Knowledge Management in Software Engineering Environments

Ana Candida Cruz Natali
Ricardo de Almeida Falbo

Computer Science Department, Federal University of Espírito Santo
Fernando Ferrari Avenue, CEP 29060-900, Vitória - ES - Brazil
{anatali, falbo}@inf.ufes.br

Abstract

Knowledge is one of the organization's most important value, influencing its competitiveness. One way to capture organization's knowledge and make it available to all their members is through the use of knowledge management systems. In this paper we discuss the importance of knowledge management in software development and we present an infrastructure to deal with knowledge management in software engineering environments (SEEs). This infrastructure is applied to manage product software quality knowledge in ODE, an ontology-based SEE.

Keywords: knowledge management, software engineering environments, ontologies, software quality.

1. Introduction

The demands on software development are increasing. Shorter time-to-market, better quality and better productivity are more and more goals to be achieved. To meet these requirements, software organizations have tried to better use one of its most important resource: the organizational software engineering knowledge. Historically, this knowledge has been stored on paper or in people's mind. Unfortunately, paper has limited accessibility and it is difficult to update [1]. Knowledge in people's mind is lost when individuals leave the company. Furthermore, in a large organization, it can be difficult to localize who knows some matter. So, knowledge has to be systematically collected, stored in a corporate memory, and shared across the organization [2]. To put knowledge sharing in practice, organizations should acquire knowledge from their members and formalize it to make it available on an organizational level. In this context, knowledge management systems can be very useful.

Knowledge management (KM) involves human resource, enterprise organization and culture, as well as the information technology, methods and tools that support and enable it [3]. A knowledge management system facilitates creation, access and reuse of knowledge, and its main goals are to promote knowledge growth, communication, preservation and sharing.

In the context of software development, KM can be used to capture the knowledge and experience generated during the software process. Although every software development project is unique in some sense, similar experiences can help developers to perform their activities. Reusing knowledge can prevent the repetition of past failures and guide the solution of recurrent problems. So, to be effective, a knowledge management system should be integrated to the software process. Since Software Engineering Environments (SEEs) integrate collections of tools supporting software engineering activities across the software lifecycle [3], it is natural to integrate KM facilities in a SEE.

In this paper, we propose a knowledge management infrastructure to enable KM in SEEs, which considers knowledge capture, store, retrieval, dissemination, reuse and maintenance. Section 2 discusses knowledge management, and why it can be better addressed in SEEs. In section 3, we present the KM infrastructure proposed. Section 4 shows how this infrastructure was developed in ODE, an ontology-based SEE. A case study using this infrastructure in the software quality domain is presented in section 5. Section 6 discusses related works. Finally, in section 7, we report our conclusions.

2. Knowledge Management and Software Engineering Environments

Success in an increasingly competitive marketplace depends critically on the quality of the knowledge, which organizations apply to their business processes. The challenge of using knowledge to create competitive advantage becomes more crucial as [1]:

- The rate of innovation is rising, so that knowledge must evolve and be assimilated at an ever faster rate;
- There is a need to replace the informal knowledge with formal methods aligned to organization processes;
- Competitive pressures are reducing the size of the workforce which holds this knowledge;
- Knowledge takes time to experience and acquire. Employees have less time for this;

In response to these needs, knowledge management (KM) has been used. There is not a unique definition for knowledge management, but according to Benjamins *et al.* [4], knowledge management is not a product in itself, nor a solution that organizations can buy off-the-shelf. It is a process implemented over a period of time, which has much to do with human relationships as it does with business practices and information technology. Thus, KM combines tools and technologies to provide support to the capture, access, reuse and dissemination of knowledge, generating benefits for the organization and their members.

Before deciding how to manage knowledge, it is essential to understand what knowledge is. According to Markkula [5], knowledge is information combined with experience, context interpretation and reflection. It is a high-value form of information that is ready to apply in decisions and actions.

Knowledge can be viewed as formal and informal knowledge [2]. Formal knowledge can be expressed in a structured form, and easily communicated and shared. Formal knowledge includes software engineering methods, document templates, components, software artifacts, and so on. Informal knowledge is highly personal and hard to formalize, making it difficult to share with others. It is embedded in an individual experience and involves intangible factors such as personal belief, perspective and value. Examples of informal knowledge are discussions and lessons learned.

In the context of software development, lessons learned are one of the most important informal knowledge. Lessons learned are gained as a result of the work of the organization itself. They may describe both successful reports and problems. Successful lessons capture positive responses to crisis. Problem lessons address things that went wrong, and potential ways to solve the problem [1]. Reuse of lessons learned from past software projects promotes good software development practices and prevents the repetition of mistakes.

An efficient knowledge management approach must be able to model, capture and support the creation and use of all types of knowledge described above. One of the problems to be addressed is the fact that no software project is like another. Experience items matching the reuse needs are rarely found. Therefore, a good reuse approach must find similar experience items and let modifications on selected items.

2.1 Knowledge Management Process and Technologies

A knowledge management system should support the activities that comprise a knowledge process. According to Staab *et al.* [6], a knowledge process involves the following steps:

- *Creation:* The contents need to be created or converted, so that they fit the conventions of the company. Creation of computer-accessible knowledge typically moves between the formal and informal knowledge. It is also possible to import knowledge. Importing knowledge items into the KM system has the same or more importance than creating them. For imported knowledge, accurate access to relevant items plays an even more important role than for homemade knowledge. For homemade knowledge items, people might act as a backup index, but it is not the case for recently imported knowledge that no one has yet seen.
- *Capture:* Once you create knowledge items, the next step is to capture their essential contents. Knowledge items have to be captured in order to determine their importance and how they mesh with the company's vocabulary conventions.
- *Retrieval and access:* This step satisfies the searches and queries for knowledge by the knowledge worker and dissemination of knowledge in a proactive manner.
- *Use:* The knowledge worker will not only recall knowledge items, but will process them for further use. Many KM systems assume that once some relevant document is found, everything is done. Eventually, however, the way to use knowledge from the organization's collective memory becomes quite involved. Topics such as proactive access, personalization, and in particular, tight integration with user task play a crucial role for the effective reuse of knowledge.

To support the KM process, knowledge management systems should facilitate knowledge access and reuse. To do that, several emerging technologies, such as ontologies, XML and software agents, have been applied.

