Data & Analysis Center for Software
ITT Industries
Griffiss Business & Technology Park
775 Daedalian Drive Data & Analysis
Rome, NY 13441-4909

A Data & Analysis Center for Software Task
Contract No. SPO700-98-D-4000
Knowledge Management in Software Engineering
A State-of-the-Art-Report

29 November 2001

Prepared for:
Mr. Paul M. Engelhart
Air Force Research Laboratory
Information Directorate/IFED
32 Brooks Road
Rome, NY 13441-4505

Produced by Fraunhofer Center for Experimental Software Engineering
Maryland and The University of Maryland

By

Ioana Rus, Mikael Lindvall, and Sachin Suman Sinha

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
16 November 2001	N/A	N/A

4. TITLE AND SUBTITLE

A State of the Art Report: Knowledge Management in Software Engineering

5a. CONTRACT NUMBER
SPO700-98-4000

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
65802S

6. AUTHOR(S)
Ioana Rus, Mikael Lindvall, and Sachin Suman Sinha

5d. PROJECT NUMBER
R266

5e. TASK NUMBER
00

5f. WORK UNIT NUMBER
01

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Fraunhofer Center for Experimental
Software Engineering Maryland and
The University of Maryland, College
Park, Maryland

8. PERFORMING ORGANIZATION REPORT NUMBER

DACS SOAR #8

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Technical Information Center (DTIC)/AI
8725 John J. Kingman Rd., STE 0944, Ft. Belvoir, VA 22060
and Air Force Research Lab/IFED
32 Brooks Rd., Rome, NY 13440

10. SPONSOR/MONITOR'S ACRONYM(S)

DTIC-AI and AFRL/IFED

11. SPONSOR/MONITOR'S REPORT NUMBER(S) N/A

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
Available from: DoD Data & Analysis Center for Software (DACS)
PO Box 1400, Rome, NY 13442-1400

14. ABSTRACT

Is knowledge management one of those hyped concepts that rise quickly, ambitiously claims to cure organizational headaches and then fails and falls quietly? Or is it an instrument that will really help organizations address some of the problems they face while trying to achieve their business objectives? In particular, is knowledge management valuable to software development organizations? What kind of problems can KM help address and solve? What kind of solutions does KM propose to these problems? How can a KM system for a software organization be implemented? What are the challenges? What are the success factors?

This report addresses the above questions and attempts to provide answers that resulted from an extensive research of the state-of-the-art and state-of-the-practice of this subject.

15. SUBJECT TERMS
Knowledge management, Software tools

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER *(include area code)*
unclassified	unclassified	unclassified	UL	57	315-334-4900

Introduction

Knowledge Management is an emerging discipline that promises to capitalize on organizations' intellectual capital. The concept of knowledge is far from new and phrases containing the word *knowledge* such as "knowledge bases" and "knowledge engineering" have been around for a while. The artificial intelligence (AI) community has, for example, long dealt with representation, storage, and application of knowledge. The concept of *Knowledge Management* (KM) emerged in the mid-1980's from the need to derive knowledge from the "deluge of information" (Lawton, 2001). In the 1990's KM was translated into commercial computer technology, facilitated by new technologies such as Internet, group support systems, search engines, portals, and data and knowledge warehouses as well as the application of statistical analysis and AI techniques. According to Lawton "80 percent of the largest global corporations now have KM projects" (Lawton, 2001). "[O]ver 40 percent of the Fortune 1000 companies now have a Chief Knowledge Officer (CKO), a senior-level executive responsible for creating an infrastructure and cultural environment for knowledge sharing" (O'Leary, 1998).

Like any other buzzword, the term "Knowledge Management" is sometimes over-used and misused. A senior analyst at the marketing research firm *Ovum* says: "I think vendors did KM a great disservice by labeling every tool that came out as KM. People got disillusioned with it" (Lawton, 2001).

So, is knowledge management one of those hyped concepts that rise quickly, ambitiously claims to cure organizational headaches and then fails and falls quietly? Or is it an instrument that will really help organizations address some of the problems they face while trying to achieve their business objectives? In particular, is knowledge management valuable to software development organizations? What kind of problems can KM help address and solve? What kind of solutions does KM propose to these problems? How can a KM system for a software organization be implemented? What are the challenges? What are the success factors?

This report addresses the above questions and attempts to provide answers that resulted from an extensive research of the state-of-the-art and state-of-the-practice of this subject.

Some of the problems that software organizations encounter are similar to those faced by other organizations, so general purpose KM systems can be used to address them; other issues are very specific, so special systems are required.

The first argument in favor of managing knowledge in software engineering is that it is a human and knowledge intensive activity (Birk, et. al., 1999). Similar to other sectors, such as consulting, law, investment banking, and advertising, the main asset of an organization consists of its intellectual capital. Software development is a "design type process" where every person involved has to make a large number of decisions, each of them with several possible choices, as opposed to a "production", or "manufacturing"

process where, once a decision is made, many workers can carry out tasks without having to make further decisions. For example, a company must select what products to develop; a project manager must select the staff and must plan a project, which implies selecting a process and a set of methods and techniques to be used; a designer must select an efficient algorithm; a programmer has to decide on a function, or variables to use; and a tester must select a set of test cases. How do all these people make their decisions? On what are they based? Most of the time, decision makers rely on personal knowledge and experience, on their "gut feeling". But as software development projects grow larger and the discipline moves from craftsmanship to engineering, it becomes a group activity where individuals need to communicate and coordinate. Individual knowledge has to be shared and leveraged at a project and organization level, and this is exactly what KM proposes. Knowledge management demystifies the individual hero and shifts the focus to collective creativity, exploiting the emerging behavioral idea – "none of us is as smart as all of us" (Bennis and Biederman, 1998). This complements software industry initiatives like the Capability Maturity Model (Paulk et al, 1995), which tries to establish stable software processes that are independent of individual software engineers. Knowledge has to be collected, organized, stored, and easily retrieved when it needs to be applied.

In software development one can identify two types of knowledge:

1. knowledge embedded in the products (artifacts), since they are the result of highly intellectual, creative activities
2. meta-knowledge, that is knowledge about the products and processes (as explained in Section 3)

Some of the sources of knowledge (e.g., the artifacts) are stored by default in electronic form, by the very nature of software development, so the use of KM is facilitated in this industry. However, this source can be viewed as raw data and information and, as discussed in Sections 2 and 3, KM seeks to turn data into information, and information into knowledge. The most eminent problem is, however, that just a fraction of all knowledge related to software is captured and made explicit. The majority of knowledge is tacit, residing in the brains of the employees. This fact makes knowledge sharing and retaining of knowledge a challenge. Different sources and types of knowledge and the corresponding approaches to handling each of them are presented in Section 3.

The purpose of this report is to describe the state-of-the-art of Knowledge Management in Software Engineering. The report starts by presenting a set of well-known problems faced in software development and shows how KM can help solve them. Section 2 sets the context for the remainder of the report, by presenting definitions of knowledge and knowledge management, as well as models that will be used later to characterize KM tools, approaches, and systems for software engineering. Section 3 discusses KM in software engineering. The implementation of a KM system, the methodology, challenges, cultural issues and success factors are presented in Section 4. A summary section concludes the report.

The intended audience of this report consists of:

- Software development organizations' Chief Knowledge Officers (CKOs) or future CKOs (in case the organization does not have such a position yet) to provide them with information regarding KM's capacity to support organizational objectives and identify the tools and systems available to support it
- Personnel in charge of implementing a knowledge management system (or at least considering one) to inform them about the tools and methodologies available, the issues involved when implementing such a system, and implementation success factors
- Project managers and developers who are in the trenches and need to get the job done and to deliver the software in time, to show them the advantages of having a KM system in place and using it. We also show that taking the time to document and share personal knowledge pays-off (sometimes in the long run but also on a short term basis).
- Vendors who need to know what segments of the KM tools market are covered and need to identify the demand for new tools. The vendors can also learn what concepts and proof of concepts have been developed and their potential for being transformed into commercial systems.
- Researchers who are looking for new research topics
- Anyone who just wants to find out what KM is and how it can be applied to support software development

Section 1.
Motivation for Knowledge Management in Software Engineering

Software engineering is a complex business that involves many people working in different phases and activities. Constant technology changes make the work dynamic: New problems are solved and new knowledge is created every day. The knowledge in software engineering is diverse and its proportions immense and growing. Organizations have problems keeping track of what this knowledge is, where it is, and who has it. A structured way of managing the knowledge and treating the knowledge and its owners as valuable assets could help organizations leverage the knowledge they possess. This section discusses software organizations' needs related to knowledge management. We also discuss some of the challenges that software organizations might face when they try to implement knowledge management, as well as some opportunities that might make implementation easier.

The Need for Capturing and Sharing Process and Product Knowledge

Each software product and process is different in terms of goals and contexts. A single software development approach cannot be assumed for all projects or products. To develop software for the space shuttle is not the same as to develop software for a dishwasher (Lindvall and Rus, 2000). Software developers are often exposed to this diversity, which makes the software discipline inherently experimental (Basili and Rombach, 1991) and we constantly gain experience with each development project. "Knowledge emerges in work practices, often being defined by the first project to address the issues involved" (Henninger, 1997).

Ideally, we would apply that experience to future projects in order to avoid mistakes and leverage successes. This does not always happen because often these work practices are not captured (Henninger, 1997). Development teams work on similar kinds of projects without realizing that results would have been achieved more easily if they followed a practice adopted by a previous project. (Basili, et. al., 2001; Brössler, 1999). The bottom line is that development teams do not benefit from existing experience. Instead they repeat mistakes over and over again (Brössler, 1999). This was manifested by the fact that "a large number of cases showed a lack of knowledge in the specific project, while this knowledge was actually available in the company" (Brössler, 1999). These problems are also tied to the problem of transferring knowledge to novices in the organization.

Knowledge Management addresses the issues of capturing and sharing knowledge, while the problems of project diversity and product singularity make it clear that such a system must be flexible enough to encompass variations on the same theme. Most artifacts guiding a software project and developed during a software project can be represented as documents. Therefore, these are the main explicit assets of the software organization.

These assets directly support the core business and must be managed so that they do not get lost. The problem of transferring knowledge from experts to novices is facilitated if the knowledge is readily captured, stored, and organized, possibly as documents. Therefore, Document Management is the main corner stone of our knowledge management model.

The Need for Domain Knowledge

Software development not only requires knowledge about its own domain, but also about the domain for which software is being developed. Sometimes a new domain requires learning a specific technique or a new programming language or application of a new kind of project management technique. Therefore, acquiring the experience and skills needed in projects takes a long time (Brössler, 1999). Even when the organization has been working on a particular domain extensively, the deep application-specific knowledge required to successfully build complex software is thinly spread over many software development engineers (Curtis et al, 1988). Although individual development engineers understand different components of the application domain, building large and complex software like the Space Shuttle software for NASA requires integration of different pieces of domain knowledge. Most of the system engineers working in these complex domain software development units say, "writing code is not the problem, understanding the problem is the problem" (Curtis et al, 1988).

There is no shortcut to learning. Domain knowledge that no one in the organization possesses must be acquired either by training or by hiring knowledgeable employees. Knowledge Management can, however, help organize the acquisition of new knowledge and it can help identify expertise as well as capture, package and share knowledge that already exists in the organization.

The Need for Acquiring Knowledge About New Technologies

Software development is becoming a more complex domain to master due to the constant change and stream of new technologies. The result is that it "is very difficult to keep the organization ahead in the competition" (Tiwana, 2000). Many industries have similar problems, but the software industry is probably worse than other industries due to the fact that the pace of change is faster.

The emergence of new technologies makes software more powerful, but at the same time "[new technologies] is every project manager's worst nightmare" (Brössler, 1999). Every emerging technology cannot be mastered overnight and it is extremely hard to accurately estimate the cost of a project when the technologies it will be using are new and unproven, and may even change during the project.

Lack of time causes a lack of experience, constantly pushing the boundaries of an organization's development set of skills and competencies (Henninger, 1997). When

developers or project managers use a technology that is new to all team members of a given project, the engineers all too often resort to the "learning by doing" approach. (Brössler, 1999). This often results in serious delays of projects.

Knowledge Management fosters a knowledge sharing culture within the company that helps facilitate sharing of knowledge related to new technologies. Knowledge Management also makes the point that time should be spent on actively searching for knowledge both within the organization and outside. Knowledge sharing occurs within communities of practice and interests, which can help speed up the learning curve.

The Need for Sharing Knowledge About Local Policies

To be a successful developer, one needs to have very specific knowledge concerning the existing software base and local programming conventions. This type of software knowledge typically exists as folklore and is informally maintained within the brains of the experienced developers. This knowledge is disseminated to inexperienced developers through informal meetings with the effect that not everyone gets the knowledge they need (Terveen et al, 1993). The practices of passing knowledge during the coffee break and at the water cooler are important aspects of a knowledge sharing culture which must be encouraged. At the same time, it must be made sure that this knowledge is also passed into the knowledge base of the organization and made available to everyone that needs it.

Knowledge Management can help set up a system that encourages both informal knowledge sharing sessions and more formal ways of communicating. Lightweight knowledge management approaches attempt to capture the informal knowledge that is shared on a daily basis so that it can be disseminated on a larger scale.

