knowledge.* The purpose of computer software is to support the processes and procedures that exist in the organization using it. Therefore, for a solution to fit the needs, the specifics of the domain the software is developed for must be obtained.
- *Sharing knowledge about policies and practices.* Each organization or project has some commonly known standards, which must be followed by all peers. Learning about those specifics becomes one of the first needs of newcomers in an organization. Software development is very fragile in this perspective, since each delivery must be very carefully coordinated, and lack of knowledge about policies makes coordination impossible.
- *Knowing who knows what; locating sources of knowledge.* To become experts in the technical area, individuals have to narrow the scope of their interests and responsibilities. Projects usually involve a large variety of technologies and consist of a large number of elements. Knowing exactly who knows what becomes a critical part of each project. Since most knowledge in projects is tacit, and making it explicit through codification can be very difficult (Desouza 2003, Rus & Lindvall 2002), locating knowledgeable individuals becomes even more important. In addition, if knowledge necessary for performing a particular task has been codified, it must be possible to easily locate its sources.
- *Collaborating and sharing knowledge.* Being a group activity, software development requires coordination between peers, which are often spread apart, working in different timezones. Effective collaboration requires constant communication and sharing of knowledge. Thefore, the need for coordination and sharing knowledge arises.

4.2 Implementing knowledge management

The following section contain a couple of different proposals of how to implement knowledge management in software development organizations. They are given as examples representing the approach that scholars in the software engineering area have towards knowledge and knowledge related activities. They are not intended to be a complete review of the works in the subject, but rather be an overview of some of the approaches. In addition, they show that codification is the preferred strategy in the software engineering field.

4.2.1 The experience factory

The *experience factory* is an approach proposed by Basili *et al.* (Basili, Caldiera & Rombach 2002). It focuses on continuous reuse and improvement, based on past experiences of successfull delivery of software products. By implementing a set of practices that are based on a quality improvement paradigm, organizations can create a knowledge base, that contains a package of analyzed and synthesized past experiences, which could be reused in future projects. The authors define the experience factory as

> ... a logical or physical organization that supports project developments by analyzing and synthesizing all kinds of experience, acting as a repository for such experience, and supplying that experience to various projects on demand (Basili, Caldiera & Rombach 2002).

In the experience factory organization, improvement of processes and products is achieved through continuous collection and evaluation of experiences, learning in a form that allows understanding and modification, which results in experience models.

In this approach there is a logical distinction between a process of project development and a process of learning and packaging gained experiences[1]. This distinction is present

[1]An excellent example can be found in (Komi-Sirvio, Mantyniemi & Seppanen 2002).

because each project implements its own practices and processes during product development, and having a shared way for codifying those experiences is necessary for the successfull operation of the experience factory.

The quality improvement paradigm consists of six basic steps: characterize, set goals, choose process, execute, analyze, and package. Product development is preformed by following the above steps. Experience come into play in two feedback cycles; a project feedback cycle and a corporate feedback cycle. In those cycles, new experiences are created that need to be packaged. *Cost models, resource allocation models for staffing, schedule, and computer utilization, and the relationship between resources and various factors that affect resources* (Basili, Caldiera & Rombach 2002) are just a few examples of what experience packages may contain. Depending on their nature, the experiences can be stored in one of the six types of packages: product, process, relationship, tool, management and data packages. Packages are typically sets of documents containing manuals, diagrams and algorithms, but may also contain software.

4.2.2 Identification of relevant knowledge

The theory behind the experience factory defines a general approach for capturing and managing past experiences for the purpose of reuse. The framework, however, is not very precise about defining which artifacts[2] should be included in a package. The key question that arises is how to make sure that a package contains only relevant and valuable elements. In order to be reusable, packages has to be well structured, contain only relevant elements and be as compact as possible (Vegas, Juristo & Basili 2003). This is due to the fact that people will be more likely to reuse this information if it will be clear how to apply it. In addition, badly structured information usually leads to failure of knowledge management initiatives (Komi-Sirvio, Mantyniemi & Seppanen 2002).

Vegas *et al.* (Vegas, Juristo & Basili 2003) have proposed a process for identifying relevant artifacts with the use of a characterization schema, a process which is empirical and iterative. *A characterization schema* is a set of characteristics that is used to inspect a certain module to determine if the module suits defined constraints or not. The proposed approach can be divided into two main parts: *schema generation* and *schema testing.*

4.2.2.1 Schema generation

This part can be divided into four different stages that if followed, can help to create characterization schema.

1. *Development of a theoretical schema* helps to define the artifact itself. In this stage a set of artifacts that are to be characterized are analyzed for the similarities and differences. The purpose of this process is to define a set of parameters that reflect differences which becomes a first draft of the whole characterization schema. The construction process should be guided by deductive reasoning.

2. *Development of an empirical schema* allows to incorporate the diverse viewpoints by consulting people related to the artifact area: potential consumers and producers. The purpose of this step is to assure that created final schema is workable *i.e.* can actually be used in practice. This can be achieved by addressing potential customers with the question about *what information consumers needs in order to select an artifact from the experience base* (Vegas, Juristo & Basili 2003). The question should be asked until stopping criteria is met, that is to the moment in which at least 25% of subjects will not cause changes in the schema with their response.

3. *Synthesis of perspectives* is required to unite theoretical and practical schemas so the resulting *preliminary schema* would contain elements from both. In this step the information that appears in at least one schema, should be entered directly into the result. In case of similar information existing in both, it must be assured that the synthesis of those two, information is not lost, but also that there is no redundancy.

[2]Techniques, products, processes, methods, and so on (Vegas, Juristo & Basili 2003).

4. *Expert peer review* is the last step in which a independent expert inspect and offer his opinion on how suitable the schema is for testing. The goal of the review is to identify possible defects that can be related to the *form* and *substance* of the produced schema. Vegas suggest, that the number of reviewers should be not less than three, even though less is also acceptable.

4.2.2.2 Schema testing

Through testing of the schema it could be possible to evaluate its validity and detect possible improvements. In the *start-up stage* it is necessary for the preliminary schema to be tested on a number of sample artifacts, by individuals representing both producers and consumers.

4.3 Barriers to KM in SE

For knowledge management to be successful there are a number of obstacles that an organization must overcome or at least address when performing the practices. Different sources recognize various problems.

Desouza *et al.* (Desouza, Raider & Davenport 2003) primarily emphasize the lack of commitment among project managers in sharing knowledge. Organizations tend to ignore the effort of employees involved in codifying their knowledge, which leads to demotivation towards the whole idea of knowledge management. For example, codification efforts are usually not included in performance reports. Project managers, stressed by deadlines and tight schedules, are very unlikely going to use the resources assigned to them for purpose of codification. Further, when codified knowledge is reused by someone in the organization, managers commonly give credit to the person that reused it, rather than to the one that originally created it.

The next problem is related to the nature of how technical jobs are performed. A software engineer can be afraid that when he contribute to the body of knowledge in a certain area, he will be recognized as an expert in this field. This might result in assigning him to the same task over and over again instead of giving him new and challenging tasks.

In addition, Desouza *et al.* (Desouza, Raider & Davenport 2003) state, that by making private knowledge public, developers are more likely become the subject of judgment and verification, which naturally is something that may discourage knowledge publication.

Another barrier is that technology is changing rapidly, and software engineers do not want to analyze the knowledge gained during a project because they believe it quickly will become outdated (Rus & Lindvall 2002).

The last obstacle that prevent knowledge management in software engineering, is that most knowledge in software development projects is tacit. Making it explicit through codification is very difficult and time consuming, sometimes it is not possible at all (Desouza 2003, Rus & Lindvall 2002).

4.4 Summary

The purpose of this chapter was to present how current researchers and practitioners propose knowledge should be managed in software engineering.

Acquiring knowledge about new technologies, accessing domain knowledge, sharing knowledge about policies and practices, knowing who knows what; locating sources of knowledge and collaborating and sharing knowledge, are the knowledge related needs that are present in all software projects. A conscious and defined approach towards knowledge in software development should focus and satisfying those needs.

The most commonly quoted framework, used to address the above needs, is called the experience factory. It focuses on continuous reuse and improvement, based on past experiences of successfull delivery of software products. Those experiences should be analyzed and synthesized into packages which could be reused later. The framework, however, does not suggest how to identify the relevant knowledge to include in the packages. One way to deal with this issue, is to create a knowledge characterization schema. If derived correctly, a knowledge characterization schema can be used to inspect artifacts created during software development to determine if they suit defined constraints and should be included in the experience package.

Naturally, there are a number of obstacles that cause knowledge management initiative in software engineering organizations to fail. These obstacles are mostly related to resistance among project managers and developers. To yield positive results in knowledge management, it is important to consider how to defeat that resistance.

In the next chapter, the nature of open source projects will be investigated.

It is not the strongest species that survie, nor the most intelligent, but the one most responsive to change.
– Charles Darwin

Chapter 5 Open Source Development

In this chapter we introduce and give an overview of what open source is, and where it originates. We describe what project and development perspectives are common to open source development, and describe the differences from traditional, closed source, software development. We also explore how the development model can be perceived as an ecology.

5.1 A brief retrospective review

The open source movement originates from the American university world of the 70's. At that time computers were time shared accessed through terminals. In that environment the source code of software was freely shared between developers, students and staff who gained more by giving their changes back to the research community than trying to sell it. In the 80's this culture vanished from the universities when the computer industry hired the good developers, and restricted source code access with non-disclosure licensing (Raymond 2000).

At this time Richard Stallman realised that there was a philosophical need of free access to software. 1983 he initiated the *GNU Project*, which later on would become the *Free Software Foundation* (DiBona, Ockman & Stone 1999). The core principles of the *GNU Project* is that software and the source from which it is built should be be freely accessible to the user. It should be free in the sense of liberty, which implies that a user should also be free to alter the program, and redistribute the altered result and its source freely. Ten years later, in the mid 90's, the operating system *Linux* and *Linux distributions* had resulted in a easy accessible free software environment and a growing community of open source developers.

The term Open Source, and the *Open Source Initiative* was coined 1998, when Netscape released the source code of their web browser. The *Free Software Foundation* use the phrase *free software* as a label for software that adhere to their philosophy – the problem with this label is that users falsely believe that it implies *without cost* and associates free source code with freeware – this issue among others was the reason for Bruce Perens to define the new label *open source,* which since have been broadly accepted in the community (Raymond 1998*a*). At that time, open source software already was widely used, and had millions of users.

The open source movement owe much of its success to the Internet. As a community open source spread as Internet became a commodity in the early 1990's (Abrahamsson, Salo, Ronkainen & Warsta 2002), and open source development has inherited many of internets traits; development is distributed and independent, it can be continued even if a specific resource disappears. The Internet also makes it easy for developers to share their code, and communicate. In open source, email and usenet has been important tools for development coordination from the beginning (Moon & Sproull 2002).

5.2 An Open Source Development Model

5.2.1 The Cathedral and the Bazaar

In *The Cathedral and the Bazaar* (Raymond 2000) Eric S. Raymond argues that there is a fundamental difference in open source development vis-a-vis traditional software development. Raymond compares the traditional closed source development model to a Cathedral, where there is a high priest controlling what goes on in the cathedral. In a software engineering environment this would be the lead designer or business managers. The initiatives for new ideas is assumed to come from above, and the developers are working on the areas they are instructed to. As opposed to this controlled environment, Raymond argues that open source is a bazaar where anyone has access to the merchandise, and is free to set up a competing stand at any time. If one merchant is more successfull, other merchants will want to copy his ideas, or in software terms, if a development practice is successful in development, other projects will be interested in adopting the practice to their environments.

The difference between open source software (OSS) and open source development (OSD) is similar. OSS does not have to imply OSD. Some software products, though released as open source, are developed in rather traditional closed source project environments. These projects still maintain a cathedral approach to development, and the only thing that makes it open source is really the license model of the released product. Other projects, follow the basic principles of open source development, works in open view, using tools which support the input from external parties.

5.3 Core principles and attributes

Even though open source development projects vary immensely, and practices come and go, a few key principles are shared by most, if not all open source development projects. A study of 80 open source projects identified two major traits that all the projects shared: the projects adhered to the open source definition and the developers were all users (Gacek & Arief 2004).

In addition to these most fundamental traits, open source development have been analysed in other studies, which describe other common principles or attributes of open source development projects (Johnson 2001, Capiluppi, Lago & Morisio 2003, Robbins 2003). In the following sections some of these, most commonly identified principles and attributes, are described.

5.3.1 Adheres to The Open Source Definition

Obviously, the most fundamental common attribute of open source projects is that the source code is freely available, normally distributed over the Internet.

In an effort to create understanding and acceptance for the term Open Source an open source definition was proposed at the inception of the *Open Source Initiative* (Initiative 1997). This definition is focused on the technical aspects of open source, especially licensing, since source code is protected by copyright regulations and is mainly shared through licensing. The definition can be outlined in three groups of requirements which must be fulfilled before a project can be certified as Open Source.

- *Source code availability.* Naturally the source code of the software must be freely available to any user. This also gives the user the right to redistribute the source code or any derivative works under the same license as the original work.

- *Non-discriminatory access.* The license may not discriminate against any person or group, neither can it restrict the field of use, and it must be technology-neutral.
- *The right to create derivative works.* Any user must have the right to create derivative works, i.e. the user must be allowed to alter the behavior of the software, use, and redistribute the alterations.

5.3.2 The developers are users

The second most common attribute in open source is that the developers are users too. This has some interesting ramifications. The developer has himself as a customer, and knows exactly when the requirements, his needs, have been met. Also, the motivation for becoming a developer often is to *"scratch an itch"* as a user, and thus is highly driven by personal needs (Hars & Shaosong 2001).

5.3.3 Community driven

The community around open source projects is a combination of the users and developers. There is a continuous flow of individuals, in and out of the community. Users join, some of them become contributing developers, and while some developers stay on a project indefinitely, some move on. As individuals assumes different roles, their type of contributions also vary (Gacek & Arief 2004).

As previously mentioned, open source communities are Internet based. To communicate, the most widely used communication method is e-mail, and mailing lists are used both for user support and development (Yamauchi, Yokozawa, Shinohara & Ishida 2000). The ability to attract and cultivate new developers from the user base affects the success of the project. This also means that a community can be destroyed, or at least severely crippled, by people who join mailing lists, without submitting to the implicit culture and basic principles (Cox 1998).

While the community around a open source project is typically informal, that does not imply that the community is not managed. Consensus is a natural tool for making decisions, since contributions are made on a voluntary basis, and no-one thus can be forced to adhere to a certain decision.

The availability of the source code, means that anyone can fork the project at any time. To fork a project means, to start a new separate project, based on the current state of the project being forked. Forking is used when the developers cannot agree on a topic, be it political, managerial or technical. Through forking, the community have a form of built-in self control on the community leaders. If a decision is unpopular enough, and consensus can not be achieved, the leaders risk that the project will be forked, and the current project will be abandoned, for the benefit of joining the newly created fork.

This is an example of how the technical aspect of source code availability affects the project management aspects in open source.

5.3.4 Meritocracy

In *Homesteading the Noosphere* (Raymond 1998*b*) another principle is explored, the open source *gift culture*. As discussed, open source projects have a community around the projects, this community is normally a *meritocracy*. A meritocracy is a culture where *one's work is one's statement*, that is, a culture where people who provide the most utility to the community also gains the most respect, status and influence. It also implies that a person that only complains without actually providing any constructive work towards resolving an issue, will not gain much respect.

Linus Torvalds, the creator of the Linux kernel, captures the essence of meritocracy well in a quote from the Linux kernel mailing list: *Talk is cheap. Show me the code.* In arguments over what way to implement certain features, the one that actually provides working code as a proof of concept is often the one whose argument is listened to.

In the Apache development community, the meritocracy is even more elaborate. The developers group is divided in several levels of membership, initially a developer has no committing rights to the versioning system, but if the developer is devoted, and shows consistency in providing correct patches he can be promoted to become a *committer*. As a committer he can add patches directly to the source code repository, without having to get it approved by another commiter first. Once a committer, if he proves himself reliable, he can be elected to the project management board, and from there become a *Apache Software Foundation* member and ASF Board member (Hann, Roberts, Slaughter & Fielding 2002).

Another sign of the influence of meritocracy was displayed by *Red Hat*, the company behind the well known Linux distribution, when they emphasized meritocracy as a core value, as they merged their gratis version of the distribution with the Fedora project. While describing how the new project would be managed they state (project 2003):

> The Fedora Project is an openly-developed project designed by Red Hat, open for general participation, led by a meritocracy, following a set of project objectives.

