2007

Making Knowledge Sharing Visible in Software Engineering

S. Vesiluoma
Teleca AB, sari.vesiluoma@teleca.com

MAKING KNOWLEDGE SHARING VISIBLE IN SOFTWARE ENGINEERING

Vesiluoma, Sari, Teleca AB, Torikatu 13 G, 60101 Seinäjoki, Finland,
sari.vesiluoma@teleca.com

Abstract

Most of the problems in software engineering projects seem to have their origin in knowledge sharing difficulties. While software engineering is one of the most knowledge intensive professions, the understanding of the role of knowledge sharing is vital. It requires making knowledge and knowledge sharing more visible than those currently are in practice. Making visible means here to identify what kinds of knowledge sharing there exists and especially what kinds of problems there are in knowledge sharing. This study produced a Knowledge Sharing Framework (KSF) to help making knowledge sharing visible in software engineering work. The KSF was utilized in an empirical case study. The resulting findings can be used for improving knowledge sharing in the organization.

Keywords: knowledge management, knowledge sharing, IS development, software engineering.

1 INTRODUCTION

Knowledge management has been studied extensively and some research has also been undertaken in the context of software engineering (e.g. IEEE 2002 and Aurum et al. 2003). Software engineering is one of the most knowledge intensive professions (Handzic 2003). Due to inadequate knowledge management, many software development organizations face problems identifying the content, location, and use of their knowledge (Rus and Lindvall 2002). Improved use of knowledge is the basic motivation for applying knowledge management in software engineering. As noted by Turner and Makhija (2006), the issues associated with knowledge *sharing* have received little systematic attention compared to the knowledge *acquisition* issues. Here, the focus has been on knowledge sharing.

To improve knowledge sharing in software engineering, we first need to understand the current situation. Humphrey (e.g. 1998), uses the term "unfreezing" for actions meant to make existing problems visible. Through understanding, or unfreezing, the motivation is created for improvement activities. This study aims at unfreezing, i.e. at understanding the role of knowledge sharing in software engineering and especially to support identifying possible problems in knowledge sharing.

This research has its origins in problems identified by the author in her work over several years as a process improvement specialist in software development companies. One part of this work has been the close study of several software engineering projects while they have been having problems. After analyzing these, it became apparent that knowledge sharing difficulties are a root cause for many of the problems encountered, emphasizing the need to understand more about the existing knowledge sharing and how it could be systematically improved.

A new Knowledge Sharing Framework (KSF) is introduced in this paper. It has been developed to make knowledge sharing more visible, i.e. to look what kind of knowledge sharing there exists or should exist, and what kind of problems exist related to knowledge sharing. This framework is intended to be used by organizations in the software engineering project business. Very often the presented models of knowledge management or knowledge sharing are too general for immediate practical usage (Reifer 2002) or tool/technology oriented (e.g. Tiwana and Ramesh 2001 or Dingsøyr and Conradi 2003). The knowledge sharing models introduced in software engineering (e.g. Oliver et al. 2003) have likewise a rather general character, failing to describe where knowledge sharing should be implemented. Some introduced models, like Experience Factory (Basili et al. 1994), are too specific for creating more holistic understanding about the nature of knowledge sharing in software engineering.

Empirical data was required to develop understanding of knowledge sharing in practice, so case study was selected as the research method. This has been a theory testing case study, where the KSF was firstly introduced based on literature and refined based on the case study. The phases defined by Eisenhardt (1989) were used in the case study.

Knowledge is defined here as something people justify as true (Nonaka 1994), is useful for them in a certain context (i.e. situation dependent, Blackler 1995) and has both tacit and explicit[1] dimension (Polanyi 1966). Also here it is recognized that the knowledge possessed by a person, group or organization develops continuously (Blackler 1995) based on the new knowledge gained.

Sveiby and Lloyd (1987) introduce two main types of knowledge (their original term: knowhow) in knowledge intensive companies. Those are professional knowledge and managerial knowledge. With professional knowledge they refer to the core knowledge of the knowledge intensive company. The success of the company depends ultimately on the abilities of its professionals. In addition to

[1] Explicit knowledge is codified knowledge which can be transferred in the form of a systematic language. Tacit knowledge then has a personal quality and is deeply rooted in action, commitment and involvement in a specific context. (Polanyi 1966).

professional knowledge, managerial knowledge is needed. They define managerial knowledge to be the “ability to preserve and enhance the company's value”.

