Journal of Universal Computer Science, vol. 9, no. 7 (2003), 641-658
submitted: 3/2/03, accepted: 21/7/03, appeared: 28/7/03 © *J.UCS*

Organizing the Knowledge Used in Software Maintenance

Márcio Greyck Batista Dias
(Universidade Católica de Brasília, Brazil
myck@brturbo.com)

Nicolas Anquetil
(Universidade Católica de Brasília, Brazil
anquetil@ucb.br)

Káthia Marçal de Oliveira
(Universidade Católica de Brasília, Brazil
kathia@ucb.br)

Abstract: Knowledge engineering emerged as a very promising area to help improve software engineering practice. One of its possible applications would be to help in solving the numerous problems that affect the software maintenance activity. Maintainers of legacy systems developed years ago with obsolete techniques and tools, and not documented, need all kinds of knowledge (application domain, programming skills, software engineering techniques, etc.) It is generally assumed that formalizing all this knowledge and recording it would be a worthwhile effort. However, research is still in a early stage and numerous questions need to be answered: What knowledge should be targeted first? Where to find this knowledge? etc. To answer these questions, one needs a precise understanding of what knowledge is at stake here. We, therefore, propose an ontology of the knowledge needed to perform software maintenance. This ontology would be most useful as a framework for future research in knowledge engineering for software maintenance.

Keywords: ontology, software maintenance, knowledge management.
Categories: K.6.3, D.2, D.2.7, D.2.9

1 Introduction

Knowledge management techniques are raising great expectation in the software engineering community. Of particular interest are the possibilities that knowledge management opens to solve the numerous problems in maintenance. Software maintenance must still cope with systems developed years ago, with languages and processes now considered deficient, for computers with severe limitations imposing convoluted algorithms. This is a knowledge intensive activity, maintainers needs 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 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 [Pfleeger 2001] (p.475), [Pigoski 1996] (p.35).

To help maintainers face these difficulties, one could envision specialized tools providing easy access to the various domains of knowledge required. However there is no clear, exhaustive definition of what knowledge would be useful to perform software maintenance. In this paper we describe our research in the identification and organization of this knowledge using ontology.

This work is part of a long-term project that aims at building a knowledge management system to assist in the software maintenance activities (e.g., system investigation, postmortem project review, etc.). This system will address the main requirements for promoting knowledge management in an organization as emphasized by [Rus and Lindvall, 2002]: first, to access domain knowledge, not only knowledge about software engineering itself (in our case specifically maintenance) but also knowledge about the domain for which the software is being developed/maintained; second, to share knowledge about local policies and practices, since new developers/maintainers in an organization need knowledge about the existing software base and local programming conventions; third, to know who knows what, for efficiently staffing projects, identifying training needs, and matching employees with training offers; fourth, to collaborate and share knowledge, independently of time and space. With these needs in mind we started our research in knowledge organization focusing specifically on software maintenance.

In the following sections we first discuss the importance and problems of software maintenance (section 2). In section 3, we give a short definition of ontology, what are its possible uses and how it may be build. Section 4 is the core of the article with the presentation of the ontology for maintenance. Then in section 5, we discuss the evaluation of this ontology. Finally, we discuss related work in section 6 and conclude in section 7.

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 [Pigoski 1996]). 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 has 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 [Lehman 1980]). This gradual loss 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 loss 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 approach for software maintenance. One of the first steps of our research was to identify what knowledge is needed during maintenance and formalize it in an ontology.

3 Ontology Definition and Methodology

An ontology is a description of entities and their properties, relationships, and constraints [Grüninger and Fox 1995]. Ontologies can promote organization and sharing of knowledge, as well as interoperability among systems. There are various methodologies to design an ontology (e.g. [Grüninger and Fox 1995]), all consider basically the following steps: definition of the ontology purpose, conceptualization, validation, and finally coding. The conceptualization is the longest step and requires the definition of the scope of the ontology, definition of its concepts, description of each one (through a glossary, specification of attributes, domain values, and constraints). It represents the knowledge modeling itself.

We defined our ontology using theses steps. The *purpose* is to define an ontology describing the knowledge relevant to the practice of software maintenance. The *conceptualization* step was based on a study of the literature and the experience of the authors. We identified motivating scenarios and competency questions (i.e., requirements in the form of questions that the ontology must answer [Grüninger and Fox 1995]). It resulted in a set of all the concepts that will be presented in the next section. The *validation* will be discussed in section 5. The *formalization* was done using first order logic. There are various editing tools available to describe an ontology (see for example [Staab et al. 2000], [Grosso et al. 1999], [Domingue 1998]), each one using a specific language and having particular features. In this first work, we chose to focus on the identification of the knowledge itself, and did not study any of these tools. We opted for a manual representation of the ontology, which should be later entered into one of these tools. For the same reasons, we described the constraints on relations and concepts, in first order logic.

4 An Ontology for Software Maintenance

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 perform those activities.