While this should be recognised as a sign of the importance of meritocracy in open source, in reality there has been some grumbling on mailing list during the spring of 2004[1]. Meritocracies depend on people's communication skills, as well as what deliverables they contribute to the project. If a person is unable to participate in discussions, for example due to language difficulties, he will perhaps not be able to impact the project as much, as had he been verbally skilled.

5.3.5 Motivation

The basis of a meritocracy is the motivation of a developer to prove himself in the community. But increased influence is not the only motivation for contributing to open source development.

A survey conducted by Hars *et al.* (Hars & Shaosong 2001) has investigated the motivations. Almost 400 open source developers answered the survey. The respondents were grouped by type, different forms of paid developers, students and hobbyists. Their responses were divided in two types of motivations, internal - such as altruism, self-determination, and community identification – and external – personal needs, human capital, peer recognition, product marketing and self-marketing.

Altruism To deliberately increase the welfare of others, without any self-interest.

Personal needs When a OSS project solves a problem for a developer, who thus to fulfill his need starts to work with the project.

Self-determination The natural human instinct to persue certain paths, to fulfill the satisfaction of feeling competent.

Community identification To fulfill the human need of belonging to a social context, and devotion to others.

Human capital To increase ones personal skill, capabilities, and knowledge.

Peer recognition To satisfy the need for fame and self-esteem, through the persuit of recognition of ones capabilities, from peer developers.

Product marketing To use open source projects as a platform to market products and services.

Self-marketing To use open source projects as a mean for marketing, by showing ones skills as a developer and good citizen.

[1]For example, see this discussion, spurred by a parody in the form of a IRC log, titled: "Revealed: how Fedora and the community interact" http://lwn.net/Articles/83360/

The survey showed that the student and hobbyist groups were most inclined to be driven by internal factors, while the employed developer groups were more driven by towards self-marketing and personal needs. Another conclusion was that the motivation was a complex issue, and most respondents were motivated by combination of the above described motivators.

5.4 The role of shared project practices

The description of open source development using a wider scope, as in this chapter so far, have limitations. It is important to understand the principles, to understand the consequences of the details, and as such the overview it is relevant. On the other hand, an analysis needs to be performed on actual work performed, not on principles, to be able to lead to any tangible result. As described, open source development is not authoritatively defined anywhere – each community has their own interpretation on how open source development is supposed to be performed in their particular project – and there is more variance than similarity, in development methods, between projects. Thus the impact of each individual work practice becomes perhaps more influential on a particular project, than the overall principles.

While work practices vary between projects, still a set of widely use practices have been identified in open source projects (Johnson 2001, Robbins 2003), for example:

- An iterativ and incremental development style, supported by stable-unstable development cycles or branches in version management tools.
- Concurrent development, supported by CVS or a similar version management tool.
- Issue tracking systems, supporting coordination of bug tracking and feature requests.
- Mailing lists, supporting development and the general user community.
- Unit testing, and automated build tools, supporting quality assurance.
- Sourceforge, and other collaborative development environments.
- A free development environment, supporting a wide adoption of the project.

From the above examples, it is possible to observe that the open source practices are tool oriented. For a practice to establish itself in the community, the development tools, and previously existing practices should support, or be able to adapt to, the new practice.

5.5 The ecology of open source development

As opposed to other more established software development methods, where the process is often well defined and thoroughly described by an authoritative source, the development model in open source development is harder to define (Abrahamsson et al. 2002). An explanation of this is that open source development, as a model, have never explicitly been developed. Instead it has evolved, taking inspiration from best practices introduced in various open source projects.

Open source development is used in projects of varying scope and size. The projects range from one developer sharing his personal hacks, to large and very complex systems, like the Linux kernel which have almost six million lines of code, and supports more than ten different hardware architectures and involves hundreds of developers world wide. Further open source development practices are used differently depending on the individual needs of each project (Gacek & Arief 2004). All these variations make it clear that open source development perhaps is not as much a well defined model in the traditional sense, but a more loose set of practices guided by a set of core principles. In this way the open source development model shares a lot with agile development models such as Extreme programming (Abrahamsson et al. 2002).

One approach to describing the open source development model as a phenomenon, is to characterize it as a ecology as Lanzara *et al.* (Lanzara & Morner 2003). According to Lanzara, the advantage with this form of comparison, with an ecology, is that it better captures the dynamic features of the development method. Open source development emphasises the interaction between people – in the form of communication – rather than organizations, and technological artifacts – centered around development tools and practices – rather than organizational routines. This does not imply that the open source community is unorganized, but that the organization is centered around interaction through communication. The ecological interaction is according to Lanzara et al, present in the form of interaction between processes and practices too. They argue that open source development model itself is a complex system, which evolves through continuous steps of variation, selection and stabilization. Variation is offered in the form of new problems occurring, and solutions to them, in the form of new development practices.

Eric Raymond (Raymond 2000) also argues that there is a advantage in considering open source as a ecology. Raymond explains that the networked, but still independent, open source community can continue to operate even though some people or resources vanishes. This is because the already committed code has been published, and someone else can pick up the work where it was left by his predecessor. This is closely related to knowledge markets, as described by Thomas Davenport (see section 3.2.1), where knowledge is exchanged on a market with buyers and sellers. In this case the market resource is the open source code and the knowledge brought into the community by individual developers. The buyers are for example the managers of Linux distributions[2], and other projects which require functionality or developer knowledge.

5.5.1 The Linux kernel development - an example of evolution

For a long time the Linux kernel development have been a role model for release management in open source projects. The project has maintained several branches, out of which one is the development branch where more intrusive changes, like changes in programming interfaces, have been introduced. This kernel branch which have odd version numbers is not always stable, and is targeted only to developers. If you run the development branch on your computer you are expected to be prepared for loss of data.

Parallel to the development branch a stable branch is maintained. The release branch, which is identified using even version numbers, is the kernel that is used by most of the Linux distributions and is normally stable and reliable. Only a few changes go into the release branch of the kernel, examples would primary be security fixes, but support for new hardware is also added continuously, as long as the changes are not too intrusive.

Periodically, normally in a cycle of one to two years time, the unstable development branch is "frozen" and released as a new stable version. When the development branch is frozen, the development work is shifted from adding features to extensive testing and bug fixing, often by encouraging normal users to try the new kernel, thus increasing the number of different hardware combinations tested. When the freeze has been in place for couple of months and the branch is deemed stable enough, it is renumbered to an even version number and a new unstable branch is started.

The release management used in the Linux kernel, with a stable branch denoted with even release numbers and a unstable development branch using odd version numbers, have been used as a model for many open source projects.

At the time of writing this thesis during the summer of 2004, the kernel development team have been discussing a change in this development practice. Instead of maintaining a development branch which only at a cycle of a couple of years is stabilized, the new proposal is to continuously integrate new changes from the development branch to the stable branch. In reality this means the developers only maintain one kernel branch, the stable one, and a set of alternative development pre-patch-series. The idea is that instead of having several years between releases, and losing resources to the double maintenance of having two branches, having to add and adopt security and stability fixes to both, these resources can be better

[2]Distributions are groups of open source software, packaged and delivered in coherent sets, which can be installed by end users. Normally a distribution is delivered as CD-ROM images.

utilized by giving the full attention to one branch.

This is an example of a rather profound change of development practices, in this case release management, where we see the evolutionary aspects of open source. In reality the change proposed has already been present for some time, only not officially, but in the way Linux distributions have managed their kernels, incorporating changes from the development branch to their distributed kernels (Corbet 2004). As is seen in this case, the best practice which originated as unofficial practices used by Linux distributors, has evolved, showed that it is advantageous compared to the current practice, and thus has overtaken the role as main practice in kernel development release management.

5.6 Summary

In our view, one of the most interesting aspects of open source, is how the small and rather technical aspect of source code availability, so vastly impacts the other aspects of open source project. In this chapter we have described how free software and open source originated from a need to be able to alter the functionality of software. This prominent attribute, the availability of the source code, affects both the ability to control development, and harness external resources, in the form of other developers. The community always have the choice of taking their resources and leave, which motivates the leading developers towards consensus, and cooperation.

We also describe the meritocracy which is a central guideline for open source developers, *i.e.* the ones who share the most, whether source code, helpful hints, ideas, documentation or support on a user mailing list, is the one who gets most credibility and influence in the open source community where he is active.

Further, we have discussed how the open source development method is guided by a set of core principles. These principles affects the organization as a whole more than individual practices.

We have also shown that it is beneficial to compare the open source methodology to an ecology. How the evolution of open source projects is hard to predict and thus have ecology-like attributes. The interaction between different practices offers solutions to problems, and problems occurs in a unpredictable manor, but open source have used this as a advantage – instead of setting practices and rules in stone, the methodology have been modeled to adapt to necessary changes – resulting in evolution, instead of revolution.

Finally we have discussed how work practices in open source. Practices are not only the tools used to perform a task, or the individual activities performed in the practice, but also the knowledge and perception the practitioners have of the practice . This descripton will be helpful when we select what form of aspects to consider in the analysis of practices.

In our view, the ecology characterization also implies that the effects individual practices have on knowledge management in open source, is as relevant as trying to approach knowledge management on the open source development model as a whole. If individual practices solves specific problems, and thus come and go as problems occurs and are alleviated, it would be reasonable to think that it is the effect the individual practice have on the project, that is relevant to knowledge management.

Honey: What are you doing here, looking for shells?
Bond: No, I'm just looking.
– Honey and Bond walking on the beach in Dr. No

Chapter 6 Analysis of Practices

This chapter contains the analysis of some of the open source development practices with the use of an analysis model that is based on the literature study. The description of the model is followed by an discussion about how we selected practices to analyse. Finally we present the result of applying the framework, the analysis itself.

6.1 The analysis model

Our model is a characterization framework for analysing open source development practices from a knowledge management perspective. We approach the analysis using our own model because we want to concisely cover the two major perspectives of knowledge management, which were identified in the literature study. In addition, we wanted to complement the analysis with a discusson on the relevance of software engineering knowledge management theory to the practices analysed.

6.1.1 The reasons for selecting several aspects

Our main goal in this thesis is to put knowledge management labels on practices present in open source development. We have chosen to avoid trying to determine whether the technical *i.e.* the codification perspective, or the social, *i.e.* the personalization perspective, can offer a more complete picture when approaching knowledge related practices. Instead we acknowledge that both the commodity view and the community view offer different, but equally interesting and relevant, perspectives. This approach is supported by the two strategies in knowledge management, namely codification, which corresponds to the commodity view, and personalization, which correspond to the community view.

A pure commodity analysis, in our view only illustrates the technical aspects of a practice, mainly resulting in focus on information. Therefore, analysis from the commodity view is more closely coupled with information management than knowledge management. By including the community view, we try to show how knowledge is created and shared, not only through technical means, but also as a social persuit between humans.

Open source projects are software projects, and we believe that the needs of software engineering, typically found in traditional development, also are relevant to open source development. In our analysis of practices we identify the needs satisfied with the help of each practice.

6.1.2 The model

Our analysis model consists of four parts. It starts with a *general description*, which gives an introduction to the practice. It is followed by an analysis from knowledge management perspectives, divided in two consecutive parts. First, we analyse the *commodity aspects* of the practice, *i.e.* using the western knowledge management view. Secondly, we analyse *community aspects*, consonant with the eastern knowledge management perspective. In the fourth part we identify which *software engineering needs* are fullfilled by the practice.

General description

This is a brief introduction to the practice, including a functional description. The purpose of the description is to give the reader enough information about the mechanisms in the given practice, to be able to understand the analysis from the commodity and the community views, and see the relevance of the recognized software engineering needs.

Commodity view

In this part we look at the practice trying to recognize what properties corresponds to the selected capabilities. We try to show how a given capability is identified, and how it is performed in the context of the practice. If a practice incorporates a specific tool, we show which tool is used in the projects we look at. The following commodity capabilities are analysed:

Accuracy and value We search for mechanisms that support deciding wheterwhether the codified knowledge is *relevant*, *structured* and *compact*. We try to see if the process of identification of the relevant knowledge for the purpose of packaging includes some of the characteristics of the framework suggested by Vegas *et al.* (Vegas, Juristo & Basili 2003) as summarized in section 4.2.2. Even though this framework is a formal definition of a process which should be followed in order to deliver a characterization schema, we believe that some of its characteristics can be recognized in the practices. For example development of *characterization schema as a result of synthesis of perspectives*, and *peer review*s. In addition, we recognize peer reviews of the actual contents of a repository as an element increasing the value and accuracy.

Dynamics Recognition of the type of the knowledge, *i.e.* if it is *static* or *dynamic.* We want to see whether is it *easy to maintain* the knowledge that has already been codified and if so, how it is supported.

Validity The possibility of *validity verification of knowledge.* Is the knowledge updated to remain in accordance with reality? We will try to see whether the practice supports verification of that codified knowledge is *relevant* and *valid.*

Concurrency Does this practice support *simultaneous work,* conducted by a number of individuals?

Confidentiality, accessibility and roles Here we try to recognize means for *protecting knowledge* and defining *different access rights* using a *role system.*

Categorization and mapping Can the practice be used as the yellow-pages, to identify knowers, pointing to knowledge rather than actually containing it. Does the practice have some kind of categorization system. Those are questions that are answered under the term of categorization and mapping.

Search Does the practice have any mechanisms for locatingrelevant knowledge, and if so, with what level of accuracy?

Expert localization Apart from the possibility of recognizing knowledgeable individuals and finding a way to contact them, we look for the possibility to recognize their *competence* and *proficiency*, by observing whether the knowledge of an individual is *purely theoretical* or is supported by some *practical experience.*

Community view

When analysing the practices from the perspective of the community view, we use Nonaka's four modes of knowledge creation: *socialization*, *externalization*, *combination*, and *internalization*, described in chapter 3.3.1.1. Nonaka's model is widely accepted as a mean to describe the sociocultural aspects of knowledge creation, and thus is suitable to describe the community view. The conversion modes are useful because they emphasize the effect that the interaction between knowledge and information has in a social context. That is, the conversion modes show how different communication between people affects knowledge creation and sharing.

Analysing from a community view perspective in this way is problematic, since it concerns knowledge as an internalized part of human consciousness, and how the individual create and use knowledge in interaction with others. Therefore we are limited to try to show the physical evidence of that form of interaction, as a sign of what form of knowledge conversion is supported. According to Nonaka typical signs of support are (Nonaka 1994):

Socialization is performed in a social context. Typical examples are team building, sharing of experiences, story telling, opportunities for apprentices to observe how experienced persons perform a task, and other behavior which shares knowledge more indirectly, without direct attention to the knowledge shared.

Externalization All forms of electronic communication in some sense is externalization, since the person sharing knowledge is forced to articulate his knowledge in messages to transmit it in an electronic medium. An important result of externalization is feedback, when an idea or problem is iteratively refined through articulation and interaction between people, such behavior supports externalization.

Combination occurs when an activity helps structure externalized knowledge, examples are sorting, filtering, validation, comparison, cross-referencing and similar activities.

Internalization is when information and experiences are converted to knowledge internal to the human mind. This is perhaps the hardest conversion mode to prove using secondary level sources. Learning by doing is a prime example of how knowledge is internalized.

We would like to underline that the community view analysis, as it is performed in this thesis, does have some weaknesses. We are primarily software engineers, and have very limited knowledge of the field of socio-cultural anthropology. In the analysis, we have been limited to primarily describe the opportunities and examples of interaction as we perceive them. The research is thus of course limited by our experiences and prior knowledge. A thorough analysis of the community view would require more extensive work than would be reasonable for us to perform within the boundaries of this thesis.

Needs supported

Since the practices we analyze are used in software development, we believe that it is desirable to see how the software engineering needs described in the chapter 4.1 are satisfied. We achieve this by describing their use in the projects by looking at the purpose they serve and their actual usage patters. We base this analysis in the open source projects that we consider to be successfull and popular that is Apache, GCC, Gnome, KDE, The Linux kernel and Mozilla.

6.2 Identification of general practices

As we described in the previous chapter, the open source model is configured around a set of interchanging practices. To be able to describe what a practice is, we use definitions introduced from software process improvement. Sami Zahran, a recognized software improvement advocate, makes a holistic definition of the term process, which adds two aspects that are interesting from a knowledge management perspective.

In this thesis the term practice is used to describe something that closely resembles Zahran's description of processes. The purpose of using this definition is to emphasize that the analysis will contain both the behavioral aspect and the objects, such as tools and capabilities, involved.