The Need for Knowing Who Knows What

Much knowledge can be recorded, but, nevertheless, the assets of a software engineering organization are mainly its employees and their tacit knowledge. "[I]t is more important for Software Engineering organizations to exploit and manage their intangible assets in contrast to their physical assets" (Tiwana, 2000). Management of intangible assets includes knowing who knows what and is part of competence management. Knowing who knows what can help reduce the time it takes employees to find experts. One study found, for example, that "software developers need to apply just as much effort and attention to determine who to contact in the organization in order to get the job done" (Perry et al, 1994). Getting expert help could reduce the long time it takes to find information. Another study found that people in software organizations spent 40% of their time in searching for and accessing different types of information related to their projects (Henninger, 1997). In the absence of any knowledge about other employees' expertise, people are even found spending as many as 3-4 days locating experts[1]. This

[1] Personal communication with professional software engineers.

gives rise to the belief that, apart from having knowledge, it is also important to know what other people know.

Another reason to keep track of who knows what in the organization is the fact that each employee wants to leverage the benefits of his knowledge and expertise. Employees, especially software engineers, are ready to switch companies even for a paltry increase in salary. In the absence of a knowledge base, the problem of a 'brain drain' can take horrible proportions. "Software organizations are heavily dependent on tacit knowledge, which is very mobile" (Tiwana, 2000). If a person with critical knowledge about processes and practices suddenly leaves the organization, severe knowledge gaps are created (Brössler, 1999). Often this person is the *only* expert in his specific area, accelerating the dilemma. The problem is that probably no one else in the organization knows what knowledge he possesses (Basili et al, 2001). In the current scenario of economic slowdown, when organizations are laying off personnel by the thousands, knowing what your employees know has become much more important, creating a necessity to retain the most important people.

Knowledge Management can never tap the brains of the employees, but it can help build structures and frameworks for capturing key information that can help retain some knowledge when employees leave. This key information would at least help in understanding what the employee who left knew and what profile his successor needs to have to fill the position. Knowledge Management can help establish routines for identifying knowledge, as well as the people who own the knowledge --- the experts. Competence Management, which also aims at identifying gaps in the knowledge structures, is, besides document management, the second cornerstone in our knowledge management model.

The Need for Distance Collaboration

Any larger software development is a group activity. The division of work into phases often means that different groups are involved at the same or different time. Due to globalization, these groups are often spread out geographically and it is common that group members live and work in different time zones. Outsourcing of subsystems to subcontractors also results in geographically co-located teams that need to work together. These groups need to communicate, collaborate, and coordinate independently of time and place.

Knowledge Management can help solve this problem as it acknowledges the need to capture, organize and store knowledge, as well as the necessity of knowledge transfer. Communication in software engineering is often related to the transfer of knowledge. Collaboration is related to mutual sharing of knowledge. Coordination that is independent of time and space is facilitated if the work artifacts and their status are stored and made part of an organizational memory.

Challenges for Knowledge Management in Software Engineering

Implementing knowledge management in any organization is a challenge because of the time and effort that is required before it starts to return on the investment. Software organizations seem to have even less time than others because of the fast pace of the business. The lack of time is a direct threat against knowledge management. An example is that "the specialist was so involved in his or her own project that there was no time to support the other project" (Brössler, 1999). People often have no time to even search for knowledge. This cultural behavior has the effect that a long-term investment such as knowledge management and learning for the next project are not prioritized. Instead, project managers are interested in finishing the current project on time. As long as management does not allow the culture to change and does not allow employees to invest in managing their knowledge, knowledge management is likely not to happen.

Another challenge is the elusiveness of software. Unlike products of other domains, software is not visible (compare with buildings in the civil engineering domain) (Devanbu, et. al., 1990). Invisibility leads to less reuse of the system. A developer, while implementing or modifying a system, cannot find out if the work has already been done. Many times, developers reinvent a system instead of reusing it, and this results in lower productivity (Devanbu, et. al., 1990). Another result is that software developers are not accustomed to reuse, which is a problem because the idea behind knowledge management is reuse of assets.

The most problematic challenge to knowledge management is that most of the knowledge in software engineering is tacit and will never become explicit. It will remain tacit because there is no time to make it explicit. There are very few approaches and tools for turning tacit knowledge into explicit knowledge, and most of the tacit knowledge is tacit in the most extreme way. Therefore, it is difficult to express and make explicit. A way to address this problem can be to develop a knowledge sharing culture, as well as technology support for knowledge management, never forgetting that the main asset of the organization is its employees.

Opportunities for Knowledge Management in Software Engineering

There are several reasons to believe that knowledge management for software engineering would be easier to implement than in other organizations. To start with, it is clear that a knowledge management system needs to be supported by appropriate IT technology (Brössler, 1999). While IT technology can be intimidating to many people, this is not the case for software engineers (Schneider, 2001). Instead, they are already accustomed to, and probably willing to try, new software tools, as long as they believe the tools will help them do a better job.

The other obvious benefit with software engineering activities is the fact that all artifacts are already in electronic form (Schneider, 2001) and, thus, can easily be distributed and shared. This is not the case in many other design-oriented industries in which the product

takes some physical form. An example is interior auto design, which has so many similarities with software design that it can even use the same approaches for managing experience (Lindvall, et. al., 2001), but which also is different in that the end product is a physical artifact that cannot easily be sent across the world in a second.

The most encouraging fact is probably that knowledge sharing between software engineers already does occur to a large degree. A great example of peer-to-peer knowledge sharing is Google[2], formerly Deja's Usenet Archive, where software engineers and others share knowledge without any form of compensation. This is encouraging because it shows that software engineers are willing to share their knowledge. Another highly appreciated knowledge sharing forum is Sun's support for Java programmers[3], which not only provides discussion forums and knowledge bases on Java technologies, but also offers technology chats with Java experts. These chats are even captured so that the knowledge they encompass does not get lost.

[2] www.google.com
[3] http://developer.java.sun.com/developer

Section 2.
Knowledge Management Background

This section discusses concepts related to knowledge and knowledge management. It describes models for knowledge evolution, as well as frameworks and initiatives for managing knowledge in software organizations.

Knowledge

According to a dictionary definition, knowledge is "the fact or condition of knowing something with familiarity gained through experience or association; acquaintance with or understanding of a science, art, or technique; the fact or condition of being aware of something" (Merriam-Webster, 2001).

With an orientation to knowledge management in software development organizations, Davenport and Prusak describe knowledge as "a fluid mix of framed experience, values, contextual information, and expert insights and grounded intuitions that provides a framework for evaluating and incorporating new experiences and information. It originates and is applied in the minds of the knower. In software organizations, it often becomes embedded not only in documents or repositories, but also in organizational routines, processes, practices, and norms" (Davenport and Prusak, 1998).

There are different levels of refinement to the items related to knowledge, the lowest one being data, followed by information, and knowledge at the highest level. ***Data*** consists of discrete, objective facts about events. It says nothing about its own importance or relevance. Data is essential raw material for the creation of information; it can be quantitative or qualitative. ***Information*** is data that is organized in a way that makes it useful for an end-user when making decisions. ***Knowledge*** is broader than information and data and requires understanding of information. Knowledge is not only contained in the information, but also in the relationships of information, its classification, and its meta-data (i.e., information about information, e.g., who has created the information) (Kappe, 1999). ***Experience*** is applied knowledge. For the remainder of this report we will use the terms experience and knowledge interchangeably. Knowledge is valuable, yet difficult to manage. In particular, technology alone cannot manage knowledge directly. Knowledge cannot be stored, but we can store information about knowledge. The human factor is required for processing knowledge and transforming information into knowledge. New knowledge can be created by experiences, observations, and drawing rational conclusions.

In our research we found that *knowledge* can have various meanings to different people. We have tried to incorporate these different meanings, keeping in mind that we are talking about knowledge in a specific context, namely in software development.

Knowledge Characteristics

Knowledge has many features, attributes, and dimensions. We discuss next the attributes and criteria that will be used further in the report to characterize and classify knowledge items, systems, approaches, and tools.

Knowledge can be ***documented*** or ***undocumented*** (i.e., personal, group, or organizational knowledge that has not been captured). ***Explicit*** knowledge, also known as codified knowledge, corresponds to the information and skills that are easily communicated and documented, such as processes, templates, and data that are captured in media. Explicit knowledge, thus, is easier to reuse across an organization. ***Tacit*** knowledge is highly personal knowledge that is gained through experience and largely influenced by beliefs, perspectives, and values embedded in the individual experiences of workers (Agresti, 2000).

Another attribute of knowledge is ***awareness*** about the existing knowledge, and about the lack of knowledge that is needed (Agresti, 2000).

The ***level of refinement*** of knowledge is related to the definitions of data, information, and knowledge. What we capture in software development (besides the work products that are stored in electronic format) is:

- Point data (quantitative or qualitative) related to a single project or event. In this category are metrics collected for a project, lessons learned from a specific event, etc.
- From a set of these data, collected for multiple projects, we can build models that contain more information and are applicable to new projects (for example the prediction models discussed in Section 3)
- At the next level of abstraction and generalization, we can derive knowledge represented as “best practices” and standards.

Depending on the set of activities in software engineering to which knowledge pertains, there can be different ***types of knowledge***, such as:

- Organizational knowledge, e.g., how to run the company, what are the business objectives, human resources aspects, etc. This type of knowledge for a software organization might not differ too much from the organizational knowledge for other industries
- Managerial knowledge is related to planning, staffing, tracking, and leading a project
- Technical (engineering/development) knowledge refers to development knowledge and skills, such as requirements analysis, designing, programming, testing, technical writing, using specific tools and methods (such as object oriented development) or specific programming languages

- Domain knowledge, related to the application domain and the specific system to which the software pertains (e.g., avionics control, NASA flight dynamics, office support tools, bank transactions)

The ***scope*** of knowledge refers to where the knowledge is applicable, to whom it is accessible, and whose activity it supports. Along this dimension, knowledge can be at the level of an individual, a group (project/team), an organization, multiple organizations, and industry wide. We will discuss next knowledge-related activities at each of these levels, from individual to industry.

Individual Learning

Ultimately, the individual is the one who performs tasks for achieving the goals set at the organizational level. Therefore, the knowledge and learning cycle at the individual level is of utmost importance. "Organizations learn only through individuals who learn. Individual learning does not guarantee organizational learning. But without it no organizational learning occurs" (Senge, 1990). In addition, groups of people who need to work together to solve a problem or perform a task need to have a memory and a learning mechanism of their own. Knowledge sharing between different components of the learning system is a critical component of the learning process.

Unlike other goods, knowledge is enriched when shared and is not diminished through use. In order for knowledge to be transferred between individuals, it must first be transformed into information (externalized), and then converted from information back to knowledge (internalized), as shown in Figure 1.

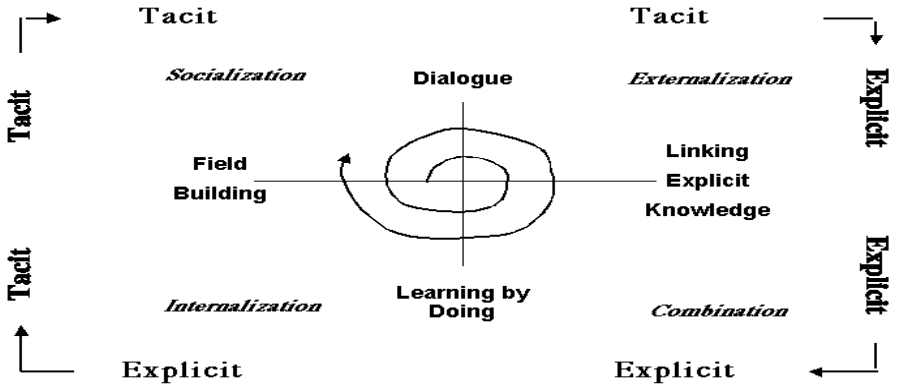

Figure 1: The Knowledge Spiral (Nonaka and Takeuchi, 1995)

Nonaka and Takeuchi define these transformations in (Nonaka and Takeuchi, 1995).

Socialization is the process of interaction between individuals that results in knowledge sharing. This process helps share employees' experiences, mental models and their

beliefs and perspectives so that the knowledge residing in peoples' brains reaches the community.

Externalization is the process of capturing information about knowledge. This can be as simple as speaking to somebody, writing a document, drawing a figure, giving a presentation, or teaching.

Combination is the phase that combines and connects existing knowledge with newly created knowledge. Combining previous knowledge that has been received from other people with one's own insights or experiences creates new knowledge.

Internalization is the process of understanding the information, putting it into context with one's own existing knowledge, and, therefore, transforming the information into knowledge. The "knowledge spiral" in Figure 1 shows how the knowledge is transformed from *tacit* to *explicit*, and then again to *tacit*, during various phases of knowledge sharing.

Knowledge sharing between individuals can be ad hoc or organized (systematic). Transfer of knowledge from one person to another can happen on an ad hoc basis within a project or an organization. This occurs when individuals initiate communication, for example, when they need to solve a problem and ask for help from other individuals who are known to have the appropriate expertise. If this communication and sharing is systematic and there is a process in place to document it, then exchanged knowledge will be captured and organized into a group (or an organizational) memory. Thus, the next time this piece of knowledge is needed, it will be retrieved from the groups' knowledge repository rather than solicited from an individual. This will lead to time-savings, both for the solicitor of the information and for the provider.

Knowledge in Software Organizations

When individuals team up to solve a problem (or to develop a product), they form a ***community of practice***. When individuals communicate and exchange information related to a common topic, but for solving different problems within or outside a company, they form ***communities of interest***, such as groups of Java programmers, (e.g., Sun's community for Java[4]), the Software Process Improvement Network (SPIN)[5], or special interest groups of the IEEE or ACM[6]. These communities heavily utilize web technology for knowledge sharing.