These and other situations induced us to organize the knowledge around five different aspects [see Figure 1]: knowledge about the Software System itself; knowledge about the Maintainer's Skills; knowledge about the Maintenance Activity; knowledge about the Organizational Structure; and knowledge about the Application Domain. Each of these aspects was described in a sub-ontology. For each one of the sub-ontologies we defined competency questions [see Section 3], captured the necessary concepts to answer these questions, established relationships among the concepts, described the concepts in a glossary and validated them with experts.

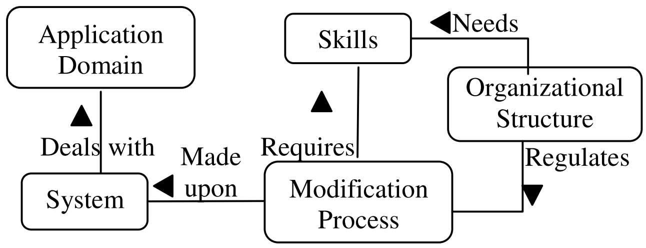

Figure 1: Ontology overview

To express the constraints over the concepts and relations, we defined 53 axioms in first order logic. These do not include axioms formalizing the specialization and composition relationships (i.e. axioms for the "is_a" and "has_a" relations). Some examples of axioms will be presented in the description of each sub-ontology.

Building such an ontology is a significant work. We spent three months to define it, the main investigator working part-time, and the two others participating in weekly validation meetings. Our first difficulty was to define clearly what was to be the focus of the ontology. This was solved defining scenarios (see beginning of the section) for the use of the knowledge. A second difficulty was to review the relevant literature in search of definitions and validation of the concepts. In this phase, we deemed important to base each and every concept on independent sources from the literature. This literature review is summarized in the concept glossary, which will not be presented here for lack of space.

4.1 The System Sub-ontology

The System sub-ontology is one of two sub-ontologies corresponding to the more computer science oriented knowledge. Knowledge about the system is also intuitively fundamental to software maintenance. The sub-ontology is pictured in Figure 2[1].

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.

[1] In all figures of the sub-ontologies, the default cardinality for the relations is 0,n. Cardinality is only represented when it differs from this default.