In order to facilitate communication and information exchange, a community may define a standard domain-oriented vocabulary using ontologies [7]. According to Uschold [8], an ontology may take a variety of forms, but necessarily it will include a vocabulary of terms, and some specification of their meaning. This includes definitions and indications of how concepts are inter-related, which collectively impose a structure on the domain and constrain the possible interpretations of the terms. Ontologies are particularly important for KM. They constitute the glue that binds KM activities together, allowing a content-oriented view of KM [6]. Ontologies define the shared vocabulary used in the KM system to facilitate communication, integration, search, storage and representation of knowledge [4]. Typical utilization scenarios comprise discussion groups, search engines, information filtering, access to non–textual information objects, and expert–user communication [9]. In these applications ontologies serve as "specifications of discourse in the form of a shared vocabulary" [9]. This "shared understanding" seems to be particularly important for knowledge management which typically deals with multi–actor scenarios.

Another interest of ontologies is their exploitation for guiding search of knowledge items. First, we have to consider that organizational knowledge must be annotated with information related to the particular ontology. Using XML (*eXtensible Markeable Languague*), it is possible to annotate a knowledge item with metadata, which describe it according to predefined organization's ontologies [7]. With annotated knowledge, searching for a specific knowledge item is made easier.

Software agents can be used to connect organizations' members to knowledge available [1]. Agents can help not only on knowledge search, but also on knowledge filtering and dissemination. If a software process is defined, agents can act in a proactive manner, searching and offering knowledge items that may be relevant for the developer's current task.

However, we must enhance that these new technologies do not create knowledge and cannot guarantee or even promote knowledge sharing in an organization which culture does not favor those activities [10]. A "knowledge-friendly" culture is one of the most important factors for the success of knowledge management [5].

2.2 KM in Software Engineering Environments

Software development is a collective, complex, and creative effort. As such, the quality of a software product heavily depends on the people, organization, and procedures used to create and deliver it. In other words, there is a direct correlation between the quality of the software process and the quality of the software developed [11]. Based on that, researchers and practitioners have been paying increasing attention to understand and improve the quality of the software process. But, to deal with complex software processes, it becomes essential to provide computer-based tools to support software engineers to perform their tasks.

Although benefits can be derived from individual CASE tools addressing separate software engineering activities, the real power of CASE can be achieved only through integration [12]. The identification of the need for integrated support for these activities throughout the software lifecycle represents the genesis of Software Engineering Environments (SEEs) [13]. Thus, SEEs can be defined as integrated collections of tools that facilitate software engineering activities across the software lifecycle [13].

But knowledge management can also be used to support developers during the software process. Using a KM approach, knowledge created during software process can be captured, stored, disseminated, and reused, so that better quality and productivity can be achieved. KM can be used to better support management activities, such as software process definition [2], people allocation and estimation, construction activities, such as requirement analysis and test case design, and quality assurance activities, such quality planning and control. Consequently, SEEs and knowledge management complements each other in supporting developers during the software process to produce better quality software.

3. An Infrastructure for Knowledge Management

To support the knowledge management process in a SEE, a KM infrastructure should be provided. The *corporate* or *organizational memory* (OM) must be at the core of this infrastructure, supporting knowledge sharing and reuse. Arranged around the OM, knowledge management services shall actively provide useful information to users working on knowledge-intensive tasks [14]. These knowledge management services correspond to the activities of the knowledge management process: creation, capture, retrieval, access, dissemination, use, and preservation of the organization's knowledge, as shown in Figure 1.

The primary requirement for an OM is to prevent the loss and enhance the accessibility to organizational knowledge by providing a centralized, well-structured knowledge repository. Since workers are often too busy to look for information or do not even know that relevant information exists, proactive services must be provided, actively reminding workers of helpful knowledge. Thus, knowledge distribution may be passive or active, as either the user can search for the required information, or the KM system itself can offer knowledge that seems relevant to the user's task [14].

To gain user acceptance, a KM system must be integrated into the organization's process, allowing to collect and store relevant knowledge as they are generated in the work. Consequently, it should be also integrated to the existing work environment [14].

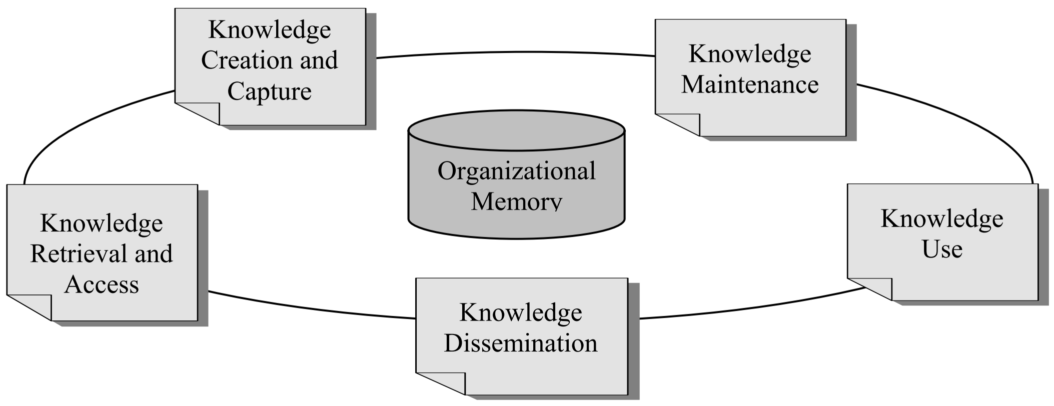


Figure 1 - Knowledge management infrastructure.

The KM system is to be an assistant to the user, supplying him with relevant organizational knowledge, but leaving him the responsibility of a contextual interpretation and evaluation of this information. In this context, to keep an OM up to date, it is important to get feedback from its users, who must be enabled to point out deficiencies and suggest improvements without significantly disrupting their usual workflow. Therefore, user feedback is essential for OM maintenance and evolution [14].

Even though the advantages of having an OM are generally recognized, organizations are reluctant to invest time and money into a novel technology whose benefits are distant and uncertain. Thus, a KM system must exploit readily available knowledge, provide benefits quickly, and be adaptable to newly arising requirements.

4. Knowledge Management in ODE

As pointed above, a KM system should be integrated into the organization's process and into its work environment. In the context of software development, this environment is a Software Engineering Environment (SEE). The main advantage of integrating knowledge management into a SEE is that KM is put into software engineers' workflow, since software development activities occur inside the computational environment rather than in the external world.

