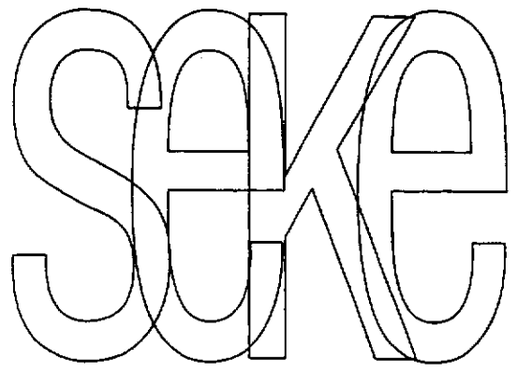
INTERNATIONAL
JOURNAL OF
SOFTWARE
ENGINEERING
AND
KNOWLEDGE
ENGINEERING



VOLUME 19 NUMBER 2 March 2009

Modeling and Analysis of Knowledge Flows in Software Processes through the Extension of the Software Process Engineering Metamodel

O. M. Rodríguez-Elias, A. I. Martínez-García,
A. Vizcaíno, J. Favela and M. Piattini

World Scientific

NEW JERSEY • LONDON • SINGAPORE • BEIJING • SHANGHAI • HONG KONG • TAIPEI • CHENNAI

International Journal of Software Engineering
and Knowledge Engineering
Vol. 19, No. 2 (2009) 185–211

World Scientific
www.worldscientific.com

MODELING AND ANALYSIS OF KNOWLEDGE FLOWS IN SOFTWARE PROCESSES THROUGH THE EXTENSION OF THE SOFTWARE PROCESS ENGINEERING METAMODEL

OSCAR M. RODRÍGUEZ-ELIAS*,‡, ANA I. MARTÍNEZ-GARCÍA*,§, AURORA VIZCAÍNO†,¶, JESÚS FAVELA*,‖ and MARIO PIATTINI†,**

*Computer Science Department, CICESE,
Km. 107 Carretera Tijuana-Ensenada, Ensenada,
Baja California, 22860, Mexico*

*Industrial Engineering Department,
University of Sonora, Luis Encinas y Rosales,
Hermosillo, Sonora 83000, Mexico*

†*Alarcos Research Group, Escuela Superior de Informática,
Universidad de Castilla-La Mancha,
Paseo de la Universidad 4, Ciudad Real, 13071, Spain*
‡*orodriguez@industrial.uson.mx*
§*martinea@cicese.mx*
¶*Aurora.Vizcaino@uclm.es*
‖*favela@cicese.mx*
**Mario.Piattini@uclm.es*

Received 26 May 2006
Revised 16 May 2007
Accepted 2 November 2007

Knowledge is a key asset in software engineering. Facilitating access to the knowledge that software engineers require for the task at hand can therefore bring many benefits. To accomplish this, it is important to understand how knowledge flows through the organization, to identify problems that may hinder a suitable flow, and to define strategies with which to address them. Process modeling has proved to be a useful technique for analyzing knowledge flows. Traditional process modeling languages do not, however, provide primitives to explicitly represent the knowledge involved in the processes within the models. In this paper, we illustrate how the Software Process Engineering Metamodel (SPEM) can be adapted, to be used as a process modeling language for analyzing knowledge flows in software processes. We have extended SPEM to represent knowledge and its sources in process models in an explicit way. We also discuss the experiences obtained from using this extension in a software organization and the lesson we have learned from it.

Keywords: Knowledge flow modeling; knowledge management; process modeling; software process; SPEM.

‡Corresponding author.

1. Introduction

One of the main assets of a software organization is its organizational knowledge, and there is thus increasing concern about the adequate management of software engineering knowledge [1]. A knowledge management (KM) strategy [2] can bring many benefits to software organizations which include time and cost reduction, quality increase, and improvement of the work environment or the software developer's performance [1, 3–5].

The main objective of KM is to facilitate the flow of knowledge in organizations [6]. Knowledge flow is formed of the processes through which knowledge is transferred throughout different places and in its different states. For instance, from where it is created or stored to where it is used; or from people to formal sources, and vice versa [6]. By facilitating knowledge flow within a software organization, it should be easier to integrate such knowledge within the software development activities, thus producing a positive effect upon the developers' performance [5]. However, it is first important to identify the knowledge which is required to perform those activities, the way in which that knowledge actually flows, and the manner in which that flow can be improved.

Process analysis techniques can provide many advantages in defining KM strategies that are integrated to the real knowledge needs of an organization [7]. A useful means by which to identify such needs is by using process modeling to analyze how knowledge flows through the processes [8–10]. Modeling can help to identify knowledge inputs and outputs in organizational activities, knowledge sources and knowledge flows between activities, roles, or systems [11, 12]. Furthermore, by modeling processes we can detect problems that may affect the correct flow of knowledge. It is then therefore possible to start defining strategies to improve that flow.

In this paper, we show how the Software Process Engineering Metamodel (SPEM) [13] can be used to analyze knowledge flows in software processes. We present a SPEM adaptation for modeling software processes, with a focus upon the identification and analysis of knowledge flows. We illustrate the approach with an example extracted from a case study in a software maintenance organization, and provide some lessons learned from this study. The content of the paper is organized as follows: first, the importance of knowledge flows in software companies is outlined. Next, some related work is exposed, particularly studies which focus upon identifying knowledge needs in software engineering in general, and in software maintenance in particular, in order to approach later the use of process modeling to identify knowledge flows in software processes. In Sec. 4, we describe the extension made to SPEM to help identify and analyze knowledge flows, and we present a case study carried out to illustrate the use of the SPEM extension in a real situation. The lessons learned from this case study are summarized in Sec. 5. Section 6 presents a discussion of the evaluation of our proposal. Finally, conclusions and future work are presented in Sec. 7.

2. Software Companies and Knowledge Flows

Improving knowledge flow is a highly important goal in KM initiatives [14], since "*the main purpose of KM is to make sure that the right people have the right knowledge at the right time*" [15]. Consequently, in order to provide KM support in an organization, one important step is to understand how knowledge flows through it [16].

The flow of knowledge in software companies may be affected by many factors. In fact, some of the most frequent problems in software organizations can be traced to knowledge flow problems, such a lack of documentation, or frequent personnel turnover; see, for example, [17–21]. Documentation is one of the most important sources used to transfer knowledge between developers and maintainers in software companies [22]. However, a common problem is the lack of updated documentation. On the other hand, turnover in personnel causes the loss of part of the most important knowledge for software organizations [4]. This problem becomes even worse if we consider that inexperienced developers often take up the posts of experienced employees who leave the organization, or who are assigned to other projects [23]. Hence, it is important to identify the knowledge that those inexperienced developers may require; equally necessary is to make the sources that are useful in obtaining that knowledge easily available.

To accomplish the above goals, we must study the current situation of the organization in which the KM support is going to be provided. This includes the identification of the main activities performed, the knowledge required and generated in those activities, and the channels through which that knowledge is flowing. We also need to identify the knowledge that is not flowing and the causes of this, in order to define possibile means by which to amend this situation [24]. Carrying out a process analysis focused on knowledge flow has the potential of bringing various advantages to software companies, as it can help to:

- ***Identify knowledge-related problems.*** The graphical representation of a process can be used as a communication channel between the actors of the process, and the process analyzers. This can enable brainstorming to identify problems such as knowledge bottlenecks, and a search for possible solutions to solve them [9].
- ***Increase the information of the knowledge and knowledge sources involved in the process.*** The explicit representation of elements in process models facilitates the analysis of those elements [25, 26]. People analyzing models with explicit representation of knowledge and its sources can be induced to think about those elements and, as a consequence, provide more information about them.
- ***Identify tools that can be integrated within the KM initiative.*** An important part of successful KM initiatives is to integrate the current technical infrastructure [27, 28], and to connect the KM support to the real work of the organization [29]. Many tools often used in software development may be

powerful knowledge sources or KM systems [30]. However, those systems might not be used to their full potential as KM systems. Analyzing software processes and having a special focus upon the way in which the tools used in the process can provide support to the knowledge flow may facilitate their integration as a part of the KM initiative.

- ***Identify requirements in order to acquire or develop new tools through which to improve the knowledge flow.*** Once the problems affecting the knowledge flow are identified, it is easier to define requirements through which to modify the tools currently used, or to acquire or develop new tools to solve those problems.
- ***Analyze the effects of including KM strategies in the process.*** Process models which consider the knowledge flowing through the process can be used to analyze the effects caused by the inclusions of changes in the knowledge flow [31]; this can, for instance, be carried out by modifying the process models of the current process to reflect the changes, and then comparing both models in order to analyze how these changes may affect the process.
- ***Improve the assignment of human resources.*** Assigning people appropriate roles in the development process is crucial to the attainment of a productive development team [32]. One of the main steps for this is to identify the profile required for each role, which includes the identification of the knowledge and skills that are required for the people that will play a specific role [32]. Software process models that consider the knowledge and skills required in the activities performed by the different roles can be used to obtain information with which to define those profiles.