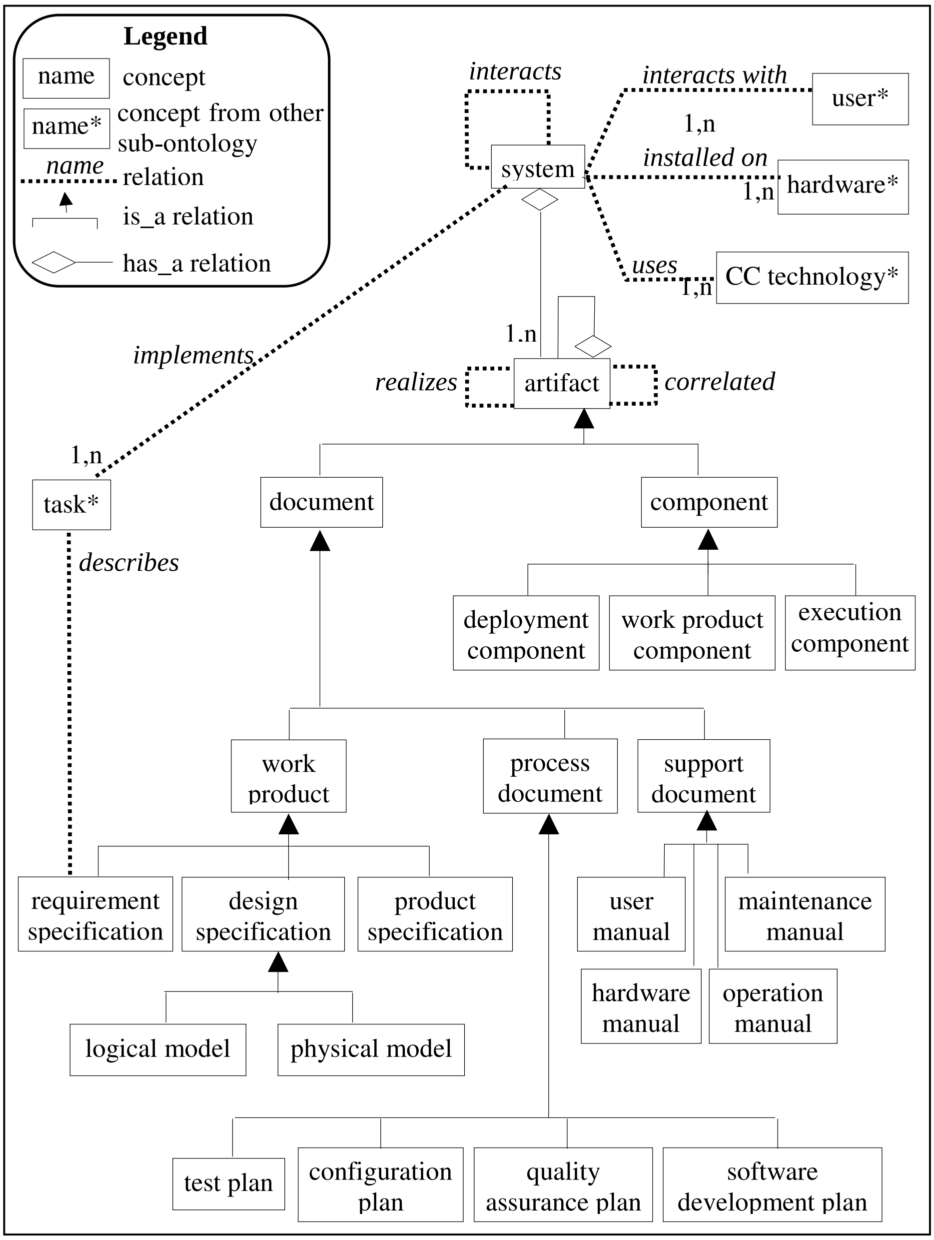

Figure 2: System sub-ontology

The artifacts of a system can generally be decomposed in documentation and software components. Briand [Briand et al. 1994] considers three kinds of documentation: product related, describing the system itself (i.e., software requirement specification, software design specification, and software product specification); process related, used to conduct software development and maintenance (i.e., software development plan, quality assurance plan, test plan, and configuration management plan); and 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 into logical and physical models.

Software components represent all the coded artifacts that compose the software program itself. Booch [Booch et al. 1997] classifies them in: execution components, generated for the software execution; deployment components, composing the executable program; and 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 this first kind of relation *realization*, relating two artifacts of different abstraction levels. Another relation between artifacts is a *correlation* between artifacts at the same abstraction level. And finally, artifacts may be composed of other artifacts (e.g. one document may be composed of several parts).

Other relations in this sub-ontology are: the software system *is installed* on some hardware, the system may *interact* with other systems, the user *interacts* with the software system, the system *implements* some domain tasks to be automated (the functionalities of the system), and finally, the software requirement specifications *describe* these domain tasks. To express the constraints over the relations (e.g. realization or correlation) we defined a set of axioms like $(\forall a_1,a_2)\ (correlation(a_1,a_2) \wedge requirementspec(a_1) \rightarrow requirementspec(a_2))$ and $(\forall a_1,a_2)\ (realization(a_1,a_2) \wedge requirementspec(a_1) \rightarrow \neg\ requirementspec(a_2))$. The fist one specifies that if a_1 is a requirement specification and a_1 correlated to a_2, then a_2 must also be a requirement specification (i.e. the correlation relation stands between artifacts of the same type). Similarly, the second axiom specifies that realization may only stand between two artifacts of different kind.

4.2 The Skills Sub-ontology

Figure 3 shows the second sub-ontology, on the skills in computer science needed by software maintainers. A scenario of use would be to be able to select the best participants for a given type of maintenance. Some competency questions we identified are: What kind of CASE tools does the software maintainer know? What kind of procedures (methods, techniques, and norms) does s-he know? What programing and modeling languages does s-he know?

There are several things a maintainer must know or understand to perform his-her task adequately: s-he must *know* (be trained in) the specific maintenance activity s-he will have to perform (e.g. team management, problem analysis, code modification), 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. To express those relations, we defined axioms like: $(\forall m)\ (maintainer(m) \rightarrow (\exists t)\ (tecnology(t) \wedge knowCCT\ (m,t))$