We have tried this integrated approach to KM in ODE (Ontology-based Development Environment) [15], a Process-Centered Software Engineering Environment that integrates CASE tools into a cohesive environment, each one supporting a software process activity, and working together to build the product during the entire software process.

ODE is being developed at LabES/UFES. Its main feature that distinguishes it from other SEEs is that ODE is developed based on ontologies. ODE uses some defined ontologies, such as a software process ontology [16] and a software quality ontology [17, 18], as its basis for integration. In its current stage, ODE has several integrated tools and its integration approach considers the following issues:

- data integration: the way tools share data;
- process integration: linkage between the tools and the software development process;
- control integration: the ability for one tool to notify and initiate actions in another;
- presentation integration: commonality of user interface;

ODE's design premise is based on the following argument: if the tools in a SEE are built based on ontologies, tool integration can be improved. The same ontology can be used for building different tools supporting correlated software engineering activities. Moreover, if the ontologies are integrated, integration of tools built based on them can be highly facilitated. However, the integration problem is not solved yet. Knowledge integration should be considered to provide knowledge management support and to evolve ODE to what we are calling a Semantic SEE [15].

A Semantic SEE can be viewed as a SEE in which part of the information handled has a formal meaning (semantics) associated, augmenting its tools' ability to work in cooperation with each other and with human developers. Tools committed themselves with an ontology can share knowledge, since the ontology defines the common meaning. The term "Semantic SEE" was coined using an analogy with Semantic Web [19]. Semantic Web aims to organize Web information, adding meaning to them, and allowing machines to process and analyze Web contents. The main goal of a Semantic SEE is analogous: to organize software engineering information, adding meaning to them, and allowing tools to share information. In a Semantic SEE, software engineering knowledge is accessible not only to human developers, but also to automated tools. Adapting the discourse of Bechhofer et al. [19] to our context, the key idea is to have software engineering data on the SEE defined and linked in such a way that its meaning is explicitly interpretable by software tools rather than just being implicitly interpretable by human developers.

ODE's architectural style reflects its basis on ontologies. It has two levels. The base or application level concerns application classes, which model the objects that address some software engineering activity. The meta-level (or knowledge level) defines classes that describe knowledge about objects at the base level. Figure 2 shows these two levels concerning software process integration and quality control [15].

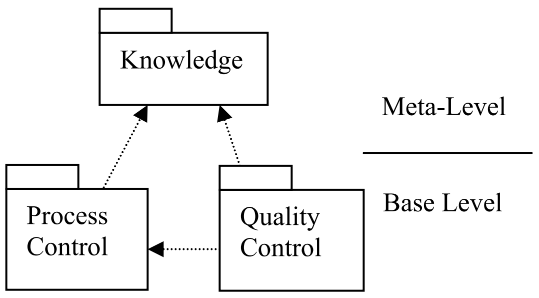


Figure 2 - ODE's two-layered architecture.

The classes at the meta-level are derived directly from the ontologies, using the systematic approach to derive object frameworks from ontologies described in [20]. All classes derived directly from the ontology are prefixed by the character "K", indicating that they constitute the knowledge in ODE. We can view the meta-level objects as items of an ontology instantiation [15].

The classes in the base level are also built based on the ontologies. The main classes and associations are derived from the ontology, preserving the same constraints as Knowledge's model. Also several classes at the base level have a corresponding Knowledge class in the Knowledge package. In this way, the meta-level can be used to describe base-level objects' characteristics. However, since an ontology does not intend to describe all the knowledge involved in a domain, but only that one that is essential to conceptualize the domain (minimal ontological commitment [21]), new classes, associations, attributes and operations are defined to deal with specific design decisions made in the application level. In fact, the ontology is a general, common sense model, and thus it does not contain all necessary modeling elements to treat applications' requirements [15].

In the context of the knowledge management, ontologies define the shared vocabulary used in the KM system to facilitate communication, search, storage, and representation. Ontologies constitute the glue that binds knowledge subprocesses together. Ontologies open the way to move from a document-oriented view of KM to a content-oriented view, where knowledge items are interlinked, combined, and used [6]. In ODE's knowledge management approach, ontologies are used to structure the OM, as well as to support the main knowledge services, such as search and reuse of knowledge items.

ODE's organizational memory contains three types of knowledge: *artifacts, instances of ontologies and lessons learned*. Artifacts and instances of ontologies correspond to the formal knowledge. Lessons learned are the informal knowledge. The OM holds information from previous projects so that users can use them to solve similar problems and to perform similar tasks.

The knowledge management approach adopted in ODE follows the one described in section 3. At the core of the knowledge infrastructure, there is an OM, supporting knowledge sharing and reuse. As shown in Figure 1, arranged around the OM, there are services supporting the following knowledge management activities:

- *Knowledge Capture*: Since ODE deals with three kinds of knowledge, it must offer facilities to capture each one of these type:
    - When dealing with lessons learned, we have to consider that project-level knowledge can be useful, but it is not always the case. Generally, project-level knowledge must be handled to become an organizational knowledge. A tool supporting a workflow for approving a lesson learned was developed in ODE. First, a developer inputs a lesson learned in the OM. At this moment, this knowledge is not available for other developers. The knowledge manager must evaluate and adapt the lesson learned so that it can be considered knowledge at the organizational level. Once approved, the lesson learned is made available.
    - The knowledge manager is responsible for creating the instances of the ontologies that are useful to the organization. In ODE, for each ontology, there is a tool supporting its instantiation.
    - Finally, artifacts created during the software process must also be available as knowledge items. Artifacts must be submitted to configuration management. ODE has a prototypical configuration management system that controls not only artifacts produced by ODE's internal tools but also artifacts from external tools that are put under version control. So, in the current stage, the ODE's configuration management system is the base for dealing with artifacts as knowledge items.
- *Knowledge Search*: Knowledge management in ODE supports information access through searching. An ODE user can search for any kind of knowledge in the OM: formal knowledge (artifacts and ontology instances) or informal (lessons learned).
- *Knowledge Dissemination*: While knowledge search is a user-initiated search, knowledge dissemination is initiated by the system, without requiring the user to explicitly formulate a query. Software agents monitor the users' actions as they work and inform them about potential relevant knowledge. Users can browse the various knowledge items and then select and reuse one of them. Knowledge dissemination is particularly important when users are not motivated to look for information or when they are not aware of the need for information in the first place.
- *Knowledge Use*: Once a knowledge item is selected for use, the user can identify what part he/she wants to use and a new knowledge item is created based on the previous one. Some reuse information is shown, including when and how often this item has been used and who used it. Finally, the user must evaluate the reused item to help knowledge maintenance. It includes evaluation information about if the item was useful, problems that appeared when reusing it, and solutions which have been applied.
- *Knowledge Maintenance*: For maintenance and evolution of the OM, it is necessary to take into account users' feedback. Based on the user feedback, the knowledge manager can decide what knowledge item is obsolete or which one had never been

