Knowledge for Software Maintenance.

Conference Paper · January 2003

Source: DBLP

5 authors, including:

Nicolas Anquetil
University of Lille Nord de France
152 PUBLICATIONS **3,017** CITATIONS
SEE PROFILE

Káthia Marçal de Oliveira
Université Polytechnique Hauts-de-France
170 PUBLICATIONS **1,634** CITATIONS
SEE PROFILE

Marcio Dias
Centro Universitario de Goias - UNIGOIAS
5 PUBLICATIONS **141** CITATIONS
SEE PROFILE

Knowledge for Software Maintenance

Nicolas Anquetil, Káthia Oliveira, Márcio Greyck Dias, Marcelo Ramal, Ricardo Meneses
UCB - Universidade Católica de Brasília
SGAN 916 Módulo B - Av. W5 Norte
Brasília - DF - 70.790-160, Brazil
{kathia, anquetil}@ucb.br

Abstract

Knowledge management is emerging as a promising area to support software engineering activities. The general idea is to use knowledge gained in previous projects to help future ones. We believe this approach is even more relevant when considering software maintenance where maintainers often have to modify a system that they did not develop, that has no documentation, and that nobody knows intimately. Contrasting with this lack of information on the system, maintainers need a lot of knowledge: about the application domain, the organization software maintenance procedures, the system itself, the language used, past development methods, etc. Although one can readily agree with this fact, there is no clear, exhaustive definition of what knowledge would be useful to perform software maintenance. In this paper we describe our research to identify these needs. This research is part of a long term project that aims at building a knowledge management system for software maintenance.

1. Introduction

The knowledge used in software engineering is diverse, its proportions are immense and it is steadily growing. Based on this premise, research in software engineering has invested in knowledge management approaches trying to make better decisions and provide organizations with different kind of knowledge needed during software development and maintenance [1]. Software maintenance, in particular, is a knowledge intensive activity. Maintainers need knowledge of the application domain, of the organization using the software, of past and present software engineering practices, of different programming languages (in their different versions), programming skills, etc. Concurrently to this knowledge need, a recurring problem of software maintenance is the lack of system documentation. Studies report that 40% to 60% of the software maintenance effort is devoted to understanding the system [2, p.475] [3, p.35]. Having a knowledge management system in this domain would seem of great interest for research and industry.

Following the same trend of thoughts, we started a project aiming at building a knowledge management system that could assist in the software maintenance activities (e.g., system investigation, reverse engineering, etc.). However, research is still in an early stage and numerous questions need to be answered: What knowledge should be targeted first? What knowledge is really important during software maintenance? Where to find this knowledge? How to best make it available?

To answer these questions we decided to first investigate what knowledge is really useful for software maintenance.

In the following sections we first (section 2) discuss software maintenance: its importance and problems. Then (section 3), we present the steps we followed to investigate and define the knowledge used during maintenance. Section 4 discusses some related work. Finally, in section 5 we conclude and propose future work.

2. Software Maintenance

The last decade or so has seen huge progress in software development techniques: new processes, languages, tools, etc. have been proposed and adopted. Software maintenance, on the contrary, seems to lag behind: "this extremely relevant subject receives relatively little attention in the technical literature" (R.S. Pressman in the foreword of [3]). Systematization of maintenance is difficult because it is fundamentally a reactive activity, hence more chaotic than development. Maintenance results from the necessity of adapting software systems to an ever changing environment. In most cases, it can be neither avoided nor delayed much: One as little control on the promulgation of new laws or on the concurrence's progresses. Organizations must keep pace with these changes, and this often means, modifying the software that support their business activities.

As a consequence, software maintenance happens in a relatively disorganized way and naturally leads to the deterioration of software systems' structure (Lehman's second law of software evolution [4]). This gradual lost of structure is as much the result as the cause of the lack of knowledge, maintenance teams have on the software systems they work on. Lacking a complete knowledge of all the implementation details, they apply modifications

that will result in a lost of structure which in turn makes the systems more difficult to understand fully and therefore to maintain.

To break this vicious circle we aim at developing a knowledge management system for software maintenance. One of the first step of our research was to identify what knowledge is needed during maintenance and also what knowledge appears more dearly needed.