To define what knowledge sharing means, the theory of Boland and Tenkasi (1995) is used here. They define that a knowledge intensive company relies on multiple specialties and knowledge disciplines to achieve their objectives. A community of knowing is defined to be a community of specialized knowledge workers. A knowledge intensive company consists of several of these overlapping communities of knowing.

The basis for knowledge sharing within and between communities of knowing is the process of perspective making and perspective taking. With perspective making, Boland and Tenkasi (1995) mean the process where a community of knowing develops and strengthens its own knowledge domain and practices. With perspective taking, they mean the communication required for taking the knowledge of other communities into account. In this study, knowledge sharing is defined to be this process of perspective making and perspective taking.

Brown and Duguid (2001) have found “sticky” and “leaky” types of definitions of knowledge in literature. Sticky discussions focus on the challenge of moving knowledge inside organizations. Leaky discussions concentrate on the external and undesirable flow of knowledge, especially the loss of knowledge across the boundaries of a firm to competitors. Here, the approach is to understand where the knowledge is too sticky, not shared as it should, and where the knowledge sharing works well. It would also be important to study why some knowledge is not shared but that is out of the scope here.

Knowledge management covers also other processes than knowledge sharing. Turner and Makhija (2006) define four critical stages of management of an organization’s knowledge. Those are: 1) knowledge creation and acquisition, 2) the transfer of knowledge to other individuals or organizational units, 3) the interpretation of this knowledge in a manner conductive to the objectives of the organization, and 4) application of the knowledge toward organizational goals. Knowledge sharing refers here especially to the second stage, the transfer of knowledge. However, it can’t and shouldn’t be fully separated from the other stages. Davenport and Prusak (1998) define three knowledge processes: generation, codification and transfer of knowledge. Knowledge sharing includes elements of these all, but mostly knowledge codification and transfer. Codification is at place to make perspective making possible. Perspective taking could then be seen as knowledge transfer, but emphasizing that it is not only objective transfer of something but it requires a human process when receiving.

The rest of this paper is organized as follows; Section 2 introduces the Knowledge Sharing Framework. The case study and the findings are presented in section 3. Matching between case study results and the KSF are discussed in section 4. Section 5 provides the conclusions and ideas for future work.

2 FRAMEWORK FOR KNOWLEDGE SHARING IN SOFTWARE ENGINEERING

The Knowledge Sharing Framework (KSF) is introduced here first through its dimensions and then summarized as a conceptual framework.

2.1 Knowledge Sharing Interface

Knowledge sharing, the process of perspective making and taking, is studied here in a software engineering environment. Most software engineering work is organized as projects and is implemented in project companies. A project company consists of company structures and project structures (Artto 1998). A project company has, according to Gareis (1996), a lean base organization and a variable portfolio of projects. In Figure 1, the structure of a project company is described. It is

based on the general dichotomy: base organization and projects. The base organization consists of line management, resource pool and support functions. Based on the situation, a variable number of projects exist in different phases of their life cycle. Every project has its own project team, which consists of persons from the company's resource pool and perhaps from suppliers' and/or customer's personnel.

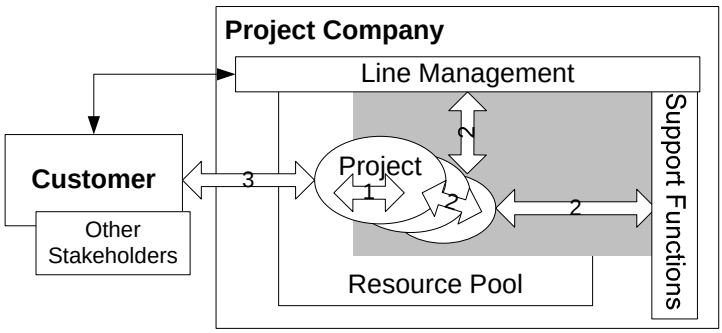

Figure 1. Structure of a project company.

Software engineering work requires many stakeholders. A stakeholder is a group or individual affected by the achievement of organizational goals, and whose own needs must be noticed to satisfy the stakeholder and not to cause difficulties for the organization (Freeman 1984). Knowledge sharing takes place between stakeholders, or with the terminology of Boland and Tenkasi (1995), between communities of knowing. When knowledge sharing happens or should happen between certain stakeholders, here it is defined that there is a knowledge sharing *interface* between these stakeholders.