used. The knowledge manager can exclude knowledge items by himself or can require the support of a software agent. To realize theses tasks, the knowledge manager has an interface to search for knowledge items, to exclude them, and to configure a software knowledge maintenance agent. The software agent can be set to alert the knowledge manager to realize an OM's maintenance at defined time intervals or when the OM has reached a defined size. The software agent can also suggest some knowledge items to be excluded based on knowledge manager criteria.

The knowledge management approach proposed is to be applied to the entire SEE, and not only to one of its tools. But to illustrate our approach, in the next section, we focus on ControlQ, a tool that supports software quality planning and tracking. So, we discuss software quality knowledge management. We should emphasize, however, that this does not mean that we are restricted to this scenario. We are also using this approach, for example, to treat software process definition, resource allocation and estimation in ODE. Thus, not only software quality knowledge can be managed in ODE, but all the knowledge created by an ODE's tool.

5. Software Quality Knowledge Management in ODE

To support software quality planning and tracking in ODE, we developed ControlQ. ControlQ's functionalities include:

- quality characteristic and metric knowledge management;
- quality planning, allowing to define quality evaluation activities that will be carried along the project. The quality manager defines for each one of these activities: *when* and *what* will be evaluated, which *quality characteristics* will be evaluated and from which *metrics* these characteristics will be computed;
- quality control, allowing to register the measurement results.

ControlQ was developed based on ODE's architectural style, which reflects its basis on ontologies. Based on ODE's two-layered architecture, the tool architecture was composed of two packages: Knowledge package, shown in Figure 3, and Quality Control package, shown in Figure 4.

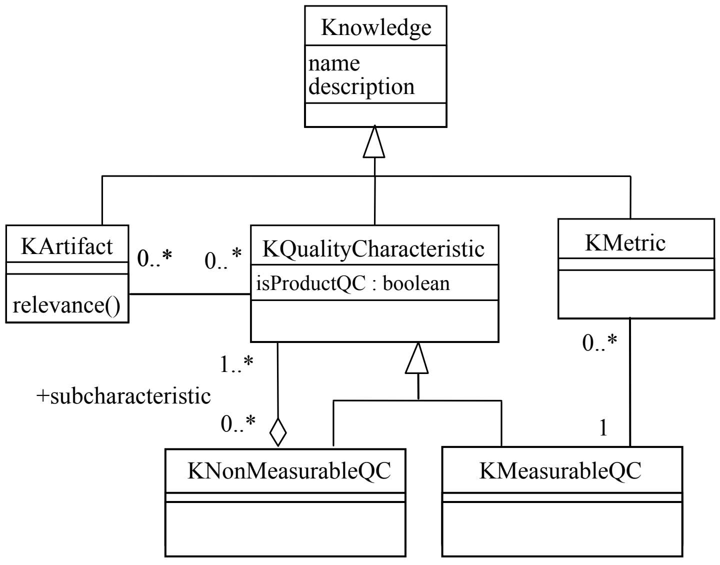


Figure 3 - Part of the Knowledge Package.

The *Knowledge* package directly reflects the concepts of the ontology, representing the common knowledge of this domain. Its classes were derived from the software quality ontology developed in [17], using a systematic approach to derive object frameworks from ontologies. However, to support quality planning and control, other classes are necessary beyond those shaped. To address the specific ControlQ's requirements, we developed the *Quality Control* package. The classes of this package represent specific concepts of the application, necessary to accomplish its goals.

As shown in Figure 4, a *quality control plan* defines all *quality evaluation activities* of a project. Theses activities define not only *what* will be evaluated (an artifact), but also *how* this evaluation will occur, i.e. which *quality characteristics* will be used to evaluate the artifact.

A *non measurable characteristic* must be decomposed into *subcharacteristics* to be computed by the aggregation of their *subcharacteristic* measures. For each one of these *subcharacteristics,* it is necessary to define its *weight* in the measurement. A *measurable characteristic* can be directly measured choosing a *metric* to quantify it. For each *choice*, indicating which *metric* will be used to quantify each *measurable characteristic*, the corresponding measure value is stored.

We can notice that the Quality Control Package requests services from the Knowledge Package. It is not only an incident. In fact, this two-layered architectural style is the basis of ODE architecture. The application level concerns application classes, which address the application requirements. The knowledge level defines domain knowledge, which can be used by several applications.

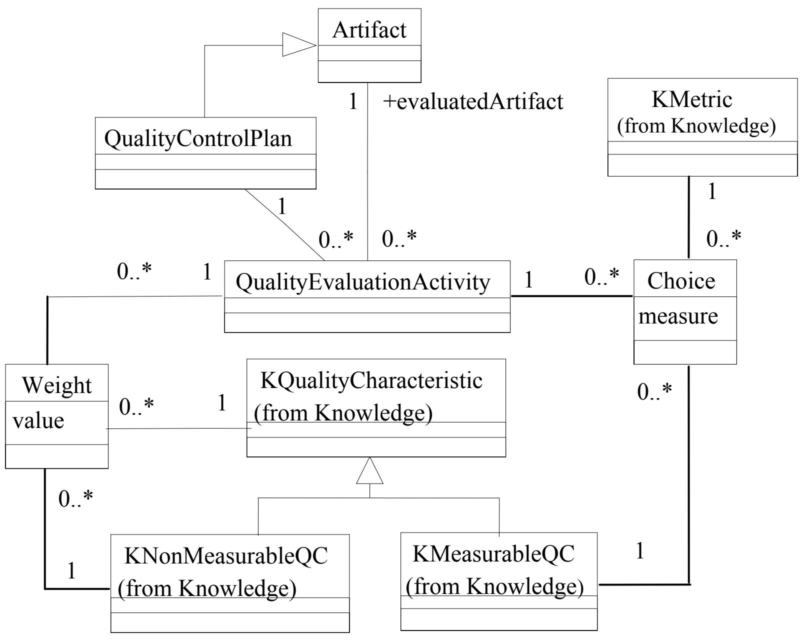


Figure 4 - Part of the Quality Control Package.

