Organizing the Knowledge Used in Software Maintenance

Márcio Greyck Batista Dias, Nicolas Anquetil, Káthia Marçal de Oliveira
UCB - Universidade Catlica de Braslia
SGAN 916 Mdulo B - Av. W5 Norte
Braslia - DF - 70.790-160, Brazil
myck@brturbo.com, {kathia,anquetil}@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.

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 [Pfl01, p.475] [Pig96, p.35].

To help maintainers face these difficulties, one could envision specialized tools providing easy access to the various domains of knowledge required. However, gathering all these informations would be a tremendous work with probably mixed results. There are few studies to indicate what aspects to prioritize. For example, although it is generally assumed that application domain knowledge is a fundamental asset for software maintenance, an initial study conducted by one of the authors [MFR02], hinted that it was quantitatively

much less important than computer science knowledge.

In this article, we propose an ontology of the knowledge relevant to software maintenance. In the following sections we briefly define what an ontology is and how it is being developed (§2), we present our ontology on the knowledge used during software maintenance (§3), we discuss some initial results on validation (§4) and related work (§5). The article ends with a conclusion and proposition of future work.

2 Ontology Definition and Methodology

An ontology is a description of entities and their properties, relationships, and constraints [GF95]. Ontologies can promote organization and sharing of knowledge, as well as interoperability among systems. There exist various methodologies to design an ontology (e.g., [GF95]), 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 these steps The *purpose* is to define an ontology describing the knowledge relevant to software maintenance. The *conceptualization* step was based on 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 [GF95]). It resulted in a set of all the concepts that will be presented in the next section. The *validation* will be discussed in section 4. The validation can lead to review the conceptualization phase. The *formalization* will consist in the implementation of a tool to gather and make the knowledge available (see §6)..

We spent three months to define this ontology, the main investigator working part-time, and the two others participating in weekly validation meetings. Our first difficulty was to define clearly which was to be the focus of the ontology. This was solved defining scenarios (see down) for the use of the knowledge. A second difficulty was to review the pertinent literature in search of definitions and validation of the concepts. As we were both the domain experts and knowledge engineers, we deemed important to base the concepts on independent sources.

3 An Ontology for Software Maintenance

We started the ontology definition 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; or what knowledge a maintainer should have on a system s-he will modify. Those and other situations showed us that we need to orga-

nize the knowledge around different aspects: knowledge about the *software system* itself; knowledge about the needed *skills in computer science*; knowledge about the *maintenance activity*; knowledge about the *organization* structure; and knowledge about the *application domain.* Consequently, we divided our ontology into five sub-ontologies to cover each of these aspects. 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 (the three authors). In the following, we present each sub-ontology, their concepts and relations. For lack of space, some concepts will not be discussed, or only alluded to.

Figure 1 shows the first sub-ontology on the *system*. The competency questions for this ontology are: What are the software artifacts of a system? How do they relate to each other? Which software and hardware resources are used by a system? What type of human resources interact with the system? Answering these questions leads to a taxonomy of software artifacts that compose a software system and a taxonomy of resources (human, hardware, and software) the system uses.

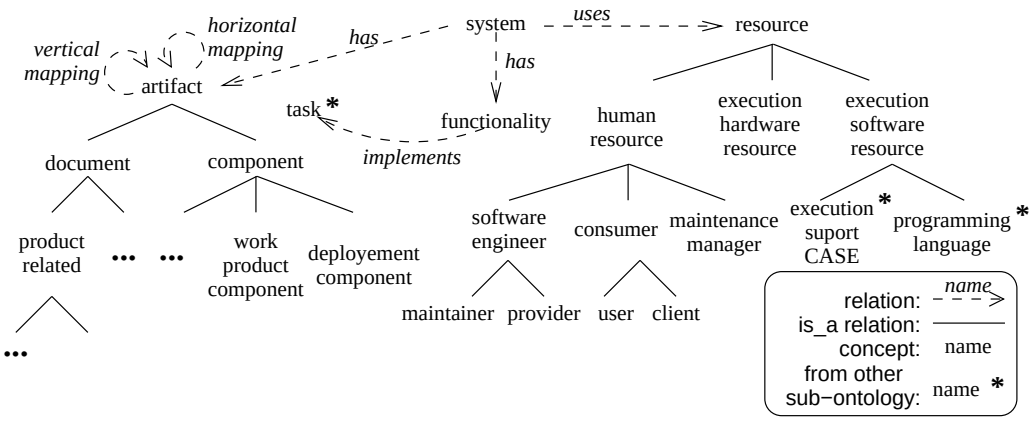

Figure 1: System sub-ontology (for lack of space, some concepts were omitted)

The artifacts of a system can generally be decomposed in documentation and software components. Briand [BBKS94] 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).

