4

EXPLICIT AS ENABLER FOR UNDERSTANDING THE TACIT

Anna Börjesson Sandberg

Ericsson AB and IT University of Gothenburg
Gothenburg, Sweden

Carl Magnus Olsson

Software Engineering & Management
IT University of Gothenburg
Gothenburg, Sweden

Abstract

Knowledge that can be expressed in words and numbers (i.e., explicit knowledge) only represents the tip of the iceberg in organizational knowledge. Explicit knowledge may be communicated in formal, systematic language, using standard notations. This paper reviews how explicit knowledge is used within software engineering, and has found that this tip is important to store organizational knowledge, discuss improvements, and communicate competences. We further discuss how the tip can be viewed as a facilitator for understanding tacit knowledge, such as context dependent actions, commitments, and involvement. To support these arguments, we have studied software engineering units within Ericsson AB during the period 2005–2007 to learn more about software engineering process descriptions and notations in practice. Findings indicate that unambiguous process descriptions are dependent on obvious software process notations and that the explicit nature of described software processes serve as a vital catalyst in initiating discussions of tacit knowledge.

Keywords

Software engineering, explicit knowledge, tacit knowledge, software process notations, software process descriptions

Please use the following format when citing this chapter:

Sandberg, A. B., and Olsson, C. M., 2008, in IFIP International Federation for Information Processing, Volume 287, Open IT-Based Innovation: Moving Towards Cooperative IT Transfer and Knowledge Diffusion, eds. León, G., Bernardos, A., Casar, J., Kautz, K., and DeGross, J. (Boston: Springer), pp. 63-82.

1 INTRODUCTION

Dividing knowledge in the categories explicit/tacit or codified/personalized has, since the mid-1990s, been widely accepted through the seminal work of Nonaka and Takeuchi (1995) and Hansen et al. (1999). Explicit knowledge is knowledge that is transmittable in formal, systematic language, while tacit knowledge is knowledge that is rooted in actions, commitment, and involvement in a specific context. For the purposes of this paper, we are listing explicit and tacit knowledge (Nonaka and Takeuchi 1995) as comparable with codified and personalized knowledge (Hansen et al. 1999). Explicit knowledge has previously been argued as representing only the tip of the iceberg of the body of all possible knowledge (Polanyi 1966).

Three process notation techniques dominate today, focusing on activities, objects, or roles (Kung et al. 1996). Activity oriented techniques are supported by BPMN (business process modeling language), object oriented techniques are supported by UML/SPEM 2.0 (unified modeling language/software process engineering meta-model), while role oriented techniques are supported by RAD (role activity diagramming). In our study, we found that none of these have clearly established itself to be state-of-the-art, nor do they have well known, deployed, and successfully diffused tools supporting them (which in itself hinders effective management of explicit knowledge).

This study reports on three cycles of action research at a major development unit within the telecom company Ericsson AB we knew was heavily involved in this issue. Specifically addressed in this paper, we asked ourselves, how do software process descriptions affect understanding of software engineering knowledge? We argue that we cannot describe explicit knowledge in ways where it can be understood by the mass without a standard notation for process descriptions, and, without describing the explicit knowledge, we are hindered from effectively understanding and managing tacit knowledge. Through quantitative studies of process descriptions and qualitative interviews with senior engineers and managers, our research elaborates on the role played by explicit knowledge in forming an understanding of tacit aspects. As explicit knowledge, we have focused on the role software process descriptions, software process notations, and tools supporting formalization of software processes play in our study. Results indicate that the main reasons for having a described software process were storing organizational knowledge, discussing improvements, and communicating competences. The described process was identified as necessary for getting a common view of the current software engineering situation to discuss both improvements and competences for specific contexts. The explicit knowledge was thereby found to be an enabler for managing actions, commitment, and involvement between individuals for specific situations (i.e., for understanding and managing the tacit knowledge). More specifically, we have found that the explicit nature of described software processes serves as a vital catalyst in initiating discussions of tacit knowledge.

We conclude that effectively managing explicit knowledge through commonly described software processes is far more important than just managing the tip of the iceberg of all possible knowledge. Aside from our contributions to process notations, this work also draws on the established concepts of tacit and explicit knowledge—well known and with considerable use primarily within knowledge management—to illustrate that these still have a considerable role to play. In our literature study, we could find no previous linking of software process notations (or models) with these concepts.

We present the theoretical context in section 2 by discussing knowledge management and software process notations. In section 3, we then describe the action research approach taken, and present the findings of each action research cycle in section 4. Discussion and analysis of the results follow in section 5, while the paper's conclusions are summarized in section 6.

2 THEORETICAL CONTEXT

This section explains the two main theoretical parts used in this paper. First, we present basic knowledge management theories and explain how we have made use of them in our study. Second, we present existing software process notations, and research related to them, while reflecting on how the growing interest in globally distributed software development makes our study even more relevant. For both sections, we have included promising future extensions of the context.