In software development, learning occurs during projects. For organizational learning, knowledge from all projects must be documented, collected and organized into a repository that will support decision making for future projects. A concept that supports this idea is the Quality Improvement Paradigm (QIP) (McGarry et al, 1994). As shown

[4] http://developer.java.sun.com/developer/community/
[5] http://www.sei.cmu.edu/collaborating/spins/
[6] http://www.acm.org/sigs/guide98.html

in Figure 2, learning occurs at each project level by analyzing and drawing conclusions about the project's results, both during execution and post mortem. These results are then taken up one level, analyzed again, packaged in an organizational context, and stored in an ***experience database***. This experience repository will support planning for future projects (i.e., choosing the processes, methods, techniques, and tools that proved to be useful to the organization).

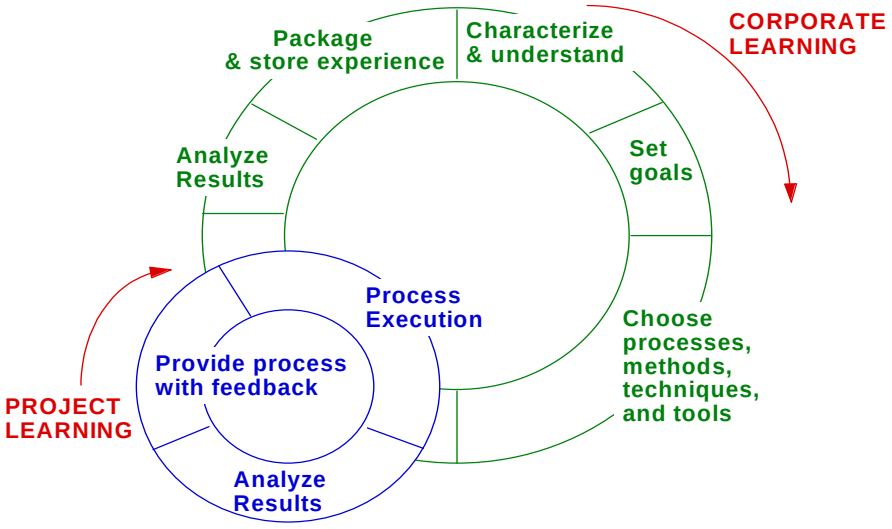

Figure 2: The Quality Improvement Paradigm (McGarry et al, 1994)

A framework for QIP implementation is the *Experience Factory* (Basili, et. al., 1994a). The approach has been successfully applied to software development at NASA for more than 25 years and recently at other organizations. The Experience Factory enables organizational learning and acknowledges the need for a separate support organization that supports the project organization in order to manage and learn from its own experience. The support organization helps the project organization observe and collect data about itself, builds models and draws conclusions based on that data, packages the experience for further reuse, and most importantly, feeds the experience back to the project organization.

The Experience Factory approach was initially designed for software organizations and takes into account the software discipline's experimental, evolutionary, and non-repetitive characteristics. Recent work shows that tailored versions of the Experience Factory approach are also beneficial for creating learning organizations whose main business is not software. This is especially true for organizations whose processes are design-oriented rather than production-oriented (e.g., manufacturing).

The Experience Factory approach has components that address capturing, storing, distributing, applying, and creating new experience. It also has components that address analysis and synthesis of knowledge (Basili, et. al., 2001). A physical implementation of the Experience Factory in an organization is called the *Experience Management System*

(EMS). The EMS is composed of content, structure, procedures and tools. The content can be data, information, knowledge or experience, which for simplicity will be called *experience* from here on. The structure is the way the content is organized. The content and the structure are often referred to as the *experience base*. Procedures are instructions on how to manage the experience base on a daily basis, including how to use, package, delete, integrate and update experience. Tools support managing the content and the structure, and carrying out the procedures, as well as helping capture, store, integrate, analyze, synthesize and retrieve experience (Basili, et. al., 2001).

Inter-company Learning

Organizations learn not only from their own experiences, but also from external sources (e.g., software vendors and other software development companies). Several software vendors provide web-based knowledge bases. Examples include Adobe's Knowledge Base[7], Microsoft's Knowledge Base[8], Oracle's Support Center[9], and Perl's Frequently Asked Questions[10]. For facilitating inter-company learning, organizations whose main business is software development or who have a significant software component embedded in their products (such as avionics, telecommunication, and automobile industries) have formed software engineering consortia, a systematic and well defined form of communication and knowledge sharing. Examples of such communities are:

- The Maryland Software Industry Consortium (SWIC), a group of Maryland information technology and software companies seeking excellence through continuous product, service and organizational improvement. The Consortium enables individual companies "to sharpen their competitive edge through focused strategic and tactical programs tailored for specific business needs." [11]
- The Software Experience Consortium[12] (SEC), a joint endeavor among a set of member companies, where applied research organizations play the role of supporter and facilitator in the collection, analysis, and dissemination of experience assets among the members of the consortium
- The Software Program Managers Network (SPMN), whose purpose is to "seek out proven industry and government software best practices and convey them to managers of large-scale DoD software-intensive acquisition programs."[13]
- The World Wide Web Consortium (W3C), a focused organizations community with 512 members. Its goal is to develop "interoperable technologies (specifications, guidelines, software, and tools) to lead the Web to its full potential as a forum for information, commerce, communication, and collective understanding."[14]

---

[7] http://www.adobe.com/support/database.html
[8] http://search.support.microsoft.com/kb/
[9] http://www.oracle.com/support/index.html?content.html
[10] http://www.perl.com/pub/q/faqs
[11] http://www.mdswic.org/
[12] http://fc-md.umd.edu/
[13] http://www.spmn.com/
[14] http://www.w3.org/

Industry-wide Knowledge Leverage

At the software industry level, there are committees or groups of experts that identify patterns (e.g., software design patterns (Gamma et al, 1995)) and generate handbooks and standards generally applicable to software development (e.g., IEEE and ISO standards) in order to leverage the experience and knowledge of all software development organizations.

One such initiative is the *Software Engineering Body of Knowledge*[15] (SWEBOK). SWEBOK was initiated in 1998 by the Software Engineering Coordinating Committee (SWECC), a joint effort between IEEE Computer Society and ACM. The *body of knowledge* represents what a practicing software engineer needs to master on a daily basis. SWEBOK is intended to be a guide to the knowledge already existing in the form of the accumulated software engineering literature. SWEBOK aims at documenting fundamental knowledge that is relatively stable over time instead of knowledge that is subject to rapid technological changes. It can be used to describe job positions in terms of the knowledge that candidates need to possess, analyze the knowledge gap between needed and current knowledge and guide the acquisition of knowledge, whether acquisition is realized by hiring new software engineers or by training existing employees. It can also direct the education of students that are planning to enter the field of software engineering and serve as the basis for certifying people that need to prove they have the knowledge required to do certain kinds of jobs. SWEBOK can be viewed as the content of a potential knowledge base that can be used by software engineers to solve different problems. Currently it is stored as an electronic document, but it could be reformatted in the form of knowledge packages. Such a form would make it possible for software engineers to search and retrieve knowledge and to provide feedback to the initial authors on the application of knowledge. Such feedback is essential to the evolvement of any knowledge and experience base. SWEBOK covers the following ten knowledge areas:

1. software requirements
2. software design
3. software construction
4. software testing
5. software maintenance
6. software configuration management
7. software engineering management
8. software engineering process
9. software engineering tools and methods
10. software quality

There are also initiatives and projects whose goal is to build knowledge bases for empirical software engineering. Such projects are the Center for Empirically-Based

[15] http://www.swebok.org/

Software Engineering (CeBASE[16]) (funded by the National Science Foundation (NSF)) and ViSEK[17] that "accumulate empirical models in order to provide validated guidelines for selecting techniques and models, recommend areas for research, and support software engineering education"[18].

Organizational Knowledge Cycle

So far we have talked about knowledge at different levels (from individual, to project, to organization, to industry). What we focus on in this report is *organizational* knowledge.

The next sub-section discusses the phases that organizational knowledge goes through in any industry, as well as the activities performed upon knowledge, as it transitions from one phase to another. We will call this the ***knowledge evolution cycle***. According to Wiig, this is a five-phase cycle. "The cycle begins with the emergence of knowledge in the organization; information about it is captured in explicit forms; the explicit knowledge is structured and classified; and the tacit and explicit knowledge are accessed and applied" (Wiig, 1999). Figure 3 shows this cycle.

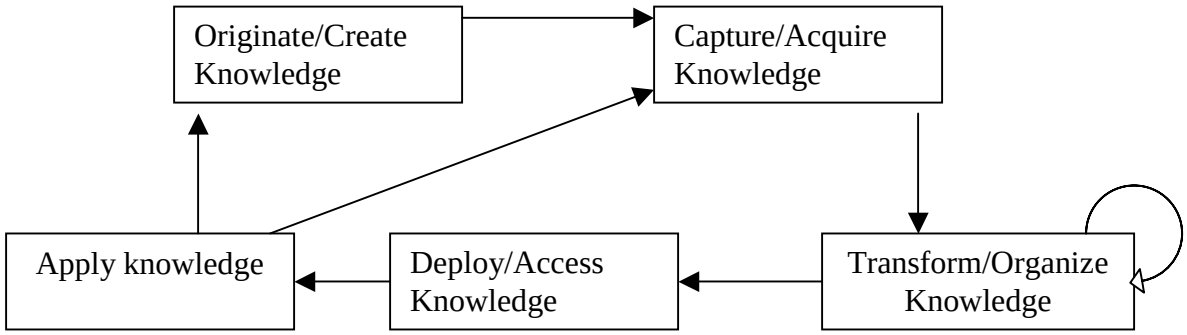

Figure 3: Organizational Knowledge Evolution Cycle [Wiig, 1999]

The boxes represent activities performed upon knowledge and are defined based on (Wiig, 1999) and (Agresti, 2000) as follows:

- ***Originate/Create Knowledge*** - knowledge is developed through learning, innovation, creativity, and importation from outside sources
- ***Capture/Acquire Knowledge*** - information about knowledge, as well as knowledge itself, is acquired and captured in explicit forms
- ***Transform/Organize Knowledge*** - knowledge is organized, transformed, or included in written material and knowledge bases. This also includes any action that renders tacit or explicit knowledge residing in different locations within the organization more accessible and more easily internalized.

[16] http://www.cebase.org/
[17] http://www.iese.fhg.de/Projects/ViSEK/
[18] http://www.cebase.org/

- ***Deploy/Access Knowledge*** - knowledge is distributed to points-of-action through education, training programs, and automated knowledge-based systems, or expert networks
- ***Apply Knowledge*** – Applying the knowledge is the ultimate goal and the most important part of the whole life cycle. Activities are aimed at making the knowledge available wherever and whenever it is needed. By using (applying) knowledge, it becomes the basis for further knowledge creation, innovation and learning.

The activities above need to be carried out in a systematic manner, which leads to the necessity of having a *knowledge management process* in place.

Knowledge Management

Knowledge management has been defined in many different ways. We have extracted the commonalities occurring in several definitions that are appropriate to the context of this report (Broadbent, 1998), (Peters, 1997)[19,20].

Knowledge management is seen as a strategy (or practice, systematic process, set of policies, procedures and technologies) that creates, acquires, transfers, brings to the surface, consolidates, distills, promotes creation, sharing, and enhances the use of knowledge (or information, intellectual assets, intellectual capital) in order to improve organizational performance; support organizational adaptation, survival and competence; gain competitive advantage and customer commitment; improve employees' comprehension; protect intellectual assets; enhance decisions, services, and products; and reflect new knowledge and insights.

KM involves the identification and analysis of available and required knowledge assets and knowledge asset-related processes, and the subsequent planning and control of actions to develop both the assets and the processes in order to fulfill organizational objectives. Knowledge management is concerned with realizing the value of this intellectual capital, which consists of *tangible* and *intangible* assets (Kappe, 1999). Tangible assets (corresponding to explicit knowledge) may vary for different industries and applications, but generally they include manuals; directories; customer information; correspondence with (and information about) clients, vendors, and sub-contractors; news; competitor intelligence; employees; competence; patent licenses; and knowledge derived from work processes (working papers, proposals, artifacts). Intangible assets (corresponding to tacit knowledge) consist of skills, experience and knowledge of people within the organization. Knowledge management should not only deal with explicit knowledge, which is generally easier to handle, but also with tacit knowledge.

[19] http://www.icasit.org/km/glossary.htm
[20] http://www.bus.utexas.edu/kman/glossary.htmest

By analyzing KM vendors, researchers and consultants, as well as KM users, Sveiby (Sveiby, 1996) identifies two tracks of activities and, consequently, two components of knowledge management systems:

1. The *IT-track*, which focuses on information management. People in this group are involved in constructing information management systems, artificial intelligence (AI), reengineering, group ware, etc. For them, knowledge consists of objects that can be handled in information systems
2. The *people-track*, which focuses on the management of people. People in this group are primarily involved in assessing, changing and improving human skills and/or behavior. They perceive knowledge as a process, a complex set of dynamic skills and know-how.

These two tracks have to be integrated to fulfill the objectives of knowledge management in any organization.

Fundamental Questions Related to Storing and Sharing Knowledge Items

Any approach that intends to capture, store, and disseminate knowledge items must address some fundamental questions:

1. ***Assuming that individuals have personal knowledge repositories, how can these repositories be shared?*** One approach would be to change the corporate culture and allow co-workers to search each other's personal repositories. While this might be a viable solution in some cases, especially where one employee is a back up for another employee, it is not a generally viable solution due to other problems. One such problem is that personal repositories often contain items that are not of general interest. Another issue is that some kind of information filtering would be needed. In some cases, filtering might not be sufficient and some kind of packaging would be necessary so that the content becomes useful for a third party.