3. Knowledge for Software Maintenance

To achieve our goal of identifying what knowledge is used in software maintenance, we followed four steps. The first step was to observe software maintainers in their daily activities to get a first idea of what knowledge they use. In this pilot study, we tried to observe the software engineers with as little preconception as possible. For example knowledge categories were not predefined but extracted from the results of the study. In the second step we formally modeled the knowledge needed for software maintenance using this pilot study, our experience and a literature review. The third and the fourth steps were, respectively, the validation of this formal model and some field research to identify the knowledge used in maintenance. In this section we present each one of those steps.

3.1 The pilot study

In the pilot study, we observed software engineers performing maintenance. From what they did and said (they were asked to "think aloud" [5]), we deduced the knowledge they were using. This knowledge was decomposed in "knowledge atoms" that we then categorized in general knowledge domains. We finally analyzed the repartition of knowledge atoms in the various domains and draw some conclusions from this (the study was published in [6] where more details on the experimental conditions and the results may be found).

We studied six software engineers in two different organizations (a bank and a public administration) during 13 sessions of 75 minutes on average. The sessions followed a protocol called think-aloud [5] where the maintainers were asked to say everything they did and why they did it. These sessions were recorded and later transcribed on paper to be analyzed. As said earlier, during the analysis, we tried to identify the kind of knowledge that the software engineers were using at each moment. For example, a session would contain: (i) "The maintainer looks for the class containing a method to consult the database", (ii) "The maintainer opens this class …", (iii) "... and goes to the method", (iv) "The maintainer removes the part of this method that references the old tables", etc. From this we would conclude that (i) the maintainer had some previous knowledge of the source code, as well as knew how to use the programming environment; (ii) again the maintainer was using a particular functionality of the programming environment; (iii) and again s-he was showing that s-he knew the system beforehand; and finally, (iv) the maintainer demonstrated programming knowledge as well as more knowledge about the system implementation (what tables were to be modified). From the study of the 13 sessions we extracted the following domains of knowledge:

- **Computer science knowledge**: Knowledge that a typical software engineer would have. This domain was subdivided in:
  - A1) Programming: Knowledge of how to use basic instructions such as loops or tests, object orientation, or understanding a piece of code.
  - A2) Programming language: Knowledge of the syntax and peculiarities of a given programming language.
  - A3) Development environment: Knowledge of the tools available in the organization to help in software development/debugging.
  - A4) Application implementation: Knowledge of how the application considered is implemented.
  - A5) Diagramming: Knowledge on design techniques and the "languages" used such as DFD, the different UML diagrams, etc.
  - A6) Organization's programming rules: Knowledge of the organization's rules that apply to computation such as naming conventions.
- **Business knowledge**: Knowledge that a typical member of the organization considered (not limited to computer scientist) would have. This domain was subdivided in:
  - B1) Application functionalities: Knowledge of what the application does, what are its functionalities.
  - B2) Organization: Knowledge of the day to day living in the organization: where to find ink for the printer or the latest backup tape, who is responsible for what.
  - B3) Problem analyzed: Knowledge of the particular situation at hand for having already encountered it.
  - B4) Production environment: Knowledge of the environment a typical user is confronted with every day such as using some given database (in production).
- **General knowledge:** Common sense knowledge that "anyone" could have. It was subdivided in:
  - C1) Additional tools: Knowledge of foreign languages, web browser, email.
  - C2) Help: Manuals, Technet.
  - C3) Other knowledge

As already mentioned, we tried to start this pilot study with as little preconceptions as possible, and this classification was established bottom-up, grouping

together knowledge items that were found in the sessions analyzed.

Among the results of the study we found that:

- Maintainers very rarely looked for new knowledge, they tended to work formulating hypotheses from what they already knew.
- Maintainers made –quantitatively– very little use of business knowledge during these sessions. Overall, 12% of the knowledge atoms found were classified in the business domain, 13% in the general domain and 74% in the computer science domain.
- We identified differences between the two organizations, which could be cultural. Maintainers from one organization typically used more business knowledge than general knowledge whereas it was the opposite for the other organization. The difference was explained by the fact that maintainers from the second organization make more use of such help tools as MSN (www.msn.com). We actually do not know whether this difference stems from different organizational cultures or different needs

Other issues of interest could not be settled, they include: study of the influence of experience on the knowledge used, and study of the influence of the type of maintenance (correction, evolution) on the knowledge used.