2.1 Knowledge Management

Knowledge management is an old management tradition where skills have been passed on in generations from parents to their children within family businesses, and exchanging know-how has been vital to survival. Over time, economies have shifted from natural resources to intellectual assets and managers started in the mid-1990s, to get interested in the concept of managing knowledge. However, as early as the 1960s, Polanyi (1996, p. 4) argued "we can know more than we can tell" and he divided knowledge in two categories. *Explicit knowledge* refers to knowledge that is transmittable in formal, systematic language. Explicit knowledge is more simply explained by saying it is knowledge that can be expressed by words and numbers. This type of knowledge only represents the tip of the iceberg of all possible knowledge. *Tacit knowledge* is rooted in action, commitment, and involvement in a specific context. Nonaka and Takeuchi (1995) made these two categories well known when analyzing Japanese innovation companies, where they proposed four different modes of knowledge conversion: socialization (from tacit to tacit), externalization (from tacit to explicit), combination (from explicit to explicit), and internalization (from explicit to tacit).

The research of Hansen et al. (1999) is also well-cited in the knowledge management area. They argue a company can choose to adopt two different knowledge management strategies, and claim that companies without a knowledge management strategy sooner or later die. Hansen et al. distinguish between *codified* and *personalized knowledge*. Knowledge is, in their view, codified to support a people-to-document strategy (i.e., knowledge is extracted from the person who created it). It is made independent of that person and is reused for a variety of purposes. In contrast, personalized knowledge is knowledge handed person-to-person (i.e., typically knowledge shared in brainstorming sessions and one-on-one conversations). As Hansen et al. argue, "A company's knowledge management strategy should reflect its competitive strategy: how it creates value for customers, how that value supports an economic model, and how the company people deliver on the value and the economics (1999, p. 3).

Mathiassen et al. (2003) argue that both Nonaka and Takeuchi and Hansen et al. make no distinction between tacit and personalized knowledge, or explicit and codified knowledge. For the purposes of this study, we have knowingly elected to do the same as Nonaka and Takeuchi, and Hansen et al., as this paper represents an analysis of how explicit knowledge is understood in an organization. Adding a further level of analysis to our study prior to having established the role explicit knowledge plays in diffusing software management practices seems somewhat premature. Depending on outcomes, Mathiassen et al. could possibly be a useful next iteration in the action research conducted at the site. Somewhat differently from Nonaka and Takeuchi, as well as Hansen et al., Mathiassen et al. argue that codification of knowledge plays a crucial role both in relation to maturing professional practices and in relation to the use of information technology to support knowledge management. It is, therefore, important that people engaged in knowledge management activities appreciate the problems and opportunities related to the codification of knowledge. Mathiassen et al. distinguish between four different types of professional knowledge: situated (personalized and tacit), exemplary (personalized and explicit), community (codified and tacit), and procedural (explicit and codified).

At this time, we do not intend to argue whether or not there is a difference between codified and explicit knowledge, but we acknowledge arguments made to distinguish between them in favor of understanding and appreciating the importance of codified and explicit knowledge within software engineering.

2.2 Software Process Notations in the Context of Distributed Development

Ellmer and Merkl (1996) present strong arguments for the importance of described development processes to be successful as software organization, and describe its role as organizational memory: “A process model is an explicit representation of process knowledge and may thus serve as a means for storing and retrieving organizational knowledge about software process execution” (p. 60). Such a described software process is typically both codified and explicit. Arguments such as those of Ellmer and Merkl have been accepted increasingly in research as well as practice over the last decade. Today, however, we are seeing software development increasingly become a global activity. This move toward distributed work challenges us to revisit our established work practices, and has seen growing academic interest (see the *MIS Quarterly*’s Special Issue on Information Systems Offshoring (32:2), June 2008). Considerable work has been reported on cultural differences, communication, and distributed communities (reviewing a task-technology fit perspective—Maruping and Agarwal 2004; elaborating on political, cultural, and global aspects—Nicholson and Sahay 2001; leveraging time-zone management in distributed work—Carmel 2006). Remarkably little is to be found currently when looking at the role of formal process notations, but Sahay (2003) represents an exception to this. Considering his perspective is on global software alliances of small to medium sized companies, rather than multinational ones such as what our study examines, this paper settles for acknowledging that an interesting comparison could be made in future publications.

Looking specifically at existing process notations, Johansson (2007), as a partial input to this study, found four groups of software process notations that we recall below.

- Activity-oriented process notations supported by business process modeling language (BPMN). These notations focus on the definition of processes as a sequence of activities. Examples of activity-oriented techniques are further discussed by Owen and Raj (2005) and White (2004).
- Object-oriented process notations supported by unified modeling language/software process engineering meta-model (UML/SPEM 2.0). These notations leverage the more comprehensive modeling constructs of object-orientation to capture processes. Object-oriented process notations are further discussed by Haumer (2006), Kalnins and Vitols (2006), and Russel et al. (1994).
- Role-oriented process notations supported by role activity diagramming (RAD). These notations describe business processes based on the specific organizational roles and responsibilities involved. These techniques are discussed by Ould (1995) and Ould and Huckvale (1995).
- Speech-act oriented process notations (not supported by any identified process notation language). The speech-act oriented notations view processes in the context of the speech-act language action paradigm. These notations are discussed by Kueng et al. (1996). Considering the speech-act oriented process notation is barely mentioned outside of Kueng et al., we will be limiting our use of it to recognizing its existence, but refrain from further research on it before we bring it into the ongoing study on which this paper reports.

When reflecting on what makes a good software process notation, we recognize that Ould and Huckvale (1995, p. 334) list a number of features that must be met to achieve high-quality process descriptions. Concepts, objects, and relationships represented in the process description should be intuitively familiar, so that people can readily understand and talk about them. The notation should be easy for readers to grasp, after a limited verbal introduction. The notation should be unambiguous (another way of saying that it should have formal syntax and semantics) so that it can be analyzed and, possibly, enacted. It should be possible for the notation to draw attention to the purposes of what people do rather than the detail of how they do it. Finally, the notation should be able to handle complexity.