3. Previous and Related Works

Our primary focus is on small to medium size software organizations. It is clear that the requirements of these organizations are different to those of big software companies [33–35]. For instance, they frequently not only have problems in adopting standardized processes, or their own processes are not well defined, but they also do not have enough resources to be able to adopt new KM systems or processes. Instead, they must first see their work processes in terms of knowledge, and see KM as a concept with implications in their current processes and infrastructure [33]. Hence, studying software processes from a knowledge perspective can contribute to helping small to medium software companies see their processes in knowledge terms and to facilitate the initiation of KM strategies which consider the current processes and technical infrastructure. Two main limitations have been found in current research:

- Most literature in the field has been conducted in big companies, and from the point of view of new developments (see [1]). It is, therefore, not directly applicable to our context. Moreover, researchers in the field of KM (for instance [36–38]), state that KM initiatives must be based on the particular needs of each organization, and that it is important to study their knowledge workers' specific needs.
- Although traditional process modeling techniques are useful in the identification of certain issues related to the knowledge involved in software processes, some important aspects are difficult to analyze. These include dependencies between knowledge, knowledge sources and activities, and, flows of particular knowledge between sources and/or activities, among others.

The work related to our study, therefore, deals with two areas: the identification of software process knowledge needs and the modeling of software processes with a focus upon the knowledge involved. We shall now present existing work in both these areas.

3.1. *Knowledge identification in software processes*

The identification of the knowledge required by people performing software engineering activities is an important research area. For instance, Lethbridge in [39] and Kitchanham *et al.* in [40] present studies aimed at identifying the knowledge that software professionals require in practice. It can be observed that much of such knowledge is obtained during practice, and thereafter, depends on the context of the organization in which that knowledge is applied. These works provide an appropriate reference through which to identify the main knowledge areas related to software engineering. However, they are too wide to be practical when classifying the knowledge involved in specific small software organizations.

In spite of the importance of software maintenance [41], there are few studies dealing with software maintenance knowledge needs. Most of these works can be classified in the field of program comprehension (examples are [42, 43]). This may be because source code is the main source of information for software maintainers [21]. However, they frequently also consult other sources [17], and it is important to identify how these are related to the maintainers' knowledge needs. Two types of studies address this issue:

- Koskinen *et al.* [44] have proposed a program comprehension tool by which to facilitate software maintainers' access to information sources in order to fulfill certain information needs defined from a review of literature. On the other hand, our research group [45, 46] has proposed the use of software agents [47] to facilitate access to knowledge sources in software maintenance. This last approach is not focused upon facilitating code comprehension, but upon facilitating software maintainers' access to sources which will be useful for obtaining information from other domains, such as the application domain, users' behavior, and previous maintenance requests, amongst others.
- Oliveira *et al.* [48] have performed a study through which to identify the knowledge involved in software maintenance, from which an ontology was developed [49]. Oliveira *et al.*'s ontology is similar to the ontology of Ruiz *et al.* [50], since

both ontologies identifies the main elements that are involved in the maintenance process, and the relationships between them. Additional, both ontologies are based on the Kitchenham *et al.*'s ontology of software maintenance [51]. Nevertheless, Anquetil *et al.* in [52] illustrate how such types of ontologies can be used as frameworks for obtaining knowledge about software maintenance projects. They used their ontology to discover what to request during postmortem analysis in maintenance projects.

The works of Koskinen *et al.* [44] and Rodriguez *et al.* [45] show that relating knowledge sources to specific knowledge needs can make a contribution towards aiding software maintainers to access the knowledge sources that might be useful for specific activities. On the other hand, Oliveira and her colleagues [48, 52] illustrate that having a model of the tasks to be carried out, and the elements involved in those tasks, such as knowledge and sources, can facilitate the identification of such knowledge needs, and the knowledge sources that might be related to them. Following this observation, we decided to focus on facilitating the identification and modeling of the sources used to obtain the knowledge that developers or maintainers require to perform their tasks, and the mechanisms that (positively or negatively) affect the flow of that knowledge within the software processes.

3.2. *Knowledge flow analysis through process modeling*

We have found very few works addressing the use of Process Modeling Languages (PML) in the study of knowledge in software processes. Because of the lack of PMLs with a knowledge perspective, traditional PMLs for business processes are frequently used for this purpose. Hansen and Kautz [9], and Rodriguez *et al.* [53], have used Rich Pictures [54] to study the knowledge involved in software processes. Rich Pictures is a flexible PML which can easily be adapted to interleave different perspectives in a process model. Nevertheless, it may be easier to analyze models represented in a more formally constrained PML than those represented in a flexible and general PML [25]. In fact, we experienced some problems in the analysis of knowledge flows in a software maintenance process when using Rich Pictures. It was difficult, for instance, to identify a detailed structure of activities and sub-activities, the sequence of activities, and dependencies between the activities and the knowledge and its sources involved [8, 53]. Therefore, we considered it more appropriate to use a formally-constrained language capable of enabling us to identify these details.

Woitsch and Karagiannis [55] propose a tool and a modeling approach which focus upon facilitating the analysis, documentation and implementation of enterprise KM systems. They use various types of models to represent different elements of the process, such as activities, people, etc. They illustrate the approach with a case study in a software organization and show how those models can be used to design an organizational memory. Nissen and Levitt [31] have used a tool to design virtual teams with which to analyze knowledge flows in a software process. They use this example to illustrate a theoretical model of knowledge flow. Neither the tools

nor the PMLs used in these two works have explicit primitives for representing knowledge. Zhuge [56] has proposed an approach for analyzing knowledge flows in software organizations by decomposing the processes in tasks and knowledge nodes, and defining the workflow of tasks and the sequence of the knowledge transfers between knowledge nodes, based on the inputs and outputs of tasks and knowledge nodes (for instance knowledge consumed or generated in the node). Finally, Strohmaier and Tochtermann [57] propose a framework with which to define a knowledge process and a tool to facilitate the development of a knowledge infrastructure, and this is exemplified by the development of a knowledge portal for a software company. However, this approach focuses upon the development of KM systems and not upon the analysis of software processes.

Bera *et al.* [10] have adapted a knowledge engineering approach to the analysis of knowledge requirements in business processes in domains different of software processes. They illustrate the use of their approach by identifying the knowledge sources related to the process activities, the manner in which those sources change during the process, and the main information or knowledge that those sources may contain. Kim *et al.* [58] have also proposed a specific type of diagram with which to represent and analyze flows of knowledge between processes. Finally, Papavassiliou and Mentzas [59] have proposed a metamodel with which to structure workflows which differentiates KM tasks from normal tasks, and knowledge objects from information or data objects.

Three observations can be made about the last review: first, we have found no examples of the use of a software-process-oriented PML in these works. Second, the approaches proposed in the works listed here are not based upon standardized PMLs, even in the case of general business processes, and less still for software process modeling. Third, it is clear that general PMLs can be used to identify certain issues related to knowledge flows in an implicit manner, such as the information sources which are required, generated, or modified by an activity [12]. However, it is important that a PML used to analyze knowledge flow provides explicit representation of these issues, such as the knowledge consumed or generated in activities or that required by the roles participating in those activities, the sources of that knowledge, or knowledge dependencies [31]. One way through which to address the latter situations is to adapt a standard PML to integrate the representation of knowledge, as we have done with SPEM.

SPEM is a UML based metamodel which has been specifically designed for software process modeling [13]. SPEM was developed to describe software processes and their components, following an object-oriented modeling approach based on UML. SPEM is defined as a UML profile; which means that it is a variant of UML which uses the extension mechanisms of UML in a standardized form for the purpose of modeling software processes. There are various advantages of using SPEM as a software PML in our work. First, UML is the most extended modeling language for software development; this may facilitate the assimilation of SPEM as a standard software PML. Secondly, it is possible to use any UML modeling tool supporting

UML profiles to produce SPEM models [60], and there are many software modeling tools which enable the use of UML as a modeling language: commercial, open source, and free tools (in our study, we have used a general commercial modeling tool). Finally, SPEM enables the modeling of a software process from different views, at different levels of abstraction and with a formally constrained language. These advantages of the SPEM notation are also being taken into account in the modeling of processes in widely-used software development environments, such as the Eclipse Process Framework (http://www.eclipse.org/epf/). The latter may help to promote the adoption of SPEM as a standard PML for process-based software development environments. Therefore, using SPEM to model software processes with a focus upon the knowledge involved, opens possibilities for its use as a basis for the introduction of process-based KM support in such environments.