Software components represent all the coded artifacts that compose the software program itself. Booch [BRJ97] 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 vertical mapping, relating two artifacts of different abstraction levels. We call the second kind of relation an horizontal mapping, relating two artifacts of the same level of abstraction.

The identification of the resources used by a software system are based on [BBKS94, Pig96, KTvM$^+$99]. At the upper level, we considered the human, software, and hardware resources. The human resources are all the people related to the system (software engineers, clients, users, and managers), the software resources are all the CASE tools used in a software execution (e.g., DBMS, network, and operational system).

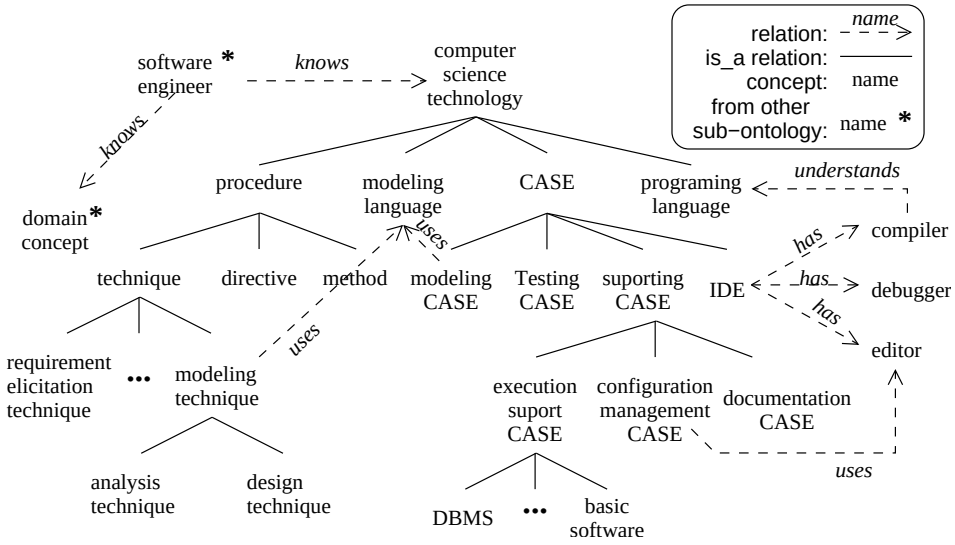

Figure 2: Computer Science skills sub-ontology

Figure 2 shows the second sub-ontology on the *skills in computer science* a software maintainer needs. A scenario of use would be to be able to select the best participants in maintenance of a given type. Some competency questions we identified are: What kind of CASE does the software maintainer have experience with? What kind of procedures (methods, techniques, and norms) does s-he know? What programing and modeling languages does s-he know?

Pressman [Pre01] gives a very complete list of CASE tools, with tools for designing, developing, testing, and supporting. The support tools may support the execution, documentation, or configuration management.

According to [KTvM$^+$99] procedures are all structured guidelines used in a software development activity like methods, techniques, and directives. Based on [CR96, LW00, Pre01], we classified the techniques in: reverse engineering (e.g., slicing), requirement elicitation (e.g., interviews, brainstorming, ...), programming (e.g., structured or object oriented), testing (e.g., white/black box), and modeling (e.g., analysis or design).

Finally the modeling language represents the graphical, and semantic rules used in a design methods, and the programming language, the syntactical and semantic rules defined to code a program.

Figure 3 shows the main concepts of the third sub-ontology on the *maintenance process*. Here, we were 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?