We also recognize that there are different types of information from which it should be possible to retrieve a process description (Curtis et al. 1992, pp. 77). This includes what is going to be done, who is going to do it, when and where will it be done, how and why will it be done, and who is dependent on it being done. If a described process is used there are, according to Ould and Huckvale, several ways that this model can help organizations to be more effective. It may be used

- *As a focus for discussion.* If we use a good process notation that has a clear syntax and semantics together with process description guidelines, it will help the organization to ask the right questions and bring important points in focus for discussion.
- *As a mean for communicating a process to others.* In this way other people can use the process as a guide and they will save valuable time since they do not have to

develop a new process over and over again. This view is also supported by Curtis et al. (1992).

- *As a basis for analysis.* It is possible through analysis to find weaknesses in the process or to identify if a certain process does not give any value.
- *For designing a new process.* When creating different process descriptions and comparing them, it is possible to find the best solution and to use it.
- *As a baseline for continuing process improvement.* It is possible that the process models can be used to find different measures in how well the process works and in this way initiate improvements to the process. This view is also supported by Curtis et al.

Today there are three acknowledged ways to describe software processes that are supported by a process notation language. We also understand that a high quality process description needs to meet several different requirements. This is the type of process description that we define as a valuable explicit record of the organizational memory.

3 METHOD

This study has the dual goal of both understanding software process notations used in practice and contributing to the body of knowledge on the same theme. Collaborative practice research (CPR) (Mathiassen 2002) supports the realization of this dual goal, while at the same time supporting the insider/outsider perspective (Bartunek and Louis 1996) that has been a beneficial aspect of this research project. One of the two authors has been working within Ericsson with process descriptions, tools for describing processes, try-outs, learning sessions, and data collection during the period 2005–2007 and taken the insider role. The other author joined the research project in the later phases, taking the outsider role and contributing with interview design, interviews, analysis, discussions, and questioning in an unbiased way. This study has also benefitted from the assistance of a master's thesis (Johansson 2007). The data collection design and the research method used (presented below) have helped us to answer the research question: How do software process descriptions affect understanding of software engineering knowledge?

The study is based on action research (Baskerville and Wood-Harper 1996, 1998; Davison et al. 2004; Galliers 1992) with a focus on understanding the current process description situation, the awareness of the current situation, and the value of and use for a described process. Baskerville and Pries-Heje (1999) argue that the fundamental contention of action research is that a complex social process can be studied best by introducing changes into that process and observing the effects of these changes. We collected data throughout the research project in three iterative cycles as defined by Susman and Evered (1978). Each cycle includes the activities diagnosing, action planning, action taking, evaluating, and specifying learning. The activities in respective iteration are summarized in Table 1.

Table 2 summarizes the data sources used in the study. The many different data sources have facilitated triangulation (Patton 1987; Yin 1994) and analysis in an unbiased way.

Table 1. The Three Iterations Explained through Susman and Evered’s (1978) Action Research Phases

Phase	Iteration 1: Process Description Situation	Iteration 2: Process Description Awareness	Iteration 3: Value of and Use for Described Processes
Diagnosing	Need to understand actual process description situation	Need to understand the level of awareness around the process description situation	Need to understand value of and use for process descriptions.
Action Planning	Planned to investigate all existing process descriptions within one major development unit	Planned to present the iteration 1 investigation for selected units and review literature for process notations standards	Planned to interview 12 senior managers and engineers. Questions defined based on learning from iteration 1 and 2.
Action Taking	Searched and found 13 different process descriptions, which were analyzed with help of 10 different attributes (see section 4.1)	Presented the iteration 1 investigation for selected units and reviewed the literature for process notation standards	Interviewed the 12 senior managers and engineers.
Evaluating	Found processes differed in most of the defined attributes	Main concerns for the current process description situation: <ul style="list-style-type: none">• Lack of a tool for process modeling• No standard used when describing processes	Wanted to use the described process to store organizational knowledge, discuss improvements and communicate knowledge and competence.
Specifying Learning	Very varying process descriptions	Process descriptions do not follow any obvious standard process notation	Process descriptions (explicit knowledge) enable tacit knowledge.

Table 2. Data Sources

Data Sources	Explanation
Direct involvement	We were directly involved in investigating process description situation, deploying process situation results, and conducting interviews
Formal interviews	We executed 12 formal interviews with senior managers and engineers of 1 hour each
Literature review	We reviewed the literature to understand what standard process notations existed
Open-ended, semi-structure interviews	We had informal interviews and discussions with practitioners who were users of the process descriptions
Process descriptions	We reviewed and analyzed 13 different formal and approved processes
Process description presentations	We presented, analyzed, and discussed the process description situation with two selected units

The questions used during the formal interviews in iteration three were designed based on understanding gained from both the process description investigation in iteration one and the literature review in iteration two. The table in Appendix A presents the questions used. Extensive use of laddering (Reynolds and Gutman 1988), that is, asking a series of neutral probing questions to stimulate respondents to elaborate and explain their answers, was employed during the questioning.