Zahran defines three aspects that describe a process (Zahran 1998). The first is the process definition, normally documented, which describes the steps and sequence of operations that constitutes the process. Secondly the process is executed by people, which have an understanding and knowledge of the process. The process has been transferred to people, when learning to perform it, and this aspect also affects the process. The third aspect of a process is the result of the process activities.

It is interesting to note that this definition states that a process is not limited to information, if it were, it would be limited to being shelfware. Instead Zahran argues that a process must to some extent be tacit knowledge.

On the other hand, in casual everyday language, the word practice refers to *an action that is performed habitually and with purpose* (Zahran 1998). Here, the emphasis is on the habitual action which is performed, disregarding possible tools used, or what outcome the action results in.

6.2.1 Knowledge management practices

The general goal of software development, and the practices involved, is to produce software deliverables which solves a specific problem. From a knowledge management perspective this implies that software development is about the ability to act, which is referred to in common definitions of knowledge. Thus, the deliverables, and many of the tools and practices used in development, have primarily a technical purpose.

For example, versioning systems like CVS are used to maintain and coordinate source code changes between a team of developers. When changes are committed to the system, it logs change messages which can be viewed by other developers, to track reasons for changes. This feature can be called a form of information or communication tool. But the main purpose is not to communicate, but to share the source code and manage changes in the code. Correspondingly, some tools and practices like e-mail, focus on information exchange.

For the purpose of investigating knowledge management in open source projects, it is reasonable to assume that practices which have information exchange as their main purpose have more influence, and thus are more relevant.

6.2.2 The identification method

The selection of practices used a two step semi-formal method, which lead us to identifying the specific practices.

1. First we selected a preliminary set of practices which we believe influence knowledge sharing, acquisition and creation during development of open source project.

2. Secondly we verified the presence of the preliminarily derived practices in a set of open source projects.

6.2.3 Selection of the preliminary set of practices

The selection of the preliminary set of practices was based on our subjective observation of open source projects, and previous experience of open source projects.

Those experiences were complemented by the observation of open source project collaboration tools, like SourceForge (Sourceforge 2004) and NovellForge (Novell 2004), which provide the open source community with a place to host projects and tools necessary for distributed software development. The first one, is for the moment the most common place

	Apache	GCC	Gnome	KDE	Linux	Mozilla	Σ
FAQs	Y	Y	Y	Y	Y	Y	6
Homepage	Y	Y	Y	Y	Y	Y	6
Howtos	N	Y	Y	Y	N	Y	4
IRC	Y	N	Y	Y	N	Y	4
Issue tracking	Y	Y	Y	Y	Y	Y	6
Mailing lists	Y	Y	Y	Y	Y	Y	6
Weblogs	Y	N	Y	Y	N	Y	4
Wikis	N	N	Y*	Y	N	N	2

* – the practice has been identified in the project, but not strictly for software engineering purposes

Table 6.1: The presence of the selected practices in open source projects.

for developing open source applications. In the case of NovellForge, we have explored its functionality by creating a project (Rudzki 2004) and observing what options are available to administrators.

6.2.4 Verifying that the practices were relevant

After deriving a preliminary set of practices, we approached a number of open source projects to verify that the preliminary set really were practices which are widely used. We based the verification on the following projects: Apache, GCC, Gnome, KDE, The Linux kernel and Mozilla. We chose these projects, because they are components in many widely known Linux distributions, and are used by scholars in literature on open source development. In addition, all these projects are highly complex and have many developers. Our goal was not to evaluate the level of success or failure of knowledge management in the projects that we used for verification.

In the verification process, the aim was to satisfy one of the software engineering needs recognized by Rus *et al.* (Rus & Lindvall 2002), *identification of local policies and practices.*

The policies and practices which we have been looking for, were generally related to the remaining software engineering needs. We wanted to find out what kind of actions need to be performed by a person who wants to contribute to the project and needs to locate explicit knowledge describing the ways for performing tasks related to software development. For example, take the perspective of an individual trying to contribute a small modification in the functionality or a new idea for approaching the problem, or someone who wants to find information about testing practices or report a bug, and nontechnical contributor willing to suggest a change that could improve application's usability. We wanted to see how the remaining knowledge related needs of software projects *i.e.* acquiring knowledge about new technologies, accessing domain knowledge, knowing who knows what and general need of collaborating and sharing knowledge are can be satisfied in open source development model.

The search consisted of the following steps:

1. Start the search on the homepage of the project
2. Look for a way to perform the task of interest
3. Identify the context in which the given knowledge was acquired
4. If the context that was identified, was one of the practices from the primary set, we considered the verification successful.

6.2.5 Verification results

Except for one, the practice that we preliminarily selected, was found in most of the projects (see table 6.1). This supports the relevance of our preliminary selection of practices.

The only exception is the use of wikis. It can be argued that wikis is a relatively new practice in open source development. The *KDE Wiki*, which is the one that contains the most explicit knowledge, was created as recently as August 2003 (KDE 2004*b*). The change log of the *Gnome Wiki* main page states that the page was created in February 2003 (Gnome 2004*a*). Still, it can be observed, that the usage of wikis is slowly becoming a common thing in open source projects (Cubranic, Holmes, Ying & Murphy 2003). For this reason we have decided to include wikis in our analysis, even though the practice is absent from the majority of our selection of project. As we show in the analysis, wikis are probably one of the most supportive practices from a knowledge management perspective.

6.2.6 Limitations of the selection process

The purpose of the selection was to illustrate which practices currently can be considered general to open source. As we described in the previous chapter, the open source development method, acts as a ecology which constantly evolves, and local practices change continuously. It is very possible that if the selection process is replicated in a couple of years it yields different results.

We also recognize that the process we have followed is subjective and it is possible that some practices have been unconsciously omitted, but we believe that for the purpose of satisfying the goals of this thesis, the ones we have selected are sufficient.

As the purpose of the selection process was to deliver a wide set of practices, we used large and widely acknowledged projects in the verification procedure. We did not try to evaluate whether the projects are successfull or not, but chose projects that used a large variety of practices. It is possible that small projects use different practices, than the ones that we have used in our method of selection.

6.3 The analysis

In the following section we present the outcome of the actual analysis. The practices are presented in alphabetical order, mostly the presentations are independent. We have not tried to grade or weigh the practices against eachother.

6.3.1 Frequently Asked Questions

Frequently Asked Questions (FAQ) is a document type that contains a compilation of common questions and their answers. The idea of gathering often asked questions in a document originate from Usenet, the Internet based discussion groups that predates the web by some 10 years (Raymond 2003). As Usenet grew, a lot of questions were repeated by people when they joined groups. After a while, these frequently asked questions became tedious to repeat, and compilations of them started to appear regularly posted, normally by group regulars. On Usenet and mailing lists, it is common to introduce newcomers to the topic of the group and netiquette, the rules of the forum, presented in a FAQ.

Today, the tradition of FAQs have spread, and detached from the news groups and mailing lists they once belonged to. FAQs have become an easy way to start documenting open source projects. Questions sometimes appear before they even become frequently asked.

In the open source community, it is common to see a combination of technical and more community centered questions. For example, one of the first thing Netscape published, when releasing the source code of their web browser under an open source license, was a FAQ. It contained both technical questions, such as *What programming language does the source code use*, and more philosophical ones, like *Why is Netscape releasing the Communicator source code.*

6.3.1.1 Commodity view

Accuracy and value Being structured and compact is the main advantage of FAQs. Their structure is well defined, and consists of questions and answers. Normally a FAQ is compact and give precise answers to the questions.

Dynamics Even though FAQs are static documents, the well defined structure makes it easy to add new questions and answers. As the frequency of a question asked about a given issue increases, it is more likely to be included in the document.

Validity When treated as entities separated from their original context, for example on project web site, FAQs risk becoming outdated quickly. Further, it is difficult to verify their validity.

Concurrency Not applicable.

Confidentiality, accessibility and roles Not applicable.

Categorization and mapping A FAQ is often structured in a such way, that the list of questions can be grouped into categories, within the subject of focus of the given FAQ.

Search Not applicable.

Expert localization In some cases, expert localization is the subject of a given FAQ. They may also contain information as about how to get further help on the particular issues.

6.3.1.2 Community view

Despite being rather simple documents, and as such mainly have their emphasis on externalization, FAQs actually fulfil needs in the socialization area too. Though the main emphasis of FAQs are on combination.

Socialization The FAQs itself, being a rather static document, has very little support for communication, but they do play a role for newcomers in the open source community as a mean to quickly pick up proper behavior on mailinglists for example. The FAQ in this contexts can even set a tone. for instance, some mailinglist FAQs recommend new subscribers to post a short introduction about themselves to the list, as a way to break the ice.

Externalization Apart from the basic externalization which always occurs when anything is written, the FAQ as a practice have a limited role to play in externalization.

Combination The main role for FAQs, in the community perspective, is as knowledge maps and information filters. There is a form of inherent filtering, knowledge selection performed already when questions become candidate material for a FAQ. Not that it is some formal process, rather it is natural, the most asked questions, are by design the ones included in the FAQ, thus the community in the process of publishing a FAQ creates a knowledge filter. This makes it easier for the newcomer to find help. Instead of having to sort through multiple answers in a mailing list archive, and being forced to determine which answers actually are relevant and correct, the answer is served in a document.

Internalization Since FAQs are targeted towards quickly providing answers, the practice generally do not lend itself towards the tutorial style of writing. Thus internalization is better supported, for example with the next practice: Howtos.

6.3.1.3 Needs supported

FAQs often contain references on how to get further help, this is one form of expert localization, as in knowing who knows what. The Linux kernel mailinglist FAQ for instance, includes a introduction to some of the more influential developers (Gooch 2004). This FAQ also gives pointers to procedures and rules which should be adhered on the kernel mailinglist. The FAQ basically acts as a knowledge acquisition manual, fulfilling the needs of sharing information on policies and practices, and knowing who knows what.

6.3.2 Howto

To write a Howto is a common method of documenting software in open source projects. A Howto is, as the name implies, a document that describes how to configure and use some specific software to perform some task. It is normally written in the style of a cook book, with precise descriptions of every little step. The step-by-step instructions are sometimes complemented with a more conceptual description of the software system or configuration.

Examples of Howto topics are “Linux Ethernet-Howto” which describes how to add network card devices to a Linux system, and the “C editing with Vim Howto”, which describes how to become more efficient in C editing using the text editor Vim. Note that the topics are rather narrow, one of the key factors in the utility of a Howto is that it is focusing on solving one specific issue well, instead of just scratch a little on the surface on many separate topics.

The Linux Documentation Project (LDP) is probably the most well known documentation project that works with Howto (LDP 2004*a*), however, as the name implies, it is focusing on Linux specific writings rather than general purpose open source projects. This project works, as a coordinating body for authors of Howtos and other documents, like books, manuals, and FAQs. Amongst other things the LDP supports authors by publishing a Howto on writing documentation. The LDP also assists authors with CVS repositories and a pool of volunteers who perform document reviews, and help with constructive feedback on new or updated Howtos.

6.3.2.1 Commodity view

In general, Howtos have exactly the same capabilities as any other static documents.

Accuracy and value When they are part of a project, Howtos can be the subject to peer review and comments.

Dynamics Howtos are simply static documents written by individuals. In order to make changes to the contents of the document, it is necessary to contact the author, or the current maintainer of the text.

Validity Each Howto often contains the document history and description of revisions. When it is a part of a project, usually it has a comment on whether it is up to date or not. However, if it is a standalone entity, it is very difficult to verify their validity.

Concurrency Concurrency can be achieved in the same way as in any other document writing process by simply the distribution of topics and sections to different individuals.

Confidentiality, accessibility and roles Not applicable.

Categorization and mapping Not applicable.

Search Not applicable.

Expert localization Apart from names, the document metric usually contain the e-mail addresses of authors and contributors.

6.3.2.2 Community view

As mentioned, Howtos in themselves are simple static documents. What makes them interesting from a knowledge sharing perspective is how and why they are used. In the process of creating and using Howtos, knowledge sharing is emphasized, and thus externalization and internalization are the two conversion modes which are most influential. In externalization it supports the author, while internalization is mostly performed by the reader.

Socialization There is little general socialization support inherent in the practice of using Howtos. Though, the effort of writing Howtos occationally create a subcommunity, where a group of users team up as authors of a Howto, and this way starts to share their experiences and problems with the project.

Externalization According to the LDP FAQ (LDP 2004*b*) usually the authors of Howtos, are either developers or new users who themselves have solved issues with Linux. This is the typical externalization perspective of Howtos. A person or group who uses, has had a problem, or develops some specific software, tool or system configuration, and wishes to share his or their knowledge on the topic.

Combination Combination is for example present in the form of filtering. The author has the opportunity to emphasize and elaborate on complex parts, while skipping over irrelevant details. Thus, by skimming through a Howtos, a reader can determine how difficult different parts are, and based on that decide where to put his effort.

Internalization Learning by doing is a central point in internalization. Howtos are good examples of how learning by doing can be supported in a environment where people do not meet in person. The contents of Howtos is often explicitly targeted towards a specific group of readers. For example, the “CD Writing HOWTO” is targeted specifically on users, while the Howto “Writing an ALSA Driver” is targeted towards developers. In either case, the target audience is explicitly described, and the contents adapted to the expected level of knowledge of the reader. Thus a reader, can by himself determine if he fits the expectation, and if he does, use the Howto to guide him through, step by step, until he has grasped the contents.

6.3.2.3 Needs supported

Considering software engineering needs, Howtos fulfil the need of *access to domain knowledge.* In a Howto, practices and procedures are described. For example, take the ALSA driver Howto previously mentioned. It describes the programming interface and files and devices used to control and use the driver. But, it also describes the normal way to implement a driver, and provides source code examples. Further the Howto describes how to get the newly written source code to fit in the main ALSA driver tree.

Another example of a need fulfilled is found in the “Gnome Bug Writing Howto” (Villa 2004). It describes the practices and rules, or perhaps rather recommendations, on how to file a correct and complete bug report for Gnome. This is an example on how the Howto is used to *share knowledge about policies and practices* in the community.

6.3.3 Internet Relay Chat

Internet Relay Chat (IRC) is another internet technique, which have been adopted by the open source community. IRC was created by Jarkko Oikarinen in 1988, and the goal was to allow Usenet type of discussions, but in realtime (Oikarinen 1997).

IRC is a client-server technique that allows users to connect to IRC servers and join channels. Each channel normally is focused on a specific topic or user community. In the channel discussions are conducted in real time with other users, also present in the channel. As a user posts a message it is immediately displayed in the clients of all other users who have joined that particular channel. As more messages are posted, messages are pushed back in the log, so that only the currently active conversations in the channel are given focus. The clients are normally able to log conversations, or sessions to a file, for later review.

Channels can be public or private, if a channel is public anyone who can connect to the server, is allowed to join the channel. If a channel is private, a user can only join it by invitation from other users. If a user does not follow the channel rules, he can be banned from the channel. That means that he can no longer post messages on the channel. Some servers even use functionality which automatically can detect undesired behavior and ban users. Otherwise the task of banning, and managing a channel, is performed by the channel operator, who acts as a discussion moderator.

As IRC has grown, IRC networks have formed. These networks are a group of individual IRC servers which act as one single server from the user perspective, but distributes the client load.

The Gnome community use several IRC channels, on a private IRC-server irc.gnome.org, both for user and developer discussions[1]. Channel names and their topics are described like this:

- *#bugs - irc.gnome.org - THE place to learn how to triage bugs in bugzilla*
- *#gnome-love - irc.gnome.org - Getting started with GNOME development.*

Note how the channel #bugs is used to combine different tools, in this case IRC with the issue tracker bugzilla, to simplify the work in the community.

6.3.3.1 Commodity view

IRC is used as a real time communication media and is not suitable for codification. The capabilities typical to practices that have strong support for codification thus are not applicable to IRC. Even though it is possible to use logs from chats to read past discussions, the logs are not generally used in that way.

Accuracy and value Not applicable.

Dynamics IRC is a discussion tool, therefore it is dynamic.

Validity Not applicable.

Concurrency Many users can share their ideas at the same time.

Confidentiality, accessibility and roles Each channel have *operators* that can block users from joining.

Categorization and mapping Different channels are used to separate different topics.

Search Not applicable.

Expert localization Posting a question on a given channel may assist in finding knowledgeable individuals.

6.3.3.2 Community view

As the commodity analysis describes, IRC is primarily a real time communication tool that has little role in any codification strategy. Instead IRC emphasizes on socialization and possibly combination.