Most of the knowledge sharing interfaces, in a project company, are visible in Figure 1. The arrows present different knowledge sharing interfaces between different stakeholders. The stakeholders in this context are individuals, project teams, base organization, customers and other stakeholders. Based on these, different knowledge sharing interfaces in software engineering can be defined: 1. between the individuals forming the project team (project team level), 2. between the project team and the rest of the organization including other projects (organization level, grey area in Figure 1), and 3. between companies (a company and its customers, suppliers and other related parties). The interface between companies has been applied here to customer communication.

These three interfaces cover the formal knowledge sharing implied by the used process model. However, in practice, there is a lot of unofficial knowledge sharing between individuals. This means all situations where an individual is sharing knowledge or communicating with another individual, for example, based on friendships. This can happen at any interface. These relationships are hard to control or to effectively take as part of the knowledge sharing structure, but can be supported by creating opportunities for individuals to have free communication. Attempts can also be made to restrict this knowledge sharing, for example, information security restrictions using confidentiality agreements. This could be an example of making an attempt to hinder knowledge “leaking” in the sense of Brown and Duguid (2001).

2.2 Software Engineering Elements

A knowledge sharing interface is defined based on the structure and stakeholders of a project company. However, more software engineering specific approach is still required. One possibility could be to divide software engineering work, for example, according to the traditional waterfall model (Royce 1970) phases. However, that division is the basis for the “traditional” plan-driven software development processes, so some other division was required here, not to limit thinking with the vocabulary of traditional processes. As a result, software engineering work is here divided into

three elements: value adding software engineering, verifying software engineering and managing software engineering.

Value adding and verifying software engineering elements have been introduced by Kylmäkoski (2006). Value adding software engineering is that part of a project where new results are gained and the progress is achieved. It can be understood based on the idea of value added analysis activity in Business Process Improvement (e.g. Harrington 1991). For each activity Harrington (1991) asks whether or not the activity is necessary to produce output. This separates e.g. analysis, design and implementation from quality assurance related activities like reviewing and testing. Value adding software engineering, as it is defined here, must be distinguished from Boehm's Value Based Software Engineering (Boehm 2006) despite the apparent similarity in their names.

Verifying means the activities that are required to assure that the results from the value adding actions are the ones wanted and carrying good quality. Examples of these are inspections/reviews and testing. Actually the word verifying is used here in a broad sense to include some validation related activities too.

In addition to the value adding and verifying element, also an element is required for managing the work. It is required especially because software engineering work is team work requiring coordination. Very seldom a software engineering project can be made by one person only. Because of time pressure and competence requirements in projects, several persons are required to achieve project goals. The work is divided into tasks and the tasks are distributed to different persons. This results in the need for communication and knowledge sharing and requires some kind of managing element.

Value adding engineering and also verifying engineering are largely based on professional knowledge. Sveiby and Lloyd (1987) point out that in addition to professional knowledge managerial knowledge is needed. The managing element is based on this managerial knowledge. Managing element includes activities that continue throughout the software engineering work. Examples of managing elements are project planning, monitoring, and controlling activities.

2.3 Project Lifecycle Dimension

In addition to knowledge sharing interface and software engineering elements, there are other important features of software engineering work to be noticed. In particular, the project lifecycle dimension is essential for understanding knowledge sharing in software development. From knowledge sharing perspective, a project has two very critical phases regarding knowledge sharing, the beginning of a project and the closure of a project. In the beginning (here called establishment) it is critical to make sure that all required knowledge is adequately available for a project in the form of people, documents, requirements etc. At the project closure, the challenge is to assure that the knowledge gained in the project is properly shared in the organization to the parties needing it or storing otherwise so that it is accessible when required. To complete the project lifecycle, also the project realization phase must be included. The resulting three phases are establishment, realization and closure.

2.4 Knowledge Sharing Framework (KSF)

The Knowledge Sharing Framework (KSF) utilizes the interfaces and elements described in the previous chapters. A summary of the dimensions of the framework is introduced in Table 1. The basic part of the KSF consists of the dimension I, knowledge sharing interfaces. Those are: knowledge sharing in a project team, in an organization, in the customer and supplier relationship, and the unofficial knowledge sharing. The element I2 has been further divided to knowledge sharing between current projects, from previous to future projects and between a project and the organization where it is undertaken.

Dimensions	Interface (I)	Software Engineering (S)	Project Life Cycle (L)
Elements	I1 In project team I2 In organization I2.1 Btw current projects I2.2 From previous to future projects I2.3 Btw a project and the base organization I3 In customer supplier relationship I4 Unofficial knowledge sharing	S1 Managing S2 Value adding S3 Verifying	L1 Establishment L2 Realization L3 Closure

Table 1. The dimensions of the Knowledge Sharing Framework

Dimension S is based on the software engineering elements: managing, value adding and verifying software engineering. Dimension L then follows the project life cycle, as discussed above.