3.2 Formalizing the knowledge

After this pilot study, we started a more formal approach. Based on the concepts identified in the pilot study, a literature review, and our own experience, we defined an ontology of the knowledge needed during maintenance (further details on this part of the study may be found in [7]). An ontology is a description of domain through the identification and description of its entities, their properties, relationships, and constraints [8].

We started the ontology construction by looking for motivating scenarios where the knowledge captured would be useful. Some of those scenarios are: deciding who is the best maintainer to allocate to a modification request based on her-his experience of the technology and the system considered; learning about a system the maintainer will modify (which are its documents and components and where to find it); defining the software maintenance activities to be followed in a specific software maintenance, and also the resources necessary to do those activities.

These and other situations induced us to organize the knowledge around five different aspects: knowledge about the Software System itself (covering domains A4 and B1 of the pilot study); knowledge about the Maintainer's Skills (covering domains A1, A2, A3, A5, B3, B4, C1 and C2 of the pilot study); knowledge about the Maintenance Activity (not identified in the pilot study); knowledge about the Organization Structure (covering domains A6 and B2 of the pilot study); and knowledge about the Application Domain (covering domain B1 of the pilot study). Each of these aspects was described in a sub-ontology. For each one of the sub-ontologies we defined "competency questions", captured the necessary concepts to answer these questions, established relationships among the concepts, described the concepts in a glossary and validated them with experts.

For lack of space, we will not give a detailed description of the ontology here but will only outline the System sub-ontology (Figure 1) to give a sense of what we did. The reader interested in more details may want to look at [7]. The System sub-ontology is one of two sub-ontologies corresponding to the more computer science oriented knowledge which showed the higher use in the pilot study (74% of the knowledge atoms). Knowledge