Socialization To start with a pure utility perspective, IRC is a place for coordination of work and meetings in the open source community. The real time communication makes it possible to maintain almost real world forms of meetings, with a leading chairman, and someone using the IRC logs as a base for minutes. But it is also common for developers to have an IRC window open while they develop, for quick feedback on problems while programming. This way of almost always being present in a developer channel, and keeping an eye on it, also supports team building.

[1]See http://gnomesupport.org/wiki/index.php/IrcChannels

Externalization From an externalization perspective, the disadvantage with IRC is that the messages are not systematically logged by the system, in this way it is very much like people talking to eachother in the hallway at an office. The messages are received by the people who are present in the channel at the time of posting, but then vanished in oblivion. Thus the knowledge shared in IRC channels are not conveniently stored and searchable, as threads in a mailinglist. Similarly, since the messages quickly are dropped from the log, the thorough reflection over the message, is not really present either.

Combination The prime form of combination support in IRC channels is probably when it is used as some form of immediate expert location. As it is a instant communication tool, it is perfect for quick questions. It is relatively easy to start a IRC client, and immediately join any channel on any IRC network. This makes IRC more convenient, compared to joining a mailing list only to post one short question.

Internalization Generally IRC in it self has little support for internalization. It can of course serve as support when internalizing knowledge from other sources, like the quick question while hacking.

6.3.3.3 Needs supported

Obviously the software engineering need most strongly supported by IRC as a practice, is the ability to *collaborate and share knowledge*. IRC communication is relatively close to normal conversations, at least for a textual forum. IRC can also be used for expert localization, thus helping the community *knowing who knows what.* The community around gnome uses, as mentioned, different channels for different topics. This is a way to help the knowledge seeker to map where he is most likely to get the right answer to a question within a certain field.

6.3.4 Issue tracking systems

During software development a number of problems can emerge. They are of different nature: faults, bugs, errors, and requests from users for changes in the functionality in previously developed components (Callahan, Khatsuriya & Hefner 1997). The name *Issue tracking* – a term related to problem management, software configuration managent and bug tracking – reflects fact that all the above problems and the actions related to their resolution are of similar nature and can be grouped under the above term (Johnson & Dubois 2003).

Usually, issues are documented and delivered to the persons who are responsible for the part of the application that is related to the issue. In software projects, where the number of developers are limited, informal issue tracking procedures are usually sufficient. However, for larger projects with many developers, issue tracking needs to be managed more formally. This need of support for formal procedures has lead to the creation of central repositories that are used to store the issues in a way that allows quick and easy access to all interested parties. Those repositories, however, have functionality that usually goes far beyond simple database for storing documents. These issue trackers *are forming a dialogue between parties in which discrepancies can be raised, discussed, and resolved* (Callahan, Khatsuriya & Hefner 1997). They have simply become tools for general planning, coordination and control in projects, a central point of communication for any team (Lohmeyer & Hassel 2004), supporting the whole life cycle of the product.

Johnson *et al.* (Johnson & Dubois 2003) have recognized that in issue trackers provide users[2] with the following functionalities:

- Create and maintain user accounts
- Classify users based on their privileges
- Reporting new issues with a number of properties such as description, keywords, category, severity etc.

[2]In this context we refer to the user of the issue tracker which can be a developer, tester or non-technical staff realted to the project.

- Classify issues with respect to priority and area of interest
- Search through issues using different criterias
- Browse issue lists
- Support communication with users through forums
- Assign a person for fixing an issue
- Send e-mails to parties involved, about changes related to an issue – the ones that are listed on so called *nosy lists*.

The most common issue trackers used in open are accessed via the web interface. Some offer command line interface through e-mail, similar to the control interface available for mailing lists (see 6.3.5).

6.3.4.1 Commodity view

Even though a number of open source tracking solutions are available, all of the projects that we have considered use Bugzilla. For this reason, when talking about product specific functionalities, we refer to Bugzilla (Bugzilla 2004).

Accuracy and value By assigning a *priority* to an issue, users can indicate that some issues are more important and needs to be taken care of primarily, whereas other issues can be fixed later. The s*everity* is another parameter which describes how serious a given problem is. It can range from *blocker*, where the application is unusable, to *trivial*, where its a matter of a minor cosmetic issue. The severity factor is also used to indicate *enhancement requests* (Bugzilla 2004).

The mechanism of voting, allows users to indicate that they would like certain issues to be fixed in the first place. This adds even more value to the mechanism of priorities, since it allows the elicititation of issues that have high-priority and thus likely are the most critical ones, for many developers, testers and nontechnical participants in the project (Bugzilla 2004).

Dynamics In general issue trackers are considered to be tools that support constant development and adoption of systems (Ramler, Wolfmaier & Weippl 2004). They allows the collection and maintainance of continually evolving requirements of the application (Ramler, Wolfmaier & Weippl 2004). When issue trackers are used for bug tracking, only by defining the status that a given problems has, and which changes continously depending on the input from users, the dynamics is being enforced into the issue tracker. In addition, the built-in discussion forums add even more dynamics.

On the other hand, it is possible to take a snapshot at any moment by exporting the contents of a repository in the form of the report, to a text file, thus providing an interface to spreadsheet or word-processing programs (Lohmeyer & Hassel 2004). This way it is possible to obtain static version of knowledge relevant to a project.

Validity Each issue that has been put into a tracker, have a certain status. For example: new, assigned, verified, closed, re-opened, and so on (Bugzilla 2004). This makes it easy to verify whether a given issue still needs to be resolved. In addition, the user interface is usually constructed in a way that it shows only relevant and valid problems.

Concurrency The issue trackers used in open source can normally be accessed either though a web interface or with the use of e-mail. These issue trackers are developed using a client server architecture and therefore support large number of simultaneous users.

In addition, a single instance of an issue tracker can simultaniously support multiple products and multiple releases of each product (Bugzilla 2004).

Confidentiality, accessibility and roles Public installations typically allow a user to query for issues without any need of creating an account. To be able to perform any other operation than browsing and querying, a user have to create an account and login to the system, normally by using their e-mail address as identification.

The individuals that have complete power over issues repositories are the administrators. They have the ability to create groups that contain users or other groups. Using this mechanism, it is possible to control users ability to view, enter and edit issues related to different products (Bugzilla 2004).

Categorization and mapping An administrator has the possibility to define keywords which should be used to tag and categorise issues. For example *The Mozilla Project* has keywords like *crash*, *regression*, *privacy*, *hang*, and *top100* (Mozilla 2004). The keywords can be selected by users when adding or maintaining issues in the repository.

Usually, issue trackers have functionality that allows users to specify dependencies between different issues. This makes it possible to decide if problems should be resolved first, base on if they block other bugs from being fixed.

Obviously, the previously mentioned properties: priority, severity and status, are means of categorization on their own. In addition it is worth to mention possible resolutions for defects which are no longer considered open. Those can be one of the following: *fixed*, *not reproducible*, *false alarms*, *to be fixed later* etc.

Search One of the basic functionalities of issue trackers is a search engine that allows users to query for existing problems. The search engines are usually flexible, and can locate the most relevant information, succinctly described by Lohmeyer *et al.* (Lohmeyer & Hassel 2004) in the following passage:

> Only a few mouse-clicks are required for such queries as "Show me all bugs in our e-commerce Web site, in the shopping cart component, submitted by Alice between December 1 and December 24 with an attachment containing the phrase 'resolve after Christmas rush"'

Expert localization Any kind of action performed in order to solve an issue is logged by the issue tracker. Apart from the fact that the individuals resolving the problem are easily identifiable, after finding the solution, they describe it in the system. Therefore, not only can these problem solvers be contacted, but by analysing their solution their expertise can be verified.

6.3.4.2 Community view

As described the issue tracker have attained the role of coordinating and planning in the open source community. Thus the main emphasis of the issue tracker, is in sharing experiences – in this case bugs, and solutions to the same – and coordinating work – who fixes what, and when is the work planned – rather than learning. This reflects in the role of the four conversion modes, where the main focus is externalization, and to some extent combination.

Socialization Bug tracking is a common way to introduce the development community to a new member. Reporting a bug is an example of how itches can be scratched in open source projects. Features as well as bugs are reported in issue trackers, and the bug tracker is one place to try to ascertain if someone else has asked about a particular issue before, or if it is new. As a new community member comes into contact with the developers through the bug resolution process, he can intercept basic routines and expectations, through the actions of the core developers.

For example, if the solutions for most bugs reports is presented as a unified patch[3], it is natural for a newcomer to pick up that habit and start using unified diffs himself, when he later submits fixes. Such behavior can be identified without anyone explicitly requesting the use of unified diffs.

[3]A unified diff is special file notation which identifies differences between two versions of a file.

Externalization Bug reports are excellent examples of how externalization leads to reflection. Especially reports from external parties, reports which often start out lacking the specific information needed to debug, and only after several iterations between the bug's assigned manager and the reporting user results in bug identification[4]. This is understandable, initially the user is not very familiar with the vocabulary specific to this project, or implementation details, and perhaps uses his own words to describe how he perceived the fault, and how it occurred. This behavior makes the developer ask followup questions, trying to narrow down the bug to a reproducible case, and identify the source. As the interaction evolves, and the developer asks for clarifications, possibly introducing new tools to help locate the source of the bug, both the newcomer and the developer have an opportunity to see each other's perspective. The responses they give each other reflects how they perceive the problem individually.

Combination The ability to cross reference bugs is a good example of how externalized information in issue trackers are used to combine knowledge, in the form of previous experiences, to transfer knowledge. The extensive opportunity to use categorization, flagging bugs, prioritizing them and sorting them under different modules or owners, are other examples of how combination is present in issue trackers.

Internalization As mentioned, the emphasis of issue trackers, is sharing and coordinating knowledge. Of course, there is some form of trial and error in the progress from the initial report, through the definition phase, optimally ending with a resolved and closed bug. But the emphasis is still in communication and externalization, rather than learning through internalization.

6.3.4.3 Needs supported

As it has been described in the introduction to the practice, issue trackers has become in open source development tools for planning, coordination and control, a central point of communication. This is very much satisfaction of the need of *collaborating and sharing knowledge* and *knowing who knows what; locating sources of knowledge.*

In order to verify that, we have decided to approach Gnome project with the following purposes: submit a suggestion as for the behavior of one of the Gnome Desktop applications, namely *Sticky Notes*, and find the people that are either related to the development of this component. We were able to satisfy our goals by simply using *sticky notes* as a search term in full text search mechanism. After performing this action, we have been presented with the list of all issues related to the application. It just happened that the improvement suggestion we were about to make has already been submitted (Gnome 2004*c*).

While browsing through the issues related to *Sticky Notes*, we were able to see comments from developers actively working on the software, but also actual source code that has been submitted in order to resolve one of the previously spotted bugs.

In general, the following more specific needs related to collaboration, can be satisfied with issue trackers (Bugzilla 2004):

- Track bugs and code changes
- Communicate with other team members
- Submit and review patches
- Manage quality assurance

They also help in identifying defined milestones or deadlines, keeping track of the information about the time it took to fix certain issues in the past which obviously offers valuable knowledge on how much time it can be expected the similar projects to take in the future (Lohmeyer & Hassel 2004).

[4]See for example, the dialog in this bug: http://bugzilla.Mozilla.org/show_bug.cgi?id=60281

6.3.5 Mailing lists

Traditionally, the collection of addresses that organization or individual can send mails to, is called a *mailing list.* Their electronic version[5] works in a similar way, except for two major differences; first, mails are send over the Internet in a electronic form, instead of the traditional paper form, and secondly, it allows the individuals from the list to send mail to all other people on the same *distribution list* (Manheimer, Warsaw & Viega 1998). In general, mailing lists are community-forging tools (Viega, Warsaw & Manheimer 1998) used for collaboration and news delivery.

In the terminology of mailing lists, the following terms are commonly used (Oda 2003):

Post a message sent to the list

Members and subscribers people who are part of a list

List administrators responsible for maintaining a list

List moderators people in charge of reading posts and deciding whether it should be sent to all members.

There are generally two type of lists: *announcement* and *discussion* lists. When used as an announcement media, mailing lists allows an individual or a group to send posts in manner similar to the way publishers send out magazines to their subscribers (Oda 2003). A discussion lists is used to discuss topics among list members. In this mode the list is very similar to regular conversations, and everyone can send mail to everyone. Posts are grouped in threads, which allows a number of discussion to take place at the same time on the same list. It is possible for moderators to allow only certain posts to be distributed, limit distribution to one particular group, or allow to send messages to the whole lists only to a number of users.

In general administrators have a number of privacy options available, which allow them to set policies for *subscribing and unsubscribing – open, confirmation required, or approval required – policies for posting to the list – open, moderated, members only, approved posters only – and some limited spam defenses* (Warsaw 2003).

Mailing lists makes it possible to categorize posts in two basic ways. By defining topics in the mail using standard syntax, which is later parsed server software to determine which category given a post falls into, or by specifying a list of keywords that must be included in the first line of the message body. It is up to the subscriber to make sure that either a proper topic or suitable keywords are used. Users can specify which topics they are interested in, and limit the messages delivered to their mail box to only the ones that are relevant to them.

Typically, all posts are kept inpublic archives, which makes it possible to perform full text search (Warsaw 2003).

Since the functionality depends on the software that is used to run particular mailing list service, there are differences between features available to users. However, in general it is possible to:

- keep anonymity by hiding ones name from appearance in posts
- set the preferred language of interaction
- obtain a list of all subscribers
- specify that posts are to be grouped in packages before delivery
- define filters
- receive confirmation of the delivery
- set auto-responders, content filtering and specifying topics. (Warsaw 2003)

[5]Also sometimes called electronic mailing lists, mailservers, and listservers.

There are two ways in which users can perform day-to-day maintenance and set preferences which determine the way they interact with the list(Viega, Warsaw & Manheimer 1998). They can use a e-mail based interface or a web based interface (Cawley 1995). To set preferences in the first way, users have to send e-mails with an appropriate command setting a given parameter. In the web based interface, they simply use forms available through a web browser. Both interfaces usually require users to authenticate before performing any operations.

6.3.5.1 Commodity view

Accuracy and value Depending on whether a mailing list is used as a discussion or an announcement media, there are different requirements on accuracy and value of transferred knowledge and different ways for achieving it.

In *discussion lists,* the mailing lists is used as media for exchanging ideas which evolve and change. The list becomes a place for conversations, and therefore it depends on the list members if the ideas shared are valuable. Thus the the accuracy and value are hard to evaluate on discussion lists. If the message send to the lists is irrelevant or has little value, the original sender can expect some negative feedback from the other members, in hope of that the poster will improvement in the future. However, in general it is almost impossible to enforce value and accuracy on a discussion list.

When used as *announcement list*s, mailing lists offer two ways for achieving value and accuracy. The first method involve moderators, who make sure that each post is relevant to the scope of the list, that it has an acceptable level of quality, and that the contents is accurate. It is very easy to recognize the corollary between the idea of peer-reviews and the work of a moderator. The second way to achieving accuracy and value, is if the mailing list allow users to receive only posts that belong a particular topic. A user can simply select a set of keywords, or topics that should be forwarded to his mail box, rather than all messages posted to the list.

Dynamics It is not possible to change the contents of the messages that have already been distributed. But it is possible to send additional information in a follow up message, the discussion that cause is dynamic.

Even though most mailing lists have archives, which contain all past previous posts sent, it is not possible to modify the contents of those archives.

Validity It is rather difficult to verify the validity of the information in the archives. Probably, the only way to check if something is still valid, apart from checking the date of posting of a message, is to ask people on the list. It is custom for long time list members to refer to previous posts, when answering newcomers in discussions related to the issue of those previous posts.

Concurrency Mailing lists work in client-server architecture and allow simultaneous use of many users. It is possible that two people send answer to the same questions or issues posted to the list, but that only gives the interested parties the possibility to verify the relevance of those independent answers, and possibly obtain a more complete picture.

Confidentiality, accessibility and roles It is possible to create two types of mailing lists: *public* and *invitation only.* The public type is available to anyone that has an access to the mailing lists server. Private lists are only available after newcomers have been approved by an administrator.

There are basically two roles defined on mailing lists; the administrator, who has full control over the list, and the moderator, who is responsible for approving, after prior manual content verification, posts to be send to all relevant subscribers.

Categorization and mapping In addition to the functionality of keywords and topics, previously mentioned, it is also a custom to create number of separate mailing lists within one open source project. Each instance of the list represents one category. For example the

Gnome Project has lists that are related to different applications, technologies, distributions, localizations, libraries and etc. (Gnome 2004*b*).