3 CASE STUDY AND FINDINGS

3.1 Research Method and Background

To systematically collect evidence of the knowledge sharing reality in software engineering, a case study was implemented in a Finnish telecom software company. The work in the company is organized as projects, so in this study the unit of analysis has been a project. The projects in this company are normally undertaken for external customers.

Two software development projects were selected as cases. Both projects introduced new technologies for mobile phones. Projects were studied nearly at the end of their life cycles (before maintenance phase), meaning that there was a large number of different kinds of material to study. The key people in the selected projects were interviewed. The projects are referred to here as Alpha and Beta.

Main data collection method has been semi-structured interviews. For project Alpha there were four interviews and for project Beta two. Project Alpha had 19 participants, the core team being about 10 persons. The project team in project Beta was six persons. The interviews started with general questions targeting to find out the key successes and problems. After that, the questions were prepared to follow the KSF dimensions and elements. Based on the KSF, a three dimensional matrix was made. For each cell of the matrix, the related software engineering project activities requiring knowledge sharing were identified. Questions were made so, that the knowledge sharing (or lack of it) related to the activities should be identified. Because the amount of time available for an interview was limited, some cells in the matrix had to be ignored. The result was 30 questions, some of those covering more than one cell in the matrix. In addition to those questions, there were questions clarifying the project background etc.

The interviews were typed word by word and possible findings searched and categorized. When more than one source supported the possible findings, those were defined as findings. In addition to the interviews, existing project material (project plans, progress reports, final reports etc.) and observations have been used. Validation of results is made through triangulation and through confirming the findings together with some interviewees and line managers.

3.2 Project Alpha

3.2.1 Overview of the Results

Even though the project Alpha produced the results required, it confronted several problems during its life time. 15 problem areas were found, as introduced in Figure 2. Problems have a consequential nature; some problems could be defined as root causes causing some other problems later. The arrows in Figure 2 describe the consequential relations between the problems. To find out the actual root causes the whole environment should be studied, but here the analysis is limited to the project and its close environment.

Two problems have been defined as the key problems in this case. Those are: P1 Weak Project Definition and P2 Resourcing with Inexperienced Resources. Those are selected as key problems, because they are root causes (several other problems result from these) and the other three root causes (P3-P5) have been minor problems compared to these two.

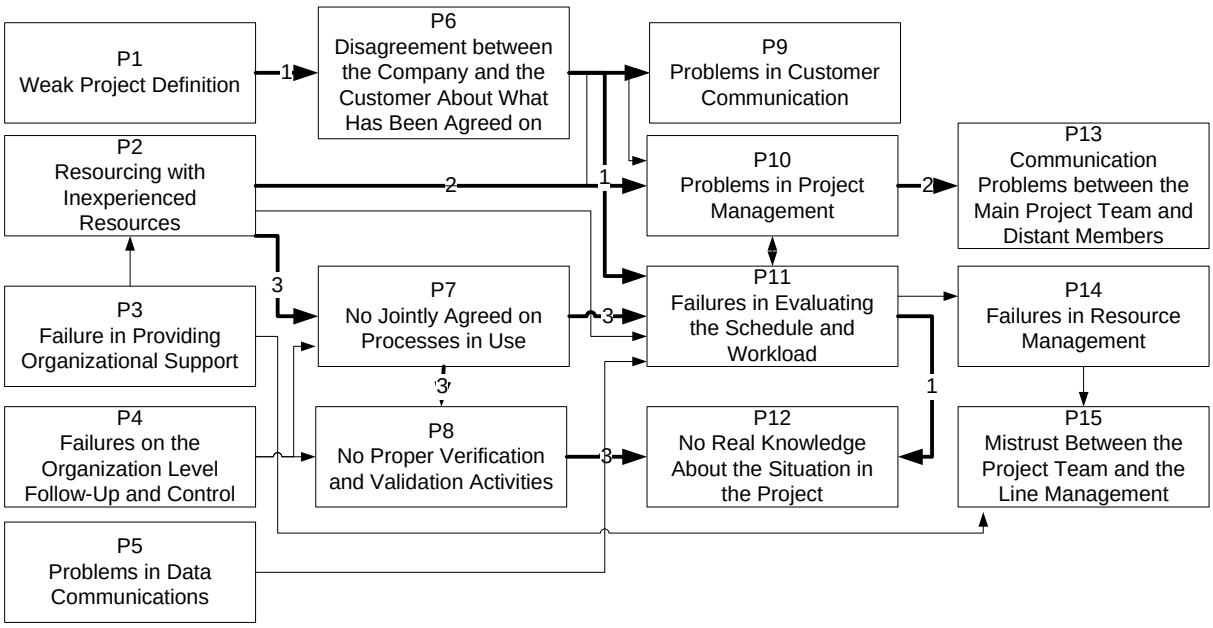

Figure 2. Results from the problem analysis of the project Alpha. Numbered arrows show the main problems and their consequences.