4 THE CASE

In mid-2005, we started to analyze the use of different process notations for managing explicit software engineering knowledge (i.e., process descriptions used in software development). We wanted to understand what hinders effective management of explicit software engineering knowledge. The investigation was made in three main iterations. When starting iteration one, it was not obvious what iterations two and three would be, or if they would be necessary. Iterations were defined as a result of the previous iteration and a need to understand more about process notations for managing explicit software engineering knowledge. In iteration one, we wanted to understand the actual process description situation for software engineering. In iteration two, we wanted to create awareness within Ericsson about the situation and, by that, receive valuable feedback on how to proceed. The third and last iteration was designed to understand in detail the value of and use for a described process for software engineering. The three following sections describe how the iterations were executed and their results.

4.1 Iteration One: Process Description Situation

To understand the actual process description situation for software engineering, we analyzed a major development unit within Ericsson AB consisting of more than 1,500 employees, geographically distributed over five sites, in two different European countries. First, we searched the Ericsson intranet for approved formal processes for software engineering and then analyzed the different processes based on a number of attributes that would help us understand their quality (see Table 3). Thirteen different approved formal process descriptions were found in the search and formed the basis of the further analysis. It is important to note that these 13 process descriptions covered different parts of the software engineering scope (e.g., products development, software development, systems development, and project management).

The analysis revealed the following situation:

1. There was no obvious structure used for the entry pages. Processes used within the same development unit often had similar entry pages, but not always.
2. Seventy percent the analyzed processes were web-based and 30 percent were document-based. The document-based processes used Word, Frame Maker, Power Point, Excel, or Visio.
3. None of the processes followed specific notation guidelines.
4. The size of the processes varied from 70 to 3,600 links and from 22 to 230 pages.
5. The percentage of broken links varied from 1.3 to 67 percent.
6. The number of templates varied from 0 to 1,500.
7. The number of unavailable templates varied from 0 to 1,050.
8. Almost all (90 percent) of the process had an appointed and known process owner.
9. Seventy percent of the processes had recently (within 6 months) been updated and 20 percent had been updated within the past year.
10. Eighty percent of the processes had a formal revision state.

Table 3. Attributes Used to Analyze the Quality of the Approved Formal Processes

#	Process Description Attribute	Explanation
1	Entry page	The style of the entry page of the process
2	Layout	Web based or document based process description
3	Notation	Following specific process notation guidelines (either home-made or standard)
4	Size	Number of links (web based) or number of pages (document based)
5	Percent of broken links	Number of the links not working compared to number of links in web/document
6	Number of templates	Number of templates
7	Number of unavailable templates	Number of unavailable templates
8	Process owner	If there was an appointed and known owner of the process
9	Process updates	When was the process last updated
10	Approve revision	Has the process a formal revision state

The analysis of the process descriptions revealed a varying situation when it came to appearance, notation, size, and quality. We understood that no formal process notation was used to describe the explicit software engineering knowledge, but we still did not understand how much of a problem this was.

4.2 Iteration Two: Process Description Awareness

At this point, we knew that the process description situation varied and that there was no obvious process notation for managing diffusion of software engineering knowledge. To create awareness about this situation, we made a presentation explaining these facts and communicated it to selected Ericsson units working with improving software engineering and the described processes. Specifically, at the end of 2005 and during 2006, we communicated the result within the Ericsson unit globally responsible for processes, methods, and tools for research and development and within a local Ericsson unit developing a product for the data communication network.

The immediate reaction was that the lack of a tool for process modeling was the main reason for the current process description situation. This conclusion was mainly based on the fact that numerous variants for different layouts (#2, Table 3) were used and, if a tool would have been used, specific notations would be followed (#3, Table 3). The lack of quality (#4 – #7, Table 3) of the described processes was mainly understood as a result of difficult manual maintenance that could be helped by a tool for describing processes.

Both the global and the local unit started initiatives to describe their engineering processes in a tool. They elected to start using rational method composer (RMC). The communication of the process description situation ended with an awareness that helped initiate actions to improve the situation. Within the started initiatives, it very soon became clear that only a tool would not be enough to deal with the current process descrip-

tion situation. Process notations (#3, Table 3) and process modeling guidelines were requested to assist in describing the processes in the tool. This awareness helped manifest an initiative to understand and learn more about process notation languages, starting with researching the literature of known process notations. This was done as a part of Johansson's (2007) master's thesis, which found that there were three different process notation techniques (activity-, object-, and role-oriented) (see section 2.2).

4.3 Iteration Three: Value of and Use for Described Processes

The awareness of the current process description situation, combined with the newly acquired knowledge about the need for and lack of deployed process notations, made us interested in understanding the actual value of and use for a described process. Maybe we would find explanations for the current process situation if we understood how engineers valued a described process and what they wanted to use it for.

We interviewed 12 senior engineers (see section 3 and Appendix A). The interviews revealed both what the interviewees valued as important for a well described process and what they wanted to use it for (Johansson 2007). Figure 1 presents a diagram of what the

Figure 1. What Is Important for a Well Described Process?

interviewees valued as important for a described process. Number in parentheses give a summary of the score, with not important equal to one point and very important equal to six points.

