Available online at www.sciencedirect.com
ScienceDirect
Information and Software Technology 49 (2007) 515–529
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

Software maintenance seen as a knowledge management issue

Nicolas Anquetil [a,*], Káthia M. de Oliveira [a], Kleiber D. de Sousa [a], Márcio G. Batista Dias [b]

[a] *Catholic University of Brasilia, Knowledge Management and IT Management, SGAN 916, Modulo B Brasilia, DF, Brazil*
[b] *University Center of Goiás–Uni-Anhangüera, GO, Brazil*

Received 25 April 2006; received in revised form 18 July 2006; accepted 24 July 2006
Available online 25 September 2006

Abstract

Creating and maintaining software systems is a knowledge intensive task. One needs to have a good understanding of the application domain, the problem to solve and all its requirements, the software process used, technical details of the programming language(s), the system's architecture and how the different parts fit together, how the system interacts with its environment, etc. All this knowledge is difficult and costly to gather. It is also difficult to store and usually lives only in the mind of the software engineers who worked on a particular project.

If this is a problem for development of new software, it is even more for maintenance, when one must rediscover lost information of an abstract nature from legacy source code among a swarm of unrelated details.

In this paper, we submit that this lack of knowledge is one of the prominent problems in software maintenance. To try to solve this problem, we adapted a knowledge extraction technique to the knowledge needs specific to software maintenance. We explain how we explicit the knowledge discovered on a legacy software during maintenance so that it may be recorded for future use. Some applications on industry maintenance projects are reported.

Keywords: Software maintenance; Software evolution; Knowledge management; Post mortem analysis; Ontology; Project revision; Post partum analysis

1. Introduction

To maintain legacy software systems, software engineers need knowledge on many different domains: application domain, system's architecture, particular algorithms used, past and new requirements, programming language, development environment, etc. More often than not, this knowledge is extracted at great costs from the detailed analysis of the system's source code: according to [28, p.475] or [29, p.35], from 40% to 60% of the software maintenance effort is devoted to understanding the system maintained.

One could argue that software development[1] suffers from the same knowledge needs, however these needs are more difficult to fulfill during maintenance. For example, it is not uncommon in software maintenance to have a very vague knowledge of what were the exact requirements for the system, whereas during software development, one is expected to have access to the requirements relatively easily.

Our position is that this constant quest for knowledge is one of the prominent problems of software maintenance and should be dealt with accordingly, for example, using knowledge management methods. We believe that adopting a knowledge management point of view on software maintenance could bring in a new light on the problem and may help improve the conditions in which it is performed.

In this paper, we present some experiments we did on software maintenance projects in the industry to capture the knowledge gained during the maintenance and record it. Our experiments use two tools from knowledge management: an ontology of the knowledge used in software maintenance; and Post-Mortem Analysis, a method to elicit knowledge from software maintainers. These tools have

* Corresponding author. Tel.: +55 61 3448 7148; fax: +55 61 3347 4797.
E-mail address: anquetil@ucb.br (N. Anquetil).

[1] We use the phrase "software development" to refer specifically to the creation of a new software system. We oppose software development to software maintenance.

0950-5849/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2006.07.007

been adapted so as to fulfill the specific knowledge needs of the maintenance activity.

The organization of the paper is the following: First, in Section 2, we give a short introduction to Knowledge Management, with a definition of knowledge and the goals of knowledge management. The two tools we are using, ontology and Post-Mortem Analysis, are also presented and discussed in more details. In Section 3 we review some basic facts about software maintenance to clarify our views, and the relation between maintenance and knowledge management. Section 4 presents our ontology of the knowledge used in software maintenance. The ontology is important as it serves as a framework for knowledge extraction. In Section 5, we discuss the second tool we used: Post-Mortem Analysis. We show how we adapted it to software maintenance projects. After presenting our approach, we discuss in Section 6 the results of some experiments we performed. Finally, in Section 7, we discuss related work before concluding in Section 8.

2. Knowledge management

Knowledge management came out as a reaction to the recognition that employees in an organization gather, as part of their daily activities, knowledge that is valuable to the organization. The typical image is that knowledge is an asset that has legs and walks home every night (cited for example in [1]). Another common image, in IT departments, is that of "immortals" on whom the continuing operation of a critical system depends exclusively.

In this section, we will provide some basic definitions for knowledge, knowledge management, ontology, and Post-Mortem Analysis.

2.1. Definitions

The definition of knowledge is normally built bottom-up from *data*, to *information* and then *knowledge* (see for example [34]): *Data* are raw facts, for example: 1.15. *Information* is data in context, for instance saying that 1.15 is the exchange rate between the US dollar and the Euro currency. *Knowledge* is a net of information based on one's particular experience, for example one's knowledge of the currency exchange mechanisms. "Knowledge is a fluid mix of framed experience, values, contextual information, expert insight and grounded intuition […] It originates and is applied in the minds of knowers" [9] (cited in [39, p.5]). There is much discussion on whether one may actually manage knowledge since it is tied to one's own experience and life. In this view, knowledge would be highly personal and impossible to express explicitly.

Without entering into this debate, we will consider that individuals can actually learn from each other and exchange knowledge – or information – to fulfill some goal. We will use a practical definition of knowledge management: "Knowledge Management enables the creation, communication, and application of knowledge of all kinds to achieve business goals". [39, p.5]. In this view, one differentiates tacit from explicit knowledge: *Explicit* knowledge is knowledge that has been captured and organized in a form that allows its distribution (for example in a book). In software maintenance, explicit knowledge could be an architectural model of a system, or a requirement specification. *Tacit* knowledge is particular to each individual and difficult to share as one is usually not even aware of all one knows. In software maintenance, tacit knowledge could be the understanding one gained on how a system is organized by working on it, or some special debugging technique one developed over time.

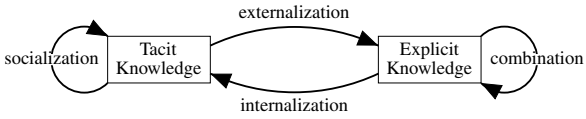

Fig. 1. Knowledge sharing according to Nonaka [26].

Nonaka [26] proposes a framework for knowledge sharing illustrated in Fig. 1. *Socialization* is the process of sharing knowledge doing things, knowledge is not made explicit, but rather a knower shows to one who does not know, how to do things. In software maintenance, this could be the case when an experienced maintainer helps a novice finding his-her way in a system, thereby giving hints on how the system is organized, where to look for things, etc. Knowledge may be slowly disseminated among small groups by this method. *Externalization* is the process of expliciting what one knows. Through externalization, a knower may express (e.g., writing a manual) what he knows and this knowledge may then be circulated among a large group or across time. In maintenance, a typical case would be the redocumentation of a system, but it may also happen during a meeting when one explains to one's colleagues how something works. *Combination* is the process of combining various sources of explicit knowledge to create a new one, as one would do in a literature survey. There may not be many examples of this in software maintenance as creating explicit documentation is not often performed when there is already some available. Finally, *internalization* is the process by which one makes some explicit knowledge one's own, by integrating it to one's own net of information. This could be the case of a maintainer studying various bits of documentation (may be a data model, a user manual and some comments in the code) to build a geneal understanding of what a system does and how it does it.

We are looking for ways to help these activities happen in a maintenance organization where knowledge of the systems being maintained is of key importance. Different techniques and tools have been proposed to support these activities of knowledge management. In this paper we will study two of them: ontology and Post-Mortem Analysis.

2.2. Knowledge organization: ontology

An ontology is an explicit specification of a simplified, abstract, view of some domain that we want to describe, discuss, and study [40]. The primary goal of an ontology is

to represent explicited knowledge, it is typically the result of a *combination* effort (see Fig. 1) where one gathers various authoritative sources on the domain and creates a consensus. There are different types of ontology in [17], we use a domain ontology to describe the domain of software maintenance. A domain ontology should contain a description of entities (of the domain) and their properties, relationships, and constraints [16].