Search All posts can be indexed, and through archives many mailing lists offer full text search functionality.

Expert localization Mailing lists are very limited in localization of experts. But competence management can be achieved by analysing the contents of posts while searching for correct answers. Some mailing lists offer functionality to search for posts by specific authors, which makes expert localization much easier.

6.3.5.2 Community view

Mailing lists probably have the most varied impact on conversion modes, to some extent mailing lists play a role in all four conversions. This can probably be explained by the wide application of e-mail, everything from announcements, to source code patches and project planning can be managed over e-mail.

Socialization Socialization is supported through the communication and sharing of experiences by e-mail. As an example, imagine when a user describes some difficulty using a certain functionality in a utility. This creates the opportunity, not only to solve the problem, which would be externalization, but to incite understanding in others, of how the user perceives the utility and intended usage. Another example of socialization is when new developers lurk on mailinglists, to be able to get an impression on the present mailing list community. Lurking is a term for when someone silently follows the discussions on a mailing list, to pick up a sense of conduct and what the community is like (Raymond 2003). Such information can be retrieved partly through the explicit information, such as who are the active members, what is the tone in messages, what kind of topics are common, and are there any topics or behavior frowned upon. But also, the absence of information, what topics are not discussed, what knowledge seems to be assumed from the posters, can be as relevant to the lurker.

Externalization Externalization is apparent, everytime someone writes a new post on the mailinglist, he automatically externalized his knowledge. At least in the sense that he expresses, how he perceives the events which made him post. Further, mailing lists gives the poster an opportunity to express his intentions, and based on the feedback received from the community reflect on his knowledge. This can be expressed as a question – answer relationship, which gives both the poster and the respondent a opportunity to reflect on their individual understanding of the concepts and models involved in the discussion.

Combination Combination also has a strong presence. Mailing list archives are a constant source of solutions to problems other's have had before you (Raymond 2001). Search engines can locate several separate discussions of problems, and with a bit of experience the solutions proposed in these separate sources can be used to validate and elect the proper solution. Also, when a topic has several aspects or solutions, the advantages and disadvantages can appear in different threads, making combination essential in electing the solution. As a part of proper etiquette, users are normally encouraged and expected to search the mailinglist archives for solutions, before posting a question to the list.

Internalization As mentioned internalization is both a driver and a result of other conversion modes, such as externalization. When a person externalized his knowledge, by writing an e-mail for instance, he simultaneously reflects on the knowledge expressed, and concurrently internalizes it.

It is common that newcomers use mailinglists for question and answer sessions, a topic can start with a trivial question, and then as the thread evolves, as his understanding increases, the topic discussed moves on. This form of interaction on mailing lists works as a facilitator to the individuals learning effort. When knowledge is attained by performing learning task, the questions can become more elaborate.

6.3.5.3 Needs supported

The main purpose of using mailing lists is to support *collaboration and knowledge sharing* in open source projects. When used for announcements, mailing lists helps distribute information to all parties involved. When used for discussion, mailing lists allows all parties to take part in discussion. In the latter case, it supports coordination and collaboration beyond the limits of informal, face to face contacts, putting on communication some well defined, non-ambiguous, easily enforceable rules. It is enough to look at Linux Kernel mailing list archive to see that between January and July 2004, the average amount of messages was *6350* per month (Linux 2004). The amount of posts and success of Linux supports the conclusion that this way of communication is effective. What makes it even more valuable is the fact that all discussions are codified and accessible for future reference. Naturally, it is a media that supports coordination and collaboration that is independent of time and space.

To some extent all other needs are also supported. Acquiring knowledge about new technologies, accessing domain knowledge, sharing knowledge about policies and practices and knowing who knows what; locating sources of knowledge can be satisfied by in the same as it would be in informal environment where face to face communication is a basis for collaboration that is simply by asking other team members. Again, mailing lists have an advantage over traditional communication, because if given questions has already been answered, the answer can be located in archives.

6.3.6 Project websites

The technology behind the web in general needs no introduction. In open source projects, it is common that several separate web sites fulfill different needs of the community.

A general project web site often presents the deliverables, the software itself and end user documentation. Also common, are pointers to how to join the community and how to get help. For example instructions on what mailing lists are available, how to join them, and if the project uses IRC, what channel and server to use. Further, links to Howtos, and FAQs are usually posted on the general project web site. Essentially the general project website operates as a source of information on what other project practices exist in the project.

A developer web site is more focused on presenting the project's development status, and development practices. As such it is common to have instructions on how to access the source code repository, issue tracking tools and similar resources.

SourceForge (Sourceforge 2004) has taken the idea of the developer site and automated it. SourceForge is a project repository web, where any open source project can register and access a prepared default project environment. This web based environment includes a CVS, a project home page, an issue tracker, the ability to create mailing lists, web forums and FTP site. It is interesting to note that many reseach studies on open source, since the inception of SourceForge, has used it as a base for research.

Note that we analyse interactive web practices, like weblogs and wikis separately, and that such content is not a part of this practice.

6.3.6.1 Commodity view

The amount of technology and ways that allow to create and maintain contents of websites is truly great. Therefore, it is difficult to talk about general properties of the web as a practice. For the purpose of our analysis we look at the web site as a set of static documents that can be edited by limited number of people, typically involved directly in the project.

Accuracy and value It is up to the authors to make sure that the information presented on the website is relevant, and structured.

Dynamics Unless some kind of CMS system is used, maintaining the knowledge that is presented can be difficult.

Validity There is a big risk that information presented on the web site is outdated.

Concurrency Read only access to the presented content of course support concurrency, but to allow concurrent editing some form of tool, like CVS or a CMS that support concurrent editing, must be used.

Confidentiality, accessibility and roles Project site serves the purpose of general introduction into the project and is intended to be public. It is either maintained by developers or people designated to the role of maintainer.

Categorization and mapping Categorization and mapping are the basic roles of the project web sites. They contain pointers to other sources, people, etc. It is up to maintainer to make sure that the content is categorized properly.

Search It is difficult to make a general comment for this capability. For this reason we looked at projects we have previously defined as successfull and popular. Three out of six projects t had full text search available on the site. In two cases they were affiliated with Google.

Expert localization There are pages that contain contact infomation to different individuals involved in the project.

6.3.6.2 Community view

The project website is probably the most static form of “communication practice” analysed in this thesis. It thus has little impact on socialization, the support is in externalization and combination.

Socialization The content of websites has little effect on team building issues, or any of the other common forms of socialization. This is due to the little communicative support in static web content.

Externalization The authoring and publishing of documents on the website, naturally contains externalization. A common mean of showing what a project supports in terms of features, is to publish a set of screen shots of the running program, at least for projects which produces graphical interfaces.

A weakness in this practice, is that there is no explicit mean of giving the author of a document feedback. Thus reflection, guided by feedback from others is limited. Weblogs and wikis are a better choice, if an author wish to get feedback on the material published.

Combination Again, we find a practice where combination is present in the form of filtering and categorization. As mentioned the project web site functions as a hub, linking to other development resources and project practices. Thus the project website can be used to determine the most commonly mean of communication, for a specific project. This way, depending on how the project presents the practices on the website, it can steer how people choose to interact with the community.

The project collaboration web service Sourceforge, puts combination to use in an automated manner. Based on CVS commit statistics, and other data which can be gathered automatically by the service, a project activity level is calculated. This activitylevel can be used by newcomers to determine how active the project is. A very high activity level normally means that the project is in active development, while a lower activity level could mean that the project is dormant.

Internalization Published documents are used as base for some internalization, but the project website itself is primarily mean of spreading information, supporting combination, and not internalization.

6.3.6.3 Needs supported

As described in the general introduction, project websites works as knowledge maps, guiding the developer and user to information resources, what project practices are used, and other useful hints on how the community around the project operates. This fulfills the need of *locating sources of knowledge,* and *accessing domain knowledge.*

Project websites also serve as a authoritative source of infomation, for example the freedesktop.org project has published standards documents which the Gnome, and the other desktop environment project KDE relies on. The website in this situation also fulfills the need of *sharing knowledge about policies and practices.*

6.3.7 Weblog

Oxford English Dictionary (Oxford 2004) defines the term[6] *weblog*, or *blog*, as:

> A frequently updated web site consisting of personal observations, excerpts from other sources, etc., typically run by a single person, and usually with hyperlinks to other sites; an online journal or diary.

In their functionality, weblogs are similar to computer system logs mechanism, which logs events related to software work, in a special log facilities by recording time, usually category and textual description of the event. Blogs work in a similar manner. Events become *entries*, which are also called *posts*, are usually presented on the web site in reverse-chronological order.

Blog engine is a software that typically resides on server connected to the Internet. Apart from displaying stored entries to the readers visiting the site, they have the web interface allowing writer to add new entries usually with the use of the HTML language. In order to allow users without knowledge of HTML, slowly WYSIWYG[7] applications appear that also have the functionality that allows to create and editing text for people which are not in continuous connection with the Internet.

The content of entries mostly consist of written texts, but can also include multimedia elements such as pictures and video clips. Depending on focus and editing culture, blogs can be divided into four groups that can be described in two dimensions:*personal* vs *topical* and *individual* vs *community* (Nardi, Schiano & Gumbrecht 2004, cited in). Personal, mostly take a form of the online diary, record-keeping or chronicle where writer shares its own reflections and ideas about various subjects. Topical, servers a purpose of newsletter containing information about single or a fixed set of categories. They contain links to the external resources which are shortly commented by the author of the post. This style of blogging is also called *filter* (Lindahl & Blount 2003). Many of blogs show the combination of serious commentary with personal revelations (Schiano, Nardi, Gumbrecht & Swartz 2004).

There are three characteristics that have gained wide acceptance among blog users: *RSS feed*, *permalink* and *trackback*. Permalink, is a static link pointing to the particular post which allows bloggers to link to each others sites. trackback (Trott & Mena 2002), which was first implemented in Movable Type, makes it possible for bloggers to see if their post has been referred by others and eventually link back to the referrer. This is achieved by sending ping between blogs. If the writer is referring in his post to the post from another blog, then he sends ping to the engine holding posts of the original author which can automatically refer back to the referrer creating this way a list of referres to the given post. RSS feed is the XML document which contains latest sumaries of entries submitted to the blog. They can be fetched automatically from writer's site by RSS feedreaders, which are typically standalone desktop applications or implemented as a part of e-mail client or web browser software. Some web sites aggregate feeds, allowing users to search for links to entries containing in their texts the keywords that user was looking for.

All of the above elements cause blogs to become interconnected and create entity which is referred to as the *blogosphere* or *blogsphere ecosystem* (Lindahl & Blount 2003, Wikipedia 2004*a*).

[6]Draft entry, submitted in March 2003.
[7]**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et.

6.3.7.1 Commodity view

Accuracy and value Personal blogs are the place for stating private and subjective opinions of the author. Their structure is rather compact and standardized, but it is very difficult to verify their accuracy.

In both community and personal blogs, list of posts presented to the user on the main page is usually a short introduction to the complete contents that can be accessed if the short abstract presented interest the reader. Behind the scenes there exist a possibility for the users to discuss presented issues and ideas.

Dynamics Personal blogs are a way for people to share ideas on various subjects with their peers. Even though they offer the opportunity for people to submit comments about particular issues, it is a subjective judgment of the blog author whether to consider it or not. Some authors even consider immediate feedback as a threat (Nardi, Schiano & Gumbrecht 2004).

It has been recognized that habits in work on personal and individual blogs change over time. This is due to the fact that not only they create the audience, but the audience also creates the blog (Nardi, Schiano & Gumbrecht 2004). Therefore it can be concluded that their nature is dynamic enough to allow such situation to occur.

Community and topical types are much more dynamic in nature. Very rarely the post that has been submitted is altered in any way, however, the discussion that takes place behind the scenes is truly dynamic. Sometimes, entries are posted with the purpose of sparking the discussion on the particular subject to obtain as many opinions as possible.

Validity It is very difficult to verify the validity of entries in both personal and community blogs. The date of posting the entry or times for submitting comments can be some kind of indicator, but in general it is difficult to verify whether a given knowledge is still valid.

Concurrency In case of personal blogs concurrency is required for submitting comments which for the web based engines is typical per se.

Submitters of posts into community blogs can also do it simultaneously.

Confidentiality, accessibility and roles Some personal blogs can be protected from a public and be viewable only by certain individuals, but it is not the general practice. It is the purpose of their existence to be public and widely accessible.

In case of community blogs, there exist some moderating functionality that allows moderators to accept for public viewing only posts meeting strictly defined criteria.

Categorization and mapping Some applications allow titles for posts, whereas others allow to specify a category and/or a keywords for entry contents. It is later possible to browse through the entries of one particular category (Nardi, Schiano & Gumbrecht 2004).

Search Blogs are organized chronologically, what is allow their easy browsing. Finding specific information with the use of indices may be more difficult (Nardi, Schiano & Gumbrecht 2004). Some applications introduced full text search functionality which is slowly becoming standard for all engines.

Expert localization The contents submitted to the personal blogs can be a subject to analysis of the given individual but expert localization from within the context of private blogosphere is rather impossible.

Community blogs allow to see all posts submitted by a given individual and trace discussion on forums. This allows to define the scope of interest of the contributor and eventual contact.

6.3.7.2 Community view

Since blogs are web sites, they of course have much in common. But opposed to web sites, the predominant role in blogging isn't externalization, even though the expressed opinions of course are externalized when published on a blog, externalization is mainly a mean for the socialization which is the main function of blogs. The expression in blogging also have an catalytic role to internalization.

Socialization When blogging people share both technical and personal issues. The technical issues are of course what primarily makes blogging relevant to open source developers. The blog is a space where opinions and discussions can be published in a more permanent way, than say mailinglists, while still allowing discussion and feedback. Also the blog works as a mean to present progress reports to the developer community. Meanwhile, the social, team building effect, of blogging is also apparent, especially considering that personal issues are discussed on developer blogs. The social aspect is well illustrated in the fact that open source developer communities have begun to create joint blog portals, where individual developer blogs are joined into one community blog, for example Planet Gnome[8].

Herring *et al.* argue that blogs are attractive because *they allow authors to experience social interaction while giving them control over the communication space* (Herring, Scheidt, Bonus & Wright 2004). The social side of blogging can probably thrive just because it is possible for the individual blogger to control what reactions are published in relation to his posts. The blog becomes a safe place to share personal issues, without leaving the blogger completely open to attack.

Externalization As mentioned, the externalization involved in blogging can almost be labeled as a side effect. Still, all the things expressed in blogs are of course externalized. The interesting aspect of externalization is here reflection, as described in the commodity analysis, the original posts seldom change, instead the ideas are discussed, and sometimes the discussion leads the author to publish a follow up post, where he elaborates on his ideas, and considers the feedback he has received in the blog comments.As a reader it thus is possible to follow the evolution of the blogger's line of thought.

Combination Originally most blogs were some form of filters for external news sources, where the blogger posts commentary on current news items or new development (Herring et al. 2004). Although blogging pattern has changed a bit today, filter-blogging remains common, and is an example of combination. The blogger elects newsworthy issues, based on his own interests and personal judgment, thus working as a news filter, sorting and ignoring less interesting news, while highlighting interesting issues. The reader thus does not necessarily have to keep an eye on the area himself, he merely need to keep up to date on the blog covering that area. For instance, Miguel de Icaza is one of the founders of the Gnome project, and highly involved in the Mono project. In Miguel's web blog[9], it thus is easy to find commentaries on the current evolution of Mono and Gnome. For an casual visitor in the Mono community, Miguel's blog works as a aggregator, pointing out what is important, so that the visitor does not have to do that work himself.

Internalization It is easy to assume that a text medium is all about externalization, but not so in the case of blogging. When Nardi *et al.* asked bloggers why they blog, a common answer was that they use the blog as a method to structure thinking.

> Alan observed that once having started a blog, it "forced" him to keep writing, a discipline he deemed important for his work. He said, "I am one of those people for whom writing and thinking are basically synonymous." Even, a graduate student in genetics, echoed this thought in saying he liked blogging because it was "thinking by writing." (Nardi, Schiano & Gumbrecht 2004).

This kind of personal reflection and structuring of thoughts, is a way to understand the information received and through self-expression create and internalize it as knowledge.

[8]Planet Gnome: http://planet.gnome.org/

[9]Miguel de Icaza's web blog: http://primates.ximian.com/~miguel/activity-log.php

6.3.7.3 Needs supported

While looking on the open source projects of our choice we have recognized that blogs are mainly used on one hand for *collaborating and sharing knowledge* and on the other, *acquiring knowledge about new technologies.* This is achieved by using both community and individual types of blogs.