4. Using SPEM to Analyze Knowledge Flows

In general terms, a SPEM process model is defined as a set of work items such as *activities*, which are classified as *work definitions*. These *work definitions* are operations that describe the work carried out by *process roles* in a process. *Work definitions* are used to structure a process by, for example, describing its lifecycle, phases, iterations or activities. The results of a work definition are called *work products*. A work product can be anything produced, consumed or modified by a process [13].

The SPEM specification proposes a standard notation with which to represent some of the SPEM concepts which are UML stereotypes that can be used in different kinds of diagrams to represent static, dynamic and behavioral views of the process. To do this, the following subset of UML diagrams can be used to represent a process model in SPEM:

- **Class diagrams** represent the structure of the work products, such as the way in which they are related to process roles, or their relationships of inheritance, decomposition, dependencies, or simple associations with other work products.
- **Package diagrams** illustrate the organization of a process, for example by grouping related work products, activities, roles, sub-processes, etc.
- **Use case diagrams** show the relationships between roles and work definitions.
- **Sequence diagrams** describe interactions between instances of SPEM elements.
- **Statechart diagrams** illustrate the behavioral view of SPEM model elements.
- **Activity diagrams** present the sequencing of activities with their input and output work products. These kinds of diagrams can also illustrate object flow states. By using swimlanes, the responsibilities of process roles can be separated.

4.1. *An SPEM extension to represent knowledge and its sources*

SPEM provides diagrams that could be used to identify knowledge flows. For example, activity diagrams can show the documents that are being modified and can implicitly define the information that is being captured or obtained from those documents. However, SPEM does not provide primitives to represent the knowledge explicitly involved in those activities. To achieve this, we have extended SPEM with several additional elements. Our interest was focused upon the following three aspects:

(1) Illustrating the knowledge and knowledge sources involved (used, generated and/or modified) in the processes and activities.
(2) Illustrating the way in which specific knowledge flows among the activities, or how a specific source is used and modified through the activities.
(3) Illustrating transfers of knowledge between sources, and among activities.

To attain the above goals, we defined certain concepts and relationships which have been packaged in a metamodel of knowledge concepts called a KnowledgeConcepts Package. These concepts are described in the following subsections.

4.1.1. *Knowledge as work product type*

Topics and sources of knowledge are defined in our models as entities that can be used, generated, or modified in the activities; that is, they are work products if we consider the SPEM specification. Based on this, we have included knowledge concepts as a type of work product called *InvolvedKnowledge*, which is used to represent the knowledge involved in the activities (see Fig. 1). This knowledge could be specific knowledge concepts, each represented as a *KConcept* element, or groups of them. To enable the creation of sets of related knowledge concepts, a type of package called *GroupedKnowledge* has been defined. Figure 1 illustrates this, and shows how the elements of the *KnowledgeConcepts* package are related to the SPEM elements.

The knowledge concepts included in the metamodel are shown in Fig. 2. Two main elements are derived as *KConcepts*: *KSource* to represent knowledge sources, and *KTopic* to represent knowledge topics. The knowledge sources have a type of location associated with them, such as a physical, email, or electronic file

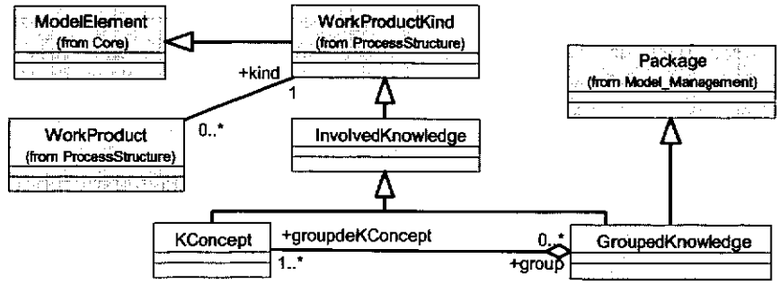


Fig. 1. Relationships between the SPEM elements and the KnowledgeConcepts package elements.

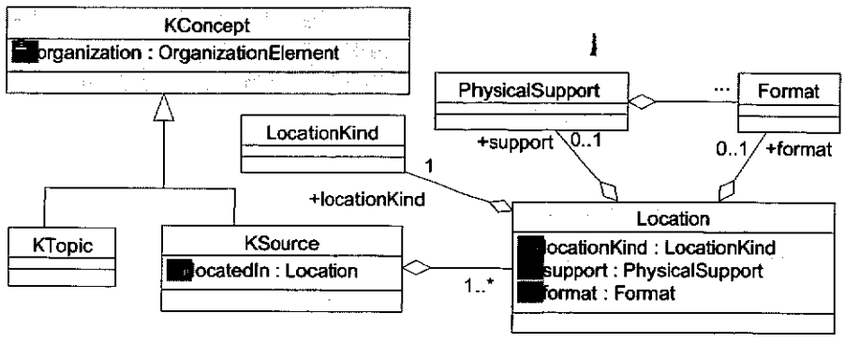


Fig. 2. Metamodel of knowledge concepts (KnowledgeConcepts package).

address, which are defined as *LocationKind*. These locations can be supported by a physical means, such as paper, video, photography, etc., and have a format, such as Word, Excel, or pdf electronic files. Each physical support can also have a set of possible formats. In this manner, we are able to integrate sources that could be replicated, such as a source that has been printed on paper and is also stored in an electronic document repository; we can define it once, and associate two or more locations to it.

Additionally, a set of organizational elements have been defined with which to enable the organization of the different knowledge sources and topics. Specifically, topics can be organized by following a schema of categories, areas, and subjects (similar to that used to classify knowledge subjects in the SEWBOK [61]). Knowledge sources are classified by following a schema composed of categories and types. By following these schemas, it is possible to begin to define taxonomies of knowledge sources and topics. The last is an initial step for many KM initiatives [37].

In order to illustrate the use of the concepts described above, let us suppose that while we are studying the knowledge required to performing a change in an application in a maintenance process, we observe that maintainers require the knowledge of which files correspond to a particular module of the system. From this, we identify the location of the source files and the internal structure of the code as knowledge topics required by the activity. We have two options: to include each topic in the models, or to group them, perhaps as a source-files-related-knowledge group. If we do the latter, we can later include other topics that may be related to the source files (such as knowledge of the programming language used to codify the files or of the application used to store or manage them) without having to modify the models.

After identifying the topics, we classify them. Our first step is to include the source-files-related-knowledge as part of the knowledge about the specific application, which is a sub-area of the maintained-products-knowledge area, and which is, at the same time, part of the maintenance-process knowledge category.

Following this schema, it is possible to define useful sources from which to obtain knowledge from an area, a sub-area, or a specific subject.

We later observe that, as is usual, maintainers tend to analyze the source code in order to identify the specific files that they must change. However, we realize that some of the maintainers have been working on writing documents to describe the relationships between the different modules in the system, and the source files that implement those modules, in order to facilitate their own work. However, since these documents are not a formal rule of the organization, and are simply for the use of their authors, they are not known by other maintainers. Additionally, we identify that some maintainers have written the files that need to be changed in order to solve the request in certain maintainance requests. Hence, those requests could be also a useful source. Based on the last point, we have now identified three different types of sources located in three different places: the source code, the personal document repositories of some maintainers, and the logbook where the maintenance requests are stored. The next step is, therefore, to include those sources as a part of the model of the process in order to identify in which activities might be useful. This may perhaps be achieved by illustrating the knowledge that can be obtained from the sources and the activities where such knowledge is required. In our models, this is done by using two main relationships which will now be described.

4.1.2. *Relationships*

Certain types of relationships were defined to represent transfers of knowledge and the knowledge that a source may contain (see Fig. 3). The *KnowledgeTransfer* relationship represents transfers of knowledge between sources, perhaps between people, a person and a document, etc. This relationship has two main properties: the sources and the work definition in which the transfer takes place. The knowledge transferred or received by the sources may be single topics or sets of topics grouped into *GroupedKnowledge* packages, and are defined with *KnowsAbout* relationships. These relationships define the knowledge that can be obtained from a source. The knowledge level (*KLevel*) property can be used to specify the level of expertise or experience of the knowledge concept that it is possible to obtain from the source, according to specific needs.

In order to illustrate the use of these relationships, let us continue with the previous example. After identifying the sources, we wish to make explicit which sources are useful for the attainment of knowledge from the source files related to a specific application. Therefore, we start relating those sources to the package that groups such knowledge through the *KnowsAbout* relationship. After doing this, we realize that the maintenance requests stored in the logbook might be an important source for other types of knowledge. We therefore begin to develop diagrams to help us to analyze how that type of source is being used in the process and particularly, what information and knowledge is being stored in them, in which activities, and who is doing it, for which we use *KnowledgeTransfer* relationships.