Practically, ontologies may serve various purposes:

- *Reference on a domain:* Explicit knowledge serves as a reference to which people, looking for detailed information on the domain modeled, may go.
- *Classification framework:* The concepts explicited in an ontology are a good way to categorize information on the domain modeled. Indication of synonyms in the ontology helps avoiding duplicate classification. Other relations among the concepts of the ontology help one browsing it and finding an information one is looking for.
- *Interlingua:* Tools and/or experts wishing to share information on the domain modeled, may use the ontology as a common base to resolve differing terminologies.

2.3. Capturing knowledge: Post-Mortem Analysis

Ontologies are used to organize the knowledge, but techniques to gather this knowledge (making it explicit) are needed, as well as techniques to redistribute it (for example to new employees). Such techniques will be discussed in Section 7, including the best-known in software engineering: the experience factory.

Maintenance viewed as a knowledge management problem is an issue little explored. In this paper we will focus on how to explicit knowledge in a software maintenance context. Once the knowledge has been made explicit, it must be stored and disseminated among other groups, but we believe that existing solutions (as the experience factory, see Section 7) should be adequate to perform this part of the whole knowledge management cycle.

A knowledge elicitation technique, well known in software engineering, is the Post-Mortem Analysis (PMA). PMA, also called project review or project retrospective, simply consists in "[gathering] all participants from a project that is ongoing or just finished and ask them to identify which aspects of the project worked well and should be repeated, which worked badly and should be avoided, and what was merely 'OK' but leaves room for improvement" [38]. In Nonaka's framework for knowledge management (Section 2.1), PMA is a tool to externalize knowledge.

The term post-mortem implies that the analysis is done after the end of a project, although, as recognized by Stålhane in the preceding quote [38], it may also be performed during a project, after a significant mark has been reached.

There are many different ways of doing a PMA, for example Dingsøyr et al. [15] differentiate their proposal, a "lightweight post-mortem review", from more heavy processes as used in large companies such as Microsoft, or Apple Computer. A PMA may also be more or less structured, and focused or "catch all". One of the great advantages of the technique is its flexibility. It may be applied on a small scale with little resources (e.g., a 2 h meeting with all the members of a small project team, plus one hour from the project manager to formalize the results). Depending on the number of participants in the PMA, it may use different levels of structuring, from a relatively informal meeting where people simply gather and discuss the project, to a more formal process as proposed in [8].

3. Software maintenance

Software maintenance consists in modifying an existing system to adapt it to new needs (about 50% of maintenance projects [29]), adapt it to an ever changing environment (about 25% of maintenance projects [29]), or to correct errors in it, either preventively (about 5% of maintenance projects [29]), or as the result of an actual problem (about 20% of maintenance projects [29]).

Software maintenance is not a problem in the sense that one cannot and should not try to eliminate or avoid it. It is instead the natural solution to the fact that software systems need to keep in sync with their environment and the needs of their users. Lehman [25] established in his first law of software evolution that "a [software system] that is used, undergoes continual change or becomes progressively less useful".

Software maintenance offers significant differences with software development. For example, software maintainers work in more restricting technical conditions, where one usually cannot choose the working environment, the programming language, the database management system, the data model, the system architecture, etc. Furthermore, these conditions are usually dictated by past technologies long superseded. Also, whereas development is typically driven by requirements, maintenance is driven by events [29]. In development, one specifies the requirements and then plans their orderly implementation. In maintenance, external events (e.g., a business opportunity, the discovery of a show stopping error) require the modification of the software and there is much less opportunity for planning. Maintenance is by nature a more reactive (or chaotic) activity.

Because of these differences, software maintenance is already more difficult to perform than software development. But, we argue that apart from these, maintenance suffers from another fundamental problem: the loss, and the resulting lack, of knowledge of various types.

A good part of the development activity consists in understanding the users' needs and their world (application domain, business rules) and convert this into running code by applying a series of design decisions [42]. All this (application domain, business rules, design decisions) represents knowledge that is embedded into the resulting application and, more often than not, not otherwise recorded.

We are not suggesting that software maintenance has knowledge requirements significantly different from

software development, but rather that whereas the knowledge needs are roughly the same in both activities, they are more difficult to fulfill during maintenance. In a proper software development project, all the knowledge is available to the participating software engineers, either through some documentation (specifications, models) or through some other member of the project. In maintenance, on the other hand, much of this knowledge is, typically, either lacking, or only encountered in the source code: the business model and requirement specifications may have been lost, or never properly documented; the software engineers who participated in the initial development (often years ago) are long gone; the users already have a running system and cannot be bothered with explaining all over again how they work. To maintain a software system, one must usually rely solely on the knowledge embodied and embedded in the source code.

As a result of this lack of knowledge, between 40% and 60% of the maintenance activity is spent trying to understand the system [27, p.475], [29, p.35]. Maintainers need knowledge of the system they work on, of its application domain, of the organization using it, of past and present software engineering practices, of different programming languages (in their different versions), programming skills, etc.

Among the different knowledge needs, one may identify:

- Knowledge about the system maintained emerges as a prominent necessity. For example, Jørgensen and Sjøberg [22] showed that software maintainers are not less subject to major unexpected problems when they have a longer experience in maintaining systems, whereas, having experience in the particular system maintained does help to reduce these problems. In other words, knowing the system greatly help maintaining it correctly whereas having done a lot of maintenance on other systems does not.
- In [4], Biggerstaff insists on the importance of application domain knowledge. He highlights that users usually report errors and enhancement requests in terms of application domain concepts (e.g., "I cannot cancel this flight reservation", "I need to be able to specify a particular seat in a flight reservation") that the maintainers must then link (trace) to some specific system component (e.g., "class XYZ", "function foo", or table "TB_ACME"). Another typical example is the need to know well the business rules of an application domain in order to test adequately a system (e.g., after a modification).
- Van Mayrhauser and Vans, already cited [42], look at the design decisions, that is to say knowledge about software development applied to the transformation of knowledge on the application domain to produce source code. They explain that these decisions impact the resulting source code, and one should know what decisions were made to understand why the program was written in a particular way. Moreover, why a possible solution was rejected, is also important information because it gives hints on the broader considerations (e.g., non-functional requirements) that guided the development.

We argue that many of the difficulties associated with software maintenance, originate from this knowledge management problem. To tackle this problem, we adapted knowledge management techniques and tools to the needs of software maintenance:

- First, we defined an ontology of the knowledge used in software maintenance, which serves as a structuring framework to develop other solutions.
- Second, we adapted the Post-Mortem Analysis technique to capture relevant knowledge in a maintenance team.

These two instruments will be detailed in the two following sections.

4. An ontology for software maintenance

We defined an ontology of the knowledge used in software maintenance to serve as a structuring framework for our research. This ontology is presented in [13], and we will not enter in a detailed description here. We will only present the main concepts of the ontology and how they relate so as to better illustrate afterward how it helped us in the rest of the work.

The ontology is divided into five subontologies: the *software system* subontology, the *computer science skills* subontology, the *modification process* subontology, the *organizational structure* subontology, and the *application domain* subontology. In the following, we present each of these subontologies, their concepts and relations. The following conventions are used: ontology concepts are written in CAPITALS and associations are underlined. Fig. 2 illustrates how the subontologies combine together.

4.1. System subontology

Fig. 3 shows the first subontology, on the *software system*.

The main concepts are the following. A SOFTWARE SYSTEM interacts with USERS and possibly other SYSTEMS. It is implemented on some HARDWARE and implements specific TASKS (of the application domain). It is composed of ARTIFACTS that can generally be decomposed in DOCUMENTATION

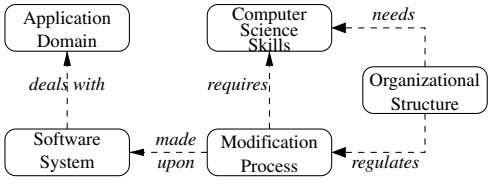

Fig. 2. Ontology overview.