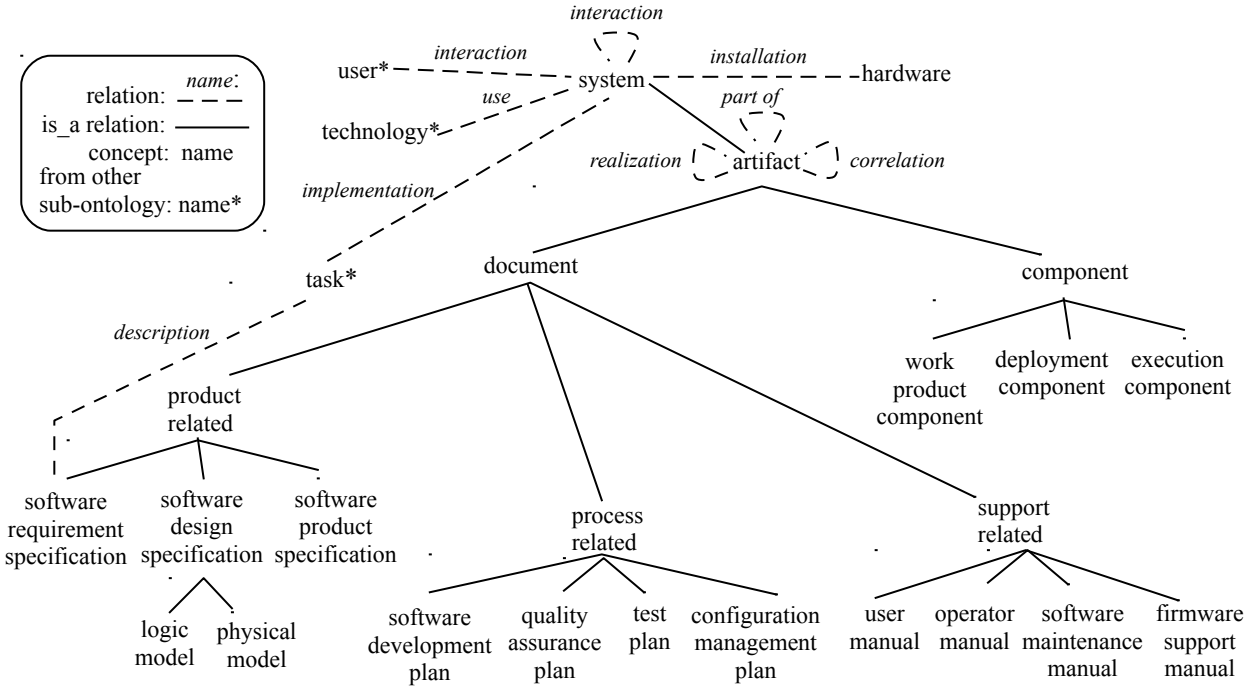

Figure 1: The System sub-ontology

about the system is also intuitively fundamental to software maintenance.

The competency questions for the System sub-ontology are: What are the artifacts of a software system? How do they relate to each other? Which technologies are used by a software system? Where is the system installed? Who are the software system users? Which functionalities from the application domain are considered by the software system?

Answering these questions led to a decomposition of the software system in artifacts, a taxonomy of those artifacts and the identification of the hardware where the system is installed, its users and the technologies that was used in its development.

The artifacts of a system can generally be decomposed in documentation and software components. Briand [10] considers three kinds of documentation: (i) product related, describing the system itself (i.e., software requirement specification, software design specification, and software product specification); (ii) process related, used to conduct software development and maintenance (i.e., software development plan, quality assurance plan, test plan, and configuration management plan); and (iii) support related, helping to operate the system (i.e., user manual, operator manual, software maintenance manual, firmware support manual). Considering that the software design specification proposed by Briand should represent the behavior and structure of the system and that we can have different abstraction models we refined the software design specification in logic and physical model.

Software components represent all the coded artifacts that compose the software program itself. Booch [11] 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. We call the first kind of relation *realization*, relating two artifacts of different abstraction levels. The second one is a *correlation* of artifacts at the same abstraction level. To express those constraints we defined a set of axioms like (i) $(\forall\ a_1,a_2)\ (correlation(a_1,a_2) \wedge requirementspec(a_1) \rightarrow requirementspec(a_2))$ and (ii) $(\forall\ a_1,a_2)\ (realization(a_1,a_2) \wedge requirementspec(a_1) \rightarrow \neg requirementspec(a_2))$. In total we defined 21 axioms for the system sub-ontology and 53 for the whole ontology.

Sub-Ontology	# concepts in ontology	instantiated concepts	
		#	%
Skill	38	28	74%
Application domain	4	2	50%
Modification	30	24	80%
System	23	16	70%
Organizational Structure	3	3	100%
Total	**98**	**73**	**74%**

Other relations in this sub-ontology are: the software system *is installed* on some hardware, the user interacts with the software system, and finally, the software specification *describes* the domain tasks to be implemented (or functionalities).

3.3 Quality Assessment

In this section we will consider the validation of the quality of the ontology with regard to the six following desirable criteria: consistent, clear, general, complete, concise, and robust. To evaluate these aspects, we asked four experts to study the ontology and fill a quality assessment report. These people were chosen for their large experience in software maintenance or for their academic background. The evaluations were good, as may be seen in Figure 2, on a scale of 0 to 4 no criterion ranked below 3 (on average).

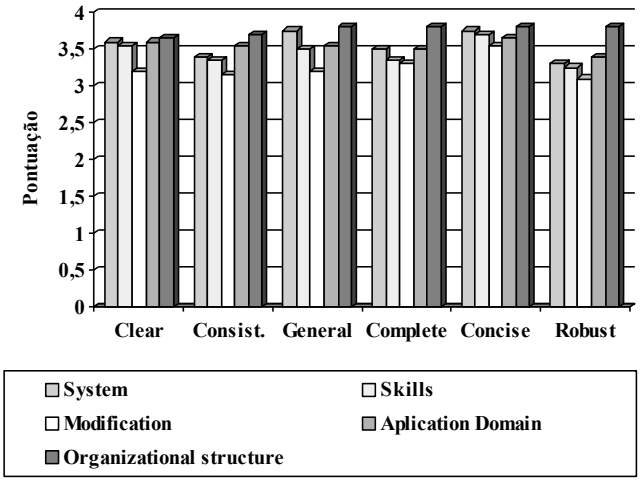

Figure 2: Result of the ontology's quality assessment

This evaluation was useful in pointing out specific problems. For example, we had not included a relation to specify that software systems may interact between themselves, the CASE taxonomy (Skills sub-ontology) did not contemplates utility tools for execution, some definitions were not clear (this is the main reason behind the lower score of the Modification sub-ontology), or some restrictions were not expressed.