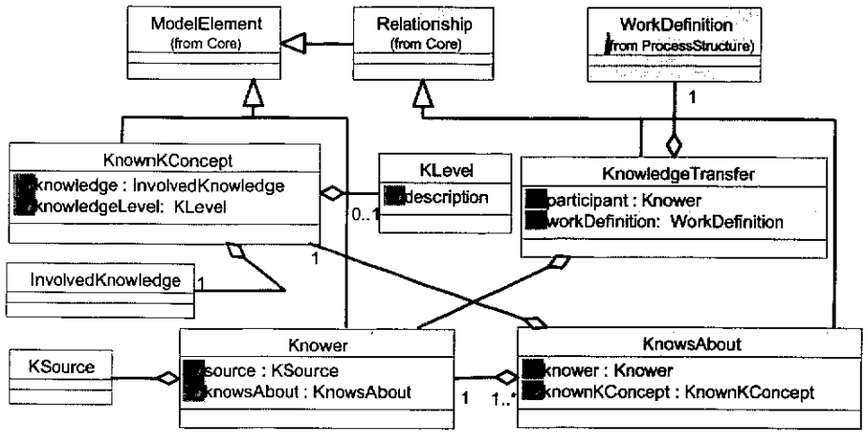


Fig. 3. Associations defined in the KnowledgeConcepts metamodel.

4.1.3. *Notation*

Figure 4 shows the notation proposed to represent knowledge topics, sources, and transfers within the models. These icons are used together with those proposed by the SPEM notation [13]. The KTopic icon represents specific skills, knowledge topics or subjects. The GroupedKnowledge package is used to represent groups of related skills or knowledge topics. KSource is used to represent knowledge sources which are primarily documents. In order to represent people or roles, the SPEM role icon is used. The KnowledgeTransfer icon represents the knowledge transfer relationship.

These notation icons are used in the different diagrams of SPEM, with the exception of the KnowledgeTransfer relationship, which is used in a new type of diagram being proposed. Use case diagrams are used to provide a general view of the process by identifying the main activities or workflows, and the main knowledge sources or packages of knowledge involved. Activity diagrams help to identify details of the knowledge involved in specific activities, and the knowledge that each role may require whilst performing those activities. Some knowledge dependencies, such as knowledge generated in one activity and applied to another, may also be identified in activity diagrams. Class diagrams identify knowledge that can be obtained from a source, and dependencies or relationships between knowledge topics and between

Fig. 4. Notation icons for the elements of the KnowledgeConcepts package.

sources. Package diagrams organize and classify knowledge topics into packages. Finally, in order to analyze transfers of knowledge, we use knowledge transfer diagrams. These illustrate the sources participating in the transfer (roles, documents, etc.), the activity or workflow in which the transfer takes place, and the knowledge being transferred. These diagrams are used to analyze the knowledge that each source shares, obtains, or that that is stored in it, along with the activity or workflow in which the transfer is taking place. If more details of the transfer are required, we use sequence or statechart diagrams.

4.2. *An application example*

In this section we describe a case study carried out to illustrate the way in which the SPEM extension can be used. In this case the approach is used in a software maintenance process in order to begin classifying knowledge topics and sources, and to identify the relationships between the sources and the knowledge that can be obtained from them. We finalize by showing that the study was also useful in identifying opportunities through which to improve the use of a tool as a knowledge flow facilitator within the process.

4.2.1. *The method*

In order to analyze knowledge flows in a maintenance process, we followed the KoFI methodology, first described in [24], and extended in [62]. KoFI proposes four main steps in the analysis of knowledge flows (see Fig. 5). The first step focuses on the identification and classification of the main sources of knowledge. The second step focuses upon the main knowledge topics which the activities of the process need to carry out. The third step focuses on the relationships between knowledge sources and topics, and the activities of the process. Finally, the fourth step focuses on the identification of the main problems affecting the knowledge flow (consult [24, 62] for more details).

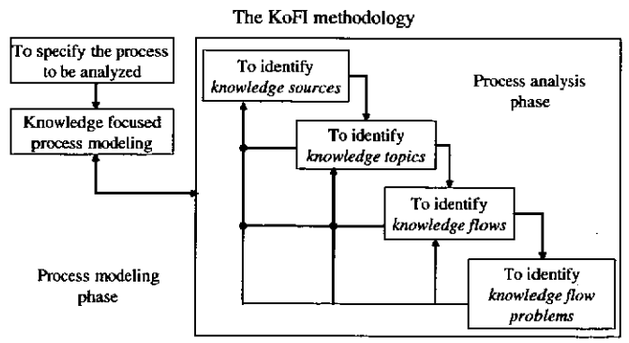


Fig. 5. General view of the steps of the KoFI methodology.

The KoFI methodology was applied for two main reasons: first, to obtain a general knowledge map of the process, i.e. to discover the main knowledge sources, the knowledge that can be obtained from them and the activities in which they could be helpful. Secondly, we focused on identifying the mechanisms used to transfer knowledge through the process, particularly the technical infrastructure currently used for that purpose, such as information systems. The goal was to identify the way in which those mechanisms contribute to the knowledge flow, and the manner in which they can be used to improve that flow. Figure 5 shows the role of the process models in the study.

4.2.2. *The subject of study*

The group of software developers who took part in the case study was the Informatics Department (ID) of a research center, which is in charge of maintaining the information systems used in this center. At the time of the study the group was composed of fourteen people: the head of the department (HD), a secretary, six software engineers (SEs), and six assistant programmers (APs). One of the SEs was a novice, with only 2 months' experience in the job; the other SEs had between 5 and 11 years experience; all of the APs had spent less than 1 year in their jobs. The group was in charge of maintaining the applications in 5 domain areas: finances, human and material resources, academic productivity, and student services. All of the applications were of about 10 to 13 years old, if we take into account the development of their first version. Most of them used the Oracle database management systems and programming tools. Normally, these applications require only one or two people for their maintenance, and can be considered as medium size applications (between 50 and 100 different modules per application, taking into account reports, views, forms, menus, etc., the ID does not have an exact measurement of their current size). Most of the maintainers did not participate as original developers of the systems. APs were assigned to the modification activities; there are often changes in the particular people involved in these tasks.

The ID does not follow standard software processes. Nevertheless, its maintenance process is similar to others (i.e. [63–65]). The work of the ID is organized into projects triggered by maintenance requests, which are of two types: modification requests (MRs) made by the users of the systems to improve or to add new features, or to adapt the system to changes in the environment (the operating system, the process, etc.); and problem reports (PRs), which describe problems the users have experienced with the systems. Each SE is responsible for the requests[a] of the systems assigned to him/her, even though some of the changes may eventually be made by another SE or by an AP. All the requests are stored in a logbook. The logbook is used to track the status of each request by the HD and the users who made them. The SEs use the logbook to track the requests assigned to them, and to consult information about the request.

[a]We use requests to refer to both, MRs and PRs.

The data of the process was obtained from interviews with the members of the ID, direct non-participatory observation, and the analysis of documents and information systems of the ID. We first developed an initial model of the process whose modification was based on observations and comments made by the members of the ID during the period of our final interviews. We used the previous models as a mechanism by which to analyze the process in the final interviews, until the SEs agreed with the models.

4.2.3. *Identifying and classifying knowledge sources and topics*

One of the main contributions of the models developed by following the extensions is that such models help to initiate the identification and classification of the main knowledge topics required by the activities and the sources in which such knowledge can be obtained. These are a part of the first two steps in the methodology followed. Figure 6 illustrates part of this; the model presents the main general topics of knowledge required by the ID maintainers to perform modifications to the systems of which they are in charge. These general topics are classified into knowledge packages of related topics, which also have sub-packages of more specialized topics. Figure 6 shows that the main knowledge needs of a maintainer are: (1) the knowledge related to the system being maintained, which includes knowledge of the structure of the system, the correct functioning of the system, and so on; (2) the application domain of the system, which includes topics related to the specific concepts used in that application domain, and the process being supported by the system; and (3) technical knowledge, which includes topics related to the programming language and the development environment being used. As can be observed in Fig. 6, it is also possible to explicitly specify dependencies between knowledge topics. For instance, some topics related to the knowledge about the system, require that the maintainers also know about the application domain.

By using the general topics as a base, it is possible to extend each package in order to define the specific systems being maintained and the different application domains. As an example, we will use the system that supports the graduate studies department activities (called SIDEP). This system includes three subsystems, one for managing the academic processes, such as student status management, course registration, or the enrolling of students. The second subsystem is for managing student scholarships. The third system is for managing academic productivity, such as the publications of researchers and students or theses directed by researchers, amongst other processes.