Contrary to what was found in the literature, the interviewees indicated "clearly shown deliverables" as most important for a well described process. As one interviewee argued, *"that deliverables are clearly shown is very important for software process quality. It is in the interfaces between deliverables that product quality is created and the deliverables should be well defined in order to reach high quality in a software processes description."* "Clearly shown deliverables" was followed by "easily understood process flows," "high correspondence to real world," and "easily understood symbols." "Use of standard notation" was valued as more important than not important by two thirds of the interviewees, but it was ranked in eighth place compared to other, more valued attributes. Some of the interviewees saw these standard notations as a form of restriction that might suffocate people creativity if they were too rigorous. On the other side, some of the interviewees saw it as mandatory that there should be a restriction in how to describe or model a specific software process. The attribute that ranked quite high in the literature, "clearly shown roles," was not valued as important at all by the interviewees. One interviewee said, *"Focus on roles often gives negative effect in that often when we start projects we begin with a plan of what roles we need and not what we will do. We don't have the free mind set about what roles are in software processes; instead we lock a role to a certain position and this causes more damage than benefit."* Another interviewee said, *"It is nice that the roles are visible, but most important are activities and that their input and outputs are visible."*

Figure 2 presents a diagram for what the interviewees valued as the important reasons for having described processes. The interviewees wanted to use the described process to "store organizational knowledge," "discuss improvements," and "communicate knowledge and competence." Several of the interviewees saw a close relation between "store organizational knowledge" and "communicate knowledge and competence." Regarding the attribute "discuss improvements," one interviewee said, *"It is first when the software process is described that we can be sure we talk about the same thing. Described software processes also play a role when initiating change [provides a given starting point] and enables comparison with actual events."* Another interviewee, who also agreed about the importance of storing organizational knowledge, said, *"Today we are not so good at storing organizational knowledge; it is boring to read documents created by other people."* Furthermore, "finding weaknesses and problems," "increasing product quality," and "measuring improvements" were valued as important, but with somewhat lower rank than the first three. In addition to these attributes, which are all mentioned in the literature as important attributes for a described process, the interviewees mentioned 11 other attributes.

We finalized the interviews by asking an open question to name tools used for describing processes that the interviewees had come in contact with, used, or heard. The interviewees named 14 different tools. The six most frequently named tools were Rational Method Composer (mention by 11 interviewees), Microsoft PowerPoint (mentioned by 6 interviewees), HWDP (a local variant was mention by 5 interviewees), HTML (mention by 4 interviewees), PLCM (a local variant was mention by 3 interviewees), and Microsoft Visio (mention by 3 interviewees). The interviewees had no common view about what tools to use for describing processes, but they all agreed about

Figure 2. Reasons for Having Described Processes

the importance of having a tool for describing processes. One interviewee said, “Today we have such advanced and complex software processes that we must use a high-quality tool to be able to control them; Microsoft PowerPoint won’t do the job anymore and we cannot do it using free-hand illustrations.” Another interviewee said, “When using Microsoft PowerPoint, [the projects need to give good reviews] in order to find gaps in the deliverables or roles,” and continued by noting that it is also “hard to keep the described software process models and links to the processes [updated].”

5 ANALYSIS AND DISCUSSION

Recollecting the notions of tacit and explicit knowledge, and motivated by the growing work in globally distributed software development, we set out to answer the question, how do software process descriptions affect understanding of software engi-

neering knowledge? We specifically looked to answer the question from the perspective of a globally active organization. In this section, we analyze and discuss our findings. First, we present the process description situation and awareness; second, we present a deeper analysis of the value of and use for described processes. We conclude the section by discussing implications for practice and research.

5.1 Lacking Obvious Standard Software Process Notation

In iteration one, we investigated a major development unit within Ericsson working with software engineering and found 13 different formal and approved process descriptions. These 13 process descriptions were analyzed through 10 different attributes as defined in Table 3. The analysis revealed that, for software engineering, there were no obvious standard software process notations to enable diffusion of knowledge. An ocular inspection of the 13 process description showed that none of the 3 identified software process notations (BPMN, SPEM, or RAD, see section 2.1) were used. Ould and Huckvale (1995) argue for the importance of unambiguous process descriptions in order for them to be useful. Also, the investigated attributes (#4 – #7, Table 3) indicated that no obvious standard software process notation was followed as there was a large difference in size, tool used, and quality.

In iteration two, we presented the result from iteration one and there was one common view on why the process descriptions varied: lack of a tool. This common understanding was likely based on the fact that numerous variants for different layout (#2, Table 3) were used and if a tool would have been used, specific notations would be followed (#3, Table 3). The lack of quality (#4 – #7, Table 3) of the described processes was mainly understood as a result of difficult manual maintenance that could be helped by a tool for describing processes. What was even more interesting for this conclusion was that no one questioned the result. In practice, this is very uncommon. It is likely that this depended on employees already having a gut feeling this was the situation, but there had been no prior attempts to visualize it. It is important to mention that several of the investigated process descriptions (the ones with good quality) were appreciated among users and provided employees with several benefits in the software engineering area.

We conclude that the investigation of the 10 attributes for the 13 process descriptions found, the reactions from employees during the process situation presentation, and the literature survey indicate that today there are no obvious standard software process notations to manage diffusion of explicit knowledge.

Unambiguous process descriptions are dependent on obvious software process notations. Established tools for describing software processes is likely an important enabler of this.

Larsen et al. (2008) study the role of modeling in achieving information systems success. One of their findings argues that the usage of UML is linked to project success. Similar to our study, we see here a close correlation between the two studies regarding the value of explicit knowledge.