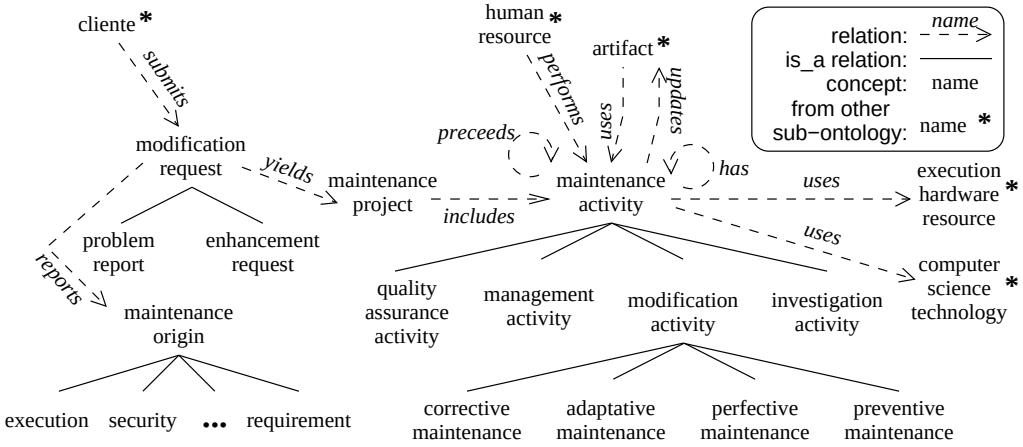

Figure 3: Maintenance process sub-ontology

According to [Pig96] a problem resolution process is initiated whenever a modification request is generated. This request is classified either as a problem report (corrective maintenance) or enhancement request (adaptative or perfective maintenance). A modification request can originate in problems with the on-line documentation, interoperability with other systems, data structure, security, new requirements, etc. All these reasons motivate the client to request some modification. The software maintenance team responds to a request initiating a modification project which will include different kinds of activity. This part of the ontology is based on [KTvM$^+$99]. A maintenance activity uses and updates some artifacts, it is inserted in a sequence of activities (the maintenance process), it uses various resources, and may be decomposed in: (i) investigation activity, assessing the impact of undertaking a modification; (ii) management activity, relating to the management of the maintenance process or to the configuration control of the products; (iii) quality assurance activity, aiming at ensuring that the modification does not damage the integrity of the product; and (iv) modification activity, taking one or more input artifacts and producing one or more output artifacts. Following [Pfl01, Pig96], we classified the modification activity as adaptative, perfective, corrective, or preventive. Other authors (e.g., [KTvM $^+$99]) only consider enhancement and correction. All activities use some computer science technology. Different people (human resource) can participate in these activities.

The fourth sub-ontology, on the *organizational structure*, is not pictured here for lack of space. We considered a traditional definition of an organization (see for example [FBG96]) which is composed of units where different functions are performed by human resources. We also included the fact that an organization defines norms or rules to be followed in the execution of its functions.

Finally the fifth sub-ontology organizes the concepts on 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 some relations and having properties which can be assigned values. This meta-ontology would best be instantiated for each application domain with a small domain ontology as exemplified for example in [OTMR99].

4 Ontology Validation

The validation of the ontology is still underway, we plan to use three different techniques and compare their results: (i) study of software engineers during their work to record what they do and identify the concepts they are using; (ii) study of the documentation and source code of a system and identification of the concepts; (iii) asking software engineers to identify what instances (identified in (ii)) of each concepts they used after a day's work.

Each method should have its own strengths and weaknesses: (i) is not biased by what the software engineers *think* they use, but imposes we guess what they are thinking which introduces another bias, it is the most difficult approach; (ii) is easier, but we have no guarantee that the concepts referred in the documentation are actually useful; (iii) does not need our interpretation of what the software engineers think, but they could use unconsciously concepts that they would not indentify.

Table 1: Some result of validation experiments (i) and (ii), see accompanying text for explanation

sub-ontologies	# of concepts			
	total	(i)	(ii)	(i & ii)
System	23	8	14	15
Skills	24	9	18	19
Maint. Activity	17	4	11	12
Organization	4	2	4	4
Domain	4	1	3	3

We already did five sessions observing two software engineers working on two different systems (method (i)), and we completed the study of the documentation of one other system (method (ii)). Some results of these two instantiation approaches are outlined in Table 1. One can observe that only about 70% of the concepts have been actually implemented. We attribute this to the small size of the experiment, for example in the Maintenance activity, there are 7 possible subconcepts of "Maintenance Origin" (e.g. documentation, execution, requirements, security, ...), unless we have a fairly large set of modification

requests (which is not yet the case), we are unlikely to instantiate all these concepts. Still, it may be the case that in the future we decide that some concepts are too rare and should be eliminated.