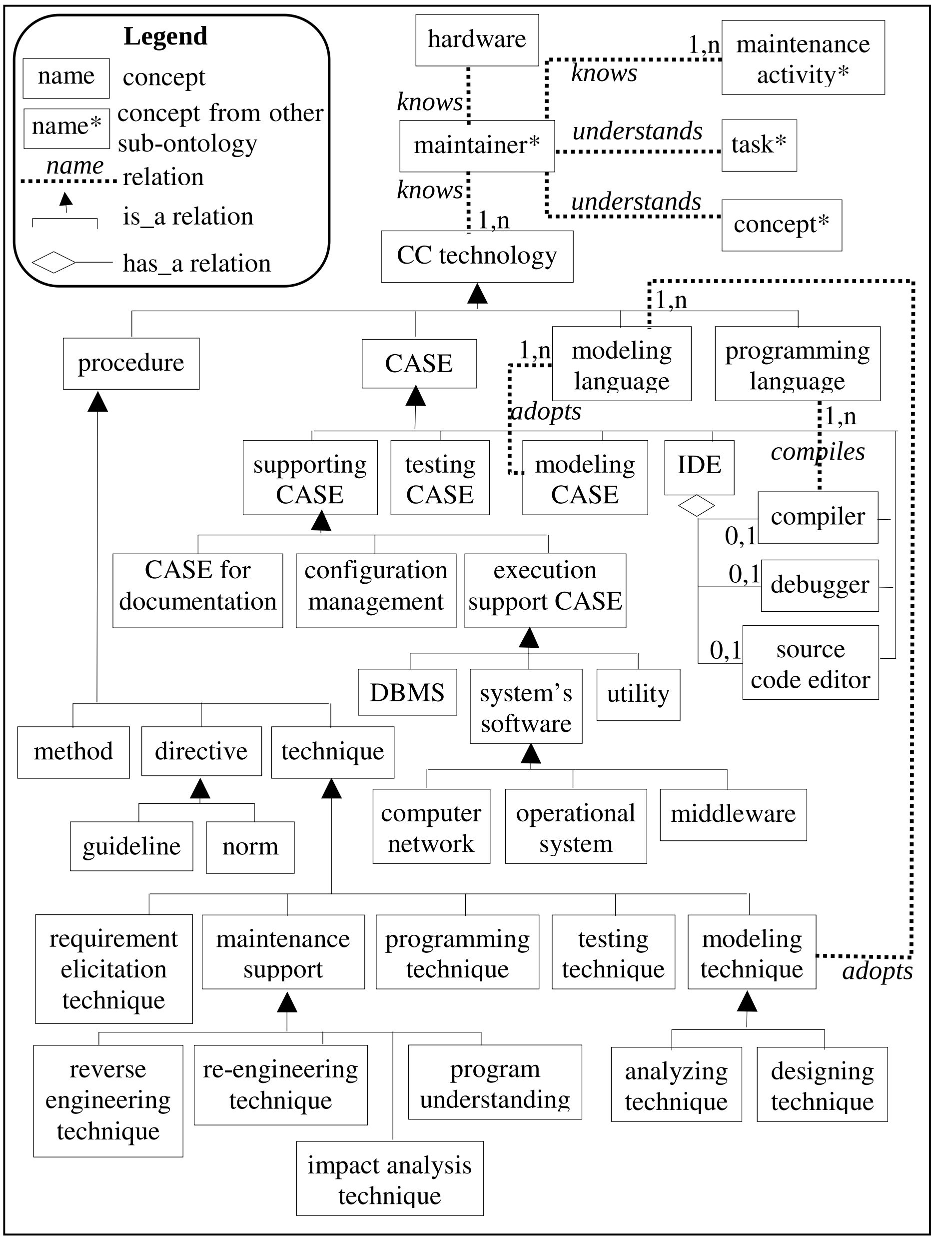

Figure 3: Skills sub-ontology

and *$(\forall m)$ (maintainer(m) $\rightarrow$ $(\exists a)$ (maintenanceActivity(a) $\wedge$ knowsActivity(m, a))* (any maintainer knows at least one tecnology and one activity).

There are four computer science technologies of interest: possible procedures to be followed, modeling language used (e.g. the UML), CASE (Computer Aided Software Engineering) tools used (for modeling, testing, suporting or developing), and finally the programming language used in the system. According to [Kitchenham et al. 1999] procedures are all structured descriptions used in a software development activity like methods (kind of sistematic procedures with semantic and syntatic definitions to be followed), techniques (sistematic procedures less formal and rigorous than a method), and directives (standards like guidelines or norms in an organization). Based on [Chandra and Ramamoorthy 1996],[Leffingwell and Widrig 2000], [Pressman 2001], we classified the techniques in: (a) requirement elicitation, procedures to assist in the identification of the requirents (e.g., interviews, brainstorming, etc.); (b) modeling, procedures, using specific modelling languages, to assist in the definition of a sistematic solution for the problem (classified in analysis and design); (c) programming, procedures for coding (e.g., structured or object oriented programming); (d) testing, procedures for testing the software (e.g., white or black box techinique); and (e) maintenance support, procedures to assist in the modification of a program (classified in reverse engineering, re-engineering, impact analysis and program compreension tecniques [Pigoski 2001]).

Pressman [Pressman 2001] gives a very complete list of CASE tools, with tools for modelling, used for the design model definition according to a specific modelling language; testing, used to define and control tests for a system; developing, that is the Integrated Development Environment (IDE - with compiler, debugger, and editor), and supporting, the execution, documentation, or configuration management. The execution supporting CASE tools represent any tool that can be used in some way during the system's execution like data base management systems, utilities, and system's software (computer network, operational system and all middleware).

4.3 The Modification Process Sub-ontology

Figure 4 shows the concepts of the Modification Process sub-ontology. Here, we are interested in organizing concepts from the modification request (and its causes) to the maintenance activities. Possible competency questions are: What are the types of modification requests? Who can submit them? What are their possible sources? What are the activities performed during maintenance? What does one need to perform them? Who perform them? What do they produce?

According to [Pigoski 1996], a maintenance project *originates* from a modification request *submited* by a client. The requests are classified either as problem report, describing the problem detected by the user, or enhancement request, describing a new requirement. Pigoski also lists the different origins of a modification request (where the problem was detected or the new requirement originates): on-line documentation (like helps and tool tips), execution (features about the execution of the system itself, like performance, instability), architectural design (like dynamic library reuse), requirement (change in a requirement or a specification of a new one), security (like not allowed access), interoperability (features related with the communication with others systems) and data structure (like structure of data files or data bases). One or more modification requests generate a maintenance project that will be composed of different software maintenance activities.