This conclusion was provided further support in the third iteration of the action research, as “use of standard notation” was leaning toward needed by two thirds of the

interviewees (see Figure 1). One interviewee argued that a standard notation is still needed, explaining that *“there should be mandatory restrictions on how to describe or model a specific software process.”* At the same time, another interviewee was afraid a standard notation would suffocate people creativity. Our interpretation of this is that while software process notations are needed, the current notations may not provide the obvious support that is needed in all instances. Nevertheless, the importance of having a tool for describing the process was still clearly argued: *“Today we have such advanced and complex software processes that we must use a high-quality tool to be able to control them; Microsoft PowerPoint won’t do the job anymore, and we cannot do it using free-hand illustrations.”*

5.2 Explicit Knowledge Facilitates Diffusion of Tacit Knowledge

Managing knowledge can be analyzed through the two categories, explicit and tacit knowledge (Nonaka and Takeuchi 1995; Polanyi 1966). Explicit knowledge is knowledge that can be expressed with words and numbers (e.g., process descriptions), while tacit knowledge is knowledge that is rooted in actions, commitment and involvement in a specific context (e.g., when designing a software architecture or defining software requirements).

From the interviews in iteration three, we learned that for a process description to be valuable, “clearly shown deliverables,” “easily understood process flows,” “high correspondence to real world,” and “easily understood symbols” were highly important. All of these but clearly shown deliverables are also recognized by the literature to be of importance for a valuable process description (Lindland et al. 1994; Ould and Huckvale; 1995). This is somewhat remarkable, considering our study indicated that clearly shown deliverables was in fact the most important driver for generating value when using described processes (see Figure 1).

Clearly described deliverables is a key driver for making the explicit knowledge—through described software processes—useful. Clearly described deliverables thus may become an important representative to understand the tacit knowledge.

Furthermore, storing organizational knowledge, discussing improvements, and communicating competences were the main reasons for using the described process. One interviewee said, *“It is only when the software process is described that we can be sure we talk about the same thing. Described software processes also play a role when initiating change [provides a given starting point] and enables comparison with actual events,”* and thus points at the close relation between explicit and tacit knowledge and in particular how the explicit can act as a catalyst for understanding the tacit. Our interviewees confirm what Ellmer and Merkl (1996) argued, that the described process was perceived as needed to manage organizational memory. The described process (i.e., explicit knowledge) was a facilitator to initiate discussion and improvement of all kinds of software engineering activities where actions, commitments, and involvement are important ingredients (i.e., tacit knowledge).

Figure 3. Explicit Knowledge Facilitates Viewing Tacit Knowledge

Explicit knowledge only represents the tip of the iceberg of the entire body of possible knowledge (Polanyi 1966). Having said this, we do not argue that it is either possible or necessary to see the complete iceberg. However, without a clear view of this tip (Figure 3a), there is a risk we might fail to see the rest of the iceberg, where challenges may need to be understood in order to be managed (Figure 3b).

Mathiassen et al. (2003) argue that codification of knowledge plays a crucial role both in relation to maturing professional practices and in relation to the use of information technology to support knowledge management. They also argue that people engaged in knowledge management activities should appreciate the problems and opportunities related to the codification of knowledge. Mathiassen et al.'s argument strengthens our view on the close relation between explicit and tacit knowledge, and our core argument that explicit knowledge is an important enabler of understanding the tacit within software engineering practice.

The explicit nature of described software processes serves as a vital catalyst in initiating discussions of tacit knowledge. These discussions enable valuable improvement suggestions to both tacit (e.g., work practices) and explicit (e.g., process descriptions) knowledge.

Nonaka and Takeuchi (1995) and Hansen et al. (1999) argue that innovation emerges from new interactions between explicit and tacit. This argument is supported by the result from our study, visualized in Figure 2, where "discussing improvements" (e.g., tacit knowledge) was valued as the second most important reason for having described processes (i.e., explicit knowledge).

We thus conclude by arguing that diffusing explicit software engineering knowledge is hindered by the lack of obvious standard notations for software processes and the tools to support them. In this way, the lack of obvious standard software process notations also hinders understanding tacit software engineering knowledge.

5.3 Implications for Practice and Research

For practice, it is important to spend time on understanding the current process description situation. It is also important to recognize the relation between explicit and tacit knowledge. Although the explicit knowledge is recognized only as the tip of an iceberg, without it, an organization would risk missing the opportunity for learning and improving when understanding of the tacit is not challenged. In daily work, where the tempo is high and unreflective actions are more common than not, a described process can indeed facilitate a common understanding of the current situation and thereby also form the basis for valuable discussions to initiate improvements. Furthermore, at this point, it may also be relevant to consider level of explicitness necessary in order to achieve maximum gains. To use our analogy, the question of how much of the iceberg it is relevant to see is becoming increasingly valid. On this note, our results provided hints that the level of abstraction from experiences from projects showed that the issue is far from obvious. On the one hand, enriching the explicit knowledge was recognized as potentially beneficial, but at the same time, several interviewees voiced concern that the process would be impossible for those not already experienced in using it, and that the overall purpose of guiding development would be defeated as any question directed toward a fully explicit process would result in so many answers—all depending on situation—that it would become almost impossible to know which experience to draw upon. Striking the right balance in what to make explicit would thus be an interesting area for further studies to see where practice may shed light on what works and why.

For research, it is important to understand that there is a lack of an obvious standard software process notation. Or at least, there is a lack of studies on deployed standard software process notations to be relied on as best practice examples. The software business would benefit from either having more focus on finding a valuable software process notation or by finding out why an already potentially valuable standard software process notation is not widely deployed. There are, of course, numerous explanations for why deployment fails, but it is likely some of them are related directly to understanding the value of a described process from a knowledge management perspective. From an agile software development perspective, it would be useful to understand more about how agile processes may be described and how they can facilitate understanding tacit knowledge.