2. ***How can the common repository be populated?*** One answer is to enable employees to contribute from their personal repositories. This is a relatively simple problem from a technology point of view, because employees are asked to share something they already have. However, it might raise a psychological issue if people do not want to give away their personal knowledge. Moreover, if employees are asked first to capture their own knowledge and then to share it, the request is likely to encounter resistance. An example is asking employees to share their lessons learned, which would not be hard to do had they already collected these lessons.

3. ***How must the common (organizational) repository be organized so that it provides easy search and retrieval of knowledge items?*** There are many different approaches for organizing knowledge items. Some are based on AI

techniques (such as case-based reasoning), some are based on advanced probabilistic indexing techniques, and others are based on a combination of manual and automatic generation of taxonomies.

4. ***How can people be persuaded to use the repository?*** In many cases this is more a cultural problem than a technological one. Software reuse, for example, is a cultural problem because software engineers do not trust code developed by others. Another issue is that in many organizational settings it is easier to find someone to ask for a solution instead of searching in repositories for answers. A third problem is having empty repositories. Inherently, any repository is nearly empty when first introduced, which gives potential users the perception that the repository and the whole initiative of sharing are worthless. This, in turn, prevents them from adding to the repository, because nobody wants to add items to something that is perceived as worthless.

5. ***How are contributions from users collected so that the repository evolves?*** Feedback from users is essential for improving the repository and the process.

6. ***How are usage and content of the repository analyzed and how can new knowledge be synthesized, based on this analysis?*** The initial seed of a repository is likely to be relatively raw material. Raw material would be lessons learned, incident reports, defect reports, project post-mortems, frequently asked questions with answers, results from projects, etc. While this raw material is useful by itself, it is also desirable to analyze and synthesize it into more refined knowledge items.

In the remainder of the report we will present some answers to these questions. Section 3 discusses mostly the technology aspects of KM systems for software development, whereas Section 4 focuses on the cultural factors that determine the successful implementation of such systems.

Section 3.
Knowledge Management in Software Engineering

As we showed in Section 1, software engineering is a knowledge intensive business and as such it could benefit from the ideas of knowledge management. The important question is, however, where does knowledge reside in software engineering?

It is clear that software engineering involves a multitude of knowledge-intensive tasks: analyzing user requirements for new software systems, identifying and applying best software development practices, collecting experience about project planning and risk management, and many others (Birk, et. al., 1999).

We identified three main categories for software engineering tasks:

1. ***Tasks performed by a team focusing on developing a software product based on customer requirements***[21]. This represents the core task of any software organization. The team leader (project manager) is responsible for ensuring that work is completed on time and within budget and possesses the intended functionality and quality. Software Engineering is document-oriented and what is produced during the project is a set of documents such as contracts, project plans, requirements and design specifications, source code, test plans and related documents. These documents are not just work products. There is also additional information embedded within them: (1) during the project they document the decisions; (2) after the project's completion, they contain the history of the project. The documents can be reused in different ways by the next project so that people can learn from them, by analyzing the solutions to different problems that these documents capture.

2. ***Tasks that focus on improving a team's ability to develop a software product (that is improving tasks in the first category).*** Here we can include tasks that might be conducted during and shortly after the project. The reason for performing these tasks is to ensure that potential knowledge gained in the project is not lost. Included here are all forms of lessons learned and post-mortem analyses that identify what went right or wrong in the project. Also included are analyses of data from the project, for example, comparisons of budgeted and actual costs, estimated and actual effort, planned and actual calendar time. Tasks in this category attempt to collect and create knowledge about *one* particular project. The results from this activity are useful by themselves, but can also be the basis for further learning. They can be stored in repositories and experience bases (for example, in *lessons learned* repositories).

[21] We include any software activity related to any software life-cycle phase in this definition to be able to discuss this issue on a general level. We refer to software development as a project-oriented task, but the same reasoning applies to many different organizational work models.

3. ***Tasks that focus on improving an organization's or an industry's ability to develop software.*** This category represents activities that analyze results from *several* previous projects in order to identify similarities and differences between them. The insights gathered by these analyses can be formulated as knowledge or experience packages and can be qualitative, quantitative, or a mix of both. Examples of qualitative packages are patterns, heuristics and best practices based on a number of experiences from different sources. Examples of quantitative packages are estimation models based on the measured attributes of previous projects and their budgeted and actual outcome. Other examples are knowledge that is packaged in terms of executable software programs that automate steps of the development process based on knowledge derived from previous projects. Industry-wide standards and recommendations also fall into this category.

The remainder of this section will discuss knowledge management for software engineering from the perspective of these three task categories (or levels).

First Level Knowledge Management

Knowledge Management Support for Core Software Engineering Activities

This section addresses core software engineering processes and activities. Birk illustrates the wide spectrum of software engineering processes that might occur in a typical software engineering project (Birk, et. al., 1999). What is common amongst the results from all these processes and activities is that they are all documents[22] (even the source code and the executable programs can be regarded as documents). The work is, many times, focused on authoring, reviewing, editing, and using these documents. Due to the fact that many software organizations are distributed over large geographic areas, these documents need to be remotely available. Because software engineering is so dominated by the documents that are produced during the various activities and processes, the foundation for a knowledge management system is a ***document management*** system. Hand in hand with document management comes the need of distributing information about the project, which calls for general ***information management***. Information management can be performed using regular office automation tools for e-mail, task management, and, scheduling. An example of such a system is Microsoft Outlook in combination with Microsoft Exchange Server. General information systems, however, fall outside of the scope of this report and are not discussed further.

Document management systems have been used for quite some time, but as the term ***knowledge management*** became popular, there was a tendency to re-label the document management tools as knowledge management tools, to accommodate the new trend.

[22] We use the term "document" in a broad sense. Typical documents are word processor files, spread sheet files, files storing project plans, presentations etc. A document can consist of one or many files.

Portal technology enables web-based communication within or outside organizations. Although managing web sites can be fairly complicated (for example, they need support for links and content management) portals can certainly be valuable to software engineering projects that need to share knowledge captured in different forms.

Document Management and Dissemination

We will illustrate the characteristics and functionality of document management tools by discussing one of these tools, *Hyperwave*[23]. Hyperwave is a relatively popular system that consists of a set of tools that implement both document management and portal technology. The *Hyperwave Information Server* enables organizations to share documents within and outside the organization. Access to documents is provided through web browsers as well as through virtual folders. Documents can be set under version control so that multiple authors can edit documents quasi-simultaneously. Hyperwave also has a mechanism for defining workflows that could support various processes (for example review processes). This is particularly useful for larger software engineering projects in which many people are involved in authoring, reviewing, and editing documents.

Interesting features related to knowledge management beyond regular document management are *Find documents* and *Find experts*. The ***Find documents*** feature allows a user to search for a wide range of document types among the documents residing on the server. Documents are automatically indexed when they are uploaded to the server. Indexing makes document searching possible by using different parameters, as well as full text search supported by a thesaurus. An extension of this feature allows users to search for similar documents as the one resulting from the search.

The ***Find experts*** feature allows users to search for experts in certain areas. Authors of documents that are related to the keywords used in the search are identified as experts.

The Hyperwave portal allows users to create new content that is published on the web site relatively quickly in form of *tracks*. A relatively large set of predefined tracks is provided. These tracks provide building blocks for the creation of more complex personal portals.

Hyperwave can be used by software organizations that need to share documents between geographically distributed locations. The mechanism for version control can be used to manage multiple versions of documents such as project plans, requirements specifications, design specifications, test plans etc. Although Hyperwave can handle source code as well, a better choice for source code management would be software development tools such as Concurrent Versions System (CVS) provided by the Free

[23] http://www.hyperwave.com

Software Foundation[24]. More information can be found at the general source for IT related topics[25].

For a more detailed description of Hyperwave and other related tools we refer to the DACS report on Software Tools for Knowledge Management (Lindvall, et. al., 2001).

Competence Management

As stated before, not all tacit knowledge in an organization can be made explicit. Therefore, in order to fully utilize the competence of the organization there is a need for keeping track of who knows what. Generally, employees do have knowledge about other employees' expertise if the group is small enough (10-15 people), but larger groups of people are exposed to the risk of "not knowing what other people know."

An elaborated solution to this problem is ***competence management***, a.k.a. ***skills management*** or ***expert network***.

Competence management systems were initially developed with the major objectives of being able to find employees with the right skills in order to staff new projects and to find individuals who have specific pieces of knowledge. Competence management has evolved over time into systems for much broader use. Now, organizations use competence management systems as sources for marketing and sales and for competence development, as well. The sales department could use them to identify which kind of project suits the company most. Marketing people could use this type of system for external marketing, for example as proof of a highly skilled workforce (Dingsøyr and Røyrvik, 2001).

Organizations like Microsoft and Hewlett-Packard, which strive to have "only the best" employees, use a competence management system to keep their personnel at the highest competency level. These organizations do not tolerate "legacy people" whose skills have become obsolete. Microsoft has employed a competence management system called SPUD in their organization (Davenport, 1997). With this system, they want to develop a structure of competency types and levels, by rating the performance of employees in particular jobs and linking the competency model to training offerings. Competence management has also improved employees' own perception of competence, helping them to better understand what skills are required to remain in the company. Hewlett-Packard (HP) has implemented a similar system named Connex (Davenport, 1996). HP's system relies on the experts entering their knowledge profiles and maintaining the profiles over time. To encourage people to use this system, they have an incentive plan of rewarding people with frequent flier miles and Dove bars. Positive experiences from using competence management systems have also been reported by (Dingsøyr and Røyrvik, 2001). Examples of tools for competence management are Skillscape[26]and Skillview[27].

[24] http://www.gnu.org
[25] http://www.informit.com
[26] www.skillscape.com

Other approaches to identifying experts in the organization…

A common problem that knowledge management is trying to address is the identification of de-facto experts. A solution for identifying experts would be to assume that one's expertise in a certain area is reflected by the documents they author. Thus, analyzing an organization's documents can identify de-facto experts. Competence management systems let experts identify themselves by editing their own profiles. An example of profiler tools is *KnowledgeMail*[28] that uses an automatically generated profile to identify experts. KnowledgeMail is based on the assumption that the content of people's e-mails reflects what they have knowledge about. KnowledgeMail analyzes e-mail repositories and builds keyword-based profiles that characterize each employee. There are two kinds of profiles for each employee: public and private. The public profile keeps a list of expertise-related keywords that the individual user has chosen to make public. The private profile keeps the keywords the individual does not want to make public. If the search for expertise using the public profiles does not result in satisfactory results, then requests can be sent out to anonymous experts based on their private profile. Revealing their own identity remains the experts' choice.

KnowledgeMail can be applied to software organizations and can help identify experts in various technical areas, for example, programming languages, database technologies, operating systems etc.

Lightweight Approaches to Knowledge Management

Lightweight approaches to knowledge management try to better manage knowledge and at the same time strive to not change how employees currently work and to become part of daily activities. These approaches are relatively easy to implement and have a great chance to pay off relatively soon (Schneider, 2000).

AnswerGarden (Ackerman, 1990) is an example of a lightweight approach that tries to capitalize on the knowledge exchanged between Help Desks and customers seeking help. All previous questions are stored in a knowledge base that the customer can browse to find answers. If the answer cannot be found in the knowledge base, then the customer can post a new question. The knowledge base is thus useful, even when it does not contain all answers for a specific area. The result is often that a knowledge base is populated relatively quickly, especially where there is a demand for knowledge. There are several examples of AnswerGarden concept implementations on the market. *AskIt*[29] is an example of a web-based application that is hosted by *AskIt* systems and allows an organization to quickly set up their own knowledge base. An important component of the AskIt system is the network of experts that will answer new questions. Another example of the application of AnswerGarden is the *Knowledge Dust Collector* (Lindvall, et. al.,

[27] www.skillview.com
[28] http://www.knowledgemail.com
[29] http://www.askit.com

2001), a system that supports peer-to-peer knowledge management where everybody is considered an expert in one or more areas. It blends the ideas of organic growth and expert supported knowledge bases with the concepts of the Experience Factory. The Knowledge Dust Creator captures the daily knowledge that employees exchange and use every day without explicitly acknowledging it. The knowledge dust is made available to other employees and, thus, is useful right away. Several feedback loops are provided so that the knowledge dust evolves over time into well-packaged experience in the form of knowledge *pearls*. This activity does, however belong to higher knowledge management levels.

Second Level Knowledge Management

Organizational Memory for Software Development

Learning from experience requires remembering history. Individual memory is, however, not sufficient and the entire organization needs a memory to explicitly record critical events. There are at least three distinguishable forms of organizational memory:

1. Memory consisting of regular work documents and other artifacts that were developed primarily to assist development of the product (examples in this category are requirements specification, and design specification)
2. Memory consisting of entities that were developed specifically to support the organizational memory (examples are lessons learned and post-mortem analyses)
3. A mix of the first two forms

In this section, we will describe regular software tools used by software engineers in their daily work that also support the creation of an organizational memory. Sometimes this organizational memory is deliberately created as part of the tool. Other times the memory is a desirable side effect.

An organizational memory for software engineers would answer the questions Why? Who? When? Where? How?