Besides the expert assessment, we also instantiated the concepts from the documentation of a real software system. This documentation came from the development of the system as well as from past maintenances.

Table 1 shows the number of concepts that were instantiated for each sub-ontology.

Table 1: Number of concepts instantiated from the study of the documentation of one system

One may observe that 26 concepts from the ontology were not instantiated. There are various explanations for this:

- Two concepts from the Application Domain sub-ontology were not instantiated because they were not identified when we did the experiment, they resulted from the quality assessment. However, a later rapid survey of the same documentation allowed us to locate instances of these concepts.
- Five concepts from the Skills sub-ontology were not instantiated because the organization does not have the technical competencies to perform them (e.g.: Reverse Engineering Technique). One concept (Modeling Language) was not instantiated because the maintenance activities were specified in (structured) natural language. Although this is a debatable question we decided at the moment, not to consider this as a modeling language. Two more concepts (Method and Guidelines) were not instantiated because the organization ceased to use any maintenance process after a change in the management. The concept: CASE for Modeling, was not instantiated because the organization does not have the necessary resources to invest in this type of tool. Finally a tenth concept (Utility) was not present in the ontology when we did the test. As for the other cases, we have proofs of its existence in the organization studied.
- Three concepts from the Modification ontology were not instantiated for lack of enough examples, there were only examples of Perfective maintenance and no Corrective, Preventive or Adaptative maintenance. For the same lack of example, three possible causes for a maintenance were not found: Security, Architectural Design and On-Line Documentation.
- In the System sub-ontology, one concept was not instantiated for the same reason: The maintenance examples we studied did not include any reference to the Product Specifications. The system studied was produced internally, thus having no reference to an external Provider. Another concept, Hardware Manual, is handled by a different unit of the organization. And finally, four concepts (Development, Configuration and Quality plans, Maintenance Manual) were not instantiated after the organization ceased to use any process, as already mentioned earlier.

It must be noted that the fact that a concept was instantiated here does not say whether it is useful for maintenance or not, but only that it was found in the documentation.

3.4 Usefulness Assessment

One of the objectives of the ontology was to represent the knowledge useful to maintenance. The preceding section presents results from the assessment of the quality of the ontology in representing knowledge. In this section we present a validation of the usefulness of the concepts represented for maintenance. To do so, we realized two types of experiment: (i) observing maintainers with the think-aloud protocol, as in the pilot study, and (ii) presenting the instantiated knowledge for a software system before the maintenance activity and then asking the software engineers what knowledge was used.

The think-aloud experiment (i) followed basically the same procedure as in the pilot study (section 3.1): the maintainers explained everything they were doing while maintaining a system, the sessions were taped, the tapes were transcribed, and finally, we identified and classified the knowledge atoms. The only difference from the pilot study was that this last step was based on the ontology. Two maintainers participated in this experiment, doing five sessions for a total of 132 minutes (26 minutes per session on average).

In the second experiment (ii), the ontology was presented and explained to the software maintainers and they were asked to fill in, every day, a questionnaire on the concepts they used. This form consisted of all the concepts we had instantiated previously (section 3.3) and the list of their instances (as we identified them). The maintainers were simply asked to tick the instances they had used during the day. They could not add new instances. The experiment was done with three maintainers and one manager. They filled 17 forms in 11 different days over a period of 10 weeks.

The results of these two experiments are given in Table 2. One may observe that there are a lot less concepts used in the first one than in the second. One reason for this is that there were fewer sessions in the first experiment and they were mostly short punctual maintenance (average of 26 minutes compared, for example, with the 75 minutes per session in the pilot study).