Once the specific knowledge topics have been identified, it should be easier to identify the sources of such topics. To illustrate the relationships between the topics and the possible sources a class diagram such as that shown in Fig. 7 may be used. In this diagram, it is possible to observe that the sources include both documents describing the procedures and norms of the research center, and the people who perform the specific tasks in the processes being supported by SIDEP. By using the model in Fig. 7, a maintainer in charge of modifying something in one of the

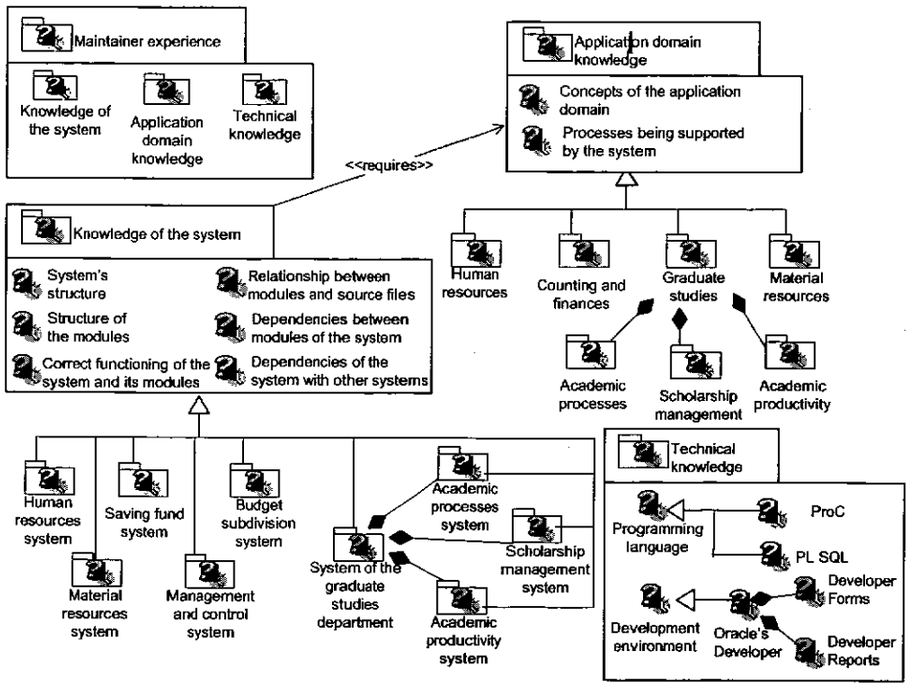


Fig. 6. Use of the SPEM extensions to classify knowledge involved in the activities of the process.

subsystems of SIDEP, can identify the sources that s/he can consult in order to obtain knowledge about the application domain of that subsystem. Moreover, if we use the extensions in their entirety, we can also assign the location in which it is possible to consult those sources, and the maintainer can thus also use the models to discover where they can be consulted.

The definition of the relationships between topics and sources was mainly carried out at a high level, similar to that in the model in Fig. 7. However, when appropriate, models can be developed in greater detail by, for instance, specifying which specific topics of a package are obtained from a specific document, people, or information system as can be seen in Fig. 8.

Figure 8 shows a document written by the maintainer in charge of SIDEP to describe the relationships between the modules of the system and their source files (ModulosSIDEP.doc). This document is a useful means by which to discover the structure of the system and its various modules, and the source files corresponding with each module. Writing such documents is not a norm in the process. However, while analyzing the models, the maintainers in charge of other systems realized that they had similar documents. Nevertheless, they did not usually know about the other documents, since they had been written for personal use, and were stored

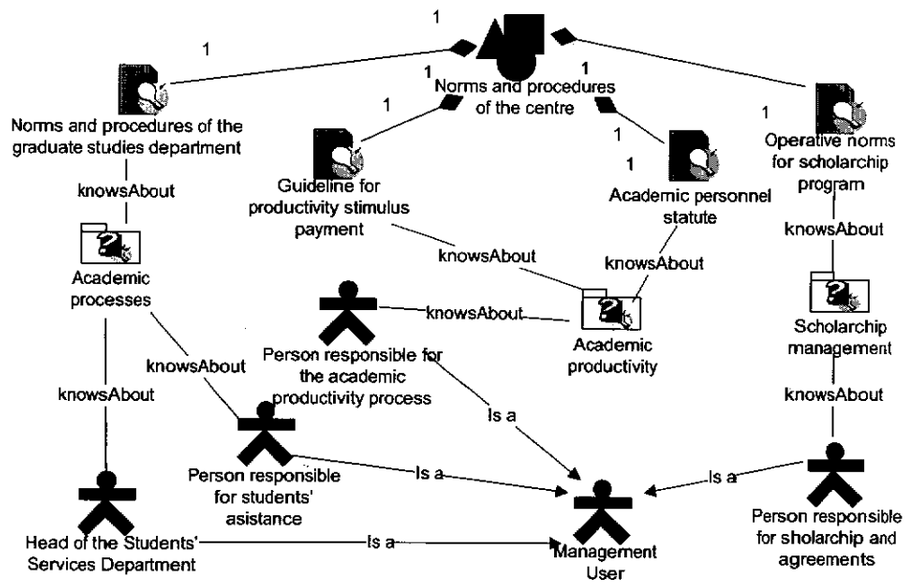


Fig. 7. Class diagram illustrating the relationships of knowledge topics and its sources.

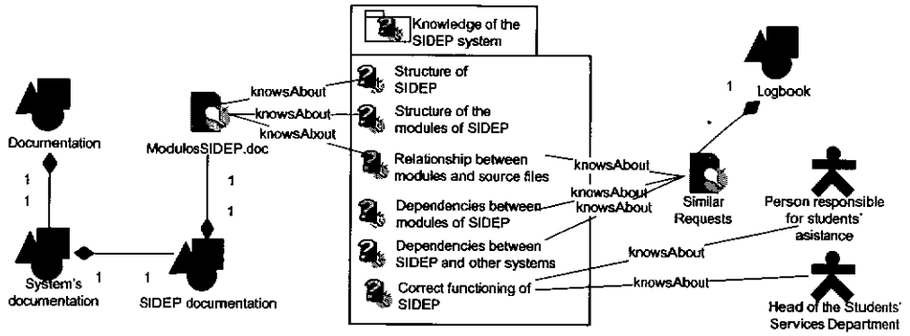


Fig. 8. Class diagram illustrating sources of specific knowledge topics.

in personal workspaces. Thus, explicitly representing the documents and the type of knowledge that can be obtained from them in the models was helpful for maintainers, as it allowed them to discover other documents which may also have been helpful in the attainment of similar knowledge. From this, we were able to integrate these types of documents into the classification structure of the process knowledge sources, thus making those documents and their location known to the whole group.

Additionally, the analysis of the knowledge required in the process activities, and the sources where that knowledge can be obtained or stored, should make the

initiation of the process through which to identify the mechanisms used to transfer knowledge between activities and/or sources easier, as is illustrated in the following subsection.

4.2.4. *Identifying knowledge flows*

We propose two types of diagrams for the analysis of knowledge flows, one to illustrate the activities in which related knowledge topics are used or generated, and the other to show transfers of knowledge between sources. To illustrate the use of those diagrams, we will use the knowledge associated with the maintenance requests, which are the main mechanism used as a knowledge flow channel in the ID.

The first step is to identify where important topics of knowledge are being used or generated within the process by identifying the activities in which this occurs. This can be done by developing the activities-knowledge transfer diagrams such as that shown in Fig. 9, which illustrates the activities in which the knowledge related to a maintenance request is being generated or used. This knowledge includes the initial data of the request (the user that made it, the date, the initial description

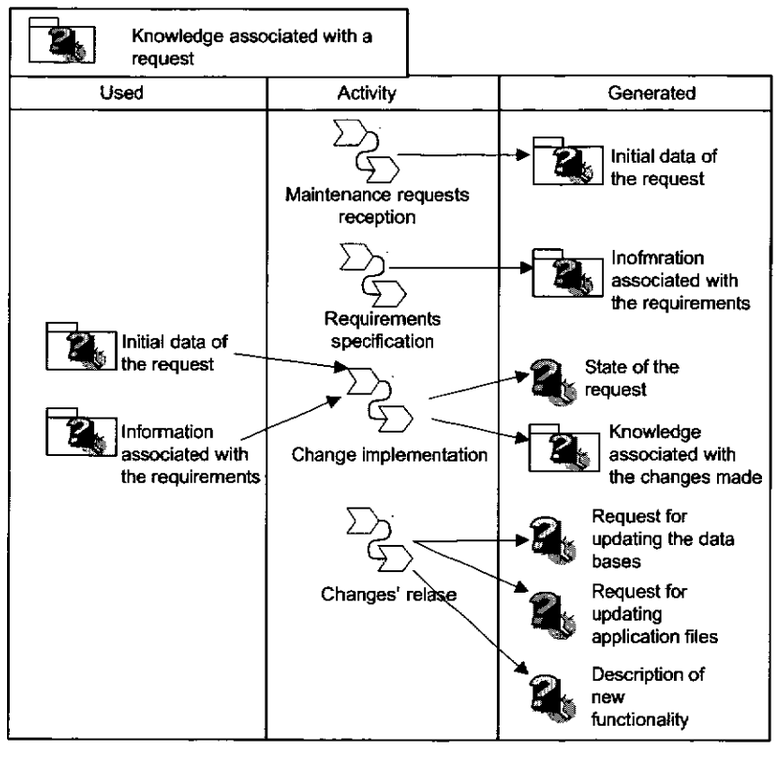


Fig. 9. Example of a knowledge transfer diagram (transfer between activities).