Design Rationale is a framework for creating an organizational memory that attempts to preserve information about the development of a software product so that the questions above can be answered. The idea behind Design Rationale is that during a software development project many different solutions to problems are tested and many decisions are made based on the results of these tests. The problem is, however, that the reasoning for these decisions is rarely captured, making it very hard for someone that was not involved in the decisions to understand why the software product is designed the way it is. Design Rationale captures exactly this information. Design Rationale also acknowledges the need for capturing information about solutions that were considered, but not implemented. The reason is that these "negative" examples could also have value

for a future maintainer of the software system. Maintainers who do not know the history of the design could be tempted to start redesigning the system and might even try to implement solutions that were earlier rejected for good reasons. If those decisions and the reasons behind them, Design Rationale argues, are captured instead, then future maintainers have a better chance of understanding the history and making better decisions regarding the evolution of the software system. An example of a Design Rationale approach is described in (Potts and Bruns, 1988).

While Design Rationale is an explicit approach to creating an organizational memory, regular tools for version control such as the Source Code Control System (SCCS) (Rochkind, 1975) represent a class of tools that indirectly create such a memory. Version Control (VC) is used to keep track of different versions of a certain document. Version Control can be applied to any kind of document and is part of many document management systems. Its origins can, however, be traced to version management of source code. A typical version control system allows users to read any document managed by the system. In order to change a document, the document must first be checked out. After the changes have been made the document is again checked in. As a part of the check-in procedure, the user comments on the changes that were made. While the actual changes can be tracked using programs that compare two documents and show the differences between them, the comment is supposed to state the reasons behind the changes. Each version of each document thus has a record attached with information about *who* made the change, *when* it was made, together with the comment that indicates *why* the change was made. The *diff* information indicates *what* was actually changed. The regular use of a version control system can enable multiple software engineers to work quasi-concurrently on the same documents and source code and can allow software engineers to retrieve the latest version of the system. The records of a VC can also be used as an organizational memory that indicates how the software product evolved during a certain time. It also provides information about the process behind this evolution. The regular software engineer can use this information to identify who made a certain change in order to find an "expert". The information has also been used for advanced analysis of software products, as well as software process (Eick et al, 2000), (Lindvall, 1998).

While Version Control helps manage versions of individual files, Configuration Management helps manage compositions of files. This is crucial for software engineers because software products consist of sets of files and every build consists of different versions of these files. In many cases different customers use different builds of the software system. Thus, there is a strong need for organizations to keep track of exactly which version of each source code file went into a particular build that a certain customer has problems with. Without having this information in the organizational memory, it would be impossible to recreate and detect the problems reported by the customers for their own version of the software system.

Software requirements drive the development of software systems, but there are many claims that the connection between the final system and its requirements is fuzzy (Soloway, 1987). This creates several problems. First, some contracts specify that the vendor should be able to relate each piece of code to one or more requirements. Second,

changes and additions to the product are often formulated in terms of new requirements, so there is a need to evaluate the impact of new and changed requirements on the product in order to determine the cost. Traceability is an approach that makes the connection between requirements and the final software system explicit (Lindvall and Sandahl, 1996). Tracing requirements contributes to the organizational memory and helps answer questions such as "What requirements led to a particular piece of source code?" and "What code was developed in order to satisfy this particular requirement?"

Third Level Knowledge Management:

Packaged Knowledge That Supports Knowledge Application

There are a large number of tools available, either as research prototypes or as commercial tools, that claim to be *knowledge-based*. Common for these tools is that they are specifically tailored for software engineering. They support the software engineer in his daily job and often result from analysis of knowledge from many previous projects. We performed a survey on tools that offer appropriate support to knowledge management in software engineering. This survey, of necessity, relies upon the claims made by tool developers, vendors and users. We could not test the tools to verify the validity of these claims.

After completing the survey, we classified the tools in two ways. First, we classified them according to the task they accomplish in the software engineering cycle. Second, we classified them according to the knowledge management cycle. Classifying according to the KM cycle resulted in three major categories

1. Tools supporting knowledge deployment and application
2. Tools supporting knowledge acquisition
3. Knowledge organization tools.

Classification of Knowledge-based Tools by the Software Engineering Activity That They Support

<u>Interactive Domain Understanding Tools</u>

Understanding the domain for which software is to be developed is a binding requirement. A lack of knowledge and understanding of the domain can significantly delay the project. Tools like *Kibitzer* (Schoen, 1991) and *RFML Workbench* (Gibson and Kevin, 1995) address this problem. These tools, while interacting with the user, acquire knowledge about the domain. Domain knowledge can exist in distinct types, such as knowledge of problem solving, knowledge of application tasks and knowledge about the user. By interacting with the user, these tools build a model of the domain, thus transferring knowledge from users to designers.

Intelligent Requirements Assistants

Requirements Assistants are tools used by requirements analysts to evaluate system requirements and codify them in formal specifications. These tools also help in requirements acquisition from the user, eventually helping with changing requirements and ill-defined specifications. Requirements assistants incorporate knowledge about application domains, system components, and design processes and support analysts in applying this knowledge to the requirements analysis process (Johnson, 1991). Other tools like KBRAS support reusability of software components by classifying and retrieving requirements acquired for other software applications (Zeroul, 1991). Specification assistants, when combined with requirements assistants, result in systems like ARIES (Johnson, 1991) that support the building of a specification through reuse of previously defined specifications from a knowledge base. These tools help transfer knowledge from customer to vendor. They also aid in acquiring knowledge about domain specific requirements, which can be used in latter projects.

Knowledge Based Program Designers

Program Designers are tools such as CAESAR (Fouque and Matwin, 1992), RT-Syn (Smith and Setliff, 1992), and Comet (Mark, 1992) that present their user with algorithms that match current requirements and specifications. These tools help transfer knowledge from the requirements phase to the design phase. Algorithms are either generated with a machine learning approach or retrieved from the knowledge base by case-based reasoning. Tools like CAESAR use case-based reasoning for retrieving algorithms from the knowledge base by matching current requirements and specifications. This great help in code design comes with a shortcoming. CAESAR tries to match every possible case with the current requirements. Although the result might be close matches, it becomes cumbersome for the user to deal with so many examples. Eventually the users might choose to generate the algorithms on their own (Smith and Setliff, 1992). While CAESAR presents the user with all possible cases that can be reused, RT-Syn tries to select one single possible algorithm. It looks into the algorithm database, choosing a likely candidate algorithm, and then makes design decisions based on the given constraints in requirements and specifications (Fouque and Vrain, 1992). *Designer Assistant* (DA) (Terveen, 1995) addresses three kinds of knowledge in its knowledge base:

- Expert knowledge of design with which most designers are not familiar
- Impact knowledge (i.e., how characteristics of a design affect another area of software)
- Fault prevention knowledge (i.e., how characteristics of design could lead to a fault)

Tools or design environments like DA not only help transfer design knowledge which exists as folklore in organizations, but they also help organizations build their knowledge bases and increase the expertise level of designers and developers.

Knowledge Based Code Generators

Most of the tools that we found in our survey are code generators. They rely on a previously acquired knowledge base that may or may not evolve, and strive to generate executable code. Such tools are KIDS (Smith, 1991), CodeBroker (Ye, 2001), and CodeFinder (Henninger, 1995). While tools like KIDS are based on development-with-reuse, other tools like CodeBroker work on the principle of reuse-within-development. KIDS and other code generators (e.g., SINAPSE (Kant, 1992)) generate code by taking partially implemented specifications and high-level design decisions as primary input. These tools transfer programming knowledge from experts to domain novices. Although these tools transfer knowledge, they still rely on the programmer's quest for acquiring and reusing that knowledge. On the other hand, CodeBroker is integrated with current development environments and tools. This system automatically creates reuse queries and locates reusable components by monitoring developer's activities. In this way, reuse of previous knowledge and components becomes an integral part of software development, rather than an added activity. CodeBroker works on the principle of *push* technology, i.e., the system will automatically let the developer know about every possible opportunity of reusing previous acquired knowledge and components. KIDS and other similar code generators work on *pull* technology, where the user has to actually search for knowledge and reusable components. While there are generic code generators like KIDS, there are also domain specific tools such as SINAPSE, which is a tool developed for generating code for supporting scientific programming. Specialty domains like scientific computing involve specific domain knowledge to be incorporated into these code generators. As a result, about 20% of the SINAPSE's knowledge base consists of code that accounts for domain knowledge and specific problem solving structures (Kant, 1992). Automation of certain tasks like coding can generate solutions more quickly, and can possibly produce more complex code, but manually created code is still considered more efficient (Kant, 1992). Still, these tools are something that programmers long for because they aid in transferring knowledge from the design phase to the implementation phase, thereby reducing their tasks by a large margin by providing guidance, reusable knowledge, and components that have been implemented in the past.

Smart Code Analysis Tools

Code analysis tools can offer substantial help during software testing and quality assurance activities. Analyzing code requires expert knowledge about the quality of the written code and good programming style. This knowledge is captured in a knowledge base and integrated with tools like OGUST (Fouque and Vrain, 1992) that analyze the code for quality and good programming style. These tools also help transfer knowledge in another way. As the programmer uses this tool, which compares current code with already gathered knowledge about good program structures, the programmer enriches his/her knowledge of elementary program structures.

Documentation Generators

Documenting software requires knowledge about requirements, specifications and design. There are tools that use this type of knowledge to generate documentation automatically (Jesus and Carapuca, 1992). This is a means of converting tacit knowledge into explicit knowledge.

Software Maintenance Tools

Software-understanding tools support the maintenance phase, where studying and understanding the software system is a necessity. Knowledge about a software system is usually spread out among source code, documentation, and the minds of designers and developers. Tools like KBSUNG (Majidi and Redmiles, 1991) centralize access to this knowledge. This tool inspects the code and, with a text explanation, tells the user about the underlying concepts in that software system. Another approach suggested by (Yu, 1991) proposes the use of a knowledge base for understanding the software and for showing consistency between new updates and existing code. This supports the transfer of designers' and developers' knowledge about the system to those who maintain it.

Predictive Models and Best Practices

For every project, team leaders and project managers need to make a series of decisions, both at the beginning of the project for all aspects of planning (staffing, schedule, process, techniques), and also during the project, to maintain the budget and estimated schedule and make the customer happy. They need support for deciding what alternative is better for their specific context, and identifying the effect of each approach on the overall project. Usually managers use their personal experience, their "gut feeling" to guide their decisions. But, since software development is such a complex and diverse process, the "gut feeling" might not be sufficient, not to mention that not all managers have extensive experience. For these reasons, predictive models can guide future projects based on past projects. This requires having a metrics program in place, collecting project data with a well-defined goal in a metrics repository (Basili, et. al., 1994b), and then processing the data for generating predictive models.

Model inputs (collected data) and outputs can be quantitative (e.g., effort, software size, number of defects, and estimated schedule versus actual) or qualitative (e.g., the type of project, application domain, staff experience, and process applied). Input data are analyzed, synthesized, and processed using different methods, depending on the purpose of the model and the type of inputs and outputs. For example, analytical models take numerical data or qualitative data (e.g., staff experience) converted into quantitative levels (e.g., 1-5) from a large number of projects and try to find formulae to correlate inputs and outputs. By using these formulae and providing them with the data that characterize a new project, one can compute estimated values for a series of project parameters. In this class are "black-box" analytical models for predicting the size and effort needed for a project, such as the COnstructive COst Estimation MOdels (COCOMO) (Boehm, 1984), COnstructive Cost Estimation for Commercial Off-The-

Shelf-based systems (COCOTS) (Abts, et. al., 2000), and the Software LIfecycle Model (SLIM) (Putnam, 1992). Another class of models in this category consists of reliability growth models that aid in predicting software reliability and deciding when to stop testing by projecting the failure rate in a system during testing. Examples are the logarithmic Musa-Okumoto (Lyu, 1996) and the exponential Jelinski-Muranda models (Lyu, 1996). These analytical models refine raw data and transform it into knowledge that, in this case, is embedded in the formulae.

Another class of models captures not only the relationship between inputs and outputs, but also the structure of the development process and the relationships between internal process variables. Examples are process models based on system dynamics (Abdel-Hamid and Madnick, 1991), the STATEMATE notation and tool (Wood and Krut, 1991), and discrete event modeling (Höst et.al., 2000), (Rus Ioana et al, 1999). These models can be executed, allowing simulation of different scenarios and analysis of different possible outcomes for multiple potential decisions. The knowledge captured by these models addresses the structure and the behavior of the projects, and is richer and more explicit than that captured by the "black-box" models (Kellner et al, 1999). Of course, the quality of the predictions offered by all these models depends on the quality (accuracy and reliability) of the collected data.

Information collected from projects can also be in a qualitative form, such as cases and situation specific lessons learned (Harrison, 2001); success and failure stories; and problems and corresponding solutions. This information can be represented in text format, or more formally as rules, indexed cases, semantic networks, etc. Applying generalization and abstraction on these data, we can generate knowledge that can be applied to other similar contexts. This is how patterns, best practice guidelines, handbooks, and standards can be derived. Artificial intelligence techniques and machine learning algorithms are being used for knowledge discovery and acquisition.

In all of the models presented above, the raw point data is transformed into explicit and more general applicable knowledge by using different refinement operators and techniques. These tools support the knowledge acquisition and application activities.