It can be seen that problem P3, Failure in Providing Organizational Support, has an arrow to problem P2. However, problem P3 is not identified as a root cause for problem P2 because it is a problem that together with P2 makes the inexperience more visible.

All elements of KSF were identified in project Alpha and the use of the framework resulted the findings presented here and several other findings, too. The most difficulties were related to the elements I2 knowledge sharing in organization and the I3 customer supplier relationship. Among these the areas of S1 managing software engineering and S3 verifying were the most critical. Problems in these resulted also problems in the area of I1 knowledge sharing in project team. The element S2 (value adding software engineering) seemed to work well, producing acceptable results, even though the problems in other elements affected it (delays etc.). The main problems originated at the project establishment phase (element L1) and affected throughout the project. At the closure phase (element L3) most of the problems had been solved and also many preventive actions were initiated to avoid these kinds of problems later.

3.2.2 Findings per KSF Element

I1 element (knowledge sharing in project team) was implemented in a very informal way. In many situations it was nearer to the type I4 (unofficial knowledge sharing). The project team felt that they had very good spirit except those distant members from the other site. However, because of not having systematic ways of knowledge sharing, the team members felt very unsure.

The knowledge sharing in the organization (I2 element) was very unsuccessful. There was a considerable lack of knowledge sharing between the right people and some of the requests for help were not taken seriously or there just were not people to help the project. Problem P2, Resourcing with Inexperienced Resources, could be an example of the findings related to the I2 element. When the project was established, the project team consisted of very new recruits in the company. Based on the understanding of the line management there were both experienced people and some less experienced people in the team. So the situation was reasonable from the management point of view. The actual problem was, however, that the experience was not exactly the type required in the project. At that time it was also impossible to release experts from other projects (P3) to support this project.

Customer communication (I3 element) started well, but after the misunderstandings appeared between the customer and the company it did not work any more. As an example of the problems in I3, one of the interviewees described the weak project definition (P1) as "The analysis for requirements specification were not detailed enough but was one that we could live with. What was missing was what the project itself should be." This meant, in practice, that the required quality level of the results was not stated clearly enough. The project definition should have been the result of the common understanding between the customer and the company. However, the parties had different understanding about the desired results. The weak or very general project definition covered the understanding of both parties, because it did not identify those parts where the differences were.

Problem P1 resulted in several other problems during the project. Firstly, it resulted in the disagreement between the customer and the company about what had been agreed on (P6). This, in consequence, resulted in e.g. problems in customer communication (P9) and failures in evaluating the schedule and work load (P11). Finally it meant that there was no real knowledge about the situation in the project (P12).

When using the S dimension of the KSF, the S1 element (knowledge sharing in managing software engineering) did not exist in a systematic way. S2 (value adding software engineering), however, worked quite well when ignoring the lack of systematic routines and the delays. Actually, the customer has even praised the architectural solutions. The last element S3 (verifying software engineering) did not exist in a systematic way.

Because of the inexperience, there were several problems in the project management (P10). In the beginning there were no systematic enough ways to define and divide tasks, to follow the project progress etc. Also three of the four interviewees said that everybody knew that things are going wrong but nobody really had the courage to raise this issue. This was not only at the project team level but also at the organization level. The main reason given for not interfering was that the project was "two weeks from ready" for a long time during its life time.

Part of the project team was on another site. Because the project was not coordinated well enough (P10) it was very difficult to manage the resources that were at the distant site of the project (P13). This cooperation between sites was strongly criticized on both sides and lack of many kinds of knowledge sharing between the persons on the main site and the distant site was evident.