Quality planning in ControlQ involves the following steps:

1. Select a project to which a quality plan will be created and define the project's software artifacts that will be evaluated;
2. For each artifact, identify which quality characteristics will be used to evaluate it.
3. For each non measurable quality characteristic, define how it is decomposed into subcharacteristics;
4. Define how to measure the identified measurable characteristics, choosing adequate metrics.
5. Define quality evaluation activities, integrating them into the software process. For each artifact defined in step 1, a set of quality evaluation activities is defined.

After defining a quality control plan, we can go to the next step: *evaluation,* that is, *measurement.* In the measurement phase, the selected metrics will be applied and, for each one of them, values are informed and registered. Measurement phase is followed by *result* *presentation*, showing a report of obtained results. The analysis of these results aids the definition of corrective actions to achieve the desired quality.

5.1 Knowledge Management in ControlQ

Software quality knowledge management can aid quality managers to perform similar quality planning activities. Reuse of past experience may avoid the repetition of mistakes in those activities. So, ODE's KM approach should be used to support quality control. Next, we present how this approach, described in section 4, was applied in ControlQ.

Software Quality Knowledge Capture

As mentioned earlier, ODE's organizational memory manages three types of knowledge: instances of an ontology, artifacts, and lessons learned. In the case of quality control, ODE's OM stores instances of a software quality ontology [18], the artifacts managed are quality control plans, and lessons learned considered are those gained during quality planning and evaluation. Consequently, ODE's KM system must support the capture of each one of these knowledge types.

Software Quality Knowledge Search

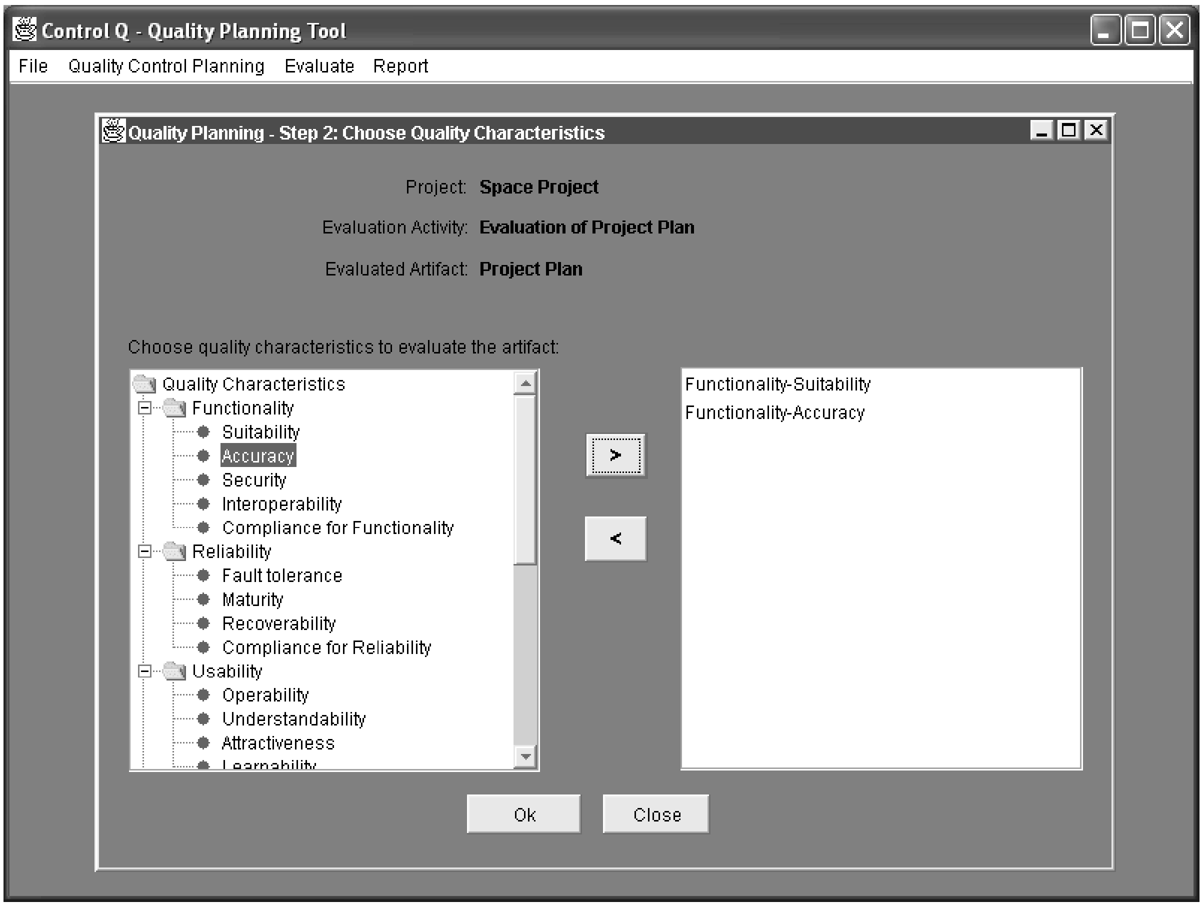
As a project manager performs a quality planning or evaluation, he/she can search for any kind of knowledge existing in the organizational memory. This search is a user-initiated search, since he/she has to define his/her needs (what knowledge he/she wants). These needs become a query, and knowledge items retrieved are presented. For example, a project manager can search for a lesson learned involving the choice of a quality characteristic to evaluate a kind of artifact. The retrieval machine will search lessons learned which refer to user's specified characteristic and artifact. Users can also search, for ontology instances or quality control plans.

Software Quality Knowledge Dissemination

Since ControlQ is defined based on a software quality ontology [17], ontology instances are used to support quality planning activities. A Knowledge manager is responsible for instantiating the ontology. These instances are stored in ODE's organizational memory, and they are used to support some steps of ControlQ's quality planning, such as:

- Defining which quality characteristic can be used to evaluate a specific artifact (step 2): as shown in Figure 5, based on the predefined ontology instances, ControlQ presents only those quality characteristics that are considered to be useful to evaluate an artifact;
- Decomposing a non measurable quality characteristic into subcharacteristics (step 3): again, based on OM's knowledge, ControlQ presents only those quality characteristics that can compose a specific non measurable characteristic;
- Defining which metric is to be used to quantify a measurable quality characteristic (step 4): as Figure 6 shows, ControlQ presents only those metrics that can be used to evaluate a specific measurable quality characteristic.