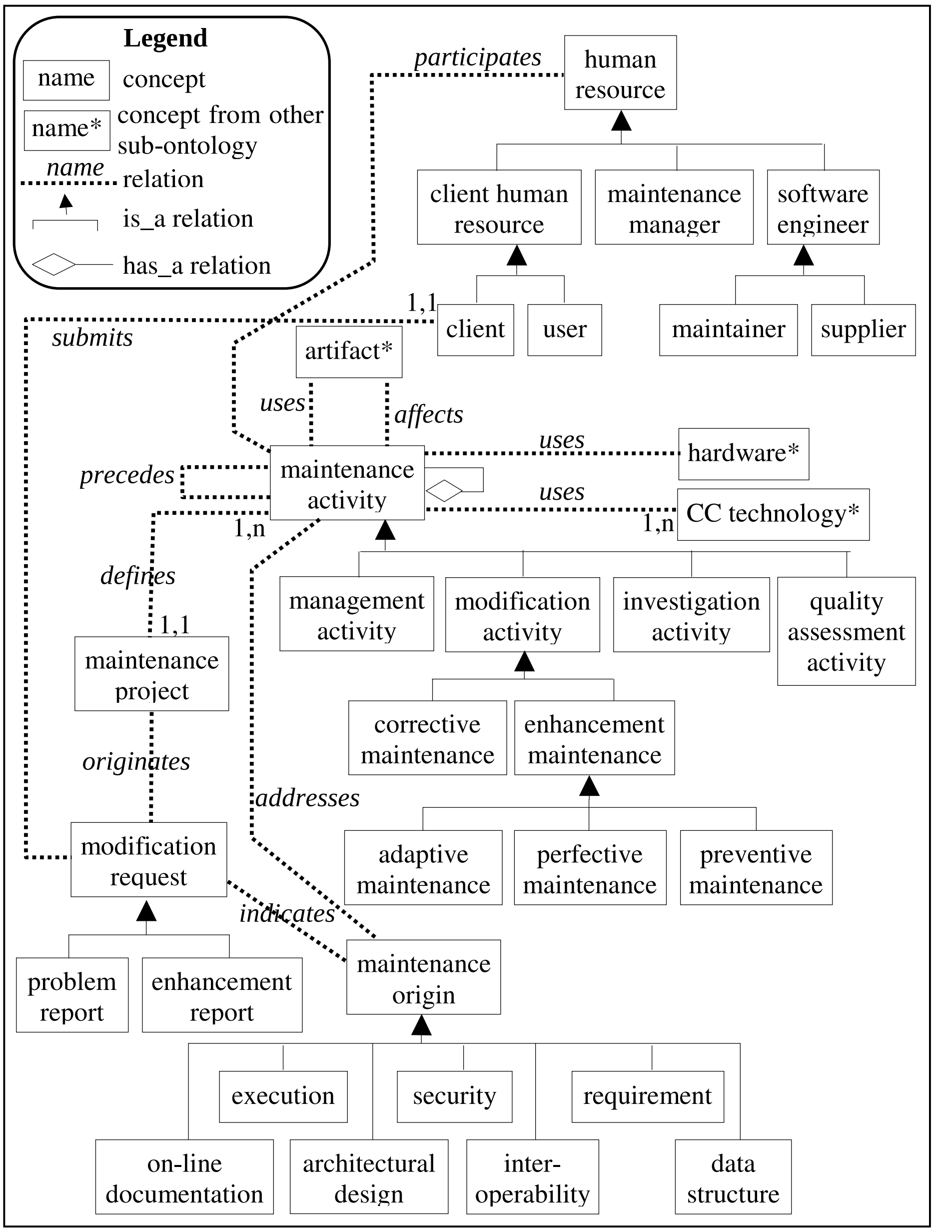

Figure 4: Modification Process sub-ontology

Based on [Briand et al. 1994], [Kitchenham et al. 1999], and [Pigoski 1996] we classified the maintenance activities in the following types: (a) investigation activity, assessing the impact of undertaking a modification; (b) management activity, relating to the management of the maintenance process or to the configuration control of the products; (c) quality assurance activity, aiming at ensuring that the modification does not damage the integrity of the product; and (d) modification activity, which may be a corrective maintenance or enhancement maintenance (adaptative maintenance, preventive or perfective maintenance). A maintenance activity *uses* one or more input artifacts and *affects* one or more output artifacts, it is inserted in a sequence of activities (and therefore has *preceeding* activities), it *addresses* some maintenance origin (already detailed), *uses* hardware resources, and uses some computer science technologies. Axioms are used for example to specify that the maintenance activities or ordered: $(\forall a_1, a_2)\ (preactivity\ (a_1, a_2) \rightarrow \neg\ preactivity\ (a_2, a_1))$ and $(\forall a_1, a_2, a_3)\ (preactivity\ (a_1, a_2) \wedge preactivity\ (a_2, a_3) \rightarrow preactivity\ (a_1, a_3))$ (expressing the anti-symetry and transitivity on the ordering of activities).