Finally, as this study is limited to one software company (although involving many people at different sites and countries), it is possible that these findings are dependent on cultural issues related to this specific company such as process immaturity or lack of process competence. Other research in the software process improvement area (Börjesson 2006; Börjesson et al. 2007; Börjesson and Mathiassen 2004) has shown that software process awareness, attention, and focus are high in the company.

6 CONCLUSIONS

This study aimed to understand more about software process descriptions used in practice. For this paper, we specifically looked at the research question, how do software process descriptions affect understanding of software engineering knowledge? Through action research during the period 2005–2007, we investigated 13 formal and approved

process descriptions in a 1,500 employee unit working with software engineering with the telecom company Ericsson AB. The investigation was followed by presentations of the investigation and interviews of 12 senior managers and engineers. The study was framed and discussed in relation to the software process notations and knowledge management literature. The study is finalized by arguing that unambiguous process descriptions are dependent on obvious software process notations, where clearly described deliverables are a key driver for making the explicit knowledge—through described software processes—useful, and that the explicit nature of described software processes serves as a vital catalyst in initiating discussions of tacit knowledge.

Acknowledgments

We appreciate the valuable comments from the anonymous reviewers for this conference. We would also like to mention our appreciation for the high quality work by our master's thesis student, Per Johansson, during the spring of 2007.

References

- Bartunek, J. M., and Louis M. R. 1996. *Insider/Outsider Team Research*, Thousand Oaks, CA: Sage Publications, Inc.
- Baskerville, R., and Pries-Heje, J. 1999. "Grounded Action Research: A Method for Understanding IT in Practice," *Management and Information Technology* (9), pp.1-23.
- Baskerville, R. L., and Wood-Harper, A. T. 1996. "A Critical Perspective on Action Research as a Method for Information Systems Research," *Journal of Information Technology* (11), pp. 235-246.
- Baskerville, R. L., and Wood-Harper, A. T. 1998. "Diversity in Information Systems Action Research Methods," *European Journal of Information Systems* (7:2), pp. 90-107.
- Börjesson, A. 2006. "Improve by Improving Software Process Improvers," *International Journal of Business Information Systems* (1:3), pp. 310-338.
- Börjesson, A., Baaz, A., Pries-Heje, J., and Timmerås, M. 2007. "Measuring Process Innovations and Efficiency," in *Organizational Dynamics of Technology-Based Innovation: Diversifying the Research Agenda*, T. McMaster, D. Wastell, E. Ferneley, and J. I. DeGross (eds.), New York: Springer, pp. 181-196.
- Börjesson, A., and Mathiassen, L. 2004. "Successful Process Implementation," *IEEE Software* (21:4), pp. 36-44.
- Carmel, E. 2006. "Building Your Information Systems from the Other Side of the World: How Infosys Manages Time Zone Differences," *MIS Quarterly Executive* (5:1), pp. 43-53.
- Curtis, B., Kellner M. I., and Over, J. 1992. "Process Modeling," *Communications of the ACM* (35:9), pp. 75-90.
- Davison, R., Martinsons, M., and Kock, N. 2004. "Principles of Canonical Action Research," *Information Systems Journal* (14), pp. 65-86.
- Ellmer, E., and Merkl, D. 1996. "Considerations for an Organizational Memory in Software Development," in *Proceedings of the 10th International Software Process Workshop* Ventron, France, June 17-19, pp. 60-62.
- Galliers, R. D. 1992. "Choosing an Information Systems Research Approach," in *Information Systems Research: Issues, Methods, and Practical Guidelines*, R. D. Galliers (ed.), Oxford, UK: Blackwell Scientific Publications, pp. 144-162.