All concept uses detected in the first experiment were also found in the second one, it did not bring in any new instances. From this and the results in Table 2, one can deduce that only six concepts instantiated in the previous section (3.3) were not found here:

- Analysis Technique and Requirement Specification Technique (Skills sub-ontology) were not used because the maintenance operations were relatively simple and restricted to small modification to the source code. Therefore, no high level analysis was required.

- For the same reason, the Requirement Specifications (System sub-ontology) was neither studied nor modified.
- The creation, modification and distribution of all Support Documentation (including User Manual and Operation Manual, the three of them being in the System ontology) falls under the responsibility of another unit, therefore the software engineers we studied need not know about them or use them.

Table 2: Number of concepts used in two experiments

	Ontology	Think-aloud		Questionnaire	
	#	#	%	#	%
Skill	38	15	39%	26	68%
Application domain	4	1	25%	2	50%
Modification	30	16	53%	23	77%
System	23	9	39%	13	57%
Organizational Structure	3	2	67%	3	100%
Total	**98**	**43**	**44%**	**67**	**68%**

We did not yet study the importance of each concept or sub-ontology in doing maintenance. There are two reasons for this: First we fill that we need more data points (maintenance session) on various systems, possibly in different organizations to get some statistically valid results; second, the Application Domain sub-ontology is actually a meta-ontology with the concepts Concept, Attribute, Relation, and Restriction. To conduct a valid experiment, we would need an actual ontology of a given application domain with all its concepts, relations, etc .

4. Related Work

As seen in section 3, our investigation of the knowledge necessary to perform maintenance included the definition of an ontology for maintenance. Before developing it, we studied the literature on knowledge based approaches to software maintenance. The following publications were found to be relevant to our research and greatly helped in the definition of the ontology although they did not solve completely our problem.

There are various proposition of mental models to describe how software engineers go about doing maintenance [12,13]. These works offer little interest since they concentrate on the process of doing maintenance rather than on the knowledge used.

In [14], the authors studied "the knowledge required to understand a program". Knowledge is classified in 3 domains: Domain knowledge (numerical analysis), Fortran knowledge and Programming knowledge. The first one correspond to our Application Domain sub-ontology, and the two others fall into our Skills sub-ontology. The problem of this study is that it concentrates specifically on program comprehension which is just one of the many tasks performed when maintaining a system. Also it is based on a toy program (102 lines of Fortran) in conditions that do not resemble real world maintenance environment.

Briand and Basili [10] identified factors that could influence the quality and productivity of software maintenance. The work is interesting because it includes various taxonomies of important concepts as: maintenance methods and tools, maintenance documentation, human mistakes, process failures, and maintenance teams. Although the focus of this work was not on knowledge it offers valuable insights on concepts that are important to maintenance (e.g.: methods and tools taxonomy, documentation taxonomy) as well as explicit classification of these concepts. We reused several of their taxonomies in our ontology.

Deridder, in [15], 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. He follows a trend of thought very similar to ours, but concentrates exclusively on application domain knowledge whereas we identified four other sub-ontologies that had useful concepts in them. Also, he concentrates on how to acquire and use this knowledge rather than extensively identify it (which would actually depend on every single application domain).

Finally, Kitchenham *et al.* in [9] designed an ontology of software maintenance. In this ontology, they identified all the concepts relevant to the classification of empirical studies in software maintenance, these concepts are classified along four main axes: the People, the Process, the Product, and the Organization. These four axes correspond respectively to our Skills, Modification, System and Organizational Structure sub-ontologies. This was one of the most inspiring work for us and we reused many of its concepts, however due to the particular focus they had when identifying these concepts (providing a framework to help categorize empirical studies on software maintenance), we felt that many concepts were either over or under detailed. The most striking evidence of this is the idea of application domain which we developed as a sub-ontology, whereas it is only included in Kitchenham's work as an attribute of the software system.

5. Conclusion

In this article, we presented some results from our research on the knowledge useful to software maintenance. Following a recent trend in software

engineering, we believe that a knowledge based approach could help solve the difficult problems faced by software maintenance: poor documentation, lack of knowledge about the system maintained from the maintenance teams, poor quality of the code after numerous modification.