For these steps (2-4), ODE's knowledge management system can also play an active role in knowledge dissemination. Software agents monitor users actions as they work in ControlQ. When the user is working in one of those steps, specific software agents act, identifying user's knowledge needs, and retrieving past and similar experiences. These agents disseminate lessons learned that relate to success or failure. Also, they disseminate other quality control plans already defined and evaluated. So, based on similar experiences, users can make decisions based not only on their own knowledge but also on organizational knowledge.


Control Q - Quality Planning Tool
File
Quality Control Planning
Evaluate
Report
Quality Planning - Step 2: Choose Quality Characteristics
>

Project: **Space Project**

Evaluation Activity: **Evaluation of Project Plan**

Evaluated Artifact: **Project Plan**

Choose quality characteristics to evaluate the artifact:

- Quality Characteristics
  - Functionality
    - Suitability
    - Accuracy
    - Security
    - Interoperability
    - Compliance for Functionality
  - Reliability
    - Fault tolerance
    - Maturity
    - Recoverability
    - Compliance for Reliability
  - Usability
    - Operability
    - Understandability
    - Attractiveness
    - Learnability

Functionality-Suitability
Functionality-Accuracy

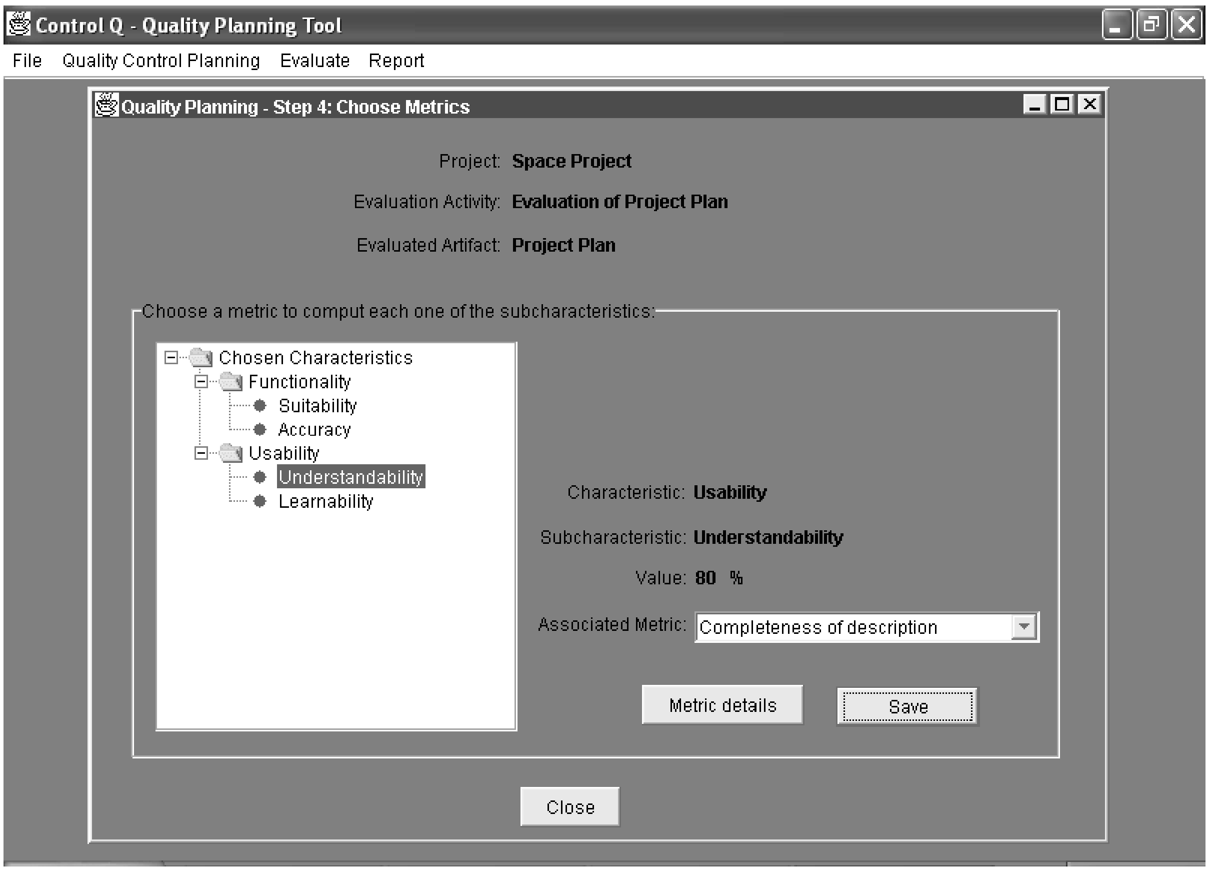
Figure 5 – Choosing quality characteristics to evaluate an artifact.

Software Quality Knowledge Use

Knowledge items retrieved are presented to the project manager. He/she can browse through this set of knowledge items and choose a knowledge item to reuse. If a quality control plan is selected for reuse, he/she must identify which part he/she wants to use in his/her own artifact. From a quality control plan, a project manager can reuse one of its defined evaluation activities or a choice made of characteristics and metrics to evaluate a project's artifact. If a software quality knowledge item is reused, the user must evaluate its content, creating a lesson learned related to this knowledge item.

Software Quality Knowledge Maintenance

The maintenance of software quality knowledge is performed in the same way of maintaining other kind of knowledge, since all of them are stored in the same organizational memory. So, based on user feedback, the knowledge manager can decide what knowledge item is obsolete or which one has never been used. These knowledge items are excluded, as previously described.



Project: **Space Project**

Evaluation Activity: **Evaluation of Project Plan**

Evaluated Artifact: **Project Plan**

Choose a metric to comput each one of the subcharacteristics:

Characteristic: **Usability**

Subcharacteristic: **Understandability**

Value: **80 %**

Associated Metric: Completeness of description

Figure 6 - Choosing metrics to quantify each measurable quality characteristics.