Process Design

In order to accommodate software development process diversity while retaining a level of discipline and standard, Henninger proposes a methodology and tool support that captures project experiences within a software process framework (Henninger, 2000). These experiences are then disseminated to developers to provide knowledge of previous development issues in the organization. Deviations from the standard process are seen as an opportunity to improve and refine the process itself. This approach is called "organizational learning" because it requires iterations to a formally defined process on the basis of deviations to this process. Apart from refining the process, the deviations also work as the *cases* in the experience base. As more and more experience is acquired in the form of cases, the process is iterated and becomes more refined. Creation of knowledge about software processes and their use is a dynamic activity and requires tool

support. A case-based reasoning tool called BORE has been developed to support this activity (Henninger, 2000). BORE captures and manages experience cases into various domains. It supports the retrieval of knowledge in the form of experience cases to help in the development process.

Classification of Software Engineering Knowledge-based Tools by the Knowledge Life Cycle Phases That They Support

Knowledge Deployment/Application Tools

This is the category that applies to the majority of the tools. Tools like KIDS and BORE work towards the dissemination of organizational knowledge to every employee who needs it. These tools apply the organizational memory or the organizational knowledge base to support tasks of the designers, developers and other software engineers. These tools can generate code (KIDS, CodeBroker), help in design and requirements analysis (SIB-ST, CODA) and assist in testing and quality control (OGUST). They are designed based on the assumption that there is a knowledge base to support them so that they can accomplish their functions. They do not address the creation or evolution of the knowledge repositories. Although most of the tools in this category are static in nature and do not evolve, they help reuse the existing knowledge and facilitate finding a solution to the problem at hand. This solution might not be the one that the engineers will eventually implement, but it still provides a guiding path to follow.

Knowledge Organization Tools

The knowledge organization phase adds context to the information that is either captured by the system or updated by the user. In the survey, we did not find any specific knowledge organization tool for software development. Organization tools maintain the knowledge base (organized according to its classification); add the relationship of each knowledge item to other items in the knowledge base; set up a hierarchy of knowledge items; maintain a history of knowledge items that have been reused during knowledge based software development; add meta-data to knowledge items; and describe relationships of specialization and generalization.

Knowledge Acquisition Tools

Knowledge acquisition tools help transfer and transform expertise from knowledge sources to explicit knowledge representations that enable effective use of the knowledge. Tool support for knowledge acquisition software engineering is rare. This is probably due to the lack of "formalization of knowledge management and problem solving processes in software engineering" (Birk, et. al., 1999) and to the fact that required knowledge exists mostly implicitly, codified, and informally (Birk, et. al., 1999).

Knowledge acquisition tools have to deal with the very difficult task of making tacit knowledge explicit. This makes knowledge acquisition in software engineering

extremely difficult. A technique used for acquiring knowledge is a dialogue between the expert and the system. The system asks the expert for all possible solutions (the solution space) for the type of problem in question, and then tries to distinguish them by questioning the expert about parameters that differentiate the solutions (Eriksson, 1992). Some tools like BORE acquire knowledge by treating every use of the knowledge base as a case and linking it to the knowledge base for further reuse (Henninger, 2000). Tools that support reuse also help acquire knowledge. RA is an example of such a tool (Reubenstein, 1991). RA captures information about the components used by a requirements engineer (data accessed, requirements model used, and related domains). This information can later be used to support the requirements engineering process.

Section 4.
Implementation of Knowledge Management

So far, this report has discussed knowledge management in general and how knowledge management in different forms could be beneficial for a software engineering organization. Implementing a KM system might, however, not be so simple, involving both challenges and obstacles. Examples of the most important issues noted by D. Rigby, an analyst for Bain&Co. (Lawton, 2001) are:

- Technology issues:
    - KM involves software technology, but it is not always simple or even possible to integrate all the different subsystems and tools to achieve the level of sharing that was planned.
    - Inadequate security. While the idea behind KM is to share knowledge, it is important not to share knowledge assets with the wrong audience (e.g., competitors and former employees). This issue might limit the extent to which knowledge can be shared in the organization.
- Lack of standards:
    - Different parts of the organization might use terms and concepts in different ways. This lack of standards can inhibit sharing of knowledge between them.
- Organizational issues:
    - It is a mistake to focus only on technology and not on methodology. It is easy to fall into the technology trap and devote all resources to technology development without planning for a KM implementation approach.
- Individual issues
    - Employees do not have time to input knowledge or do not want to give away their knowledge.

It is important to learn from failures and mistakes. An analysis of KM failure causes reveals that some users manage documents instead of meaningful knowledge (Lawton, 2001). This is easily done, because most of the KM tools on the market address document management rather than knowledge management. Another problem relates to the fact that not everything can be managed by a KM system. If everything is dumped into a common repository, the amount of information to be managed becomes un-manageable and most of it can be useless. The last failure cause mentioned by Rigby is related to the fact that users did not determine their goals and strategy before implementing KM systems. In fact, 50-60% of all KM deployments failed because many organizations did not posses a good KM deployment methodology or process, if any. Consequently, “financial and organizational costs swamped the benefits, usually as a result of inadequate attention to strategic priorities” (Lawton, 2001).

There is no doubt that recent developments in technology have made it possible and easier than ever to collect and share knowledge, but organizational cultures might not

encourage or even allow it. According to Agresti, developing a knowledge culture was frequently cited as the number one obstacle to successful KM (Agresti, 2000). It will be very hard to get acceptance from employees if the new technology is perceived as opposing the organizational culture. If the new technology can be introduced in a way that is in line with the organizational culture, then the initiative has a much higher chance to succeed. Having these challenges, obstacles, and failures in mind, this section will focus on the cultural issues of implementing a KM system. It will discuss different reward systems, give an example of a methodology for implementing a KM (namely the Experience Management Systems), and compile a list of success factors for implementing KM systems.

Reasons Why People Would Not Share Knowledge

A company's culture reflects what people think and feel about the organization. Do they trust each other and their management, and are they willing to go out of the traditional bounds of the work culture to benefit the organization?

Software organizations need to realize that employees may feel possessive about their knowledge, and they may not be forthcoming in sharing it. After all, the knowledge they have is why they are valuable to the organization, why they are paid by the organization, and why they do not want to give that knowledge away. A term which is used these days is "capturing tacit knowledge", which is similar to "picking your employees' brains." This term sounds like software organizations are picking whatever their employees know. The "capturing emotion" might scare people into withholding their knowledge, thinking they will be expendable as soon as their employers have captured all of the knowledge they need. If this was the result of successful knowledge management, then everybody should be afraid of losing their job (Vlietinck, 1999).

Here are several reasons why employees might be reluctant to share their knowledge:

1. Employees want the organization to be dependent on them. If they share the knowledge with others, they fear they will loose their "expert" status.
2. Some cultures encourage individualism and ban cooperative work and sharing. In such cultures it is harder to establish a successful knowledge management program. As a matter of fact, most Western schools do not encourage students to work together in the classroom or while doing homework, so most students have learned that sharing is cheating. In order to create a sharing culture, such values and manners have to be unlearned.
3. Employees might not be willing to share lessons learned because of their negative connotation. Lessons learned are based on incidents, some of which might be failures. Although the purpose is to learn from failures to avoid similar mistakes, many employees might fear that submitting negative lessons learned could be interpreted against them by management.

These are cultural issues that management must handle by creating a learning environment. Employees will, however, always be concerned with how management treats them, and the information that management has about them. Employees will react negatively if they fear that information will be used against them.

An Example: Harvesting Knowledge from E-mail

In Section 3 we discussed KnowledgeMail, a software tool that creates expertise profiles based on e-mail content. It is clear that the technology exists to harvest knowledge from e-mail, but does the organizational culture allow it?

There are many reasons why individual employees would not want to let a program analyze their e-mail, even though it would be possible to prohibit publication of the profile. E-mail is commonly used for a mix of work-related and private issues, and many employees fear that an analysis program would reveal to management facts that they want to keep private. The fear is that even if the keywords could be kept private, there would still exist a possibility to access them. It is a fact that many employers already monitor their employees' email traffic. The difference is that most of the time monitoring happens without the employees knowing or being aware of it. With approaches that explicitly analyze e-mail, the situation is different and employees would have to change their behavior considerably when using e-mail. They would, for example, have to strictly distinguish between personal and work-related e-mail accounts.

Creating a Sharing Culture

A critical factor for knowledge and experience sharing is that management creates trust amongst employees, and between employees and management. It should be noted, though, that this is a long-term goal. The fear that information can be interpreted and used against individuals has been acknowledged in other aspects of software engineering, for example in measurement. A solution to this problem is, for example, the one proposed by the Experience Factory (EF) approach, presented in Section 2. EF clearly states that data collected by individuals must be anonymous and should never be used to judge them (Basili, et. al., 1994a).

The core values promoted by the EF for establishing a sharing culture are based on the fundamentals of learning. In order to improve, employees need to learn from past experience, and in order for employees to learn, the organization needs to create a learning environment. The characteristics of a learning environment are that it is allowed to make mistakes and learn from them. Experience is not hidden or traded, but freely given to the employee who needs it. Experience is collected, not in order to replace, degrade or evaluate people, but in order to help them (e.g., help them remember; help them collaborate; and help them organize, spread and share data, information, knowledge, and experience). People are encouraged to share experience and help others, and are rewarded based on how much they share.

Learning and improvement can only occur in an environment where it is possible to obtain feedback about the outcome of various activities. A learning organization creates feedback loops on several levels and the design of the experience management system must allow, and even enforce, feeding data back to the system. An example of feedback loops is an honest dialogue between employees in the organization. Another way of creating feedback loops is the principle of iteration, i.e., the work is iterated and improved in steps. Iteration also facilitates removing defects early in the lifecycle (Basili, et. al., 2001).

An example of EF implementation can be found at Fraunhofer Center For Experimental Software Engineering, Maryland (FC-MD). FC-MD has implemented the EF in order to leverage its employees' knowledge and experience (Basili, et. al., 2001). FC-MD uses several strategies to set the right culture in order to encourage employees to share and use knowledge and experience. The first strategy was to establish corporate core values that explicitly address and support the core values of an experience factory. Another strategy is to weave experience-related activities into the regular work process and leverage what employees are already doing. One example of such activities is the *project presentation*, where project managers actively collect experiences and present them to the rest of the organization. The project presentation is packaged in a way that helps new employees learn about projects and facilitates project analysis. In order to show that management supports these activities, a special project account has been set up to which employees charge all activities related to the experience base, so that these activities do not increase the cost of their current projects. Employees' contribution to this initiative and to the experience base is a criterion for the individual annual performance evaluation.

Reward Systems

Implementing reward systems can help develop a knowledge-sharing culture. There are numerous ways of rewarding employees and showing that the organization appreciates employees who are willing to share their knowledge with others, and are willing to search for and use knowledge created and documented by others. Next, we present several instances of reward systems.

To encourage the sharing and reuse of knowledge, Xerox recommends the creation of a "Hall of Fame" for those people whose contributions have solved real business problems (Vlietinck, 1999). The company rewards staff that regularly shares useful information and identifies them as key contributors to the program.

At Hewlett Packard, Karney, the main evangelist of the KM initiative, gave out free Lotus Notes licenses to prospective users. When a new knowledge base was established, he gave out 2000 free airline miles for the first 50 readers and another 500 miles for anyone who posted a submission. Later promotions involved miles for contributions, for questions, and for responses to questions. Soon, more than two-thirds of the identified

educator community had read at least one posting, and more than a third had submitted a posting or comment themselves (Davenport, 1996).

Another type of reward system is the "points system" used by *ExpertExchange*[30]. People are rewarded with points for answering questions that other people asked. Everyone who opens an account receives a certain amount of points. These points can be used to "pay" for answers received from experts, thus transferring points from askers to answerers. The site is open to the public, that is to everybody who wants and is able to be an "expert" and answer questions. The people with the highest number of points have answered the most questions and are often mentioned on the front page of the web site. In order to ensure quality answers to questions, it is possible for "askers" to feed back information that indicates whether the expert's answer was satisfying or not.

The Importance of a Champion

The example from Hewlett-Packard points out that an evangelist or champion is needed in order to start and maintain the initiative. This person needs to market the effort to encourage employees to contribute and use the system, and must always be its proponent. Despite Karney's success described above, he was also frustrated. In spite of the rewards with free miles and e-mail and voice mail exhortations, he still felt the need to continually solicit fresh contributions to the knowledge base. "The participation numbers are still creeping up," he notes, "but this would have failed without an evangelist. Even at this advanced stage, if I got run over by a beer truck, this database would be in trouble." (Davenport, 1996).

Leveraging Employee's Expertise

Everyone who attempts to manage knowledge encounters the same question: "Does all knowledge need to be documented?" Most models that support experience reuse and knowledge management make the assumption that all relevant experience can be collected and recorded, but this does not necessarily hold true in practice (Wieser, et. al., 1999). The benefit of explicit knowledge or experience is that it can be stored, organized, and disseminated to a third party without the involvement of the originator. The drawback is that it takes a considerable amount of effort to produce explicit knowledge, if it is possible to produce it at all. Some definitions of tacit knowledge state that tacit knowledge is *inexpressible*, indicating that this form of knowledge cannot be expressed at all, much less documented.

Knowledge sharing does not have to be limited to the formal knowledge management system. Knowledge sharing normally occurs at many places. Knowledge sharing occurs at the coffee tables, in the lounge, and even around the water cooler. When an employee tells a colleague how a particular problem was solved, knowledge is shared. When somebody writes a five-line code on a paper napkin to help a friend, knowledge is again

[30] www.expertexchange.com

shared. Software organizations could encourage these habits in order to create a knowledge sharing culture. To reach the maximum effect of knowledge sharing, the employees should also be encouraged to document and store their knowledge in the KM system. They should be encouraged to deposit information into the KM system of the organization whenever they help somebody in a way that directly or indirectly relates to the organization's knowledge. By doing so, they are ensuring that whatever information they have passed on does not get lost in the minds of the employees. What is a problem for one can also be a problem for others (Terveen, et. al., 1993).