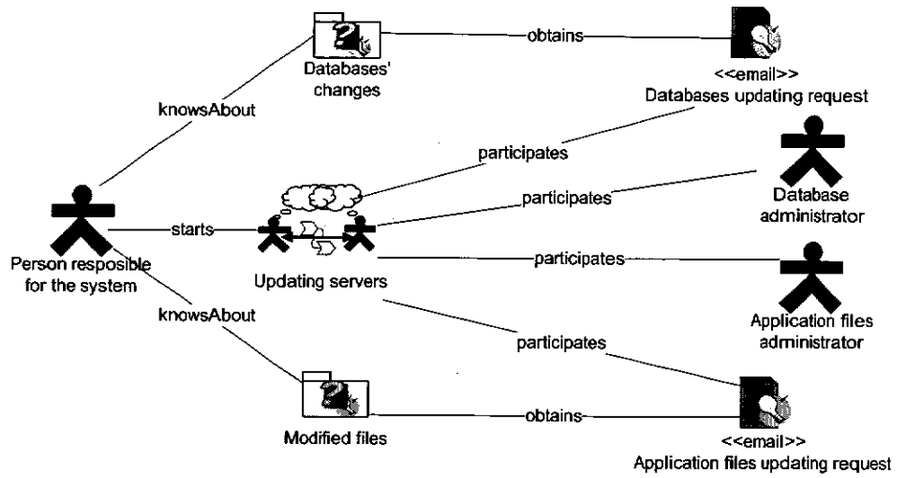


Fig. 10. Example of a knowledge transfer diagram (transfer between sources).

of the changes requested, etc.), some information related to the requirements of the requests (a more detailed description of the changes requested, documents associated with the requirements, such as forms, norms, memos, etc.), the state of the request during the process, and finally, knowledge about the solution given to that request (such as changes made, time consumed, etc.).

The second step is to identify the sources in which the knowledge is being stored or from which it is obtained. Diagrams such as that shown in Fig. 10 (knowledge sources transfer diagrams) can be used for this purpose. In the ID, the information associated with a maintenance request is stored in different sources. For instance, when a request has been addressed, the person responsible for the system sends emails requesting an update of the databases and application servers to the database administrator and to the files server administrator respectively. The applications and data accessible to the users of the systems are stored in these servers. These emails can thus be used to obtain information about the changes made to the databases, and the files that have been modified as the result of a specific maintenance request. The knowledge transfer diagrams illustrate this by defining the source that initiates the transfer, the participating sources, the knowledge being transferred, and the activity or workflow in which the transfer takes palace.

The knowledge transfer diagrams facilitate the identification of the mechanisms used as knowledge flow channels. This is an initial step towards the definition of KM strategies aimed at improving the use of the current infrastructure. For instance, in this study, it may be observed that part of the main knowledge involved in the maintenance process is that associated with a maintenance request. Although a part of such knowledge is stored in the requests which are part of the logbook system, not all the maintainers write the same information. A request may frequently

only contain the initial data. However, there are other sources that may be useful if we wish to obtain more associated information, such as documents related to the requirements, the emails for requesting the servers' updates, or users' manuals describing new functionalities.

By using the latter as a basis, we were able to recommend that the ID should modify its logbook in order to allow the inclusion of links to documents associated with each request. The goal was to enable the maintainers to access all the associated documents and information related to a request from the logbook, thus improving the use of the logbook as a knowledge flow facilitator. It will therefore be easier for maintainers to retrieve knowledge such as the effort that was required in each request, or what can be done to address a request that is similar to a previous one by, for instance, consulting the files that were required to be changed after the previous request.

As we have described, the extended SPEM diagrams were used in order to understand the maintenance process of the ID, and to identify the main knowledge involved, its sources, and the way in which the latter are being used by the actors of the process. From this analysis we have identified problems affecting the flow of knowledge, and have proposed solutions by which to address those problems and to improve the process. In the following section we shall present the lessons that were learned from the study, and some of the activities that we are now conducting as a result of this.

5. Lessons Learned from the Case Study

Here we draw on the lessons learnt from the application of the SPEM extension proposed here. It is, however, important to mention that the lessons presented are related both to the contributions of this study and to a previous one, in which a more flexible PML was used to analyze the process from a higher level of abstraction [53] and which was presented in Sec. 3.2. The lessons learned draw on two different points of view: that of the researchers and that of the practitioners (the ID perspective).

5.1. *Research perspective*

- In the first analysis we were unable to identify sequences and dependencies between activities and knowledge. With the extended SPEM we were able to identify such dependencies, and to define the activities in more detail. Although this was a tedious and time-consuming task, it enabled us to identify some important topics dealing with knowledge that were not identified in more general views of the process. These topics were useful in the recognition of important sources for the activities of the ID, such as the documents describing the structure of the maintained systems that the maintaners written for personal use (mentioned in the examples in Sec. 4.2). This was possible for two main reasons: firstly, in the present study we were performing a second analysis, and thus had previous knowledge. Secondly, SPEM is more formally-constrained and permits more detailed models than the first PML used. For instance, it was easier to identify a detailed structure of activities and sub-activities and to recognize the dependencies between the knowledge and sources which were created or modified in some activities and used in others.
- The first analysis was used in order to commence the definition of a knowledge map which thus aided us to classify the sources of knowledge, to know where they could be found, and to discover the knowledge that could be obtained from them [66]. However, using the explicit representation of knowledge topics and knowledge packages in the extended SPEM models has helped us to improve the structure and the content of this map, by improving the classification and structure of the knowledge topics, and by describing the dependencies between knowledge topics and between sources better.
- Additionally, the use of the SPEM extension helped us in the design of the internal structure of knowledge and sources which are managed by a tool that we are developing to aid SEs in accessing knowledge sources, when performing development or maintenance tasks. To do all this, we have explored the use of software agents to help identify knowledge sources which are useful for particular tasks being carried out by the SEs [45, 46]. The agents' communication language is based on an ontology of knowledge topics and sources designed by following a structure obtained from the process models. We have designed a prototype that uses this ontology and the knowledge map to identify relationships and dependencies between the activities, knowledge topics, and sources [53].

5.2. *Practitioner's perspective*

- The members of the ID are now aware of some of the problems they face in their maintenance process and are consequently developing tools to address some of these problems. They have, for instance, developed a web portal in which all the documents and information of the systems being maintained will be easily accessible.
- The logbook is being used more often as an important source of knowledge. When we conducted the study, not all the engineers used the logbook on a regular basis, but it is now a key part of their process. This might help the ID to maintain better control over the changes they make.
- The study has improved the sharing of knowledge among members of the ID. Through the analysis of the models, the members of the ID have become aware of sources of knowledge that they did not previously know existed, and these can be used to obtain important knowledge for their activities. For instance, the documents mentioned in the examples in Sec. 4.2 are now shared with the rest of the team.
- The ID personnel are now interested in formally describing their processes as a step towards the adoption of a standard software process model. They are interested in taking on a Mexican standard for small-to-medium-size software

development organizations which was approved by the Mexican government as a measure towards promoting the Mexican software industry [67]. This process model considers the integration of a knowledge base as an important part of the software process. The ID did not previously have a formal description of their processes. They discovered that the analysis of the extended SPEM models might be a useful aid in the formal description of their processes. The representation of knowledge and its sources in the models is of particular interest in aiding the definition of the knowledge base.

6. Discussion

A rigorous evaluation of the approach presented here is not easy since it should consider a formal comparison of our approach to others that may be similar. However, the modeling approaches that we have found in literature are quite different from ours, which makes carrying out such a comparison difficult. As was pointed out in Sec. 3.2, the other studies are not based on a software-focused PML and use very different types of diagrams and have different purposes.