In the beginning of the project, mostly, because of the inexperience (P2) but also partly because of the organizational follow-up failures (P4), the team and especially the project manager did not really understand the purpose of commonly agreed on processes or work flows. The project manager said "Actually processes make me feel sick. It suits better to others to do everything according to the predefined instructions." Not following the predefined processes the project had a lack of systematic

project planning and monitoring (P11) and verification and validation activities (P8), which are an example of element S3 (verifying software engineering). Of course, some reviews and testing existed in the project but those were not very well planned, scheduled and resourced. Again, these all resulted in a lack of really knowing the situation in the project (P12). Problem P8 (No Proper Verification and Validation Activities) meant that with some persons there were no ways to assure that right things were done right way and that results were usable. There were, for example, situations where a person was leaving the project and the results achieved proved to be far from what was expected when these became visible to the other team members.

3.3 Project Beta

Project Beta was a very successful project. The main problems were the defects in the development platform hardware and the availability of software components produced under third party intellectual property rights. The corrective and preventive actions for both problems actually promoted more knowledge sharing. The hardware problem initiated more communication between the customer and the project manager. The availability problem resulted in a new checkpoint on the project checklist which was subsequently used in all projects. This is an example of I2, sharing knowledge in organization, and especially I2.2, from previous to future projects.

In the project Beta the knowledge sharing in the project team (I1 element) seemed to be well implemented. Especially the interviewees thought the weekly formal project meetings (example of S1, managing software engineering, element) were very useful and the task pool system, that was used in assigning tasks worked nicely. The tasks in the task pool were planned together in the team and managed by the project manager. At the weekly meetings the situation with the current task was checked and new tasks assigned when needed. Unofficial knowledge sharing (I4) was encouraged, for example, through having all the team members in one room together with another team including the expert named for this project.

Knowledge sharing in the organization (I2 element) also seemed to work well in this project. A nominated expert was available when required and also sat in the same room as the project team. One line manager had been named as a Project Director and he also acted very actively in this role. A Quality Assurance Coordinator was named for the project and supported the project manager in quality assurance related topics. These are examples of I2.3 element, knowledge sharing between a project and the base organization. The naming of a Quality Assurance Coordinator was actually one result of the preventive actions initiated based on project Alpha and thus an example of knowledge sharing from previous to future projects (I2.2).

I2.1 (knowledge sharing between current projects) was visible in the contacts with at least two other project teams to learn more about their experiences among certain technology areas. I2.2 element, knowledge sharing from previous to future projects, was visible here so that the project was fully based on the previous project and there were also possibilities to have the next project based on the current one. The resulting designs, code etc. from this project could not be directly utilized in other projects, but the experiences and new piloted ways of working were possible to share.

Knowledge sharing between the customer and the company (element I3) worked very well. What was special here was that this relationship was very open and strong. The difficult issues were quite easy to discuss between the customer and the company. The communications included the managing element (S1) in steering groups and the value adding element (S2) in weekly technical meetings. The verifying element (S3) came from the joint reviews etc.

The interview of one software engineer produced the understanding of one set of differences between projects from the knowledge sharing perspective. It was related to the element I3, knowledge sharing in customer supplier relationship. This interviewee had been used to projects where nearly all members of the project team participated in the customer communication. Here, however, the

communications model was project manager centric where the project manager is nearly the only person in the project team talking directly with the customer.

In project Beta a question, common with the project Alpha, emerged: how to know that a software engineer producing code does progress as expected and does produce results of the required quality. This is one place where more knowledge is required, whether it is the knowledge that a certain engineer can be trusted (tacit knowledge based on experience) or to have knowledge of ways in which to check the situation in time, before the opportunities for corrective actions are lost. In project Alpha this question was the result of not having systematic enough verification and validation activities (problem in element S3 verifying). In project Beta this was not the case. Project Beta had quite reasonable and systematic verification and validation procedures in use. However, those were not always enough to follow the progress of single project team members (element S1 managing software engineering).

At this case, the project life cycle dimension L was mostly visible in the well succeeded introduction of the project to the project team members (L1 project establishment).

4 MATCHING THE KSF WITH THE CASE STUDY RESULTS

These two cases, together with the KSF, made the knowledge sharing and also the lack of it very visible. Matching of KSF with the findings, the KSF profiles, are introduced in Figure 3. The interface (I) elements are as columns and the lifecycle (L) elements as rows that are further divided into sub rows for software engineering elements (S). The changes between lifecycle (L) element rows indicate changes over time.