Finally, different people (human resource) can participate in these activities (from [Briand et al. 1994], [Kitchenham et al. 1999], [Pigoski 1996] and [IEEE-12119 1998]): software engineers (supplier or maintainer, respectively, who developed and maintain the system), maintenance manager, and client's human resources (client, who pays for the modification, and user, who uses the system).

4.4 The Organizational Strucuture Sub-ontology

The fourth sub-ontology, on the organizational structure, is pictured in Figure 5. We considered a traditional definition of an organization (see for example [Fox et al. 1996]) composed of units where different functions are performed by human resources. We also included the fact that an organization defines directives to be followed 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.

To define the scope of this sub-ontology we set the following competency questions: What organizational units compose the organization? What positions exist and who occupies each position? What directives does the organization adopt? How do the organizations relate one to the other?

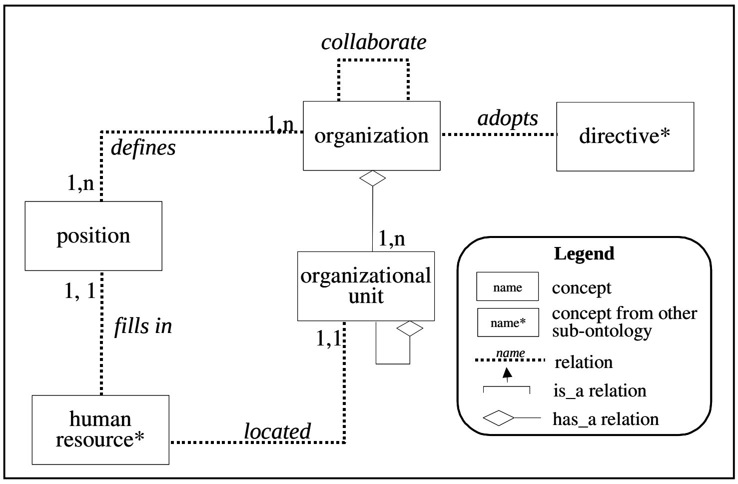

Figure 5: Organizational Structure sub-ontology

Based on [Fox et al. 1996] and [Uschold et al. 1995] we defined that any organization adopts its own directives and defines the positions to be occupied by a humam resource. Also, organizations can collaborate with each other and each one is composed of organizational units. Those organizational units are organized in a hierarchical structure where one is composed of other ones.

4.5 The Application Domain Sub-ontology

Finally the fifth sub-ontology [Figure 6] organizes the concepts on the application domain. The competency questions are: What concepts and tasks compose an application domain? What are the properties of each concept? How do the concepts relate one to the other? What concepts are used in each task? What restrictions apply to the application domain?

We choose 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 by relations 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 [Oliveira et al. 1999]. 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.

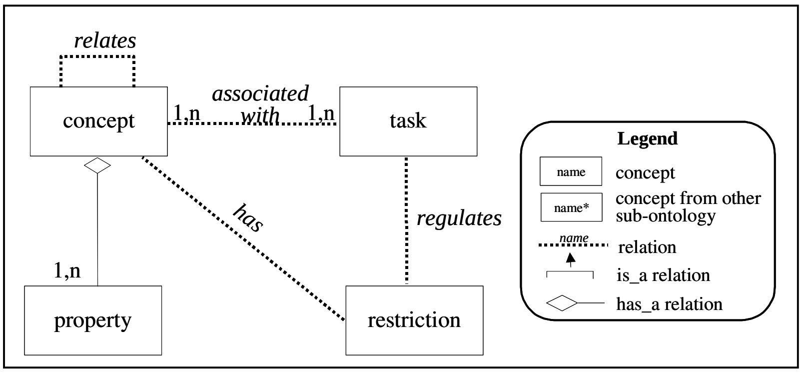

Figure 6: Application Domain sub-ontology

5 Ontology Validation

With the ontology defined, we started its validation in two ways: validation of the quality of the ontology itself (how clear it is, how complete, concise, etc.), and validation of the usefulness of the concepts for maintenance (which was the ontology's purpose as specified in Section 4).

In this section we present how we validated the ontology in these different ways.

5.1 Quality Assessment

To validate the quality of the ontology we considered the six following desirable criteria (see for example [Gruber 1995] and [Gómez-Pérez 1995]): (a) consistency, refering to the absence (or not) of contraditory information in the ontology; (b) completeness, refering to how well the ontology covers the real world (software maintenance for us); (c) conciseness, refering to the absence (or not) of needless information or details; (d) clarity, referring to how effectively the intended meaning is communicated; (e) generality, refering to the possibility of using the ontology for various purposes inside the domain fixed (software maintenance); and (f) robustness, referring to the ability of the ontology to support changes.

To evaluate these criteria, we asked four experts to study the ontology and fill a quality assessment report composed of several questions for each criterion. 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 7, on a scale of 0 to 4 no criterion has an average below 3.