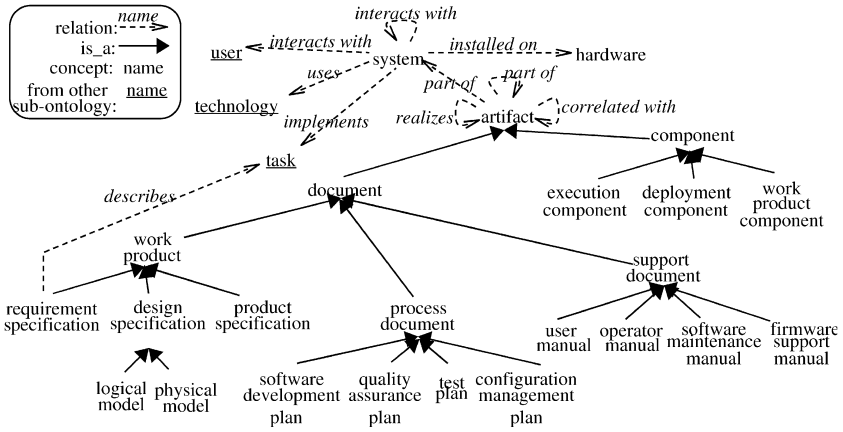

Fig. 3. System subontology.

and SOFTWARE COMPONENTS. Briand [7] considers three kinds of documentation: (i) PRODUCT RELATED, describing the system itself; (ii) PROCESS RELATED, used to conduct software projects; and (iii) SUPPORT RELATED, helping to operate the system.

SOFTWARE COMPONENTS represent all the coded artifacts that compose the software system itself. Booch [6, p.349–350] classify them in: (i) EXECUTION COMPONENTS, generated for the software execution; (ii) DEPLOYMENT COMPONENTS, composing the executable program; and (iii) WORK PRODUCT COMPONENTS, that are the source code, the data, and anything from which the deployment components are generated.

All those ARTIFACTS are, in some way, related one to the other. For example, a requirement is related to design specifications which are related to deployment components. There are also relations among requirements. We call the first kind of relation a realization, relating two artifacts of different abstraction levels (in the USDP [21], one says that a sequence diagram realizes a use case). We call the second kind of relation correlation, relating two artifacts of the same level of abstraction (for example, a class diagram and a sequence diagram realizing the same use case would be correlated).

4.2. Skills in computer science subontology

The second subontology describes the skills needed in computer science to perform maintenance. It is presented in Fig. 4.

The MAINTAINER must know (be trained in) the MAINTENANCE ACTIVITY that must be performed, the HARDWARE the system runs on, and various COMPUTER SCIENCE TECHNOLOGIES (detailed below). Apart from that, the MAINTAINER must also understand the CONCEPTS of the application domain and the TASKS performed in it. There are four COMPUTER SCIENCE TECHNOLOGIES of interest: possible PROCEDURES to be followed, MODELING LANGUAGE used (e.g., the UML), CASE TOOLS used (for modeling, testing, supporting, or developing), and finally, the PROGRAMMING LANGUAGE(S) used in the system.

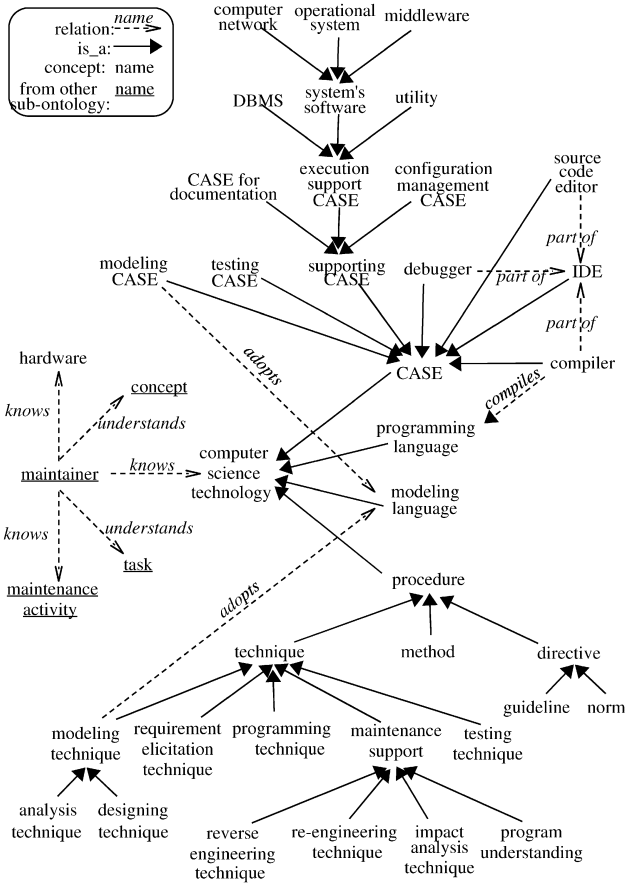

Fig. 4. Computer Science skills subontology.

4.3. Modification process subontology

Fig. 5 shows the concepts of the modification process subontology. Here, we are interested in concepts from the modification request and its causes. According to Pigoski [29], a MAINTENANCE PROJECT originates in a MODIFICATION REQUEST submitted by a CLIENT. These REQUESTS are classified either as PROBLEM REPORTS or ENHANCEMENT REQUEST. He also lists the different ORIGINS of a MODIFICATION REQUEST: ON-LINE DOCUMENTATION, EXECUTION,

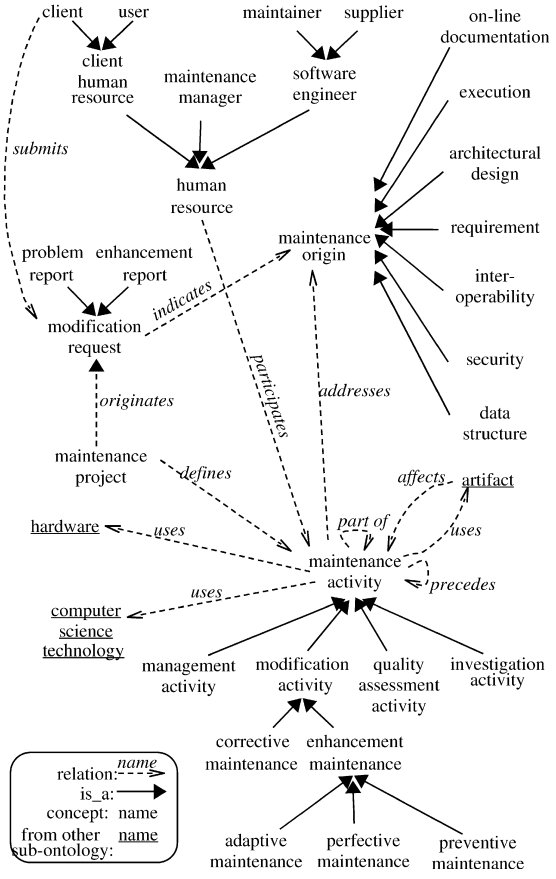

Fig. 5. Modification Process subontology.

ARCHITECTURAL DESIGN, REQUIREMENT, SECURITY, INTEROPERABILITY, and DATA STRUCTURE. One or more MODIFICATION REQUESTS generate a MAINTENANCE PROJECT that will define the different software MAINTENANCE ACTIVITIES to execute.

Based on [7,24,29], we classified the MAINTENANCE ACTIVITIES in the following types: INVESTIGATION, MANAGEMENT, QUALITY ASSURANCE, and MODIFICATION.

A MAINTENANCE ACTIVITY uses input ARTIFACTS and affects output ARTIFACTS. It is inserted (precedes) in a sequence of ACTIVITIES, it addresses some MAINTENANCE ORIGIN, uses HARDWARE resources and some COMPUTER SCIENCE TECHNOLOGIES.

Finally, different types of person (HUMAN RESOURCES) may participate in these ACTIVITIES (from [7,18,24,29]): SOFTWARE ENGINEERS, MANAGERS, and CLIENT'S HUMAN RESOURCES (CLIENT and USER).