We presented several steps of our research:

- First we did an early experiment to get an idea of what kind of knowledge was used and to what extent.
- This led to a few conclusions including the unexpected fact that application domain was – quantitatively– little used.
- One of the failures we perceived in this first step was the lack of a formal definition of the knowledge used in maintenance. We, therefore, decided to define an ontology of the knowledge used in maintenance.
- A third step was to evaluate the quality of our ontology according to traditional quality criteria. The results were good.
- Finally we evaluated the usefulness of the knowledge represented in the ontology. The results were satisfactory (68% of all the concepts were used at least once in our experiments) although we identified that a larger experimental base would be necessary to observe actual use of all the concepts.

This research is intended to be the base of a long term project aiming at building a knowledge based environment to help software maintenance. Future work includes:

- Better evaluation of the usefulness of the concepts contained in the ontology.
- Investigating the possibility of designing manual procedures (process) to populate the ontology.
- Investigating the possibility of creating (semi-) automated tools to assist in populating the ontology from existing systems.
- Build a maintenance assistant tool which would help managers and maintainers perform their task and look for needed knowledge. This tool would use the ontology as a framework to define the knowledge base.

References

1. LINDVALL, I., **Knowledge Management in Software Engineering**, IEEE Software; pp-26-38; May/June 2002.
2. PFLEEGER, S. L. **Software Engineering: Theory and Practice**. 2nd Edition. New- Jersey: Prentice Hall, 2001.
3. PIGOSKI, T. M. **Practical software maintenance: best practices for managing your software investment**. USA: John Wiley & Sons. P.87-102, Dec, 1996.
4. LEHMAN, M.. **On understanding Laws, evolution and conversation in the large program lifecycle**. Journal of Software & Systems, vol. 1, p.213 – 221, 1980.
5. LETHBRIDGE, T. C., SIM, S. E., SINGER, J. **Software Anthropology: Performing Field Studies in Software Companies**. Consortium for Software Engineering Research (CSER), 1996.
6. RAMAL, M. F., MENESES, R., ANQUETIL, N. **A Disturbing Result on the Knowledge Used During Software Maintenance**, Working Conference on Reverse Engineering, Richmond, VA, U.S.A., p. 277-287, 29 Oct-1 Nov., 2002.
7. DIAS, M. G.B., ANQUETIL N., OLIVEIRA K.M., **Organizing the Knowledge Used in Software Maintenance**, Learning Software Organization Workshop, 2003
8. GRÜNINGER, M., FOX, M. S. **Methodology for the Design and Evaluation of Ontologies**. Toronto, CANADA: Technical Report, University of Toronto, 1995.
9. KITCHENHAM, B. A., TRAVASSOS, G. H., MAYRHAUSER, A. *et al*. **Toward an Ontology of Software**. Journal of Software Maintenance: Research and Practice 11(6):365-389, May, 1999.
10. BRIAND, L. C., BASILI, V., KIM, Y., SQUIER, D. R. **A Change Analysis Process to Characterize Software Maintenance Projects**. In: Proceedings of The International Conference on Software Maintenance, 1994.
11. BOOCH, G., RUMBAUGH, J., JACOBSON, I. **UML - Guia do usuário**. Editora CAMPUS, 2000.
12. RUGABER, S., TISDALE, V. G. **Software Psychology Requirements for Software Maintenance Activities**. Software Engineering Research Center, Georgia Institute of Technology, 1992.
13. VON MAYRHAUSER A., VANS A, **Dynamic Code Cognition Behaviors For Large Scale Code**, Proceedings of 3rd Workshop on Program Comprehension, WPC'94, pages 74–81. IEEE, IEEE Comp. Soc. Press, nov. 1994.
14. CLAYTON R., RUGABER S., WILLS L, **On the Knowledge Required to Understand a Program**, Working Conference on Reverse Engineering, pages 69–78. IEEE, IEEE Comp. Soc. Press, Oct. 1998.
15. DERIDDER, D. **Facilitating Software Maintenance and Reuse Activities with a Concept-oriented Approach**, Brussels, Belgium: Programming Technology Lab, Vrije Universiteit Brussel, 2002.