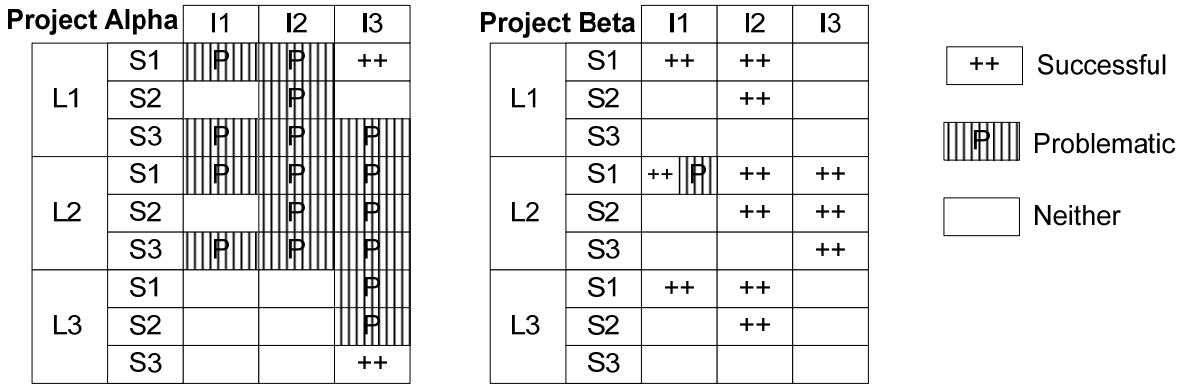

Figure 3 KSF profiles.

As can be seen, the resulting profiles are very different, reflecting the different level of success in projects. The identified problem areas indicate possible improvement areas, especially if those are visible in other projects also. Identified successes then are possible best practices that should be studied and shared with other projects. For example, many of the good practices of project Beta could have really much helped project Alpha. In project Beta, some preventive activities were taken based on the first lessons from project Alpha.

In addition to the cases introduced in this paper, a web survey has been implemented covering other projects in the company. When describing results from those with KSF profiles, those will look something between these two examples having mostly successes but also some problems. This kind of KSF profiles can be exploited in an organization to understand the level of knowledge sharing practices in projects. Project specific profiles can be further summarized into organizational profiles, giving an overview of potential improvement targets and strengths in the organization.

5 CONCLUSIONS

There are deficiencies in connecting academic discussion of knowledge management to software engineering work and processes. However, this is a very relevant question both in scientific and in practical perspectives since significant benefits can be achieved e.g. by improving knowledge sharing in software engineering work. Luckily, recently this has been understood and work for utilizing knowledge management in software engineering has been started. The purpose of this paper is to contribute to that work by defining a method for understanding the knowledge sharing situation and by mapping out the problem areas of knowledge sharing in software engineering.

This paper introduces a framework that can be used to make knowledge sharing more visible and enhance knowledge sharing. The previous models and frameworks for knowledge management and sharing in software engineering have been either very general/strategic ones or specifically concentrating on certain knowledge sharing situations. The KSF comes in between these by defining when and where knowledge sharing should take place.

Seeking efficiency is important in nowadays software engineering where the competition is global and general process best practices (e.g. CMMI, Chrissis et al. 2003) are established. To find a competitive edge the old ideas are not adequate enough. Here, it is suggested that such an edge could be found from improving knowledge sharing in software engineering. Even though the KSF is mostly used here as a descriptive framework making knowledge sharing visible, it also has normative value. The KSF defines situations where knowledge sharing should exist and thus can be used to notice the possible lack of knowledge sharing.

One of the contributions to practical software engineering work is the set of found problems and the understanding of the importance to support knowledge sharing in software engineering. Utilizing the KSF to make KSF profiles of software engineering projects and, as a summary, of the whole organization is a powerful tool to identify strengths and weaknesses. Through identifying and making the problems visible also corrective and preventive actions can be initiated.

A limitation of this study is that only two cases were studied within this framework. However, it showed that the framework is applicable and useful in making knowledge sharing visible. Also the resulting findings are not case specific ones, but could be expected to appear in other projects and organizations, too.

The introduced knowledge sharing framework and the understanding of the importance of knowledge sharing are a good basis to continue to find ways to improve knowledge sharing. This includes finding out techniques to document systematically best practices that are known to improve knowledge sharing in software engineering work. The current work of the author includes definition of Knowledge Sharing Patterns (Vesiluoma 2006) to serve in this purpose. The KSF has been used both as a method to identify situations requiring support and also as one basis for structuring the created Knowledge Sharing Pattern Language. This work continues.

Acknowledgements

This work has been supported by the Academy of Finland under grant 112068 and Tampere Graduate School in Information Science and Engineering (TISE).