4.4. Application domain and organization subontologies

The fourth subontology, on the organizational structure, is presented in Fig. 6, left part. We considered a traditional definition of an ORGANIZATION (see for example [37]) composed of ORGANIZATIONAL UNITS where HUMAN RESOURCES fill different POSITIONS. We also included the fact that an organization defines DIRECTIVES to be adopted in the execution of the tasks. Our goal here was not to define all possible aspects of an organization, but only to define that the maintenance is an activity performed by people in some ORGANIZATIONAL UNIT that compose the whole ORGANIZATION with its own rules.

Finally, the fifth subontology (Fig. 6, right) organizes the concepts of the Application Domain. We chose to represent it at a very high level that could be instantiated for any possible domain. We actually defined a meta-ontology specifying that a domain is composed of domain CONCEPTS, related to each other and having PROPERTIES which can be assigned values and RESTRICTIONS that defines constraints for the CONCEPTS. This meta-ontology would best be instantiated for each application domain with a domain ontology as exemplified in [10]. We also considered that the CONCEPTS in an application domain are associated with the TASKS performed in that domain and those TASKS are regulated by some RESTRICTIONS.

5. Post-Mortem Analysis for maintenance

As already mentioned, Post-Mortem Analysis is a commonly recommended practice for software engineering projects [5,27]. It is used as an externalization tool (see Section 2.1), for example, to gather the lessons learned during the realization of a project.

Three facts emerged as near constants in the articles reporting use of PMA [11]:

- it is mostly used for process improvement;
- it is mostly used in software development context; and
- it is mostly used at the end of projects.

In the literature, PMA is mainly viewed as a process improvement tool. For example, Stålhane et al. start their paper [38] with the affirmation: "An obvious way to improve a software development process is to learn from past mistakes". Other authors [5,15,23,30,43] assume the same point of view, either explicitly or implicitly.

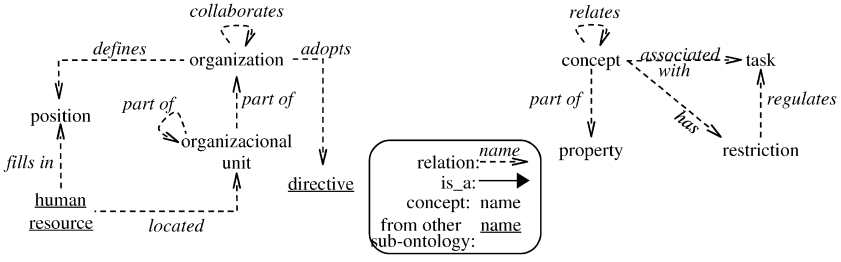

Fig. 6. Organizational Structure subontology (left) and Application Domain subontology (right).

Similarly, PMA is always cited in the context of software development (either explicitly or implicitly). For example, Kerth [23] in discussing whether to call the activity Post-Mortem ("after death") Analysis or post-partum ("after birth") Analysis, argues: "a software effort is more like a birthing experience than dying experience – after all the goal is to create something new". This view mainly holds for development projects, if we consider maintenance, for example corrective maintenance, the goal is not to create anything new.

Finally, PMA appears to be mostly performed at the end of projects (hence the name). One problem with this approach is that for long projects, the team only remembers "the large problems that are already discussed – things that have gone really bad" [38] (note that, despite raising the issue, the article does not detail any specific solution). Another difficulty raised by Yourdon [43] is a high turnover that may cause key team members to disappear, with their experience, before the end of the project. The solution proposed (but not developed) by Yourdon is to conduct mini-postmorta at the end of each phase of the projects. One of our contributions is to formalize the implementation of Yourdon's idea of intermediary mini-postmorta for software maintenance projects.[2]

We already saw, in Section 3, that software maintenance is a knowledge intensive activity including knowledge on the software system maintained and its application domain. Therefore to use PMA as an externalization technique in maintenance projects, we need to adapt it to uncover, not only knowledge on the maintenance process (e.g., how it was executed, what tools or techniques worked best), which is the "traditional" use of PMA, but also to register knowledge on the system maintained (e.g., how subsystems are organized, or what components implement a given requirement).

To define this new PMA model, we had to consider three important aspects that will be detailed in the following subsections: (i) when to insert PMA during the execution of a typical maintenance process, (ii) what knowledge we should look for, and (iii) how to extract this knowledge from the software engineers.

5.1. When to perform PMA during maintenance

Software maintenance projects may be of widely varying size, they may be short in the correction of a localized error, or very long in the implementation of a new complex functionality, or correction of a very diluted problem (e.g., the Y2K bug). For small projects, one may easily conduct a PMA at the end of the project without risk loosing (forgetting) important information, but for larger projects, it is best to conduct several PMAs during the project (as proposed by Yourdon [43]) so as to capture important knowledge before it becomes so internalized in the participants' mental models that they cannot clearly remember the details.

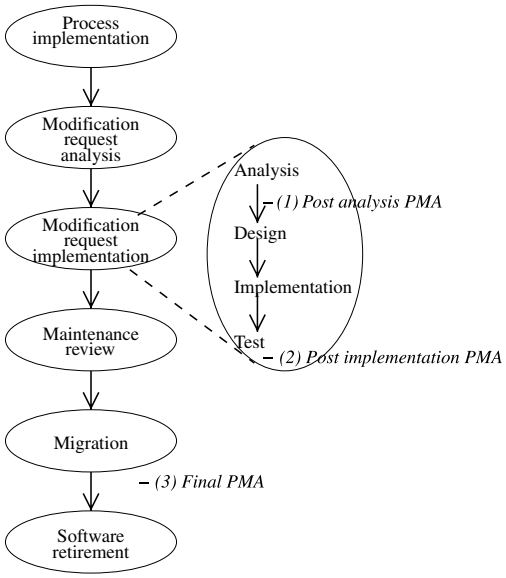

Fig. 7. Overview of the ISO 14764 Maintenance process [20] with the intermediary and final PMAs and their respective scope.

To identify the points, in a software maintenance project, where we could perform PMA, we used the ISO/IEC 14764 [20] maintenance process.[3] It is a basic process for maintenance projects with the following activities: process implementation, problem and modification analysis, modification implementation, maintenance review/acceptance, migration, and software retirement (the process appears in Fig. 7).

Process implementation: includes tasks to document the maintenance process, establish procedures for modification requests, establish the configuration management process, …

Problem and modification analysis: includes tasks to replicate the problem, analyze it, develop options to solve it, and choose one.

Modification implementation: includes tasks to implement the modification such as requirements analysis, architectural design, detailed design, coding, and testing.

Maintenance review/acceptance: includes tasks to review the modification with the authorizing organization and obtain approval for it.

Migration: includes tasks to plan the migration of the modified system, notify when, why, and how the migration will happen, train the users, review the impact of the new system, etc.

Software retirement: includes tasks similar to the preceding activity but focused on the old system to retire, instead of implanting a new one.

To be of use, the intermediary PMAs should be conducted at the end of significant milestones, evenly

[2] Note that this contribution is not intrinsically linked to software maintenance and could be applied to software development projects.

[3] This process is actually the same as the ISO/IEC 12207 [19] maintenance process.

distributed during the project. In a large software maintenance project, analysis of the modification (how it may be done, what parts it would impact, how to fit it in the existing architecture), and actual implementation (detailed design, coding, testing) would consume the major part of the project time (analysis represents 40% to 60% of a maintenance project, [29]), while other activities as validation or migration would be shorter. We identified two main points where to perform the intermediary PMAs (see Fig. 7):

- after the *analysis of the modification* which includes the first two activities (Process implementation, Problem and modification analysis) and the initial task of the third activity (Modification implementation: requirement analysis);
- after the *implementation of the modification* which includes the rest of the third activity (Modification implementation).

A third (final) PMA can then be conducted at the end of the project in order to review all its aspects and the most recent activities not yet considered in the intermediary PMAs.

Other points when to perform PMAs could be considered, for example after the important Maintenance Review activity. However, we considered that a PMA after this activity would probably be very close in time after the second PMA (post-implementation) and before the third one (final), therefore duplicating the effort for little return. Important lessons learned during the Maintenance Review activity (mostly if the modification was satisfactory and if not why) can be explicited during the final PMA.