Individual/personal blogs are maintained rather independently by developers working on a given project. Their contents is made of mixture of both personal and professional issues. There is not much difference among projects as of how individual/personal blogs are created and maintained.

The approach towards community blogs is where projects differ. Probably the most interesting usage scheme can be observed in KDE project. KDE developers use individual blogs to share their ideas and thoughts about the functionality of applications, experiences with new technologies or simply problems that they have to deal with during their daily activities. It is very common for the readers to submit their comments with suggested solutions to those problems, experiences about the same technologies or flaws in thinking about new functionality that author of the post was planning to implement. So far their usage scheme is not much different from the common individual blog. The difference comes in aggregation of feeds. KDE project maintains a website, called *Developer Journals* (KDE 2004*a*), which collects RSS feeds from all registered KDE developers. Those feeds are presented in a clean and tidy way on the common webpage, where both other developers and users can access them in an uniform way without the need of going into separate blogs in search for posts. Each post belongs to at least one category. Readers interested in particular types of issues can browse through the posts in a specific category.

6.3.8 Wiki

WikiWikiWeb, shortly *wiki*, is a collaborative software that can be used to create a set of hypertext documents, or the created document collection itself (Wikipedia 2004*c*). Originally wiki was established at the Portland Pattern Repository by Ward Cunningham, one of the creators of the extreme programming development model. Since its inception the concept has gaining acceptance and a great number of open source wiki engines has been developed.

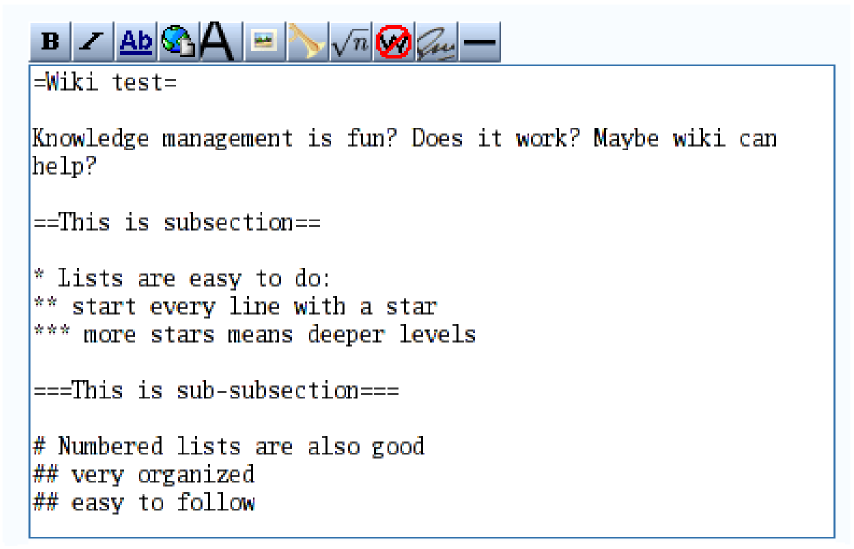
Wiki software allows an author to create and edit collections of interconnected documents that are mostly targeted towards web publishing. Editing is performed with a plain web browser and authors do not have to install additional software. Authors use a special markup language (see Figure 6.1), that was designed to be simple and easy to use. Pages created using this language are transformed, transparently to the user, into HTML documents to be rendered in the browser window (see Figure 6.2). The reason for this design is that HTML syntax is considered too complicated to allow fast editing and distracts authors from the actual content.

There exist a number of wiki engines and they slightly differ in functionality and supported markup. However, there is an underway initiative that has the purpose of delivering *Wiki Markup Standard* that trying to unify all existing variations, so all engines could share the same set of markup rules.

In general wiki is valued for its simplicity in creation and update of the contents. Normally wiki pages are open to public editing, that is anyone visiting a wiki site can change or add new information to the wiki, and the modifications do not require review before being publicized.

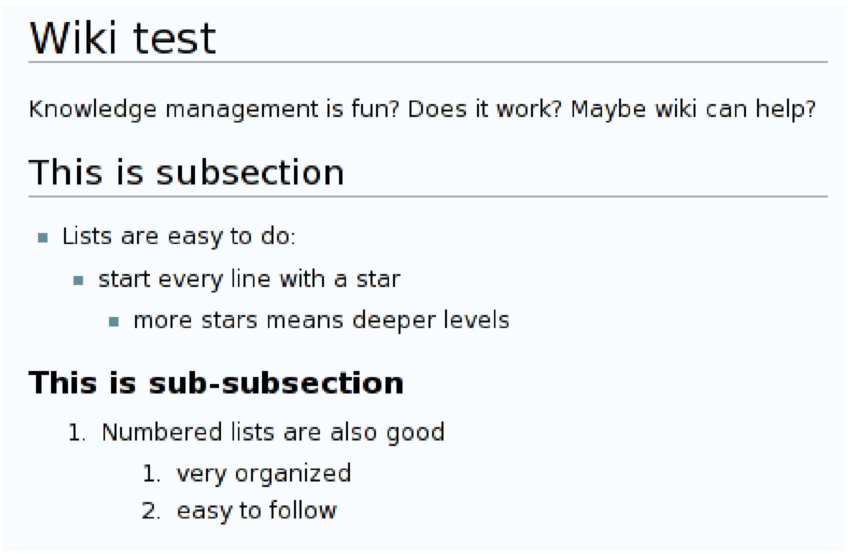
6.3.8.1 Commodity view

The concept of wiki is very general, however, it comes to particular engines to have certain functionality or not. The following descriptions of capabilities are based on the MediaWiki engine, that is used for the purposes of the community developed encyclopedia called Wikipedia. The English version of Wikipedia contains over *300 000* wiki pages as of July 2004 (Wikipedia 2004*b*).



```
=Wiki test=

Knowledge management is fun? Does it work? Maybe wiki can
help?

==This is subsection==

* Lists are easy to do:
** start every line with a star
*** more stars means deeper levels

===This is sub-subsection===

# Numbered lists are also good
## very organized
## easy to follow
```

Figure 6.1: Wiki source code edition in browser.



Wiki test

Knowledge management is fun? Does it work? Maybe wiki can help?

This is subsection

- Lists are easy to do:
  - start every line with a star
    - more stars means deeper levels

This is sub-subsection

1. Numbered lists are also good
   1. very organized
   2. easy to follow

Figure 6.2: Wiki page rendered in browser.

Accuracy and value The value of pages that are available through wiki comes from its nature. It is safe to say that it is socially determined. This is due to the fact that it is constantly reviewed and verified by number of individuals. During this process, they evolve to contain mature and accurate contents (Deursen & Visser 2002). Of course in case of vandalism (intentional destruction of value), the content at given time might be inaccurate. However, each wiki has the functionality allowing to see all previous versions of the content and this way allows to recognize whether the act of vandalism has occurred. In addition, pages are heavily crossconnected[10] and while the value of the text can also be verified by number of the references to another source.

The MediaWiki allows also to create a set of featured articles that favor the most valuable texts through the process of verification. Texts are first nominated then must be reviewed and accepted without any objections by all reviewers to become featured.

Dynamics The ease of edition, very little constraints, high availability make wiki a tool/practice that is very dynamic. Lack of the formal process allows very rapid changes. Therefore, they have been recognized to be especially useful for codification of the knowledge where the underlying structure is very likely to change (Deursen & Visser 2002).

MediaWiki has the feature of discussion about each wiki what increases interaction between contributors.

Validity Wiki engines have the functionality that allow not only to verify time of codification of knowledge, but also to see the complete revision history. Users are able to see previous versions of the page, differences between texts with highlighted changes, all authors and changes that have been made within definable timeframe. In addition, the most recent changes are exposed in particular, easy to track and be a subject to verification.

Concurrency The great benefit of wiki being a web-based solution, is that a big number of users can work from geographically dispersed locations at the same time. Obviously, concurrency problem do not appear until two users want to work on exactly the same text. To allow it, a number of features has been implemented. In general, it is impossible to verify whether given page is currently being edited. Therefore, it is being verified at the time of submission. A mechanism checks for potential edit conflicts. If such occurs, author can choose whether to employ his changes, ignore the changes done by another user or revert to previous version of the text. One of the newest features of wiki engines is automatic merge. They allow to limit a number of edit conflicts that must be handled manually to the situation in which literally the same lines of text have been edited.

Confidentiality, accessibility and roles Wiki has been designed with the openness in mind. Its functionality is based on this characteristic. Anybody can see and edit all wiki pages.

Most public wikis has the functionality that allows registration, but the general practice is not to require individuals to register. Since the contents of the repository lays in the interest of general public, there are very little limitations as for accessibility and roles in general. Most engines, however, provide certain ways to limit write access. There exist methods for protecting particular entries from edition. So-called administrators can set and revoke the protection (Wikipedia 2004*c*). In general this is considered as the violation of wiki philosophy and therefore is avoided.

Categorization and mapping It is possible to define different categories that allow to group wiki pages into different topic. In addition, they have a functionality of cross references. They allow reader to explore not only the main term, but a set of related items as well.

Search Wikis usually offer at least title search, but in most cases they have also the full text search. In addition, it is possible to search for a specific key word (Nakisa 2003). If such term does not exist, the nearest match is returned. For the convenience of users a set of special pages is available. It is possible to see a list of all images, users, pages by title, site statistics, etc,.

[10]We refer to the physical pages connected with the use of hyper links.

Expert localization Unless the changes to the page has been done anonymously, it is possible to track all contributions of the person that has edited the article which is in the interest of the reader. This allows, to some extent, verify the scope of interest of contributor, judge the quality of other contributions. People editing wikis usually create their homepages within the system which contains contact information, for example in the form of e-mail or instant messaging system. It is up to the given contributor to decide about preferred ways of contact.

6.3.8.2 Community view

As opposed to traditional static web sites, wikis are more dynamic and through the openness – it is common that any visitor is allowed to make changes at any time – can encourage more spontaneous sharing of knowledge. Like ordinary web sites, wikis are not communication focused, but documentation tools. This of course means that the emphasis in wikis is externalization and combination.

Socialization Socialization is primarily performed through some form of observation, and as wikis are poorly suited for communication, they are not suited for socialization.

Externalization Wikis support externalization, the simple interface, forms based and with a transparent syntax, makes it easy for any user to contribute to a wiki. The low learning curve, makes it suitable for newcomers to add, and work on already existing pages.

One aspect of externalization is reflection. In wikis, one reflection supporting practice is to encourage visitors, not to change old texts if they have comments, but instead to append the comments to the end of the page. This is encourage especially if the visitor wish to add a view opposite the current page offers. An alternative solution to this problem is to have special discussion pages as sub-pages to the topic discussed. This solution is for example used by Wikipedia. The ability to comment page content in connection to the page in wikis, offers the reader an opportunity to evaluate a topic, not merely on the basis of the facts presented, but also the arguments and discussion supporting the facts.

Combination As static web sites, wikis are commonly used as repositories for links to other information resources. For example, if a solution for a specific problem is proposed on a mailing list, it can be copied to a wiki, to make it possible to permanently reference it. The Gnome support wiki for example uses a wiki as a tool to create a FAQ, which is based partially on answers found on various mailing lists (Project 2004).

Internalization Internalization emphasize learning new knowledge, and wikis by themselves have little support for this. Wikis are, mainly information repositories, and though the information wikis contain, can be internalized, the wiki tool itself is not primarily targeted at this.

6.3.8.3 Needs supported

Below we present the analysis of wiki usage patters based on the KDE project (KDE 2004*b*). It is very intersting to realize that wikis can be used to satisfy all of the software engineering knowledge related needs, recognized by Rus *et al.* (Rus & Lindvall 2002).

The most critical part of the functionality is the categorization system. While browsing thought categories, user can localize necessary resources relevant only to developers, quality team, users, documentation contributors and users. Also, types of actions that one might one to perform on the applications fro the projects such as installation, configuration or generally administration.

Worth mentioning is also the fact that categories tend to overlap and not necessary follow strict rules. Even though, it is still possible to find relevant knowledge. Category and full text search also come in very handy.

Acquiring knowledge about new technologies. Apart from contraining descriptions about various technologies, wiki can be used to localize the URLs of RSS feeds that are represent all additions to articles, components, architecture and many other technology intensive areas.

Accessing domain knowledge. Just by typing in the full text search “kde technologies”, user can find the page titled *KDE, the integrative desktop* which contains the information about types of technologies, possibilities of integration or compatibilities between different desktop environments.

Sharing knowledge about policies and practices. To find out about all relevant sources for user interface guidelines, it is just enough to localize *KDE Quality Team HOWTO User Interface* wiki page which contains a general description of the subject, showing the approach in KDE towards related issues and finally points to other sources such as GUI design standards, relevant mailing lists and similar elements.

Knowing who knows what; locating sources of knowledge. KDE wiki contains pointers to various other source of knowledge, which has also been covered in this thesis such as electronic mailing lists, IRC channels, lists of Howtos, etc,. Recently all KDE related FAQs have been moved to wiki. The contents of knowledge sources is described giving the user insight into what can be obtained and achieved in different places.

Collaborating and sharing knowledge. Apart from giving the possibility to share knowledge and tips between users and developers about the functionality and usage of the application, that the project is delivering independently from the time and place of work, wiki in KDE projects is also used as coordination tool. On the page *Developers wanted* that gives current and potential developers an overview of what kind of functionality needs to be implemented in which part of the system.

6.4 Summary

This chapter contains the synthesis of all previously presented facts, applied in a characterization framework. The application of this framework for the purpose of analysis of a number of practices typical to open source development, gives an overview of the ways in which knowledge realted needs are satisfied in open source development.

In order to draw a more complete picture of knowledge management aspects of open source practices practices, we have decided to present aspects of both the commodity and the community views. In addition, since open source projects are software projects and software engineering knowledge needs typical to traditional development, exists in open source development, in the analysis of the practices we identify software engineering needs. Therefore, the complete characterization framework consists of four parts: a general description, a list of capabilities that are typical to the commodity perspective, a list of properties that are typical to the community perspective and a list of software engineering knowledge related needs.

The set of practices that are analysed, are selected using a two step method, which lead us to identifying the general practices. Our initial selection was verified by observing six common open source projects: Apache, GCC, Gnome, KDE, Linux kernel development and Mozilla. Through the verification it was recognized that FAQs, project web sites, Howtos, IRC, issue tracking, mailing lists, weblogs and wikis are commonly used in open source projects. An analysis of those practices from knowledge management perspective allowed us to recognize their knowledge related capabilities and aspects of software engineering needs.

The last chapter of this thesis, contains the conclusions of our research, a short discussion about interesting issues that have appeared, and suggestions for future research.

The greatest enemy of knowledge is not ignorance, it is the illusion of knowledge.

– Stephen Hawking

Chapter 7

Conclusions and discussion

This chapter concludes our research. Here we discuss our findings, present our conclusions, and suggest potential areas for further research. The chapter is divided into three sections: first we summarize the thesis by providing answers to our research questions. Secondly, we present and discuss the conclusions we have drawn from the research. Finally we present areas, which we believe could benefit from further research.

7.1 Answers to the research questions

The main goal of this thesis was to look upon open source development from the perspective of knowledge management. We wanted to enhance the understanding of the mechanisms present in open source development, which are related to knowledge acquisition and sharing. Putting knowledge management labels on the practices commonly performed in this type of development, can result in new means to understand and improve how to deal with knowledge in open source projects.

When we begun the thesis research, we defined a number of research questions, which aimed to satisfy our primary research objective. In the following section we provide answers to those questions, based on the results of our research. In relation to each answer, along with the description of the way used to answer it, we also describe how the answer contribute to the overall thesis goal.

1. How is knowledge management understood, and is it possible to analyse project practices from that perspective?

After performing a literature survey, we found two different but widely accepted perspectives or views on knowledge management. Both views approach knowledge with the purpose of managing it, but the methods proposed in the commodity view are quite different from the methods proposed in the community view.

In the commodity view, knowledge is treated as an object, which can be used in a similar manner as goods in a market. To make that possible, those goods need to be extracted from individuals, processes, and organizations and converted to a form that is transferable. Therefore, in this view, the main goal of conscious knowledge management is creating effective means to store and retrieve knowledge. Thus knowledge management initiatives focus on identification of the capabilities of knowledge and the selection of a proper codification media.