- Hansen, N., Nohria, N., and Tierney, T. 1999. "What's Your Strategy for Managing Knowledge?," *Harvard Business Review* (77:2), pp. 106-116.
- Haumer, P. 2006. "Software & Systems Process Engineering Meta-Model (SPEM 2.0)," Object Management Group, Needham, MA (<http://www.omg.org/cgi-bin/doc?ad/06-11-03>).
- Johansson, P. 2007. "Software Process Notations: The Role and Quality of Described Processes," unpublished Master's Thesis, Chalmers University of Technology, Gothenburg, Sweden.
- Kalnins, A., and Vitolins, V. 2006. "Use of UML and Model Transformations for Workflow Process Definitions," available through Cornell University Library (<http://arxiv.org/abs/cs.SE/0607044>).
- Kueng, P., Bichler, P., Kawalek, P., and Schrefl, M. 1996. "How to Compose an Objectoriented Business Process Model," in *Method Engineering on Method Engineering: Principles of Method Construction and Tool Support: Principles of Method Construction and Tool Support*, S. Brinkkemper, K. Lyytinen, and R. J. Welke (eds.), London: Chapman & Hall Ltd., pp. 94-110.
- Larsen, T. J., Niederman, F., Limayem, M., and Chan, J. 2008. "The Role of Modeling in Achieving Information Systems Success: UML to the Rescue?," *Information Systems Journal*, forthcoming.
- Lindland, O., Guttorm, S., and Sölvberg, A. 1994. "Understanding Quality in Conceptual Modeling," *IEEE Software* (11:2), pp. 42-49.
- Maruping, L. M., and Agarwal, R. 2004. "Managing Team Interpersonal Processes Through Technology: A Task-Technology Fit Perspective," *Journal of Applied Psychology* (89:6), pp. 975-990.
- Mathiassen, L. 2002. "Collaborative Practice Research," *Information, Technology & People* (14:4), pp. 321-345.
- Mathiassen, L., Robertson, M., and Swan, J. 2003. "Cracking the Code: The Dynamics of Professional Knowledge," paper presented at the European Knowledge Management Conference, San Francisco, CA.
- Nicholson, B., and Sahay, S. 2001. "Some Political and Cultural Issues in the Globalization of Software Development: Case Experience from Britain and India," *Information and Organization* (11), pp. 25-43.
- Nonaka, I., and Takeuchi, H. 1995. *The Knowledge Creating Company. How Japanese Companies Create the Dynamics of Innovation*, New York: University Press, Inc.
- Ould, M. A. 1995. *Business Processes*, Bath, UK: John Wiley & Sons Ltd.
- Ould, M., and Huckvale, T. 1995. "Process Modeling: Who, What and How—Role Activity Diagramming, in *Business Process Change: Concepts, Methods and Technologies*, W. Kettinger and V. Grover (eds.), Hershey, PA: Idea Group Publishing, pp. 330-349.
- Owen, M., and Raj, J. 2005. "PMN and Business Process Management," Telelogic (<http://www.telelogic.com>).
- Patton, M. Q. 1987. *How to Use Qualitative Methods in Evaluation*, Newbury Park, CA: Sage Publications, Inc.
- Polanyi, M. 1966. *The Tacit Dimension*, New York: Anchor Day.
- Reynolds, T. J., and Gutman, J. 1988. "Laddering Theory, Method, Analysis, and Interpretation," *Journal of Advertising Research*, February-March, pp. 11-31.
- Russel, N., ter Hofstede, A. H., van der Aalst, W. M., and Wohed, P. 2006. "On the Suitability of UML 2.0 Activity Diagrams for Business Process Modelling," in *Proceedings of the 3rd Asia-Pacific Conference on Conceptual Modelling* (Volume 53), Hobart, Australia, pp. 95-104.
- Sahay, S. 2003. "Global Software Alliances: The Challenge of 'Standardization,'" *Scandinavian Journal of Information Systems* (15), pp. 3-21.
- Susman, G., and Evered, R. 1978. "An Assessment of the Scientific Merits of Action Research," *Administrative Science Quarterly* (23), pp. 582-603.

White, S. 2004. "Process and Modeling Notations and Workflow Patterns," BP Trends, March (<http://bptrends.com/publicationfiles/03-04%20WP%20Notations%20and%20Workflow%20Patterns%20-%20White.pdf>).

Yin, R. 1994. *Case Study Research*, Newbury Park, CA: Sage Publication, Inc.

About the Authors

Anna Börjesson Sandberg is a senior specialist in research and development working at the telecom company Ericsson AB in Gothenburg, Sweden. Both her working tasks and research are focused on software process improvement, process descriptions, change management, and change agency. Anna has been working within the software Industry since 1994. She strives to become a reflective practitioner through combining daily industry work with action research. In January 2006, she received her Ph.D. in Applied IT from the IT University of Gothenburg. Anna is a member of IEEE and ACM. Contact her at anna.sandberg@ericsson.com.

Carl Magnus Olsson is a Ph.D. student at the University of Limerick, Ireland, and works at the IT University of Gothenburg, Sweden. He has held a guest researcher position at the Operations Management and Information Systems Department of Northern Illinois University, and was a participant in the ICIS Doctoral Consortium in Seattle, 2003. Carl Magnus has previously worked professionally with software improvement for six years. His publications include conference papers at IFIP WG 8.2, IFIP WG 8.6, the European Conference on Information Systems, and the International Conference on Information Systems. Contact him at carl.olsson@ituniv.se.

Appendix A: Questions for Cycle Three

#	Questions (for # 1–17, 19: Value on a six grade scale from strongly disagree to strongly agrees to the statement)
1	"I believe described processes are important for discussing improvements." Why?
2	"I believe described processes are important for communicating knowledge and competence." Why?
3	"I believe described processes are important for storing organizational knowledge." Why?
4	"I believe described processes are important for finding weaknesses or problems." Why?
5	"I believe described processes are important for measuring improvements." Why?
6	"I believe described processes are important for increasing product quality." Why?
7	"I believe described processes are important for other reasons." Which and why?
8	"I believe it is important for high process quality that the model corresponds to 'the real world.'" Why?
9	"I believe it is important for high process quality with a low number of levels (clicks) to reach information." Why?
10	"I believe it is important for high process quality that the symbols are easily understood." Why?
11	"I believe it is important for high process quality that the flows are easily understood." Why?

12	"I believe it is important for high process quality that a standard notation is used." Why?
13	"I believe it is important for high process quality that common process modeling guidelines are used." Why?
14	"I believe it is important for high process quality that roles are clearly shown." Why?
15	"I believe it is important for high process quality that deliverables are clearly shown." Why?
16	"I believe it is important for high process quality that activities are clearly shown." Why?
17	"I believe something else is important for high process quality." What and why?
18	What tools for modeling software processes do you know of?
19	"I believe tools for modeling software processes are important." Why?