In fact, none of the approaches used to analyze the knowledge involved in business processes that we have found in literature present formal evaluations. Based on this assumption, we therefore believe that it will be very difficult to compare results. The works in question limit themselves to exemplifying the use of the proposals in sample cases. Some, for instance [9, 10, 56, 58], provide the results of such applications, as we have done. Others, such as [31, 55, 59], propose tools for applying the approach. However, one of the reasons for using SPEM as a basis for our modeling approach is that its use does not require special tools. Finally, Strohmaier and Tochtermann [57] show the types of tools that can be developed from using their proposal. Nevertheless, our proposal is not focused upon facilitating the development of KM systems per se; it has been defined in order to facilitate the analysis of software processes from a KM perspective, thus enabling the identification of the mechanisms used as knowledge flow channels through which to improve the use of those mechanisms, following the results of studies which show that small and medium size companies must first see their processes in terms of knowledge, and improve the use of their actual infrastructure as a KM enabler, before developing or buying a new one [33].

Having compared our approach with those of literature, it is evident that the main contribution of our work is that we have illustrated the way in which a standard software-oriented PML can be adapted to the analysis of the knowledge involved in software processes, particularly for two purposes: to start the identification and classification of the main knowledge topics and their sources, and to identify the mechanisms used as knowledge flow facilitators. The first purpose contributes to one of the initial steps in many KM initiatives, which is that of developing taxonomies of knowledge topics and sources [37]. The second purpose focuses upon helping small and medium size software companies to initiate their KM efforts by first

starting to analyze their processes in terms of knowledge, and by later identifying the infrastructure they have, which may be a useful knowledge flow enabler. The case study presented in this paper helps illustrate the manner in which this was accomplished, showing that the goals of the study were fulfilled in the sense that now the ID has started seen that they actually perform KM in different activities, and that they have tools commonly used in their process that can be important knowledge flow enablers; for instance the logbook or the personal documents written by maintainers. Thereafter, we can assume that now the ID is seeing their processes from a KM point of view. We consider that this is an initial step towards the improvement and increment of KM practices in the group studied.

Another means by which to evaluate the approach might be by measuring the benefits of its application in real cases. Unfortunately, the benefits of KM initiatives can only be measured in the long run and are influenced by variables that are difficult to control, as has been pointed out by Anquetil *et al.* in [52]. Therefore, we have limited our work to applying the modeling approach in a case study in order to show its usability and some of the possible benefits of applying it; as was presented in Sec. 4.2. We have learned various lessons from this study, which are described in Sec. 5, and these have pointed us in the direction of further work.

7. Conclusions and Future Work

Since software processes are knowledge-intensive, knowledge should be considered as an important element of such processes. Software PMLs should include primitives to represent issues related explicitly to the knowledge in the process models. Few modeling approaches are able to refer to, for instance, the knowledge itself, its sources, or knowledge transfers between activities, roles, or knowledge sources, such as people, documents, software systems, etc. In this paper we have described how an extended version of SPEM can be used to analyze software processes from a knowledge perspective, and how useful it has been in detecting knowledge related problems faced by a software maintenance group, such as misused knowledge sources or knowledge flow channels. Additionally, analyzing software processes from a knowledge point of view can also help software organizations to identify the mechanisms they have at hand and which may be important knowledge flow enablers. We have, moreover, shown that the extension proposed can also aid in the classification of knowledge and sources, and the identification of their relationships, which may help in the development of a knowledge map of a process, all this being part of the initial steps towards the development of KM initiatives [37].

As future work we are planning to integrate a process-based development environment supporting SPEM models with a structure for developing knowledge maps, in a manner which will enable us to develop the structure and content of knowledge maps for software processes, through the SPEM models, and thereby integrate KM support into this environment. We wish to explore whether this can help in the design of a process-based maintenance environment which integrates KM strategies.

The work presented here is a starting point in this direction, and is part of a larger initiative focused on providing KM support for maintenance projects.

Acknowledgments

This work is partially supported by CONACYT under grant C01-40799 and scholarship 164739 provided to the first author, Mexico; the MECENAS project (PBI06-0024, Junta de Comunidades de Castilla-La Mancha, Consejería de Educación y Ciencia), the ESFINGE project (TIN2006-15175-C05-05, Ministerio de Educación y Ciencia [Dir. Gral. de Investigación]/Fondos Europeos de Desarrollo Regional [FEDER]), and by the CALIPSO network (TIN20005-24055-E, Ministerio de Educación y Ciencia), Spain.

References

1. A. Aurum, R. Jeffery, C. Wohlin and M. Handzic (eds.), *Managing Software Engineering Knowledge* (Springer, Berlin, 2003).
2. K. Haggie and J. Kingston, Choosing your knowledge management strategy, *Electronic Journal of Knowledge Management Practice* **4** (2003).
3. T. Dingsøyr and R. Conradi, A survey of case studies of the use of knowledge management in software engineering, *Int. J. Software Engineering and Knowledge Engineering* **12**(4) (2002) 391–414.
4. I. Rus and M. Lindvall, Knowledge management in software engineering, *IEEE Software* **19**(3) (2002) 26–38.
5. A. Tiwana, An empirical study of the effect of knowledge integration on software development performance, *Information and Software Technology* **46**(13) (2004) 899–906.
6. U. M. Borghoff and R. Pareschi (eds.), *Information Technology for Knowledge Management* (Springer, Berlin, 1998).
7. R. Maier and U. Remus, Defining process-oriented knowledge management strategies, *Knowledge and Process Management* **9**(2) (2002) 103–118.
8. O. M. Rodríguez, A. I. Martínez, A. Vizcaíno, J. Favela and M. Piattini, Identifying knowledge management needs in software maintenance groups: A qualitative approach, in *Proc. Fifth Mexican International Conference on Computer Science (ENC'2004)*, eds. R. Baeza-Yates, L. Marroquin and E. Chávez (Colima, México, 2004), pp. 72–79.
9. B. H. Hansen and K. Kautz, Knowledge mapping: A technique for identifying knowledge flows in software organizations, in *Proc. European Conference on Software Process Improvement (EuroSPI 2004)*, ed. T. Dingsoyr, pp. 126–137.
10. P. Bera, D. Nevo and Y. Wand, Unravelling knowledge requirements through business process analysis, *Communications of the Association for Information Systems* **16** (2005) 814–830.
11. M. S. Abdullah, I. Benest, A. Evans and C. Kimble, Knowledge modelling techniques for developing knowledge management systems, in *Proc. European Conference on Knowledge Management* (Dublin, Ireland, 2002), pp. 15–25.
12. T. H. Davenport and L. Prusak, *Working Knowledge: How Organizations Manage What They Know* (Harvard Business School Press, Boston, MA, 2000).

13. OMG, *Software Process Engineering Metamodel Specification (SPEM)*, 2002 [cited 2004 October 29]; Available from: http://www.omg.org/technology/documents/formal/spem.htm.
14. U. M. Borghoff and R. Pareschi, Information technology for knowledge management, *J. Universal Computer Science* **3**(8) (1997) 835–842.
15. M. Handzic, Why Is It Important to Manage Knowledge? *Managing Software Engineering Knowledge*, eds. A. Aurum *et al.* (Springer, Berlin, 2003), pp. 1–4.
16. M. E. Nissen, An extended model of knowledge-flow dynamics, *Communications of the Association for Information Systems* **8** (2002) 251–266.
17. C. Seaman, The information gathering strategies of software maintainers, in *Proc. Int. Conf. Software Maintenance (ICSM'2002)* (Montreal, Canada, 2002), pp. 141–149.
18. B. P. Lientz, Issues in software maintenance, *Computing Surveys* **15**(3) (1983) 271–278.
19. T. C. Lethbridge, J. Singer and A. Forward, How software engineers use documentation: The state of the practice, *IEEE Software* **20**(6) (2003) 35–39.
20. S. Dart, A. M. Christie and A. W. Brown, *A Case Study in Software Maintenance*, 1993, Software Engineering Institute, Carnegie Mellon University, Pittsburgh.
21. J. Singer, Practices of software maintenance, in *Proc. Int. Conf. Software Maintenance* (Bethesda, Maryland, 1998), pp. 139–145.
22. B. Curtis, H. Krasner and N. Iscoe, A field study of the software design process for large systems, *Communications of the ACM* **31**(11) (1988) 1268–1287.
23. R. Thomsett, The year 2000 bug: A forgotten lesson, *IEEE Software* **15**(4) (1998) 91–95.
24. O. M. Rodríguez-Elias, A. I. Martínez-García, A. Vizcaíno, J. Favela and M. Piattini, Identifying knowledge flows in communities of practice, *Encyclopedia of Communities of Practice in Information and Knowledge Management*, eds. E. Coakes and S. A. Clarke (Idea Group Publishing, Hershey, 2005), pp. 210–217.
25. B. Curtis, M. I. Kellner and J. Over, Process modeling, *Communications of the ACM* **35**(4) (1992) 75–90.
26. R. Conradi and L. Jaccheri, Process modelling languages, *Software Process*, eds. J. C. Derniame, B. A. Kaba and D. Wastell (Springer, Berlin, 1999), pp. 27–52.
27. A. Tiwana, *The Knowledge Management Toolkit: Practical Techniques for Building a Knowledge Management System* (Prentice Hall, 2000).
28. M. E. Jennex and L. Olfman, Assessing knowledge management success, *Int. J. Knowledge Management* **1**(2) (2005) 33–49.
29. T. A. Stewart, The case against knowledge management, *Business 2.0* **3**(2002)80.
30. M. Lindvall and I. Rus, Knowledge management for software organizations, *Managing Software Engineering Knowledge*, eds. A. Aurum *et al.* (Springer, Berlin, 2003), pp. 73–94.
31. M. E. Nissen and R. E. Levitt, Agent-based modeling of knowledge flows: Illustration from the domain of information systems design, in *Proc. Hawaii International Conference on System Science* (HICSS 2004).
32. S. T. Acuña, N. Juristo and A. M. Moreno, Emphasizing human capabilities in software development, *IEEE Software* **23**(2) (2006) 94–101.
33. J. Sparrow, Knowledge management in small firms, *Knowledge and Process Management* **8**(1) (2001) 3–16.
34. I. Richardson and C. G. von Wangenheim, Why are small software organizations different? *IEEE Software* **24**(1) (2007) 18–22.