In the community view, knowledge is considered to be tightly coupled with the process of knowing and experiences of the knower. Social activities and interactions between individuals thus become the most important objective of effective knowledge management. The focus of knowledge management initiatives in this view, is the recognition of means that encourage and support knowledge sharing through human networking, where trust and collaboration are key factors.

Since these two views appears to complement each other, it became desirable to analyse open source using both views. From the perspective of the commodity view, we could look at the supportiveness of different entities in the process of knowledge codification. From the perspective of the community view, the subject of analysis are the mechanisms of interactions between groups and individuals connected through networks.

Contribution to the main goal The answer to the above question, gave us means for approaching open source practices. It allowed us to define perspectives from which we could approach practices in open source, looking through a prism of genereal knowledge management theories.

2. What are the knowledge management aspects of traditional software development and which ones are relevant to open source?

As in the case of general knowledge management, we performed a literature survey of research that connect software engineering and knowledge management. Most of the proposed approaches focus on efficient codification. We have presented a summary of approaches that are influential.

In order to recognize the relevance of specific aspects of traditional software development with respect to open source development, we performed a survey of the current literature on the nature of open source development. We observed, that many of the obstacles found in traditional development were not relevant to open source development. Though, we also concluded that the knowledge related needs that are typical to traditional development, also exist in open source.

Contribution to the main goal We hoped to obtain some additional suggestions as how to approach our analysis of open source practices. This was partially satisfied by the recognition of knowledge related needs of traditional software development. We concluded, that it is possible that the practices of open source implicitly try to satisfy those needs. Therefore, we complemented our analysis of practices with a perspective on knowledge related needs.

3. What are the common, knowledge related practices in open source projects?

In order to answer this question, we followed a two step process. In the first step we chose practices that we recognized to be related to knowledge acquisition and sharing. Our choice was supported by observations of open source project collaboration tools like SourceForge and NovellForge. We selected a preliminarily set containing the following practises: issue tracking, FAQs, project websites, Howtos, IRC, mailinglists, weblogs and wikis.

In the second step, we verified that the selected practices existed in a set of widely acknowledged projects. We also verified that the practices are used to share knowledge in the projects. Except for wikis, all the practices are present in the set of projects, which supports the relevance of our preliminary selection. We decided to include wikis in the analysis because it is interesting from a knowledge sharing perspective, even though it was not widely present in the projects we used for verification.

Contribution to the main goal By answering the above questions we obtained a set of practices common in open source projects related to knowledge sharing and acquisition. Those practices were analysed in detail from the perspective of knowledge management.

4. What are the knowledge management aspects of practices in open source projects?

Based on the results of the literature survey, we derived an analysis model for assessing open source practices. The model considers three perspecives: the commodity view, the community views, and software engineering knowledge related needs. We used the model to analyse the practices related to knowledge acquisition and sharing in open source software projects.

Some practices appear to have more capabilities, better support for collaboration and community building than others. However, the purpose of our analysis was not to compare the practices to each other, only to recognize aspects of knowledge management in open source projects.

Contribution to the main goal By giving the answer to the above question, we fulfilled the main goal of our thesis. The outcome of the analysis, the conclusions, are presented in the next section of this chapter.

5. How does open source development differ from other development models in the context of knowledge management?

This question was also answered through a literature study where we identified how open source development is described, both by the open source community itself, as well as the research community.

We found that the open source development model, as opposed to traditional development models, vary immensely depending on who describes it. In essence only two basic attributes have been found general to all open source projects: that the source code is available, and that the developers are also the primary users. We observed that despite a great variation in managerial attributes, many projects share a set of common project practices and tools. This implies that the tools and the practices around those tools have a great impact on the way open source projects are managed. What we also found, was that the driving force in open source project is the meritocracy based community. The research also describe open source as an ecology where continuous change occur, not only in the project requirements, but also in the needs of project practices, and tools used.

Contribution to the main goal The observation of the project practice's role in the open source ecology supports the assumption that it is relevant to analyse knowledge management aspects of individual practices. Further, the comparison to an ecology, may be the answer to why open source developers manage to create and share knowledge, despite the fact that they seldom meet in person.

7.2 Conclusions

We present three main conclusions, based on our literature studies and the analysis of practices.

We found that the nature of open source drives the creation and adoption of development practices, such that those practices support knowledge sharing sufficient enough for the community to create a successfull software development environment. Further, we found that the methods we identified and analysed have the means for knowledge acquisition and transfer, which are advocated by many scholars in knowledge management.

We found that each practice analysed, fulfills a role in supporting knowledge management in open source projects. Each practice is centered in different knowledge conversion modes, such that all the practices complement each other, and toghether supports all conversion modes.

We found that, by using our analysis method, we have successfully been able to show that knowledge management does occur in open source projects, and further how it supports the creation and sharing of knowledge in open source projects.

7.3 General observations

Apart from the above presented main conclusions of this thesis, we have made a couple of additional observations. Even though we have not been able to conclusively analyse them through the material presented in this thesis, we still find these observations interesting enough to present them, and argue their merit.

7.3.1 Trust issues

Trust plays a very important role in knowledge management. In the commodity view, it is a necessary factor for the knowledge market to operate. In the community view it is recognized as a basis for socialization, which in turn is a basis for knowledge sharing and creation.

It is interesting to observe, that trust is a inherent part of the open source development model too. Most of the development and communication takes place using the Internet. Individuals have to to express their ideas explicitly, either through mailing lists, IRC channels or discussion forums, in order for others to reflect or comment. Source code is of course also contributed over the net. In most cases records of discussions, conversations, agreements and submissions are available for open review. Therefore it is very difficult to be dishonest; since every statement can be retrieved, lies are easy to expose.

In addition, the risk of someone getting the credit for something that he did not actually do, is rather small. In the open source community taking undeserved credit for other's work is frowned upon. Again, due to the traceability of actions in the community, individual that does not obey the rules, risk loosing professional credibility. In an environment based on meritocracy, loosing credibility means loosing the ability to act.

7.3.2 Informal development with formal practices

It is interesting to see that the open source community, which often is percieved as rather informal, have created a such number of formal but unwritten rules. Take for example Bugzilla. It is a very sophisticated issue tracking system, which almost require a structured organization to be usable. Yet, despite this complexity, Bugzilla has become a part of many open source projects.

We also think it is interesting that two very opposite approaches to knowledge sharing, wikis and Bugzilla, at the same time are the two practices which are the most knowledge-needs supportive. One is very formal whereas the other almost is an selfregulatory chaos.

This combination of informal settings and tools which requires a structured practices approach is intriguing. It lead us to believe that there is a strong self regulatory power in the nature of open source.

7.3.3 Overcoming barriers and obstacles

We also observe that most barriers and obstacles to knowledge management initiatives are either nonexistent, or have somehow been overcome, in open source development.

The first recognized obstacle is the lack of commitment among project managers. Tight schedules and a difficulty in including knowledge management related initiatives in performance reports, are simply not relevant issues in open source. The reason is that open source projects are separate from any corporate organizations involved in development. The projects have no tight schedules nor performance reports, even if the corporations involved in development has.

Also absent is any fear that someone will get credit for using knowledge that he did not actually create. As has been discussed previously, it is a part of the open source nature to give credit where credit is due. Even the smallest contributions as recognized and publicized. In addition, people do share their ideas so others could reuse them or base some new creations and innovations. This is what open source is about.

Neither is there any fear of being recognized as an expert. Individuals decide for them selves which tasks to focus on and how to contribute. Instead, being identified as an expert leads to more influence in the community.

Another obstacle is, that people are afraid to become subject to judgments and verification, is still existent, but in case of open source is rather motivator for better quality software. Since people operate in the meritocracy in which the commitment and expertise are key factors, they want to do their job as good as possible.

When people work on open source software, they need to make their ideas and knowledge explicit. They use methods that force them to unconsciously codify their ideas, even volatile issues and rapidly changing technology. That is how the last two obstacles are implicitly removed in open source.

7.4 Future research

During the research for this thesis we have found some intriguing properties of open source that might be worth exploring in the future.

7.4.1 Mapping motivation

The motivation of members of open source communities, which drives both creation of the open source software and the evolution of practices is interesting. It is not uncommon for developers, to work on open source projects, on their spare time, after a full day of work. They are eager to share their thoughts, discuss ideas and create new knowledge.

While some research have been done in the area of developer motivation in open source, it might be desirable to recognize and explore the mechanisms influencing motivation. Looking from the perspective of corporate software organizations, it could be interesting to see if this knowledge could be mapped to their own organization's environment.

7.4.2 Supporting evolution

As recognized in this thesis, evolution is an inherent part of open source. At the same time, the impact that evolution has on open source projects appears to be little known in the community. Similarily, commercially developed closed source projects also needs to be continously updated, as well in product, as in project practices.

It would be interesting to explore the technchal nature of open source projects, and analyse the mechanisms that support the evolution. Both in the perspective of improving open source projects, and in the perspective of supporting process improvement in traditional closed source development.

7.4.3 The impact of voice based communication

For the moment, most of the communication that takes place in open source development is in written form. This written communication is the basis for creating trust in the community. Trust is the key determinant for knowledge sharing and open source development in general to occur.

Recently, voice communication over Internet has become videly available. A question that could be worth to investigate is how voice communication will influence open source development. Maybe, people contributing to open source projects will ignore certain technical advancements, in order to make it possible to operate, or maybe they will adopt those technologies and create more effective and more competitive ways for sharing and creating knowledge?

7.5 Acknowledgements

We would like to express our deep gratitude to our friends and families for their support and input while researching and writing this thesis. We would also like to thank our supervisor and our opponents for their constructive critisims of our work.

Bibliography

Abrahamsson, Pekka, Outi Salo, Jussi Ronkainen & Juhani Warsta. 2002. "Agile software development methods. Review and analysis." *VTT Electronics* .
URL: http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf

Alvestrand, Harald. 2004. "Linux Counter, Project.". Last accessed April 15th 2004.
URL: http://counter.li.org/estimates.php

Aurum, Aybuke. 2003. Supporting Structures for Managing Software Engineering Knowledge. In *Managing Software Engineering Knowledge*, ed. Aybuke Aurum, Ross Jeffery, Claes Wohlin & Meliha Handzic. Springer-Verlag.

Basili, Victor R., Gianluigi Caldiera & H. Dieter Rombach. 2002. Experience Factory. In *Encyclopedia of Software Engineering online*. John Wiley & Sons Inc. Internet version, last accessed April 14th 2004.
DOI: http://dx.doi.org/doi:10.1002/0471028959.sof110

Bierly, Paul E. III, Eric H. Kessler & Edward W. Christensen. 2000. "Organizational Learning, Knowledge and Wisdom." *Journal of Organizational Change Management* 13:595–618.
DOI: http://dx.doi.org/doi:10.1108/09534810010378605

Bollinger, Terry. 1999. "Linux and Open-Source Success : Interview With Eric S. Raymond." *IEEE Software* 16(1):85–89.

Brooking, Annie. 1999. *Corporate memory : strategies for knowledge management*. London : International Thomson Business Press.

Bugzilla, Team. 2004. "The Bugzilla Guide - 2.19 Development Release." *Bugzilla Project Web Site* . Last accessed August 7th 2004.
URL: http://www.bugzilla.org/docs/tip/html/

Callahan, John R., Reshma R. Khatsuriya & Randy Hefner. 1997. "Web-based Issue Tracking for Large Software Projects." *IEEE Internet Computing* 2(5):25–33.

Capiluppi, Andrea, Patricia Lago & Maurizio Morisio. 2003. Characteristics of Open Source Projects. In *Proceedings of the 7th European Conference on Software Maintenance and Reengineering*. pp. 317–327.

Castells, Manuel. 1996. *The Information Age : Economy, Society and Culture. Vol. 1, The Rise of the Network Society*. Blackwell Publishers.

Cawley, Piers. 1995. "Majordomo : A Mailing List Handler." *Linux Journal* 1995(13).

Choo, Chun Wei, Brian Detlor & Don Turnbull. 2000. *Web work : information seeking and knowledge work on the World Wide Web.* Kluwer Academic, Dordrecht.

Corbet, Jonathan. 2004. "Another look at the new development model." Internet. Last accessed July 29 2004.
URL: http://lwn.net/Articles/95312/

Cox, Alan. 1998. "Cathedrals, Bazaars and the Town Council." Internet. Last accessed August 6 2004.
URL: http://slashdot.org/features/98/10/13/1423253.shtml

Cubranic, Davor, Reid Holmes, Annie T.T. Ying & Gail. C. Murphy. 2003. Tools for Lightweight Knowledge Sharing in Open-source Software Development. In *Workshop on Open Source Software at ICSE 2003.*

Darke, Peta & Graeme Shanks. 2000. Case Study Research. In *Research Methods for Students and Professionals : Information Management and Systems*, ed. Kristy Williamson. Centre for Information Studies : Charles Sturt University.

Davenport, Thomas H. 1997. *Information ecology : mastering the information and knowledge environment.* Oxford University Press.

Davenport, Thomas H. 1999. Knowledge Management and the Broader Firm : Strategy, Advantage and Performance. In *Knowledge Management Handbook*, ed. Jay Liebowitz. CRC Press.

Davenport, Thomas H. & Laurence Prusak. 1998. *Working knowledge : how organizations manage what they know.* Paperback ed. Harvard Business School.

Dawson, Christian W. 2000. *The Essence of Computing Projects : A Student's Guide.* Pearson Education Limited.

Desouza, Kevin C. 2003. "Barriers to effective use of knowledge management systems in software engineering." *Communications of the ACM* 46(1).
DOI: http://dx.doi.org/doi:10.1145/602421.602458

Desouza, Kevin C. & Awazu Yukika. 2003. "Constructing Internal Knowledge Markets: Considerations From Mini Cases." *International Journal of Information Management* 23(4):345–353.
DOI: http://dx.doi.org/doi:10.1016/S0268-4012(03)00056-2

Desouza, Kevin C., Jeffrey J. Raider & Thomas H. Davenport. 2003. "Intellectual Asset Reuse in Software Development." *Accenture Research Note* . Last accessed May 5th 2004.
URL: http://www.accenture.com/xd/xd.asp?it=enweb&xd=_insCresearchnoteabstract_186.xml

Desouza, Kevin C. & Thomas H. Davenport. 2003. "Reusing Intellectual Assets." *Accenture Research Note* . Last accessed May 5th 2004.
URL: http://www.accenture.com/xd/xd.asp?it=enweb&xd=_insCresearchnoteabstract_184.xml

Deursen, Arie van & Eelco Visser. 2002. The Reengineering Wiki. In *Proceedings of the Sixth European Conference on Software Maintenance and Reengineering (CSMR 02).* IEEE Computer Society pp. 217–220.

DiBona, Chris, Sam Ockman & Mark Stone. 1999. *Open Sources: Voices from the Open Source Revolution.* O'Reilly & Associates, Inc.

Dinkelacker, Jamie, Pankaj K. Garg, Rob Miller & Dean Nelson. 2002. Progressive Open Source. In *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002.* ACM pp. 177–184.
DOI: http://dx.doi.org/doi:10.1145/581339.581363

Dougherty, Dale. 2001. "GNOME's Miguel de Icaza on .NET : The New Development Environment for the Next 20 Years.". Last accessed August 1st 2004.
URL: http://www.ondotnet.com/pub/a/dotnet/2001/07/09/icaza.html

Drucker, Peter F. 1992. "The New Society of Organizations." *Harvard Business Review* 70(5):95–104.

Eisner, Howard. 2002. Information Age. In *Encyclopedia of Software Engineering online.* John Wiley & Sons Inc. Internet version, last accessed June 17th 2004.
DOI: http://dx.doi.org/doi:10.1002/0471028959.sof157

Gacek, Cristina & Budi Arief. 2004. "The Many Meanings of Open Source." *IEEE Softw.* 21(1):34–40.

Gnome, Project. 2004*a*. "Gnome Wikis.". Last accessed August 8th 2004.
URL: http://gnomesupport.org/wiki/index.php/HomePage?action=PageHistory

Gnome, Project. 2004*b*. "The mail.gnome.org Archives.". Last accessed August 8th 2004.
URL: http://lists.gnome.org/archives/

Gnome, Team. 2004*c*. "Gnome Bug Submission into Bagzilla, Bug ID 146749." *Gnome Project Web Site* . Last accessed August 10th 2004.
URL: http://bugzilla.gnome.org/show_bug.cgi?id=146749

Godbout, Alain J. 1996. "Information Vs. Knowledge." Internet. Last accessed June 17th 2004.
URL: http://www.km-forum.org/ajg-002.htm

Godfrey, Michael W. & Tu Qiang. 2000. Evolution in Open Source Software: A Case Study. In *Proceedings of the International Conference on Software Maintenance. ICSM 2000.* IEEE Computer Society pp. 131–142.