References

Artto, K.A. (1998). A Management Framework for a Project Company. In proceedings of the 14th IPMA World Congress on Project Management, Ljubljana, Slovenia, 677-681.

Aurum, A., Jeffery, R., Wohlin, C. and Handzic, M. (Eds.) (2003). Managing Software Engineering Knowledge. Springer-Verlag, Berlin, Heidelberg.

Basili, V.R., Caldiera, G. and Rombach, H.D. (1994). Experience Factory. In Marciniak, J.J. (Ed) Encyclopedia of Software Engineering. John Wiley and Sons, UK, 469-476.
Blackler, F. (1995). Knowledge, Knowledge Work and Organizations: An Overview and Interpretation. Organization Studies, 16 (6), 1021-1046.
Boehm, B. (2006). Value-Based Software Engineering: Overview and Agenda. In book Biffl, S., Aurum, A., Boehm, B., Erdogmus, H. and Grünbacher, P. (Eds.) Value-Based Software Engineering. Springer
Boland, R.J. and Tenkasi, R.V. (1995). Perspective Making and Perspective Taking in Communities of Knowing. Organization Science, 6 (4), 350-372.
Brown, J.S. and Duguid, P. (2001). Knowledge and Organization: A Social-Practice Perspective. Organization Science, 12 (2), 198-213.
Chrissis, M.B., Konrad, M. and Shrum, S. (2003). CMMI - Guidelines for Process Integration and Product Improvement. SEI Series in Software Engineering. Addison-Wesley.
Davenport, T.H. and Prusak, L. (1998). Working Knowledge – How Organizations Manage What They Know. Harward Business School Press, Boston, Massachusetts.
Dingsøyr, T. and Conradi, R. (2003). Usage of Intranet Tools for Knowledge Management in a Medium-Sized Software Consulting Company. In book Aurum et al. (2003), 49-68.
Eisenhardt, K.M. (1989). Building Theories From Case Study Research. The Academy of Management Review, 14 (4), 532-550.
Freeman, R.E. (1984). Strategic Management: A Stakeholder Approach. Harper-Collins, Boston, MA.
Gareis, R. (1996). The Application of the "New Management Paradigm" in the Project-Oriented Company. In Proceedings of IPMA´96 World Congress on Project Management, Paris, France, 687-689.
Handzic, M. (2003). Why is it Important to Manage Knowledge? In book Aurum et al. (2003), 1-4.
Harrington, J. (1991). Business Process Improvement – the Breakthrough Strategy for Total Quality, Productivity and Competitiveness. McGraw-Hill Inc.
Humphrey, W.S. (1998). Foreword for the book: Zahran, S. Software Process Improvement, Practical Guidelines for Business Success. Addison-Wesley, xiii-xiv.
IEEE (2002). A special section on knowledge management in software engineering (several articles). IEEE Software, 19 (3), 14-15, 26-62.
Kylmäkoski, R. (2006). RaPiD7: A Collaborative Method for the Planning Activities in Software Engineering – Industrial Experiments. PhD thesis. Tampere University of Technology, Publication 582, Tampere, Finland.
Nonaka, I. (1994). A Dynamic Theory of Organizational Knowledge Creation. Organization Science, 5 (1), 14-37.
Oliver, G.R., D'Ambra, J. and Van Toorn, C. (2003). Evaluating an Approach to Sharing Software Engineering Knowledge to Facilitate Learning. In book Aurum et al. (2003), 119-134.
Polanyi, M. (1966). The Tacit Dimension. Routledge & Kegan Paul, London.
Reifer, D.J. (2002). A Little Bit of Knowledge is a Dangerous Thing. IEEE Software, 19 (3), 14-15.
Royce, W.W. (1970). Managing the Development of Large Software Systems. In proceedings of IEEE Wescon, 1-9.
Rus, I. and Lindvall, M. (2002). Knowledge Management in Software Engineering. IEEE Software, 19 (3), 26-38.
Sveiby, K.E. and Lloyd, T. (1987). Managing Knowhow: Add Value by Valuing Creativity. Bloomsbury, London.
Tiwana, A. and Ramesh, B. (2001). Integrating Knowledge on the Web. IEEE Internet Computing, 5 (3), 32-39.
Turner, K.L. and Makhija, M.V. (2006). The Role of Organizational Controls in Managing Knowledge. Academy of Management Review, 31 (1), 197-217.
Vesiluoma, S. (2006). Improving Knowledge Sharing in Software Engineering. International Transactions on Systems Science and Applications. 1(2), 167-173.