ControlQ is a good choice to exemplify our knowledge management approach, because this tool has an ontology defined to support their activities, representing the common knowledge of this domain. Another benefit of using ControlQ as our example is the fact that it automates a knowledge intensive task (quality control). Finally, the main output of this task is a quality control plan, which parts can be considered knowledge items in organizational memory, and then can be reused and disseminated.

We presented the benefits of a KM-based approach to support quality planning in ControlQ. In the other hand, if ControlQ was not supported by knowledge management, some drawbacks would occur. Without knowledge management, lessons learned during software quality planning in ControlQ would be missed or stored in people's mind. For instance, in software quality plan definition, the choice of the same metric to quantify a software quality characteristic is not always the better choice. Depending on the artifact to be evaluated and the quality characteristic, a different metric would be more appropriated to quantify this characteristic. In addition, defining which quality characteristics are to be used to evaluate an artifact and also, in which subcharacteristics they should be decomposed, are not simple tasks. Executing these tasks would be harder if ControlQ's user would have to choose by himself, without any support provided by the tool.

Experience in quality planning and tracking can only be achieve in practice, that is, defining quality plans and evaluating them. Offering KM-based support to those activities is a way to improve learning. For this reason, ControlQ captures knowledge created in software quality planning, stores and disseminated it, even in a proactive manner, offering similar experiences and knowledge that can be reused. When knowledge is available, it is not necessary to construct this knowledge again, trying a solution for a problem that has already been solved. Consequently, reusing knowledge helps to prevent the repetition of past failures and guide the solution of recurrent problems.

6. Related Work

Most organizations agree that knowledge is an essential asset for success and survival in an increasingly competitive and global market. This awareness is one of the main reasons for the exponential growth of knowledge management research.

Several works have exploited the use of KM systems to support software engineering tasks, such as [2, 5, 19]. Borges et al. [2] store and share the experience obtained in software process definition. To share this knowledge, an experience repository was built, containing the organizational standard process as well as the artifacts and informal knowledge obtained throughout the projects. In order to facilitate the storage and sharing of the experience, they built ProKnowHow, a tool that supports the standard software process tailoring procedure for each project, providing KM support.

Markulla [5] describes an initiative at ICL Finland to promote software engineering knowledge sharing and reuse. The focus is on supporting development tasks, such as planning, design and coding. A framework has been developed for creating, capturing, storing, sharing and applying tacit and explicit knowledge in project and organizational levels.

Althoff et al. [22] defend that continuous reuse of software engineering experience can be supported by an organizational memory that is capable to manage all kinds of software engineering experiences. They propose a generic, scalable architecture and an underlying methodology for reuse of all kinds of software engineering experience.

In [23], a system for supporting experience management in a multinational software improvement consultancy called Q-Labs is presented. The objective is to provide a "virtual office" for Q-Labs, and to allow Q-Labs consultant to benefit from the experience of every other Q-Labs consultants.

Looking to these works, we can find many common points. All of them, including ours, are based on the concept of Experience Factory [24]. An experience factory is an organizational unit that supports reuse of experience and collective learning by developing, updating and providing, on request, past experiences to be used by project organizations. However, none of them is integrated to a Software Engineering Environment (SEE), and none offers support for quality planning. Thus, it is worth to remember that this work was developed in the context of ODE, an ontology-based SEE. The remarkable feature of our work is proposing a KM approach actively integrated into the work process and social practices of a SEE. So, a major concern is to capture information from the work process without extra effort for developers who can receive knowledge from an active OM.

Observing structural aspects of a KM system, we also find many related works in the literature. Abecker et al. [14] defined a knowledge management approach with an organizational memory at the core of the KM system. Arranged around such an organizational memory, knowledge-management services provide actively knowledge to users. Our approach shares many of the definitions proposed by them. Thus, the KM system developed also has the organizational memory acting like a central knowledge repository and around this, there are services for capturing, searching, disseminating, using and maintaining knowledge.

Ontologies have been pointed as crucial for KM systems [4, 6, 7]. Benjamins et al. [4], for example, present a knowledge management approach based on ontologies and use ontological engineering to knowledge organization and structuring. Ontologies also play an important role in our approach, since they are used to structure ODE's organizational memory. But in our approach ontologies also give rise to knowledge items, since ontologies can be instantiated.

Finally, several researches pointed out the benefits of software agents for several purposes in knowledge management. Rabarijoana et al. [25] suggest the use of agents for knowledge retrieval. Staab et al. [26] presented an approach for intelligent proactive knowledge dissemination. Agents work on knowledge created through the usual work tasks of the user and offer knowledge to the user that may be relevant for his currently task. In our approach, agents also disseminate knowledge according to users' needs. But in contrast, we embed our agent support in specific steps of an activity, based on its ontological distinctions. So, we use semantic information to guide knowledge dissemination.

7. Conclusions

Knowledge management systems facilitate access and reuse of knowledge typically by using several emerging technologies, such as ontologies, and software agents. In this paper we presented an infrastructure for managing knowledge in a software engineering environment. At the core of this infrastructure there is an organizational memory. Around it, there are knowledge management services supporting KM activities, such as knowledge capture, retrieval, search, dissemination, maintenance and reuse. We also presented how this infrastructure is being used to support software quality knowledge management in ODE, an ontology-based SEE.

Knowledge management integrated to ODE reflects a design perspective of knowledge management [27]. In this perspective, knowledge workers constantly create new knowledge as they work. Some benefits of this approach can be pointed out:

- With KM integrated to a SEE, it is easier for developers to create new knowledge. In this way, the organizational memory is not closed. It is always evolving.
- A major concern for knowledge management in ODE is to capture information during the software process without developers' extra effort. Thus, the KM system is actively integrated into the work process. An isolated KM system, on the other hand, can be a barrier to innovation, because it does not let workers share new ideas with their peers. Closed systems do not give organizations control over their own knowledge, since there is a gap between knowledge creation and integration. Innovations happen outside the KM system, and then it contains information that is chronically out of date and that reflects an outsider's view of work.
- Knowledge management users are no longer passive receivers of knowledge, but are active researchers, constructors, and communicators of knowledge. Knowledge can be constructed collaboratively in the context of the work. Attention to knowledge requires attention to people, including their tasks, motivation, and interests in collaboration. The heart of intelligent human performance is not the individual human mind but groups of minds interacting with each other and with tools and artifacts.
- A KM system must provide the information workers need, when they need it. ODE's KM system can play an active role in knowledge dissemination. Software agents monitor the actions of users as they work, and inform them about potentially relevant knowledge for the task at hand.