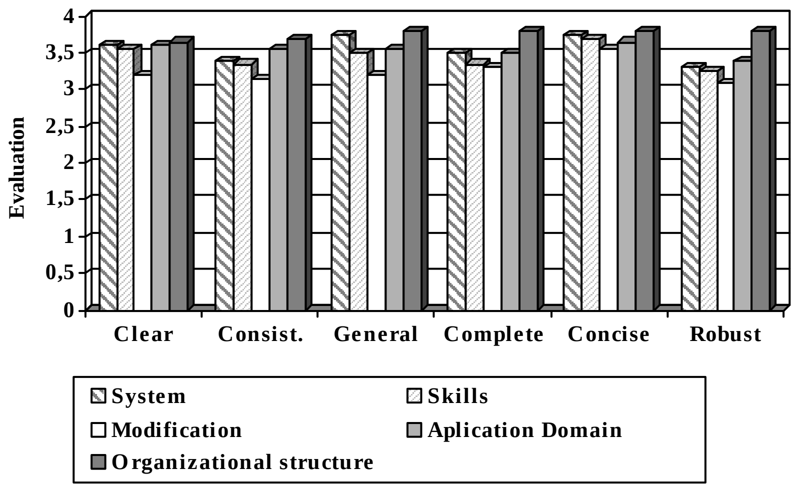

Figure 7: Result of the ontology's quality assessment

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 contemplate 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 assessments experiment, we also checked the completeness and conciseness of the ontology by instantiating the concepts from the documentation of a real software system to. 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.

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

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%**

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

- Two concepts from the Application Domain sub-ontology were not instantiated because they were not identified when we did the experiment. They were added after the first 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 (modelling language) was not instantiated because the maintenance activities were specified in (structured) natural language and, although this is a debatable question, we decided at the moment, not to consider this as a modelling 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 modelling, 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 their 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 operation 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 plan, quality plans, maintenance manual) were not instantiated after the organization ceased to use any maintenance 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.

5.2 Usefulness Assessment

One of the main 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: observing maintainers while they were maintaining a system, and presenting the instantiated knowledge to the software engineers and asking them what concepts they used.

For the first experiment we used a protocol called think-aloud [Lethbridge et al. 1996] 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. During the analysis, we tried to identify the kind of knowledge that the software engineers were using at each moment based on the defined 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, 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 5.1] 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.

All uses of concepts 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 (5.1)

	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%

Table 2: Number of concepts used in two experiments

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 modifications 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 sub-ontology) falls under the responsibility of another organizational unit, therefore the software engineers we studied need not know about them or use them.

6 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 propositions of mental models to describe how software engineers go about doing maintenance [Rugaber and Tisdale 1992], [von Mayrhauser and Vans 1994]. They offer little interest since they concentrate on the process of doing maintenance rather than on the knowledge used.

In [Ramal et al. 2002], one of us started to study the knowledge used during software maintenance. This earlier work contained a very crude identification of various knowledge domains connected with this activity. The domains identified were: Computer Science Domain, Application Domain and General Domain (common sense knowledge). The current research is a follow-up on the preceding paper and describes the result of our efforts to formally and completely identify the knowledge useful during software maintenance.

In [Clayton et al. 1998], the authors studied "the knowledge required to understand a program". Knowledge is classified in 3 domains: Domain knowledge (numerical analysis in this case), Fortran knowledge and programming knowledge. The first one corresponds 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 his coleagues [Briand et al. 1994] 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, [Deridder 2002], 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 [Kitchenham et al. 1999] 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.

7 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 defined an ontology of the knowledge used in software maintenance.

This ontology would be useful as a framework to guide future research trying to improve software maintenance using knowledge engineering techniques. It could be the base of studies to answer questions as: What knowledge should be taken into account when considering software maintenance? What kind of knowledge is most important? etc. Our ontology was based both on expert experience and a study of the relevant literature.

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 (we are conducting futher validation experiment).
- 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.

Acknowledgments

We acknowledge CNPq – Brazilian Research Council for the financial support to this project.

References

[Booch et al. 1997] Booch, G., Rumbaugh, J., Jacobson, I.: "The Unified Modeling Language – User Guide"; Addison-Wesley, 1997.

[Briand et al. 1994] Briand, L. C., Basili, V., Kim, Y., Squier, D. R.: "A Change Analysis Process to Characterize Software Maintenance Projects"; Proc. The International Conference on Software Maintenance, 1994.

[Chandra and Ramamoorthy 1996] Chandra, C., Ramamoorthy, C. V.: "An Evaluation of Knowledge Engineering Approaches to the Maintenance of Evolutionary Software"; Proc. 8th Software Engineering An Knowledge Engineering Conference, Nevada, USA, p.181-188, Jun . 1996.

[Clayton et al. 1998] Clayton R., Rugaber S., Wills L: "On the Knowledge Required to Understand a Program"; Proc. Working Conference on Reverse Engineering, p. 69–78, Oct. 1998.