The Experience Factory (Basili, et. al., 2001) is an example of an approach that is based upon the assumption that knowledge and experience can be made explicit so that they can be stored in knowledge and experience bases.

Ericsson Software Technology AB has implemented a version of the Experience Factory called the *Experience Engine* that does not focus on documented knowledge and experience (Johansson and Hall, 1999). Instead of relying on experience that is stored in experience bases, the Experience Engine relies on tacit human expertise. Two roles were created in order to make the tacit experience accessible to a larger group of employees. The ***experience broker*** is a person who connects the people who have a need for experience with people that have the experience. The experience broker must be a generalist so that he can quickly understand the problem at hand and identify the right person that has relevant knowledge to solve the problem. The experience broker must have both social skills and a well-developed personal network of people. The ***experience communicator*** is a person who has in-depth knowledge on one or more topics. The experience broker connects the experience communicator with the person owning the problem. The communicator should not solve the problem, but should educate the problem owner in how to solve it, following the philosophy that says "Give the man a fish and you feed him for a day, teach the man how to fish and you will feed him for a life time."[31]. A similar approach based on experience brokers was implemented with good results at sd&m AG (Brössler, 1999).

The approach of relying on tacit knowledge rather than explicit knowledge is appealing because it relaxes the requirement to document extensively. Although it better utilizes the knowledge, it still does not solve the problem of the organization being dependent on its employees. This is an approach that addresses tacit-to-tacit knowledge transfer. It would be natural to think that a well-implemented Experience Engine would explicitly reveal who knows what and, therefore, it can be said that this approach supports making the organization aware what knowledge it has (and, indirectly, what knowledge it does not have).

[31]Proverb Lao Tzu

Success Factors in the Implementation of a KM System

Below we present a set of factors required to exist for a successful implementation of a KM system, compiled from (Wiig, 1999), (Tan, et. al., 1998), (Moody, 1999) and (Dennis, 2000).

Knowledge friendly culture – The organization values learning and innovation, and establishes appropriate incentives and reward systems. People collaborate and have a positive attitude towards knowledge. When there is free flow of knowledge from other employees, individuals tend to respond in the same manner.

Opportunities – Employees must be placed in an environment where they have opportunities to use their capabilities to the fullest.

Motivation – Employees must be motivated to share their knowledge with other people in the organization. They must be convinced that their sharing of knowledge will be valuable to the organization and, most importantly, to themselves.

Concrete shared objectives – Develop a broadly shared understanding of the enterprise's mission, current direction, and the role of the individual in support of the enterprise and of the individual's own interests.

Knowledge base – The knowledge base should be managed the same way as physical assets. Time and effort should be invested in designing, building and maintaining its content.

Technical infrastructure – All knowledge management systems should be linked to other information systems, providing necessary security features.

Effective governance for the KM practices – Continuous monitoring, evaluation, and guidance of the KM activities and their plans, results and opportunities.

Interdisciplinary problem solving working groups – Create problem-solving groups comprised of people from a variety of disciplines. This will transfer the knowledge from one discipline to another, as well as provide solutions to interdisciplinary problems in decreased time.

Multiple channels for knowledge transfer - A variety of channels for knowledge transfer are desirable, as each adds value in a different way. It is particularly important to provide opportunities for face-to-face contact, as well as electronic forms of communication.

Measurement - Measuring the usage of the KM system and its efficiency is essential to the accurate assessment and improvement of knowledge management programs in order to be able to increase the value or prolong the duration of the sustainable competitive advantage.

Empowerment – Employees must be given permission to innovate, improvise and stretch enterprise policies and practices beyond the predetermined scopes.

Successful Knowledge Management Case Study

Unfortunately, only a few successful implementations of KM systems in software development companies are published. We present one of them here, namely the KM program for software development at Daimler Chrysler (DC) (Schneider, 2001), whose champion is Kurt Schneider. The program is called Software Experience Center (SEC) and is based on the Experience Factory approach. SEC has tried different approaches to knowledge management for software and learned that:

- Documenting experience is often a pain in the neck for experienced experts because it is very time consuming
- The end users of these experiences often want other things than what is documented by experienced authors

To achieve a successful implementation, SEC turned passive experience bases into dynamic experience magnets that "actively attract experiences." To counter the problem of documentation, they developed techniques and tools that allow capturing experiences in ways that are less labor intensive and even made the regular job easier for those who contribute experiences. One strategy is to capture experience during activities regularly performed by employees. An example of this strategy is recording everything that is said and done during prototype demonstrations. Another example is filling in an optimized one-page form that helps capture experience while one is still in action.

Prototypes are of little value by themselves. It is the concepts and rationale built into them and the gained experiences that make them valuable. The demonstrations of products and designs are considered the essence of the project. So, the presenter of the prototype is required to mould the demonstration in a way that makes it suitable in explicitly expressing the related knowledge. The demonstration is recorded in any possible way (audio, screen shots, execution paths etc.) and user feedback is collected during the presentation. Later, when the ideas of the prototype need to be revisited, these recordings can be used to refresh one's memory.

Collaboration is another example of attracting experiences into the experience base. The use of communicative power of a computational environment is used to support tasks like risk analysis. Movements of users and their chat contributions are recorded. These can be replayed later by others to learn about risk analysis by studying how the experts argued about different risk factors.

SEC attracts developers into experience elicitation by making it easier to record experience and incorporating this recording into a task that they have to do anyway. It also attracts potential end users by the amount of information and highly differentiated analyses available.

Methodology for Implementing an Experience Management System

As stated at the beginning of this section, having a methodology to guide the implementation of any system is a critical element of any KM program. We give an example of such a methodology, namely the implementation of an Experience Management System (Basili, et. al., 2001) that is relevant to any experience-based and knowledge-based approach.

Different organizations have different needs and cultures and that is the reason why each EMS implementation needs to be tailored to the target organization. The EMS methodology addresses this issue and helps in the understanding and setup of an EMS for any specific organization. The methodology helps define the content, structure, procedures and tools that will be part of the EMS. The participation of people from the organization in the application of the methodology is crucial for the success of the EMS implementation because such an initiative cannot be successful without peoples' support, contribution, and use. They are also the ones who know their culture and problems best and can help avoid obstacles and aim at solving the true problems. The following is a description of the methodology steps for developing an EMS for a particular organization and domain of experiences. It is based on best practices derived from multiple past EMS projects and has been continuously improved.

The first step is a characterization of the organization and definition of current business processes and existing knowledge. A distinction is made between knowledge that is documented, undocumented and unavailable. Many organizations already have procedures in place to manage isolated subsets of their experience, but fail to manage all crucial experience. The characterization helps understand what experience is not covered, how existing documented experience fits into the new system, and how it can be reused.

After the characterization of the organization, roles for the EMS users are defined and use cases are developed based on business processes and user roles. User roles are defined based on the culture of the organization and the type of roles that will be performed by different people. Examples of user roles are *provider* (anyone who provides experience to the EMS's experience base); *consumer* (anyone who uses the EMS to search for experience), and *maintainer* (a person who is responsible for maintenance of the experience base). User roles can be refined for each main category. An example of this refinement is *topic managers*, derived from maintainer and defined as anyone who is responsible for maintenance of experiences related to a specific topic. Use cases are defined based on the characterization of the organization, the business processes that are

relevant to the EMS and the user roles. The use cases cover procedures that are already in place and add new ones as necessary.

The next step is to define a data model, or *taxonomy*, which is suitable for the organization. The data model is used to describe and classify the experience that will be included in the EMS in order to make it easier for users to retrieve the experience. In this step, different types of experience that will be managed are identified and classified. Acceptable values for each component of the data model are also defined. The results of this step are documented in an EMS Requirements Document.

Based on the EMS Requirements Document, the architecture of the EMS is defined. COTS, glueware, and in-house built components that, together, fulfill the requirements are used to define the architecture. Applications already in place and used by the organization are strongly considered to be part of the architecture. The architecture is then implemented. Tools supporting the EMS are developed, installed and integrated.

After implementation, a set of procedures for the regular maintenance of the EMS is deployed. These procedures are tied to the user roles and will make sure that the system works and that the managed experience is always up to date.

Following the development of the EMS procedures, the EMS's experience base is populated with an initial set of experience packages (the *seed*), and the EMS is configured and launched. A rollout plan is prepared and executed to train, market and motivate people to use the EMS.

The Experience Base is a living entity and has to be treated accordingly. It has to be nurtured, cared for, and allowed to grow and renew itself. The Experience Base, therefore, has to be maintained regularly, as without maintenance the Experience Base will die because the users will not trust it anymore. Its users will soon discover that the Experience Base ages and will abandon it when they realize it provides them with less and less value. However, pure maintenance is not enough. In order to retain its users, the Experience Base must also improve over time and continuously add value. Therefore, after the system is deployed it must be constantly improved based on feedback and measurement. Types of feedback are formal evaluations (including interviews and tests with users); direct feedback from users; feedback loops embedded in the tools; and analysis of usage data for the tools. According to the received feedback, the content is constantly updated and new experience packages are analyzed and synthesized into new experience packages. This step is continuously iterated in order to improve the EMS.

Summary

The focus of this report is knowledge management in software engineering. It presents the developments in knowledge management in general, and for software engineering in particular, and discusses models, approaches, and tools for knowledge management. The report also presents resources that can provide help, inspiration, and information to organizations that want to better manage their knowledge.

Software development is a knowledge- and people-intensive activity. Groups that are geographically distributed carry out a significant amount of the work in software engineering. People in such groups must collaborate, communicate, and coordinate their work, which makes knowledge management a necessity. As a matter of fact, small and stable organizations where employees are within an arm's reach of each other can probably survive without knowledge management. However, for organizations that are large and distributed, whose environment is continuously changing, or have a high turnover, managing their knowledge assets is critical for survival.

A characteristic of software engineering that turns out to be an advantage over other industries in terms of managing intellectual capital is that artifacts are already captured in electronic form and can easily be stored and shared. In addition, software engineers often have a friendly attitude towards using new technology. This means that a software organization that implements a knowledge management system could have a good chance to succeed with this mission. However, this remains a challenging task because a knowledge management system is more than just technology. There are only a few published reports about initiatives to manage knowledge in software organizations, but all of them talk about the difficulty of achieving employees' acceptance and implementing the KM system in a way that maximizes the help provided to its users (Schneider, 2001), (Brössler, 1999), (Johansson and Hall, 1999), (Terveen, et. Al., 1993). A software organization that seeks to implement knowledge management can find these reports very useful.

Ideas about buying or developing knowledge-based and knowledge management tools can also be acquired from the tools mentioned in Section 3. Many of these tools are based on previously captured knowledge that is packaged and easily disseminated and applied. Other tools help with knowledge management in that they provide a support for a specific task within (or related to) knowledge management. Other tools provide the fundamental platform for knowledge management on top of which helpful applications can be built.

The fundamentals for knowledge management in software engineering from our perspective are Document Management and Competence Management. The artifacts resulting from software development tasks represent the explicit knowledge assets of the organization and must be managed efficiently. They not only serve the project for which they were produced, but can also be analyzed in order to generate new knowledge. A Document Management system is needed for managing these assets. Document

Management can, however, only manage explicit knowledge and not tacit knowledge. As a matter of fact, a large part of the knowledge in a software organization can never be made explicit. Tacit knowledge is embedded within the employees, thus the need for organizations to manage information about their employees' knowledge. We call this the Competence Management system. This system keeps track of all the individual employee's knowledge so that the organization is aware of who knows what.

Although new technology can help solve problems, it is also an issue because it requires that software engineers acquire new knowledge daily. The trend is that there are more and more resources on the web that help software engineers pick up new technology faster and share their experiences with others. Due to the speed at which this form of knowledge sharing has been growing so far, we expect it to grow enormously over the next several years. It creates a market place for expertise and, thanks to the Internet, the experts can serve their customers from any location.

Another problem, even harder than changing technology, is how to share knowledge within the organization. To learn and understand the local practices at an organization can be very hard and it often takes a long time until a new employee is productive. Local organizational practices involve specifics about applying technology in a certain way, in a certain domain. This becomes more of an issue if the organization has to maintain legacy systems. Often there is little or no documentation indicating why a system was designed a certain way. If the legacy systems have been used for a long time, all of the people who initially designed them might have left the organization. The technology initially used might be obsolete, and the documentation that is available might not match the actual system. The result is that no one dares to change the system because it is impossible to tell what the consequences might be. Capturing and disseminating local development practices could probably benefit the most from a knowledge management system.

Many reports on knowledge management emphasize the importance of tool support. The truth is that technology is only half of the battle. If people do not want to share knowledge, then no technology can make them do so. Thus, it is very important to get acceptance from the people who should eventually use the system. Acceptance is probably easiest to get by demonstrating that the new system will make employees' lives easier. Successful knowledge management also requires management support. This means, for example, that management must invest sufficient resources. It is possible to achieve short-term benefits from knowledge management initiatives, but, in the end, knowledge management is a long-term investment and must be treated as such. Otherwise, it might fail fairly soon.

There is a trend in the increase of the number of commercial tools that support knowledge management available on the market. These tools will make managing of diverse forms of knowledge possible and affordable for all organizations. However, making appropriate use of these tools requires that organizations pay at least as much attention to cultural issues as we have outlined in this report.