35. S. García, C. Graettinger and K. Kost, *Proceedings of the First International Workshop for Process Improvement in Small Settings*, 2006, Carnegie Mellon, Software Engineering Institute, Pittsburgh, p. 309.
36. K. Wiig, *People-Focused Knowledge Management: How Effective Decision Making Leads to Corporate Success* (Elsevier, Amsterdam, 2004).
37. M. Rao (ed.), *Knowledge Management Tools and Techniques: Practitioners and Experts Evaluate KM Solutions* (Elsevier, Amsterdam, 2005).
38. K. Dalkir, *Knowledge Management in Theory and Practice* (Elsevier, Amsterdam, 2005).
39. T. C. Lethbridge, What knowledge is important to a software professional? *IEEE Computer* **33**(5) (2000) 44–50.
40. B. Kitchenham, D. Budgen, P. Brereton and P. Woodall, An investigation of software engineering curricula, *The Journal of Systems and Software* **74**(3) (2005) 325–335.
41. M. Polo, M. Piattini and F. Ruiz (eds.), *Advances in Software Maintenance Management: Technologies and Solutions* (Idea Group Publishing, Hershey, PA, 2003).
42. A. V. Mayrhauser and A. M. Vans, Program comprehension during software maintenance and evolution, *IEEE Computer* **28**(8) (1995) 44–55.
43. A. J. Ko, B. A. Myers, M. J. Coblenz and H. H. Aung, An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks, *IEEE Transactions on Software Engineering* **32**(12) (2006) 971–987.
44. J. Koskinen, A. Salminen and J. Paakki, Hypertext support for the information needs of software maintainers, *J. Software Maintenance and Evolution: Research and Practice* **16**(3) (2004) 187–215.
45. O. M. Rodríguez, A. Vizcaino, A. I. Martínez, M. Piattini and J. Favela, Using a multi-agent architecture to manage knowledge in the software maintenance process, *Lecture Notes in Artificial Intelligence* **3213** (2004) 1181–1187.
46. A. Vizcaíno, J. Favela, M. Piattini and F. García, Supporting software maintenance in web repositories through a multi-agent system, in *Proc. Int. Atlantic Web Intelligence Conference (AWIC'2003)*, eds. E. Menasalvas, J. Segovia and P. S. Szczepaniak (2003), pp. 307–317.
47. H. S. Nwana, Software agents: An overview, *Knowledge Engineering Review* **11**(3) (1996) 205–244.
48. K. M. Oliveira, N. Anquetil, D. M. G. M. Ramal and R. Meneses, Knowledge for software maintenance, *Proc. 15th Int. Conf. Software Engineering and Knowledge Engineering (SEKE'03)*, pp. 61–68.
49. M. G. B. Dias, N. Anquetil and K. M. D. Oliveira, Organizing the knowledge used in software maintenance, *J. Universal Computer Science* **9**(7) (2003) 641–658.
50. F. Ruiz, A. Vizcaíno, M. Piattini and F. García, An ontology for the management of software maintenance projects, *Int. J. Software Engineering and Knowledge Engineering* **14**(3) (2004) 323–349.
51. B. A. Kitchenham, G. H. Travassos, A. Mayrhauser, F. Niessink, N. F. Schneidewind, J. Singer, S. Takada, R. Vehvilainen and H. Yang, Towards an ontology of software maintenance, *J. Software Maintenance: Research and Practice* **11** (1999) 365–389.
52. N. Anquetil, K. M. de Oliveira, K. D. de Sousa and M. G. Batista Dias, Software maintenance seen as a knowledge management issue, *Information and Software Technology* **49**(5) (2007) 512–529.
53. O. M. Rodríguez, A. I. Martínez, J. Favela, A. Vizcaíno and M. Piattini, Understanding and supporting knowledge flows in a community of software developers, Lecture Notes in Computer Science **3198** (2004) 52–66.
54. A. Monk and S. Howard, The rich picture: A tool for reasoning about work context, *Interactions* **5**(2) (1998) 21–30.

55. R. Woitsch and D. Karagiannis, Process-oriented knowledge management systems based on KM-services: The PROMOTE approach, *Int. J. Intelligent Systems in Accounting Finance & Management* 11 (2002) 253–267.
56. H. Zhuge, Knowledge flow management for distributed team software development, *Knowledge-Based Systems* **15**(8) (2002) 465–471.
57. M. Strohmaier and K. Tochtermann, B-KIDE: A framework and a tool for business process-oriented knowledge infrastructure development, *J. Knowledge and Process Management* **12**(3) (2005) 171–189.
58. S. Kim, H. Hwang and E. Suh, A process-based approach to knowledge flow analysis: A case study of a manufacturing firm, *Knowledge and Process Management* **10**(4) (2003) 260–276.
59. G. Papavassiliou and G. Mentzas, Knowledge modelling in weakly-structured business processes, *J. Knowledge Management* **7**(2) (2003) 18–33.
60. J. Bézivin and E. Breton, Applying the basic principles of model engineering to the field of process engineering, *UPGRADE* **5** (2004) 27–33.
61. A. Abran, J. W. Moore, P. Bourque, R. Dupuis and L. L. Tripp (eds.), *SWEBOK: Guide to the Software Engineering Body of Knowledge: 2004 Version* (IEEE Computer Society, Los Alamitos, California, 2004).
62. O. M. Rodríguez-Elias, A. I. Martínez-García, J. Favela, A. Vizcaíno and J. P. Soto, Knowledge flow analysis to identify knowledge needs for the design of knowledge management systems and strategies: A methodological approach, in *Proc. 9th Int. Conf. Enterprise Information Systems (ICEIS): Special Session on Business Intelligence, Knowledge Management and Knowledge Management Systems* (2007), pp. 492–497.
63. IEEE, *STD 1074-1995: IEEE Standard for Developing Software Life Cycle Processes*, 1995.
64. M. Polo, M. Piattini and F. Ruiz, A methodology for software maintenance, *Advances in Software Maintenance Management: Technologies and Solutions*, eds. M. Polo, M. Piattini and F. Ruiz (Idea Group Inc., Hershey, 2003), pp. 228–254.
65. N. Chapin, J. E. Hale, K. M. Khan, J. F. Ramil and W.-G. Tan, Types of software evolution and software maintenance, *J. Software Maintenance and Evolution: Research and Practice* **13**(1) (2001) 3–30.
66. O. M. Rodríguez-Elias, A. I. Martínez-García, A. Vizcaíno, J. Favela and M. Piattini, Constructing a knowledge map for a software maintenance organization, in *Proc. Poster Session of the 21st IEEE International Conference on Software Maintenance* (ICSM 2005) (Budapest, Hungary, 2005), pp. 51–54.
67. H. Oktaba, MoProSoft: A software process model for small enterprises, in *Proc. First Int. Research Workshop for Process Improvement in Small Settings*, eds. S. García, C. Graettinger and K. Kost (Pittsburgh, PA, 2005), pp. 93–101.