[Deridder 2002] Deridder, D.: "Facilitating Software Maintenance and Reuse Activities with a Concept-oriented Approach"; Programming Technology Lab, Vrije Universiteit Brussel, Brussels, Belgium, 2002.

[Domingue 1998] Domingue, J.: "Tadzebao and WebOnto: Discussing, Browsing, and Editing Ontologies on the Web"; 11th Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Apr. 1998.

[Fox et al. 1996] Fox, M. S., Barbuceanu, M., Gruninger, M.: "An Organization Ontology for Enterprise Modeling: Preliminary Concepts for Linking Structure and Behaviour"; Computers in Industry, v. 29, p.123-134, 1996.

[Gómez-Pérez 1995] Gómez-Pérez, A: "Some Ideas and Examples to Evaluate Ontologies". 11th Conference on Artificial Intelligence for Applications, p.299-305, 1995

[Grosso et al. 1999] Grosso, W.E., Eriksson, H., Fergerson, R.W., Gennari, J.H., Tu, S.W., Musen, M.A.: "Knowledge modeling at the millennium (the design and evolution of Protégé-2000)"; 12th Banff Workshop on Knowledge Acquisition, Modeling, and Management, Banff, Alberta, 1999.

[Gruber 1995] Gruber, T. R.: "Toward Principles for the Design of Ontologies Used for Knowledge Sharing". Int. J. Hum. Comput. Stud., 43(5/6): 907-928, 1995.

[Grüninger and Fox 1995] Grüninger, M., Fox, M. S.: "Methodology for the Design and Evaluation of Ontologies"; Technical Report, University of Toronto, Toronto, Canada, 1995.

[IEEE-12119 1998] IEEE-1219: "IEEE Standard for Software Maintenance". Los Alamitos, CA: IEEE Computer Society Press, 1998.

[Kitchenham et al. 1999] Kitchenham, B. A., Travassos, G. H., Mayrhauser, A. et al.: "Toward an Ontology of Software Maintenance"; Journal of Software Maintenance: Research and Practice 11(6):365-389, May 1999.

[Leffingwell and Widrig 2000] Leffingwell, D., Widrig, D.: "Managing Software Requirements: A Unified Approach", Addison-Wesley, 2000.

[Lehman 1980] Lehman, M.: "On understanding Laws, evolution and conversation in the large program lifecycle"; Journal of Software & Systems, vol. 1, p.213 – 221, 1980.

[Lethbridge et al. 1996] Lethbridge, T. C., Sim, S. E., Singer, J.: "Software Anthropology: Performing Field Studies in Software Companies"; Consortium for Software Engineering Research (CSER), 1996.

[Oliveira et al. 1999] Oliveira, K., Rocha, A., Travassos, G. H., Menezes, C: "Using Domain-Knowledge in Software Development Environments"; Proc. Software Engineering and Knowledge Engineering, p. 180-187, Kaiserlautern, Germany, Jun. 1999.

[Pfleeger 2001] Pfleeger, S. L.: "Software Engineering: Theory and Practice"; 2nd Edition. New- Jersey, Prentice Hall, 2001.

[Pigoski 1996] Pigoski, T. M.: "Practical software maintenance: best practices for managing your software investment"; John Wiley & Sons. P.87-102, Dec. 1996.

[Pigoski 2001] Pigoski, T. M.: "Software maintenance"; In: Guide To The Software Engineering Body Of Knowledge; Los Alamitos, CA: IEEE Computer Society Press. Trial Version 1.00, May, 2001.

[Pressman 2001] Pressman, R. S.: "Software Engenieering"; 5th Edition, p.225-241, McGraw Hill, 2001.

[Ramal et al. 2002] Ramal, M. F., Meneses, R., Anquetil, N.: "A Disturbing Result on the Knowledge Used During Software Maintenance"; Proc. Working Conference on Reverse Engineering, Richmond, p. 277-287, Oct-Nov. 2002.

[Rugaber and Tisdale 1992] Rugaber, S., Tisdale, V. G.: "Software Psychology Requirements for Software Maintenance Activities"; Software Engineering Research Center, Georgia Institute of Technology, 1992.

[Rus and Lindvall, 2002] Rus, I., Lindvall, M.: "Knowledge Management in Software Engineering"; IEEE Software, v. 19, n. 3, p.26-38, May/Jun 2002.

[Staab et al. 2000] Staab, S., Erdmann, M., Maedche, A., Decker, S.: "An Extensible Approach for Modeling Ontologies in RDF(S)"; ECDL 2000 Workshop on the Semantic Web, 2000.

[Uschold et al. 1995] Uschold, M., King, M., Moralee, S. et al.: "The Enterprise Ontology"; The Knowledge Engineering Review, v.13, 1995.

[von Mayrhauser and Vans 1994] Von Mayrhauser A., Vans A.: "Dynamic Code Cognition Behaviors For Large Scale Code", Proc. Workshop on Program Comprehension, p. 74–81, IEEE Comp. Soc. Press, Nov. 1994.