Currently, knowledge management has been integrated into ODE environment and firstly, its aim is to support software quality knowledge management in ControlQ. The ControlQ tool was build previously and successfully integrated into ODE, so software quality can be planed and tracked over ODE's projects. We are now working to extend our approach to other ODE's tools.

Acknowledgments
The authors acknowledge CAPES for the financial support to this work.

References

[1] O'Leary, D.E., "Enterprise Knowledge Management", IEEE Computer Magazine, March, 1998.

[2] Borges, L.M.S., Falbo, R.A., "Managing Software Process Knowledge", Proc. of the CSITeA'2002, June 2002.

[3] O'Leary, D.E., Studer, R., "Knowledge Management: An Interdisciplinary Approach", IEEE Intelligent Systems, January/February, Vol. 16, No. 1, 2001.

[4] Benjamins, V. R., Fensel, D., Pérez, A. G., "Knowledge Management through Ontologies", Proc. of the 2nd International Conference on Practical Aspects of Knowledge Management (PAKM98), Switzerland, 1998.

[5] Markkula, M., "Knowledge Management in Software Engineering Projects", In: Proc. of the 11th International Conference on Software Engineering and Knowledge Engineering, Kaiserslautern, Germany, June,1999.

[6] Staab, S., Studer, R., Schurr, H. P., Sure, Y., "Knowledge Processes and Ontologies", IEEE Intelligent Systems, January/February, Vol. 16, No. 1, 2001.

[7] Rabarijoana, A., Dieng, R., Corby, O., "Exploitation of XML for Corporate Knowledge Management", Knowledge Acquisition, Modeling, and Management, Proc. of the European Knowledge Acquisition Workshop (EKAW`99), Lecture Notes in Artificial Intelligence, LNAI 1621, Springer-Verlag, 1999.

[8] Uschold, M., "Knowledge level modelling: concepts and terminology", Knowledge Engineering Review, vol. 13, no. 1, 1998.

[9] O'Leary, D., "Using AI in knowledge management: Knowledge bases and ontologies". IEEE Intelligent Systems, May/June, 1998.

[10] Davenport, T., Laurence, P., "Working Knowledge: How Organizations Manage What They Know", Harward Business School Press, Boston, Massachusetts, 1998.

[11] Fuggetta, A., "Software Process: A Roadmap", in Proc. of The Future of Software Engineering, ICSE'2000, Limerick, Ireland, 2000.

[12] Pressman, R.S., *"Software Engineering: A Practitioner's Approach"*, 5th Edition, New York: McGraw-Hill, 2000.

[13] Harrison, W., Ossher, H., Tarr, P., "Software Engineering Tools and Environments: A Roadmap", in Proc. of The Future of Software Engineering, ICSE'2000, Ireland, 2000.

[14] Abecker, A., Bernardi, A., Hinkelman, K., "Toward a Technology for Organizational Memories", IEEE Intelligent Systems, Vol. 13., No. 3, pp. 40-48, May/Jun, 1998.

[15] Falbo, R.A, Guizzardi, G., Natali, A.C.C., Bertollo, G., Ruy, F.B., Mian, P.G., "Towards Semantic Software Engineering Environments", in Proc. of the 14th Int. Conference on Software Engineering and Knowledge Engineering, SEKE'02, Ischia, Italy, 2002 (to appear).

[16] Falbo, R. A., Menezes, C. S., Rocha, A.R.C.; "Using Ontologies to Improve Knowledge Integration in Software Engineering Environments", Proc. of SCI'98/ISAS'98, USA, July, 1998.

[17] Falbo, R.A, Guizzardi, G., Duarte, K.C., "An Ontological Approach to Domain Engineering", in Proc. of the 14th Int. Conference on Software Engineering and Knowledge Engineering, SEKE'02, Ischia, Italy, 2002 (to appear).

[18] Duarte, K.C., Falbo, R.A., "Uma Ontologia de Qualidade de Software", Anais do VII Workshop de Qualidade de Software, WQS'2000, João Pessoa, Brasil, Outubro 2000.

[19] Bechhofer S., Horrocks, I., Goble, C., Stevens R., "OilEd: a Reason-able Ontology Editor for the Semantic Web", Proc. of KI2001, Joint German/Austrian conference on Artificial Intelligence, Vienna. Springer-Verlag LNAI Vol. 2174, pp 396--408. 2001.

[20] Guizzardi, G., Falbo, R.A., Pereira Filho, J.G., "Using Objects and Patterns to Implement Domain Ontologies", in Proc. of the 15th Brazilian Symposium on Software Engineering, Rio de Janeiro, Brazil, 2001.

[21] Gruber, T., "Toward principles for the design of ontologies used for knowledge sharing", International Journal of Human-Computer Studies, 43(5,6), pp.907–928, 1995.

[22] Althoff, K., Birk, A., Hartkopf, S., Muller, W., Nick, M., Surmann, D., Tautz. C., "Managing Software Engineering Experience for Comprehensive Reuse". In: Proc. of the 11th International Conference on Software Engineering and Knowledge Engineering, Kaiserslautern, Germany, Jun, 1999.

[23] M.G. Mendonça Neto, V. Basili, C.B. Seaman, and Y-M Kim, "A Prototype Experience Management System for a Software Consulting Organization", in Proc. of the 13th Int. Conference on Software Engineering and Knowledge Engineering, SEKE'01, Buenos Aires, Argentina, 2001.

[24] Basili, V., Caldiera, G., Rombach, H. "The Experience Factory", Vol. 1 of Encyclopedia of Software Engineering, Chapter X, John Wiley & Sons. 1994.

[25] Rabarijoana, A., Dieng, R., Corby, O., "Building a XML-based Corporate Memory", Workshop on Knowledge Management and Organizational Memories In the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI99), Sweden, July 1999.

[26] Staab, S., Schurr, H. P., "Smart Task Support through Proactive Access to Organizational Memory". Knowledge-based Systems, 13(5): 251-260. Elsevier, 2000.

[27] Fischer, G., Ostwald, J., "Knowledge Management: Problems, Promises, Realities and Challenges", IEEE Intelligent Systems, Vol. 16, No. 1, January/February, 2001.