5.2. What knowledge to look for in software maintenance

Depending on the specific scope of each PMAs, we may hope to explicit different kind of knowledge. For example, information on the testing techniques used will be best discovered during the second PMA (post-implementation) just after the tests have been performed. As already explained, it is clear from previous work on PMA, that it is a successful technique to discover lessons learned from the execution of a process and thereby improve its next execution. Therefore, during each intermediary PMA, we will seek information on the tasks and activities that occurred before it. The final PMA will mainly look for information on the execution of the whole process. However, we also wish to discover new knowledge learned on the system, its application domain, or other issues not related to the maintenance process.

To identify what information we could hope to discover in each PMA, we mapped the concepts defined in our ontology to the particular tasks reviewed in each PMAs. For example, the first PMA (post-analysis) occurs after: (i) the process implementation activity, (ii) the problem and modification analysis activity, and (iii) the requirements analysis task from the modification implementation activity.

In the process implementation activity, the only task typically performed for each new maintenance project is to develop plans and procedures for conducting the activities of the project. To execute this task the project manager usually takes into account his/her experience from previous projects with a similar domain, size, and team. Therefore, the type of knowledge related to this activity is about the process execution, what MAINTENANCE ACTIVITIES are needed and may be what specific TECHNOLOGY will be required (see subontologies in Figs. 4 and 5). This implies that the first PMA should look for this particular type of knowledge.

The problem and modification analysis activity starts when the maintainer analyzes the modification request to determine its impact on the organization, on the existing system, and on other systems that may interact with it. From this analysis the maintainer defines options for implementing the modification. Based on the analysis report the manager estimates the effort to do the maintenance and sets the time frame for the project. With this information, one obtains approval to do the modification. The types of knowledge related to this activity are (see concepts from Figs. 3–6):

- detailed knowledge on the modification request (see the concepts related to the MODIFICATION REQUEST as, what was the maintenance type: correction or enhancement; what was the MAINTENANCE ORIGIN: DOCUMENTATION, EXECUTION, …; who submitted the MODIFICATION REQUEST);
- how the impact analysis (an INVESTIGATION ACTIVITY) was performed. For example, what COMPUTER SCIENCE TECHNOLOGIES were used: what CASE tool, what MAINTE-

Table 1
The three maintenance PMAs and the types of knowledge they focus on

PMA	Type of knowledge
(1) Post-analysis	Details on the modification request
	Organizational structure using the software
	Options for implementing the modification
	Effort estimation for the modification
	Negotiation of time frame to do the modification
	Documents modified
	Requirement elicitation technique used
	Tools used
	Application domain
	Details on the requirements
(2) Post-implementation	Programming languages and tools used
	Programming techniques used
	Software components modified
	Systems interrelationship
	Analysis/design inconsistencies
	Re-engineering opportunities detected
	Artifacts traceability
	Database design
	Design patterns used
	Testing technique used
	Process and support documentation modified
(3) Final	Negotiations with other technological departments
	Modification monitoring
	Maintenance process

...

Category: Negotiation of time frame to do the modification

- How was the time limit negotiated with the client? Do you think the method was satisfactory? Why?
- The time frame initially proposed by the client was realistic given the size of the modification? Why? Was it the result of a previous maintenance?

Category: Application domain

- What business concepts were involved in this maintenance?
- What business rules were involved in this maintenance?
- Did you discover any new requirement or application domain concept during this maintenance?
- If the modification request was due to a requirement modification, what law, measure, status, etc. caused the change of requirement? What is the context of the change?

...

Fig. 8. Excerpt of the post analysis questionnaire. Application domain questions are intended to instantiate the concepts of the Application Domain subontology (see text).

NANCE TECHNIQUE; also, what ARTIFACT of the SYSTEM can be impacted?;
- the organizational structure (see HUMAN RESOURCES, who uses the software, who is the CLIENT, who are the SOFTWARE ENGINEERS involved); or
- how the time frame to implement the modification was defined and negotiated.

Finally, the requirements analysis task includes updating the system documentation related to the problem being solved. Performing this task, the maintainer uses specific REQUIREMENT ELICITATION TECHNIQUES (Fig. 4) and tools to better collect and register the user requirements. During this task, the maintainer should also learn about different CONCEPTS of the domain, BUSINESS RULES, who are the USERS, which parts of the organization use the system, when and why is it used.

The types of knowledge to consider in each PMA are defined similarly, based on the activities they review and the concepts of the ontology involved in these activities. A list of knowledge types is proposed in Table 1. The next section details how we plan to discover information in each type.

5.3. How to perform PMA during maintenance

Finally, we had to define a method that would help the software engineers remember and explicit all they could have learned in the various knowledge domains considered (process, system, application domain, ...). For this we decided to perform the PMAs in two steps: First, we designed a questionnaire as a mean to pre-focus their mind on the bits of information we want to discover. This questionnaire is distributed among the software engineers that will participate in the PMA session. In a second step, we conduct a PMA session where the same topics are brought up again to effectively discover new information. The actual PMA session may take various forms (for examples, see [38]): semi-structured interview, KJ session,[4] using Ishikawa diagram, or using a combination of these.

The questionnaires are composed of one or more questions for each type of knowledge identified for that PMA. Questions are designed to instantiate the concepts defined in our ontology. Fig. 8 shows some questions from the PMA post-analysis questionnaire. One can easily trace the questions of the second part (Category: Application Domain) back to the concepts of the Application Domain subontology. There are two possible uses of the questionnaire, first, it may be used only to revive the memory of the PMA participants on the various types of knowledge sought. In this approach, the actual answers would not be considered in the PMA session. Another approach, that we actually used, is to use the answers to the questionnaires to help the facilitator focus the PMA session on the topics that appear most likely to bring new knowledge.

We have, thus far, experimented our proposal with semi-structured interviews and brainstorming sessions (KJ sessions). The results showed some interesting knowledge bits as will be discussed in the next section.

6. Discussion of experimentation

Validating a knowledge management approach in general is a difficult thing as the results only appear on the long run, and even then, it may be difficult to pinpoint a single event that clearly shows the benefit of the approach. Moreover, our proposition is more to show the importance of different

[4] A kind of brainstorming session.

kinds of knowledge for software maintenance, and how they may be collected, than a complete knowledge management approach (the experience factory, see Section 7, might be useful there). Therefore, we will limit ourselves to discuss some experiments of the PMA methodology and their results.

PMA for maintenance was applied to six maintenance projects from the industry. We applied interview PMAs in four small maintenance projects (about 1 man/week work), and KJ sessions for two larger projects (more than two months). Both experiments and their results will be discussed here.

The experiments were realized in a public organization, where the software maintenance group includes about 60 software engineers (managers, analysts, programmers, DBA, etc.). The methodology was tested on a specific group of 15 people, responsible for the maintenance of 7 legacy software systems. It must be mentioned that the organization had just undergone (2 or 3 months before) a major redefinition of its working practices with the introduction of new software processes. Unfortunately, this meant that the process issue was still a sensitive one at the time of the experiment, with many adjustments still to be done and the topic in itself more present in the mind of the software engineers. This is a bias in our experiments in the direction opposite to the one we favor (discovering more knowledge about the system or the application domain).

In all cases, the maintainers had been briefed before hand on the goals of the PMAs, particularly that it was not intended to be a witch-hunt. Actually some experimental PMAs had already been conducted in the organization previously with the same group.

In all experiments, data on the projects were gathered through a special management tool implanted earlier (about 9 months before) in prevision of these experiments. Statistics on the duration of the PMAs were collected more informally.

6.1. Interviews

We applied semi-structured interview PMAs to four short maintenance projects with few maintainers (one or two) involved. Semi-structured interviews are a systematic way to follow an agenda and allow the interviewer to find out more information on any issue that was not adequately answered in the questionnaire. We felt that interviews would be the best tool because the small size of the maintenance team allows the facilitator to easily merge the discoveries. The characteristics of these projects were as follow:

- *Project 1:* Perfective maintenance, involved 2 maintainers during 6 days for a total of 27 man/h of work.
- *Project 2:* Perfective maintenance, involved 1 maintainer during 5 days for a total of 17 man/h of work.
- *Project 3:* Perfective maintenance, involved 2 maintainers during 5 days for a total of 47 man/h of work.
- *Project 4:* Perfective maintenance, involved 2 maintainers during 2 days for a total of 10 man/h of work.