Gooch, Richard. 2004. "The linux-kernel mailing list FAQ." Internet. Last accessed July 29 2004.
URL: http://www.tux.org/lkml/

Handzic, Meliha. 2003. Why Is It Important to Manage Knowledge? In *Managing Software Engineering Knowledge*, ed. Aybuke Aurum, Ross Jeffery, Claes Wohlin & Meliha Handzic. Springer-Verlag.

Hann, Il-Horn, Jeff Roberts, Sandra Slaughter & Roy Fielding. 2002. "Why Do Developers Contribute to Open Source : First Evidence of Economic Incentives.".
URL: http://opensource.ucc.ie/icse2002/HannRobertsSlaughterFielding.pdf

Hansen, Morten T., Nitin Nohria & Thomas Tierney. 1999. "What's Your Strategy For Managing Knowledge?" *Harvard Business Review* .

Hars, Alexander & Ou Shaosong. 2001. Working for Free? : Motivations of Participating in Open Source Projects. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences, 2001.* pp. 2284–2292.

Hecker, Frank. 1999. "Setting Up Shop : The Business of Open-Source Software." *IEEE Software* 16(1):45–51.

Herring, Susan C., Lois Ann Scheidt, Sabrina Bonus & Elijah Wright. 2004. Bridging the Gap: A Genre Analysis of Weblogs. In *Proceedings of the 37th Hawaii International Conference on System Sciences, 2004.* IEEE pp. 101–111.

Hissam, Scott A. & Charles B. Weinstock. 2001. Open Source Software: The Other Commercial Software. In *1st Workshop on Open Source Software at ICSE 2001.*

IBM. 2004. "IBM's 'Linux is everywhere' campaign website.". Last accessed April 15th 2004.
URL: http://www-306.ibm.com/e-business/doc/content/lp/prodigy.html

Initiative, The Open Source. 1997. "The Open Source Definition.". Last accessed July 28 2004.
URL: http://www.opensource.org/docs/definition.php

Johnson, Jeffrey N. & Paul F. Dubois. 2003. "Issue Tracking." *Computing in Science & Engineering* 5(6):71–77.

Johnson, Kim. 2001. A Descriptive Process Model for Open-Source Software Development. Master's thesis University of Calgary Calgary, Canada: .

KDE, Project. 2004*a*. "KDE Developer Journals.". Last accessed August 8th 2004.
URL: http://www.kdedevelopers.org/blog

KDE, Project. 2004*b*. "KDE Wikis.". Last accessed July 19th 2004.
URL: http://wiki.kde.org

Komi-Sirvio, Seija, Annukka Mantyniemi & Veikko Seppanen. 2002. "Toward a Practical Solution for Capturing Knowledge for Software Projects." *IEEE Software* 19(3):60–62.

Lanzara, Giovan F. & Michèle Morner. 2003. The Knowledge Ecology of Open-Source Software Projects. In *Proceedings of 19th EGOS Colloquium.* Preliminary Draft, last accessed August 10th 2004.
URL: http://opensource.mit.edu/papers/lanzaramorner.pdf

Lawton, George. 2001. "Knowledge Management : Ready for Prime Time?" *Computer* 34(2):12–15.

LDP, Project. 2004*a*. "LDP Author Guide.". Last accessed August 8th 2004.
URL: http://www.tldp.org/LDP/LDP-Author-Guide/html/index.html

LDP, Project. 2004*b*. "LDP FAQ.". Last accessed August 15th 2004.
URL: http://tldp.org/FAQ/LDP-FAQ/

Levinthal, Daniel A. & James G. March. 1993. "The myopia of learning." *Strategic Management Journal* pp. 95–113.

Lindahl, Charlie & Elise Blount. 2003. "Weblogs : Simplifying Web Publishing." *Computer* 36(11):114–116.

Lindvall, Mikael & Ioana Rus. 2003. Knowledge Management for Software Organizations. In *Managing Software Engineering Knowledge*, ed. Aybuke Aurum, Ross Jeffery, Claes Wohlin & Meliha Handzic. Springer-Verlag.

Linux, Kernel Development Team. 2004. "Linux Kernel Mailing List.". Last accessed August 7th 2004.
URL: http://marc.theaimsgroup.com/?l=linux-kernel

Lohmeyer, Juanita & Maik Hassel. 2004. "Open-Source Bug Tracking with Bugzilla." *Linux Journal* . Last accessed April 15th 2004.
URL: http://www.linuxjournal.com//article.php?sid=7216

Lueg, Christopher. 2001. "Information, knowledge, and networked minds." *Journal of Knowledge Management* 5(2):151–160.

Manheimer, Ken, Barry Warsaw & John Viega. 1998. Mailman – An Extensible Mailing List Manager Using Python. In *Proceedings of the 7th International Python Conference.* Last accessed August 5th 2004.
URL: http://www.foretec.com/python/workshops/1998-11/proceedings/papers/manheimer/manheimer.html

Merriam-Webster. 2004. "Merriam-Webster Online Dictionary.". Last accessed June 17th 2004.
URL: http://www.m-w.com

Miller, Frank J. 1999. "I = 0 (Information has no intrinsic meaning).". Last accessed June 17th 2004.
URL: http://www.fernstar.com.au/publications/papers/i=o.htm

Moon, Jae Yun & Lee Sproull. 2002. "Essence of distributed work: the case of the Linux kernel.". Last accessed June 20th 2004.
URL: http://www.firstmonday.dk/issues/issue5_11/moon/index.html

Mozilla, The Organization. 2004. "Bugs Howto." *The Mozilla Project Web Site* . Last accessed August 7th 2004.
URL: http://www.mozilla.org/bugs/

Nakisa, Ramin. 2003. "Wiki Wiki Wah Wah." *Linux User and Developer* 29:42–48. Last accessed July 24th 2004.
URL: http://194.73.118.134/lud29-Collaborative_Software-Wiki.pdf

Nardi, Bonnie A., Diane J. Schiano & Michelle Gumbrecht. 2004. Blogging as Social Activity, or, Would You Let 900 Million People Read Your Diary? In *Proceedings of Computer Supported Cooperative Work 2004.* Submitted to conference. Last accessed July 22nd 2004.
URL: http://home.comcast.net/~diane.schiano/CSCW04.Blog.pdf

Nonaka, Ikujiro. 1994. "A dynamic theory of organizational knowledge creation." *Organization Science* 5:14–37.

Nonaka, Ikujiro & Hirotaka Takeuchi. 1995. *The knowledge-creating company : how Japanese companies create the dynamics of innovation.* Oxford University Press.

Nonaka, Ikujiro, Katsuhiro Umemoto & Dai Senoo. 1996. "From Information Processing to Knowledge Creation: A Paradigm Shift in Business Management." *Technology in Society* 18(2):203–218.
DOI: http://dx.doi.org/doi:10.1016/0160-791X(96)00001-2

Norris, Jeffrey S. 2004. "Mission-Critical Development with Open Source Software : Lessons Learned." *IEEE Software* 21(1):42–50.

Novell. 2004. "Novell Forge : A collaboration Web Site for Development of Open-source Projects.". Last accessed August 12th 2004.
URL: http://forge.novell.com

Oda, Terri. 2003. "GNU Mailman - List Member Manual.". Last accessed August 5th 2004.
URL: http://mailman.sourceforge.net/mailman-member/mailman-member.html

Oikarinen, Jarkko. 1997. "History of IRC.". Last accessed August 9 2004.
URL: http://www.ircbeginner.com/ircinfo/history-jarkko.html

Oxford, University Press. 2004. "Oxford English Dictionary Online.". Last accessed July 19th 2004.
URL: http://dictionary.oed.com

Pfeffer, Jeffrey. & Robert I. Sutton. 2000. *The Knowing-doing Gap : How Smart Companies Turn Knowledge Into Action.* Harvard Business School Press. Digital version.

Polanyi, Michael. 1966. *The tacit dimension.* Reprint, 1983 ed. Peter Smith.

project, The Fedora. 2003. "What is The Fedora Project?" Internet. Last accessed August 16 2004.
URL: http://fedora.redhat.com/about/

Project, The Gnome. 2004. "Gnome Frequently Asked Questions.". Last accessed August 10th 2004.
URL: http://gnomesupport.org/wiki/index.php/GnomeFrequentlyAskedQuestions

Prusak, Laurence. 2001. "Where did knowledge management come from?" *IBM systems journal* 40(4). Internet version, last accessed May 7th 2004.
URL: http://researchweb.watson.ibm.com/journal/sj/404/prusak.pdf

Quigley, Edward J. & Anthony Debons. 2000. Interrogative theory of information and knowledge. In *Proceedings of the 1999 ACM SIGCPR conference on Computer personnel research.* ACM pp. 4–10.
DOI: http://dx.doi.org/doi:10.1145/299513.299602

Ramler, Rudolf, Klaus Wolfmaier & Edgar Weippl. 2004. "From Maintenance to Evolutionary Development of Web Applications : A Pragmatic Approach." *Lecture Notes in Computer Science* 3140/2004(1):287–299.
DOI: http://dx.doi.org/doi:10.1007/b99180

Raymond, Eric Steven. 1998*a*. "Goodbye, "free software"; hello, "open source".". Last accessed June 20 2004.
URL: http://www.catb.org/~esr/open-source.html

Raymond, Eric Steven. 1998*b*. "Homesteading the Noosphere.".
URL: http://www.catb.org/~esr/writings/homesteading/homesteading/

Raymond, Eric Steven. 2000. *The Cathedral and the Bazaar.* O'Reilly. Internet version, last accessed April 14th 2004.
URL: http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/index.html

Raymond, Eric Steven. 2001. "How To Ask Questions The Smart Way.". Last accessed August 14th 2004.
URL: http://www.catb.org/~esr/faqs/smart-questions.html

Raymond, Eric Steven. 2003. "The Jargon File.". Last accessed August 16 2004.
URL: http://catb.org/~esr/jargon/

Robbins, Jason. 2003. Adopting Open Source Software Engineering (OSSE) Practices by Adopting OSSE Tools. In *Making Sense of the Bazaar: Perspectives on Open Source and Free Software.* O'Reilly Associates.

Rudzki, Michal P. 2004. "E-race Knowledge Recorder.". Last accessed August 12th 2004.
URL: http://forge.novell.com/modules/xfmod/project/?eracekrecord

Rus, Ioana & Mikael Lindvall. 2002. "Knowledge management in software engineering." *IEEE Software* 19(3):26–38.

Schiano, Diane J., Bonnie A. Nardi, Michelle Gumbrecht & Luke Swartz. 2004. Blogging by the Rest of Us. In *Extended abstracts of the 2004 conference on Human factors and computing systems.* ACM pp. 1143–1146.
DOI: http://dx.doi.org/doi:10.1145/985921.986009

Sourceforge, Open source development network. 2004. "SourceForge website.". Last accessed April 15th 2004.
URL: http://sourceforge.net/

Spek, Rob van der & André Spijkervet. 1997. *Knowledge Management, dealing intelligently with knowledge.* CIBIT. Last accessed June 18th 2004.
URL: http://www.cibit.com/site-en.nsf/p/Publications-Knowledge_Management,_dealing_intelligently_with_knowledge

Stenmark, Dick. 2001. The Relationship between Information and Knowledge. In *Proceedings of 24th Information Systems Research Seminar in Scandinavia. IRIS 24.*

Stenmark, Dick. 2002. Information vs. knowledge : the role of intranets in knowledge management. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences, 2002. HICSS.* pp. 1326–1335.

Stewart, Thomas A. 1997. *Intellectual capital : the new wealth of organizations.* Doubleday.

Sveiby, Karl-Erik. 1994. "What is information?". Last accessed July 2nd 2004.
URL: http://www.sveiby.com/articles/Information.html

Sveiby, Karl-Erik. 1997. "Tacit knowledge.". Last accessed August 2nd 2004.
URL: http://www.sveiby.com/articles/Polanyi.html

Sveiby, Karl-Erik. 1998. "Intellectual Capital and Knowledge Management.". Last accessed June 18th 2004.
URL: http://www.sveiby.com/library/intellectualcapital.html

Swan, Jacky, Sue Newell, Harry Scarbrough & Donald Hislop. 1999. "Knowledge management and innovation: networks and networking." *Journal of Knowledge Management* 3:262–275.
DOI: http://dx.doi.org/doi:10.1108/13673279910304014

Teece, David J. 2002. *Managing Intellectual Capital : Organizational, Strategic, and Policy Dimensions (Clarendon Lectures in Management Studies).* Paperback ed. Oxford University Press.

Trott, Benjamin & Trott Mena. 2002. "TrackBack Technical Specification.". Last accessed July 22nd 2004.
URL: http://www.movabletype.org/docs/mttrackback.html

Tuomi, Ilkka. 1999. "Data Is More Than Knowledge : Implications of the Reversed Knowledge Hierarchy for Knowledge Management and Organizational Memory." *Journal of Management Information Systems* 16:103–117.

Vegas, Sira, Natalie Juristo & Victor R. Basili. 2003. A Process for Identifying Relevant Information for a Repository : A Case Study for Testing Techniques. In *Managing Software Engineering Knowledge*, ed. Aybuke Aurum, Ross Jeffery, Claes Wohlin & Meliha Handzic. Springer-Verlag.

Venzin, Markus, George von Krogh & Johan Roos. 1998. Future research into knowledge management. In *Knowing in firms : understanding, managing and measuring knowledge*, ed. George von Krogh, Johan Roos & Dirk Kleine. Sage.

Viega, John, Barry Warsaw & Ken Manheimer. 1998. Mailman : The GNU Mailing List Manager. In *Proceedings of the 12th Systems Administration Conference (LISA '98).* Advanced Computing Systems Association, USENIX. Last accessed August 5th 2004.
URL: http://www.usenix.org/publications/library/proceedings/lisa98/full_papers/viega/viega_html/viega.html

Villa, Luis. 2004. "Gnome Bug Writing Howto.". Last accessed August 8th 2004.
URL: http://bugzilla.gnome.org/bug-HOWTO.html

von Krogh, George, Kazuo Ichijo & Ikujiro Nonaka. 2000. *Enabling knowledge creation : how to unlock the mystery of tacit knowledge and release the power of innovation.* Oxford University Press.

Warsaw, Barry A. 2003. GNU Mailman, Internationalized. In *Proceedings of the USENIX 2003 Annual Technical Conference.* Advanced Computing Systems Association, USENIX pp. 39–50. Last accessed August 5th 2004.
URL: http://www.usenix.org/events/usenix03/tech/freenix03/full_papers/warsaw/warsaw_html/index.html

Webopedia. 2004. "Open source definition.". Last accessed April 15th 2004.
URL: http://www.pcwebopedia.com/TERM/O/open_source.html

Wiig, Karl M. 1993. *Knowledge Management Foundations : Thinking About Thinking - How People and Organizations Represent, Create and Use Knowledge.* Schema Press.

Wiig, Karl M. 1999. Knowledge Management : An Emerging Discipline Rooted in a Long History. In *Knowledge Management*. Scheduled for publication fall, 1999. ed. Internet version, last accessed June 15th 2004.
URL: http://www.krii.com/downloads/km_emerg_discipl.pdf

Wikipedia. 2004*a*. “Blogsphere definition.”. Last accessed July 19th 2004.
URL: http://en.wikipedia.org/wiki/Blogosphere

Wikipedia. 2004*b*. “English Wikipedia Statistics.”. Last accessed July 19th 2004.
URL: http://en.wikipedia.org/wiki/Special:Statistics

Wikipedia. 2004*c*. “Wiki definition.”. Last accessed July 19th 2004.
URL: http://en.wikipedia.org/wiki/Wiki

Wilson, T.D. 2002. “The nonsense of ’knowledge management’.”. Internet version, last accessed August 15 2004.
URL: http://InformationR.net/ir/8-1/paper144.html

Yamauchi, Yutaka, Makoto Yokozawa, Shinohara Shinohara & Toru Ishida. 2000. Collaboration with Lean Media : How Open-Source Software Succeeds. In *Proceedings of ACM CSCW’00 Conference on Computer-Supported Cooperative Work*. pp. 329–338.
DOI: http://dx.doi.org/doi:10.1145/358916.359004

Zahran, Sami. 1998. *Software Process Improvement - Practical Guidlines for Business Success*. London: Pearson Education Ltd.