Acknowledgements

We would like to thank the following people who helped us author this report.
Patricia Costa, Scott Henninger, Raimund Feldman, Forrest Shull for reviewing the document, and Jen Dix for proofreading.

References

Abdel-Hamid, T. and Madnick, S. E., “Software Project Dynamics An Integrated Approach”, Prentice-Hall, Englewood Cliffs, NJ, 1991

Abts, C., Boehm, B., and Bailey Clark, E. "Observations on COTS Software Integration Effort Based on the COCOTS Calibration Database", The 25th Annual NASA Goddard Software Engineering Workshop, 2000

Ackerman, M. S. "Answer Garden: A Tool for Growing Organizational Memory", Conference on Office Information Systems, COIS90, Cambridge, Mass, 1990

Agresti, W., "Knowledge Management," Advances in Computers, Vol. 53, 2000, pp. 171-283

Basili, V. R., Caldiera, G., and Rombach, D. H., "The Experience Factory," Encyclopedia of Software Engineering - 2 Volume Set, Wiley, 1994a, pp. 469-476

Basili, V. R., Caldiera, G., and Rombach, D. H., "The Goal Question Metric Approach," Encyclopedia of Software Engineering - 2 Volume Set, Wiley, 1994b

Basili, V. R., Lindvall, M., and Costa, P. "Implementing the Experience Factory Concepts as a Set of Experience Bases", Knowledge Systems Institute, 13th International Conference on Software Engineering & Knowledge Engineering, 2001, pp. 102-109

Basili, V. and Rombach, H. D., "Support for Comprehensive Reuse," IEEE Software Engineering Journal, Vol. 22, No. 4, 1991, pp. 303-316

Bennis, W. and Biederman, P. W., "None of Us Is As Smart As All of Us," IEEE Computer, Vol. 31, No. 3, 1998, pp. 116-117

Birk, A., Surmann, D., and Althoff, K.-D. "Applications of Knowledge Acquisition in Experimental Software Engineering", 11th European Workshop on Knowledge Acquisition, Modeling, and Management, 1999, pp. 67-84

Boehm, B., "Software Engineering Economics," IEEE Transactions on Software Engineering, Vol. 10, No. 1, 1984, pp. 5-21

Broadbent, M., "The Phenomenon of Knowledge Management: What Does It Mean to the Information Profession?," Information Outlook, Vol. 2, No. 5, 1998, pp. 23-36

Brössler, P. "Knowledge Management at a Software Engineering Company - An Experience Report", Workshop on Learning Software Organizations, LSO'99, Kaiserslautern, Germany, 1999, pp. 163-170

Curtis, B., Krasner, H., and Iscoe, N., "A Field Study of the Software Design Process for Large Systems," Communications of the ACM, Vol. 31, No. 11, 1988, pp. 1268-1289

Davenport, T., “Knowledge Management at Hewlett-Packard”, Knowledge Management Case Study, http://www.bus.utexas.edu/kman/hpcase.htm, 1996

Davenport, T., “Knowledge Management at Microsoft” Knowledge Management Case Study, http://www.bus.utexas.edu/kman/microsoft.htm, 1997

Davenport, T. H. and Prusak, L., “Working Knowledge: How Organizations Manage What They Know”, Harvard Business School Press, Boston, MA, 1998.

Dennis, F. C., “The Role of Knowledge Management in the Development of a Sustained Competitive Advantage”, Spescom Datafusion, 2000

Devanbu, P., Brachman, R., Selfridge, P., and Ballard, B. "Lassie: A Knowledge-based Software Information System", 12th International Conference on Software Engineering, 1990, pp. 249-261

Dingsøyr, T. and Røyrvik, E. "Skills Management as Knowledge Technology in a Software Consultancy Company", Workshop on Learning Software Organizations, LSO'01, Kaiserslautern, Germany, 2001, pp. 96-103

Eick, S. G., Graves , T. L., Karr, A. F., Marron, J. S., and Mockus, A., "Does Code Decay? Assessing the Evidence from Change Management Data," Transactions on Software Engineering, Vol. 27, No. 1, 2000, pp. 1-12

Eriksson, H., "A Survey of Knowledge Acquisition Techniques and Tools and Their Relationship to Software Engineering", Journal of Systems and Software, 1992, pp. 97-107

Fouque, G. and Matwin, S. "CAESAR: A System for Case Based Software Reuse", 7th Knowledge-Based Software Engineering Conference, KBSE, 1992, pp. 90-99

Fouque, G. and Vrain, C. "Building a Tool for Software Code Analysis: A Machine Learning Approach", 4th International Conference on Advanced Information Systems Engineering, CAISE, 1992, pp. 278-289

Gamma, E., Helm, R., Johnson, R., and Vlissides, J., “Design Patterns Elements of Reusable Object-Oriented Software”, Addison-Wesley, 1995

Gibson, M. and Kevin, C. "Domain Knowledge Reuse During Requirements Engineering", 7th International Conference on Advanced Information Systems Engineering, CAISE, 1995, pp. 283-296

Harrison, W., “PSU Software Engineering Lessons Learned Database”, Portland State University, http://www.cs.pdx.edu/~warren/LLR/, 2001

Henninger, S., "Information Access Tools for Software Reuse," Journal of Systems and Software, Vol. 30, No. 3, 1995, pp. 231-247

Henninger, S., "Case-Base Knowledge Management Tools for Software Development," Automated Software Engineering, Vol. 4, 1997, pp. 319-340

Henninger, S., "Using Software Process to Support Learning Software Organizations", The 25th Annual NASA Goddard Software Engineering Workshop, 2000

Höst, M., et.al., "Exploring Bottlenecks in Market Driven Requirements Management Processes with Discrete Event Simulation", 3rd Process Modeling and Simulation Workshop, ProSim 2000, London , 2000

Jesus, L. and Carapuca, R., "Automatic Generation of Documentation for Information Systems", 4th International Conference on Advanced Information Systems Engineering, CAISE, 1992

Johansson, C. and Hall, P. C. M., "Talk to Paula and Peter - They are Experienced", Workshop on Learning Software Organizations, LSO'99, 1999, pp. 171-185

Johnson, W. L. "The KBSA Requirements/Specification Facet: ARIES", 6th Knowledge-Based Software Engineering Conference, KBSE, 1991, pp. 48-56

Kant, E. "Knowledge-Based Support for Scientific Programming", 7th Knowledge-Based Software Engineering Conference, KBSE, 1992, pp. 2-4

Kappe, F., “Hyperwave White Paper - Version 1.2”, Hyperwave Research & Development, 1999, http://www.hyperwave.com/

Kellner, Madachy, and Raffo, "Software Process Modeling and Simulation: Why, What, How," Journal of Systems and Software, Vol. 46, No. 2/3, 1999

Lawton, G., "Knowledge Management: Ready for Prime Time?," IEEE Computer, Vol. 34, No. 2, 2001, pp. 12-14

Lindvall, M., "Are Large C++ Classes Change-Prone? An Empirical Investigation," Software --- Practice and Experience, Vol. 28, No. 15, 1998, pp. 1551-1558

Lindvall, M., Frey, M., Costa, P., and Tesoriero, R. "An Analysis of Three Experience Bases", Althoff, Klaus-Dieter, Feldmann, R. L., and Muller, Wolfgang, Springer Verlag, Workshop on Learning Software Organizations, LSO'01, 2001, pp. 106-119

Lindvall, M., Rus, I., Jammalamadaka, R., and Thakker, R., “Software Tools for Knowledge Management”, Data and Analysis Center for Software (DACS), 2001

Lindvall, M. and Rus, I., "Process Diversity in Software Development Processes," IEEE Software, Vol. 17, No. 4, Aug 2000, pp. 14-71

Lindvall, M. and Sandahl, K., "Practical Implications of Traceability," Software --- Practice and Experience, Vol. 26, No. 10, 1996, pp. 1161-1180

Lyu, M., “Handbook of Software Reliability Engineering”, IEEE Computer Society Press, McGraw Hill, 1996

Majidi, M. and Redmiles, D., "A Knowledge-Based Interface to Promote Software Understanding", 6th Knowledge-Based Software Engineering Conference, KBSE, 1991, pp. 178-185

Mark, W. "Comet", 7th Knowledge-Based Software Engineering Conference, KBSE, 1992, pp. 254-255

McGarry, F., et.al., “Software Process Improvement in the NASA Software Engineering Laboratory”, CMU/SEI-95-TR-22, Department of Computer Science, University of Maryland, College Park, MD 20742, 1994

Merriam-Webster, O., “The Language Center, Merriam-Webster's Online Dictionary & Thesaurus”, http://www.m-w.com/, 2001

Moody, D. "Using Knowledge Management and the Internet to Support Evidence Based Practice", 10th Australasian Conference on Information Systems, Wellington, NZ, 1999, pp. 660-676

Nonaka, I. and Takeuchi, H., “The Knowledge Creating Company”, Oxford University Press, 1995

O'Leary, D. E., "Enterprise Knowledge Management," IEEE Computer, Vol. 31, No. 3, 1998, pp. 54-61

Paulk, M. C., Weber, C. V., Curtis, B., and Chrissis, M. B., “The Capability Maturity Model for Software: Guidelines for Improving the Software Process”, Addison--Wesley, 1995

Perry, D. E., Staudenmayer, N., and Votta, L., "People, Organizations, and Process Improvement," IEEE Software, Vol. 11, No. 4, July 1994, pp. 36-45

Peters, R. F., "Information Partnerships: Marketing Opportunities for Information Professionals", Information Outlook, Vol. 1, No. 3, 1997, pp. 14-16

Potts, C. and Bruns, G., "Recording the Reasons for Design Decisions", 10th International Conference on Software Engineering, ICSE, 1988, pp. 418-427

Putnam, L. H., “Measures for Excellence - Reliable Software on Time, Within Budget”, Prentice Hall, 1992

Reubenstein, H., "The Requirements Apprentice: Automated Assistance for Requirements Acquisition," IEEE Transactions on Software Engineering, 1991, pp. 226-240

Rochkind, M. J., "The Source Code Control System," IEEE Transactions on Software Engineering, Vol. 1, No. 4, 1975, pp. 364-370

Rus Ioana, Collofello, J., and Lakey, P., "Software Process Simulation for Reliability Management", Journal of Systems and Software, Vol. 46, No. 2/3, April 1999, pp. 173-182

Schneider, K., "LIDS: A Light-weight Approach to Experience Elicitation and Reuse", Springer, The Profes 2000 Conference, Oulo, Finland, 2000, pp. 407-424

Schneider, K., "Experience Magnets - Attracting Experiences, Not Just Storing Them", Product Focused Software Process Improvement, PrOFES'01, Kaiserslautern, Germany, 2001, pp. 126-140

Schoen, E., "Active Assistance for Domain Modeling", *In Proceedings of* 6th Knowledge-Based Software Engineering Conference, KBSE, pp. 26-35,1991.

Senge, P. M., “The Fifth Discipline. The Art and Practice of the Learning Organization”, Doubleday Currency, 1990

Smith, D., "KIDS: A Knowledge-Based Software Development System," Automating Software Design, MIT Press, 1991, pp. 483-514

Smith, T. and Setliff, D., "Knowledge-Based Constraint-Driven Software Synthesis", 7th Knowledge-Based Software Engineering Conference, KBSE, 1992, pp. 18-27

Soloway, E., "I Can't Tell What in the Code Implements What in the Specs", The Second International Conference on Human-Computer Interaction, 1987, pp. 317-328

Sveiby, K. E., “What is Knowledge Management - Working Paper”, Sveiby Knowledge Management, 1996, http://www.sveiby.com.au/KnowledgeManagement.html

Tan, S., Teo, H.-H., Tan, B., and Wei, K.-K., "Developing a Preliminary Framework for Knowledge Management in Organizations", Fourth Annual Americas Conference on Information Systems, Baltimore (Maryland), United States, 1998, pp. 629-631

Terveen, L. G., "Living Design Memory: Framework, Implementation, Lessons Learned," Human-Computer Interaction, Vol. 10, No. 1, 1995, pp. 1-37

Terveen, L. G., Sefridge, P. G., and Long, M. D., "From ‘Folklore’ to ‘Living Design Memory’", ACM 1993 Conference on Human Factors in Computing Systems, 1993, pp. 15-22

Tiwana, A., “The Knowledge Management Toolkit: Practical Techniques for Building a Knowledge Management System”, Prentice Hall PTR, 2000

Vlietinck, E., “How to Build a Culture of Knowledge Management, ITWeek Online, 1999, http://www.zdnet.co.uk/itweek/brief/1999/26/management/

Wieser, E., Houdek, F., and Schneider, K., "Systematic Experience Transfer - Three Cases from the Cognitive Point of View", International Conference on Product Focused Software Process Improvement, PROFES 99, 1999

Wiig, K., “Comprehensive Knowledge Management - Working Paper”, Knowledge Research Institute, Inc., 1999, http://www.knowledgeresearch.com/downloads/compreh_km.pdf

Wood, D. and Krut, R., “Evaluation of Process Modeling Improvements”, SEI/CMU Technical Report, 1991

Ye, Y., "An Active and Adaptive Reuse Repository System", CD – ROM [Electronic Edition], 34th Hawaii International Conference on System Sciences, Software Technology Track, 2001

Yu, B. "LARGE Software System Maintenance", 6th Knowledge-Based Software Engineering Conference, KBSE, 1991, pp. 171-177

Zeroul, K., "KBRAS: A Knowledge-based Requirements Acquisition System", 6th Knowledge-Based Software Engineering Conference, KBSE, 1991, pp. 38-47