During and after each project, questionnaires were distributed to the maintainers, then the PMAs facilitator would meet with each maintainer to interview him/her. The duration of the interviews is listed in Table 2. It is roughly constant and does not seem to depend on the duration of the maintenance project (although the number of interviews does depend on the size of the team).

We felt that this way of performing PMAs gave complete satisfaction with results as expected (e.g., knowledge gained on the system or the application domain). Examples of these results are proposed in Section 6.3. The interviews were found to be flexible, easily applied, and at little cost. However, as already mentioned, we feel that the method would not scale up well and larger teams need group meeting(s) to facilitate the convergence of ideas.

In Table 3, we present an overview of the number of concept that could be instantiated during the PMAs. For example, of the 23 concepts in the System subontology, 11 were instantiated, which means that at least one concrete example of these concepts was mentioned during the PMAs as something that was learned and worthy of remembering. When an instance of a given concept is mentioned (for example, concept USER from Fig. 5), we consider that this concept and all its super-concepts (CLIENTE HUMAN RESOURCE and HUMAN RESOURCE) are instantiated.

From the table, we can see that the Process subontology is the one that was the most instantiated, in number of concepts (21) and number of instances (135). We already knew that PMA was an adequate tool to discover lessons learned

Table 3
Concepts from the ontology instantiated during the four PMAs

	Number of concepts	Instanciated concepts	Number of instances
System	23	11	80
Process	30	21	135
CS skills	38	05	09
Organization	03	03	22
Application domain	04	04	68

Table 2
Duration of the Interview PMAs

Maintenance project	Team members	Project duration (man/h)	PMA (1): post-analysis (min)	PMA (2): post-implementation (min)	PMA (3): final (min)
1	2	27	20	20	15
2	1	17	20	30	10
3	2	47	20	30	15
4	2	10	20	25	20

from the process. We believe that the recent changes in the organization's processes are, at least partly, responsible for this higher representation of the Process subontology. Various other concepts from this subontology were not instantiated due to the characteristics of the projects. For example, all four projects were perfective maintenance, therefore the concept CORRECTIVE MAINTENANCE could not be instantiated in these cases. This is also the case for many CASE subconcepts (there are 16 in the subontology) which were not used or do not exist in the case considered (e.g., DEBUGGER).

With only four maintenance projects, we were able to instantiate almost half of the concepts from the System subontology (11 instantiated for a total of 23) with many instances (80). We see it as a sign that our method does allow to discover such knowledge and is successful in this sense. Because of the typical conditions of legacy software systems (foremost the lack of system documentation), many concepts from the System subontology could not be instantiated. This is the case of many DOCUMENT sub-concepts (there are 16 in the subontology).

Similarly, knowledge on the application domain and the Organizational Structure was gained during the PMAs. All concepts from these two subontologies were instantiated and, more importantly, many instances were found, especially in the case of the application domain subontology (68 instances). This is a good result since application domain knowledge is considered very important by some authors (e.g., [4]).

Finally, the lesser results for the Computer Science Skills subontology is seen as natural and with little impact. From the nine instances found, four were to mention interviews as the REQUIREMENT ELICITATION TECHNIQUE used (and to note that it was satisfactory). The other instances, refer to the discovery of the minus operator in a relational database environment (PROGRAMMING TECHNIQUE); the use of a new class from the programming language library (PROGRAMMING TECHNIQUE); two instances of new testing approaches (TESTING TECHNIQUES) and the discovery of a functionality of the modification request management tool ClearQuest (SUPPORTING CASE). It is natural that experienced software engineers discover less new knowledge about computer science techniques or CASE tools, and we do not see this as a problem with our approach. Computer science skills are considered background knowledge that all software engineers should have.

6.2. Brainstorming sessions

We applied KJ sessions (a kind of structured brainstorming) to two large maintenance projects with more maintainers involved (more than 10 in our experiments). The KJ sessions seemed best fitted because they are a kind of structured brainstorming where all the team members get a chance to share what they learned. When the team gets bigger, it is important to have this kind of meeting so that all opinions may be expressed and compared. A clear problem of the method is that it is more costly and difficult to organize. The characteristics of these projects were as follow:

- *Project 5:* Adaptive maintenance, involved 11 maintainers during 60 days.
- *Project 6:* Adaptive maintenance, involved 12 maintainers during 90 days.

Because of the urgency of both projects, it was not possible to realize the intermediary PMAs (post-analysis and post-implementation) and we could experiment only the final PMAs. This is a sign of the lack of clear support from the upper management toward this experiment.

As described earlier, these sessions were prepared with the distribution of questionnaires intended to revive the important points of the projects in the mind of the participants and help them focus on the topics of interest. For each project, the PMA was divided in two KJ sessions: in the first session, positive and negative points were raised, they were then summarized and organized by the facilitator to prepare the second session where corrective actions were proposed for the negative points (this second session is not really part of our experiment). The duration of each session is given in Table 4.

Although Dingsøyr et al. [15] termed KJ sessions a "lightweight post-mortem review", knowledge management in general is still a costly process. We could verify this when we had to find time for a team of more than 10 people to meet during an entire workday. Because of this, we could not apply the intermediary meetings that we were planning. This is a problem because the last PMA focuses potentially more on the process and less on the system or application domain.

The recent introduction of new processes showed still more heavily in these experiments as in the previous ones because of the need to justify the meetings to the eyes of the upper management. Consequently, the results were not as satisfying as for the short maintenance projects, with more points coming out on the maintenance process itself and less knowledge on the system, application domain, or organizational structure being elicited.

Although these PMAs used a different format (brainstorming session) than for the small maintenance projects, they still made use of the same instrument (the questionnaire) to revive the knowledge of the software engineers. Because of this, we believe that they could have met with our objective of discovering knowledge on the systems, the application domain, or the organizational structure. We see the lack of clear results as a consequence of the particular conditions we had to deal with.

Table 4
Duration of the KJ sessions (the duration is that of the whole "session" which included two actual meetings) for both projects

Maintenance project	Team members	Project duration (days)	PMA	PMA duration (h)
5	11	60	final	6
6	12	90	final	8

6.3. Some results

Examples (from the interview PMA) of knowledge gained during a maintenance project and uncovered by the PMAs are:

- redocumentation of business rules found in the source code;
- detailed understanding of how a particular module works;
- identification of re-engineering opportunities (the re-engineering was not actually performed but the need for it was recorded for future analysis);
- identification (by some software engineers) of incomplete knowledge on the programming language or of a CASE tool; or
- identification of problems in the business processes and proposition of solutions to improve these processes.

Another result that was not expected but gave us great satisfaction was that we started to create a culture of knowledge management in the organization. The maintainers were beginning to look for the PMAs to exchange information and actually asked to have them. One suggestion that came out of these PMAs was that it would be useful to design a tool to help knowledge recording during the maintenance so that it would not be lost afterward (for example, when a PMA may not happen or when it is delayed too long after the events it covers). It is possible that the tool proposed by Derrider [12] (see Related Work Section) could be of some help in this sense, however this is a difficult issue and we are not sure whether this is at all recommendable. We strongly believe that knowledge management is better done as a separate, clearly identified, activity rather than on the fly. It seems clear to us that the success of the PMA approach lays in the fact that people stop to do their usual work to start reflecting on what they know and learned. Doing this during the execution of a project may prove difficult and may be counter-productive.

7. Related work

Using some kind of knowledge management technique to help maintenance is not new, although we believe we are the first to explicitly present and deal with the software maintenance problem in terms of a knowledge management issue. We divide related work in two categories:

- Knowledge management in software engineering in general.
- Knowledge management in software maintenance.

7.1. Knowledge management in software engineering

One can trace the introduction of knowledge management techniques for software engineering back to the proposal of the Experience Factory by Basilli et al. (e.g., [2,3]). The Experience Factory is intended to gather the lessons learned from past projects and make these available to other members of the organization. It started as a process improvement tool and evolved in a more general knowledge management approach. The Experience Factory differs in several points from our proposal: It is more general, being applicable to any software engineering process, and not even restricted to software organizations [3]. All application reports relate to process improvement (including [41], see next section) and do not deal with other information as we do. Finally, it is a much heavier solution than ours, not easily implemented (e.g., see the application reports in [14,35]), whereas the use of PMA may be implemented simply and at a relatively small cost.

Actually, PMA may be used inside the Experience Factory to collect knowledge that the factory will allow to store and recover. As we already mentioned, we did not explore the aspect of knowledge redistribution. Our focus was primarily to consider the knowledge needed in software maintenance and see how we could make it explicit. The next step could be to use the full Experience Factory framework to complete the knowledge management cycle. However, due to the investment required to implement the Experience Factory, much more involvement from the higher management would be required for this step to be taken.

As already noted (Sections 2.3 and 5) Post-Mortem Analysis is already practiced in software engineering projects (e.g., [5,15,27,38]). An important difference of our approach is the context of software maintenance, which is different from what is usually practiced. Typically, PMA has been used to improve development process, we are using it to gain other types of knowledge (for more details see Section 5).

Agile software development methods propose practices that promote knowledge sharing among a team of software engineers. Agile methods focus on a relatively small team, where knowledge is implicitly shared among the members. To work well, this model requires that the team evolves slowly over time (low turnover) so that new members can catch up with the common pool of knowledge before too many old members leave. In the organization we studied, there is no fixed team, but a pool of 60 software engineers who may be assigned to the maintenance of any system. Actually, many experienced software engineers left the organization, after a change in management and a re-organization of the working habits.

Another difference is that although promoting knowledge sharing is fundamental to agile methods, they remain software development methods and not knowledge management methods. Knowledge is shared implicitly and no special attention is given to it. We put knowledge at the front of the stage and made it a goal of its own. Actually, in this sense, an important result for us was to make the software engineers conscious of the importance of knowledge, of recording it, and of sharing it.

7.2. Knowledge management in software maintenance

Other authors already applied some kind of knowledge management techniques to software maintenance.

There is an experiment using the Experience Factory in the context of Software Maintenance [41]. The objective of this experiment remains in the line of the traditional Experience Factory application: study and improve the process. The fact that much more knowledge may be involved (and explicited and re-used) in Software Maintenance is not alluded to. The only hint in this direction is that the characterization of the knowledge (a step in the Experience Factory process) includes a characterization of the existing system. This, in itself, implies that the knowledge on the system is important too. However, this knowledge was not further considered in the referred experiment, i.e., it is not explicited and stored in the Experience Factory.

There is another proposition to use the Experience Factory to help share results of researches on Software Maintenance [36]. This is of course a completely different preoccupation than ours since it focuses researchers and not practitioners.

A similar preoccupation, although not using the Experience Factory, guided the work of Kitchenham et al. [24]. They defined an ontology to help classify research work on maintenance. Although their work was pioneer, they do not aim at solving any maintenance problem, but rather help research on maintenance.

Ruiz et al. [33] published an "ontology for the management of maintenance projects". Their goal is the same as ours and they also use an ontology to identify and classify the knowledge to discover. There are two differences between our approaches: First, Ruiz et al. ontology is mainly concerned with the maintenance process and quality assurance, it is actually based on a specific process; second, they are trying to come up with tools that would help to automatically discover, classify, and recover the knowledge. On the other hand, we tried to define our ontology independently of any particular process, and we did not give any special importance to the process, but considered also the system to maintain (3 concepts in Ruiz et al. ontology), the skills needed for maintenance, etc. On the second point, although we do not reject the idea of having tools to help manage the knowledge (and the need for it was actually raised by the software engineers during our experiments), we follow Rus and Lindvall's recommendation that "it is a mistake for organizations to focus only on technology and not on methodology" [34, p.34]. We do believe it is important to first establish a culture of knowledge management before trying to automate things. This is why we concentrated first on defining a method for PMA.

Two other closely related publications [31,32] deal with knowledge for software maintenance, but once again they consider primarily knowledge on the process, ignoring the larger necessities maintenance has.

Another approach, looking to record and recover knowledge with the aid of specialized tools, is that of Deridder [12]. He proposes to help maintenance using a tool that would keep explicit knowledge about the application domain (in the form of concepts and relations between them) and would keep links between these concepts and their implementation. His approach is mainly concerned with the tool and has no backing ontology. It concentrates, a priori, on application domain knowledge (although the tool would probably allow representing any concept in other domains).

8. Conclusion

Software Maintenance is a knowledge intensive activity. Software maintainers need knowledge of the application domain of a legacy software, the problem it solves, the requirements for this problem, the architecture of the system, how it interacts with its environment, etc. All this knowledge may come from diverse sources: experience of the maintainers, knowledge of users, documentation, source code,… Most of the time however, the knowledge once acquired stays in someone's head as opposed to be formally documented for later retrieval and reuse. When a maintainer leaves the organization, all the knowledge he/she gathered on the various systems he/she worked on, is lost for this organization.

This process is costly, and studies suggest that about 50% of the cost of maintenance is spent on recreating it [29, p.35]. In this paper, we submit that this lack of knowledge is one of the prominent problems in software maintenance, and we look for some solutions to help solve it:

- We designed an ontology of the knowledge useful to software maintenance as a framework to support other knowledge management solutions;
- We experimented Post-Mortem Analysis to help in eliciting knowledge acquired during maintenance and record it.

The ontology of the knowledge useful to software maintenance may be seen as a reference, listing all the concepts we need to worry about; or it may be seen as a classification scheme to categorize pieces of information that we may gather; it could also be used as a common description of maintenance for various tools trying to exchange information. We did not explore this last part.

People usually do not know what they know. This is why techniques such as Post-Mortem Analysis (PMA) are needed to elicit knowledge. PMA is a tool now common in software engineering, however it has mainly been used in software development projects to help gather lessons learned for process improvement. In a software maintenance context, we need to elicit other types of knowledge, for example, knowledge about the system maintained or the application domain. For this, we designed a PMA method where a questionnaire is used to focus the mind of the maintainers on the bits of knowledge that we are interested in.

We experimented our PMA method on different maintenance projects of various size and obtained good results:

- the capability of PMA to uncover lessons learned from projects has not been diminished and several possible improvements were proposed;
- we demonstrated the possibility to elicit other types of knowledge, particularly knowledge gained on the system maintained and the application domain;
- we believe we actually started to create a culture of knowledge management in the subject organization. This was not a goal we actively pursued, but came as a gratifying outcome.

We are still involved with knowledge management for software maintenance, and a new goal for us will be to find a way to disseminate the knowledge gained among a large body of people. We feel this is but imperfectly dealt with by current methods such as recording knowledge on paper or in database.

References

[1] Victor Basili, Patricia Costa, Mikael Lindval, Manoel Mendona, Carolyn Seaman, Tesoriero Roseanne, Marvin Zelkowitz. An experience management system for a software engineering research organization, in: Proceedings of the 26th Annual NASA Goddard Software Engineering Workshop. NASA Goddard Space Flight Center, 2001.

[2] Victor R. Basili, Gianluigi Caldiera, H. Dieter Rombach. Encyclopedia of Software Engineering, volume 1, chapter The Experience Factory, pages 469–76. John Wiley & Sons, 1994.

[3] Victor R. Basili, Mikael Lindvall, Patricia Costa. Implementing the experience factory concepts as a set of experience bases, in: Proceedings of 13th International Conference on Software Engineering and Knowledge Engineering, SEKE'01, pp. 102–109. Knowledge Systems Institute, 2001.

[4] T.J. Biggerstaff, B.G. Mitbander, D. Webster, Program understanding and the concept assignment problem, Commun. ACM 37 (5) (1994) 72–83.

[5] A. Birk, T. Dingsøyr, T. Stålhane, Postmortem: never leave a project without it, IEEE Softw. 19 (3) (2002) 43–45.

[6] G. Booch, J. Rumbaugh, I. Jacobson, The Unified Modeling Language User Guide, Addison-Wesley, 1998.

[7] Lionel C. Briand, Victor R. Basili, Yong-Mi Kim, Donald R. Squier, A change analysis process to characterize software maintenance projects, in: International Conference on Software Maintenance/ICSM'94, pp. 1–12, 1994.

[8] B. Collier, T. DeMarco, P. Fearey, A defined process for postmortem review, IEEE Softw. 13 (4) (1996) 65–72.

[9] T.H. Davenport, P. Laurence, Working Knowledge: How Organizations Manage What They Know, Harvard Business School Press, Boston, 1998.

[10] K.M. de Oliveira, F. Zlot, A.R. Rocha, G.H. Travassos, C. Gallota, C. Menezes, Domain-oriented software development environment, J. Syst. Softw. 172 (2) (2004) 145–161.

[11] Kleiber D. de Sousa, Nicolas Anquetil, Káthia M. de Oliveira, Learning software maintenance organizations, in: Grigori Melnik, Harald Holz (Eds.), Advances in Learning Software Organizations – 6th International Workshop, LSO 2004, No. 3096 in Lecture Notes in Computer Science, pp. 67–77. Verlag, June 2004. ISBN 3-540-22192-1.

[12] Dirk Deridder, Facilitating software maintenance and reuse activities with a concept-oriented approach, Technical report, Programming Technology Lab – Vrije Universiteit Brussel, May 2002.

[13] M.G. Batista Dias, N. Anquetil, K.M. de Oliveira, Organizing the knowledge used in software maintenance, J. Universal Comput. Sci. 9 (7) (2003) 641–658.

[14] T. Dingsøyr, R. Conradi, A survey of case studies of the use of knowledge management in software engineering, Int. J. Softw. Eng. Knowl. Eng. 12 (4) (2002) 391–414.

[15] Torgeir Dingsøyr, Nils Brede Moe, Nytrø Øystein, Augmenting experience reports with lightweight postmortem reviews, Lecture Notes in Computer Science, 2188(2001) 167–181, PROFES 2001, Berlin, Germany.

[16] Michael Gruninger, Mark S. Fox, Methodology for the design and evaluation of ontologies, in: Workshop on Basic Ontological Issues in Knowledge Sharing/IJCAI'95, August 1995, Also available as a Technical Report from the Department of Industrial Engineering, University of Toronto.

[17] Nicolas Guarino (Ed.), Formal Ontology in Information Systems. Frontiers in Artificial Intelligence and Applications. IOS Press, Amsterdam, 1998.

[18] Standard for software maintenance. Technical report, IEEE – Institute of Electrical and Electronics Engineers, May 1998. ISBN: 0738103365.

[19] ISO/IEC 12207 Information technology – Software life cycle processes. Technical Report 12207, ISO/IEC, 1995.

[20] ISO/IEC 14764: Information technology – Software Maintenance. Technical Report 14764, Joint Technical Commitee International Standards Organization/International Electrotechnique Commission, 1999.

[21] I. Jacobson, G. Booch, J. Rumbaugh, The Unified Software Development Process, Addison-Wesley, 1999.

[22] M. Jørgensen, D.I.K. Sjøberg, Impact of experience on maintenance skills, J. Softw. Maint.: Res. Pract. 14 (2) (2002) 123–146.

[23] Norman L. Kerth, An approach to postmorta, postparta and post project review, On Lione: http://c2.com/doc/ppm.pdf. Last accessed on: 06/01/2003.

[24] B.A. Kitchenham, G.H. Travassos, A. von Mayrhauser, F. Niessink, N.F. Schneidewind, J. Singer, S. Takada, R. Vehvilainen, H. Yang, Towards an ontology of software maintenance, J. Softw. Maint.: Res. Pract. 11 (1999) 365–389.

[25] M.M. Lehman, Programs, life cycles and the laws of software evolution, Proc. IEEE 68 (9) (1980) 1060–1076.

[26] Ikujiro Nonaka, Hirotaka Takeuchi, The Knowledge-Creating Company. Oxford University Press, 1995. ISBN 0195092694.

[27] S.L. Pfleeger, What software engineering can learn from soccer, IEEE Softw. 19 (6) (2002) 64–65.

[28] S.L. Pfleeger, Software Engineering: Theory and Practice, second ed., Prentice Hall, 2001.

[29] T.M. Pigoski, Practical Software Maintenance: Best Practices for Software Investment, John Wiley & Sons, Inc., 1996.

[30] Linda Rising, Patterns in postmortems, in: Proceedings of the 23rd Annual International Computer Software and Applications Conference, pp. 314–15. IEEE, IEEE Comp. Soc. Press, October 25–26, 1999.

[31] Oscar M. Rodriguez, Aurora Vizcaíno, Ana I. Martínez, Mario Piattini, Jesús Favela, How to manage knowledge in the software maintenance process, in: Grigori Melnik, Harald Holz (Eds), Advances in Learning Software Organizations – 6th International Workshop, LSO 2004, Lecture Notes in Computer Science 3096, pp. 78–87. Springer Verlag, June 2004. ISBN: 3-540-22192-1.

[32] Oscar M. Rodriguez, Aurora Vizcaíno, Ana I. Martínez, Mario Piattini, Jesús Favela, Using a multi-agen architecture to manage knowledge in the software maintenance process, in: Mircea Gh. Negoita, Robert J. Howlett, Lakhmi C. Jain (Eds.), Proceedings of the 8th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, KES 2004, Lecture Notes in Computer Science 3213, pp. 1181–1188, Springer Verlag, 2004. ISSN 0302-9743.

[33] F. Ruiz, A. Vizcaíno, M. Piattini, F. García, An ontology for the management of software maintenance projects, Int. J. Softw. Eng. Knowl. Eng. – SEKE 14 (3) (2004) 323–349.

[34] I. Rus, M. Lindvall, Knowledge management in software engineering, IEEE Softw. 19 (3) (2002) 26–38.

[35] K. Schneider, J.-P. von Hunnius, V.R. Basili, Experience in implementing a learning software organization, IEEE Softw. 19 (3) (2002) 46–49.
[36] Carolyn B. Seaman, Unexpected benefits of an experience repository for maintenance researchers, in: International Workshop on Empirical Studies of Software Maintenance, WESS'00, http://hometown.aol.com/geshome/wess2000/metricsandmodels.htm, October 2000. accessed on: 02/01/2005.
[37] Mark S. Fox, Mihai Barbuceanu, Michael Gruninger, An organization ontology for enterprise modeling: preliminary concepts for linking structure and behaviour, Comput. Ind. 29 (1996) 123–134.
[38] Tor Stålhane, Torgeir Dingsøyr, Geir K. Hanssen, Nils Brede Moe. Post-mortem – an assessment of two approaches, in: Proceedings of the European Software Process Improvement 2001 (EuroSPI 2001), October 10–12, 2001.
[39] Amrit Tiwana. The Knowledge Management Toolkit. Prentice Hall PTR, 2000.
[40] T.R. Gruber, Toward principles for the design of ontologies used for knowledge sharing, Int. J. Hum. Comput. Stud. 43 (5-6) (1995) 907–928.
[41] J. Valett, S. Condon, L. Briand, Y.-M. Kim, V. Basili. Building an experience factory for maintenance, in: Proceedings 19th Annual Software Engineering Workshop. NASA Goddard Space Flight Center, November 1994.
[42] Anneliese von Mayrhauser, A. Marie Vans. Dynamic code cognition behaviors for large scale code, in: Proceedings of 3rd Workshop on Program Comprehension, WPC'94, pp. 74–81. IEEE, IEEE Comp. Soc. Press, November 1994.
[43] Yourdon (Ed.), Minipostmortems. COMPUTERWORLD, March 19, 2001.