We also considered the number of instances of each concepts, it varies from 1 to 300 (Components of a System), being normally less than 10. The frequency of apparition of an instance vary from 1 to 6.

5 Related Work

This work is part of a broader project which aims at developing methods and tools to study and support knowledge management in software maintenance.

In [MFR02], 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 [BBKS94], Briand *et al.* characterized and described various concepts related to software maintenance processes. Although they did not actually describe an ontology, their article list and discuss various concepts in such general topics as: the organization, the documentation, the tools, the process, etc.

The work most related to what we did is that of Kitchenham *et al.* [KTvM+99] defining an "ontology of software maintenance". This other ontology has for goal to "provide a framework for categorizing" studies on software maintenance and "allow to provide a context" for them. We, on the other hand, were considering the knowledge that is useful when doing maintenance. This difference of focus resulted in a different organization of the concepts, and differences in the level of detail we considered. This appears clearly in various occasions as when *domain* is an attribute of the system in [KTvM+99] and a sub-ontology in our work. Nevertheless both ontologies do have many concepts in common.

6 Conclusion and Ongoing Work

In this article, we gave a rapid overview of 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. We are currently validating this ontology studying software maintainers in their daily activities.

Ongoing and future work include developing a small tool to browse the ontology and instantiate its concepts. This tool could be used in a first step to help us do more validation experiments on the ontology, but we also plan to extend it in a second phase to help manage the individual competencies of a software team. We are also planning to start a small experience factory for maintenance which would used the ontology as a knowledge framework.

References

[BBKS94] Lionel C. Briand, Victor R. Basili, Yong-Mi Kim, and Donald R. Squier. A Change Analysis Process to Characterize Software Maintenance Projects. In *International Conference on Software Maintenance/ICSM'94*, pages 1–12, 1994.

[BRJ97] G. Booch, J. Rumbaugh, and I Jacobson. *The Unified Modeling Language - User Guide*. Addison-Wesley, 1997.

[CR96] C. Chandra and C. V. Ramamoorthy. An Evaluation of Knowledge Engineering Approaches to the Maintenance of Evolutionary Software. In *Proceedings of the 8th Software Engineering an Knowledge Engineering Conference*, pages 181–188, jun. 1996.

[FBG96] M. S. Fox, M. Barbuceanu, and M. Gruninger. An Organization Ontology for Enterprise Modeling: Preliminary Concepts for Linking Structure and Behaviour. *Computers in Industry*, 29:123–134, 1996.

[GF95] M. Gruninger and M.S.N. Fox. Methodology for the Design and Evaluation of Ontologies. In *Workshop on Basic Ontological Issues in Knowledge Sharing / IJCAI'95*, Aug. 1995.

[KTvM+99] Barbara A. Kitchenham, Guilherme H. Travassos, Anneliese von Mayrhauser, Frank Niessink, Norman F. Schneidewind, Janice Singer, Shingo Takada, Risto Vehvilainen, and Hongji Yang. Towards an Ontology of Software Maintenance. *Journal of Software Maintenance: Research and Practice*, 11:365–389, 1999.

[LW00] D. Leffingwell and D. Widrig. *Managing Software Requirements: A Unified Approach*. Addison-Wesley, 2000.

[MFR02] Nicolas Anquetil Marcelo Fenoll Ramal, Ricardo de Moura Meneses. A Disturbing Result on the Knowledge Used During Software Maintenance. In *Working Conference on Reverse Engineering, WCRE'2002*, pages 277–287. IEEE, IEEE Comp. Soc. Press, 29 Oct.-1 Nov. 2002.

[OTMR99] K.M Oliveira, G. Travassos, C. Menezes, and A.R Rocha. Using Domain-Knowledge in Software Development Environments. In *Software Engineering and Knowledge Engineering*, pages 180–7, jun. 1999.

[Pfl01] Shari Lawrence Pfleeger. *Software Engineering: Theory and Practice*. Prentice Hall, 2nd edition, 2001.

[Pig96] Thomas M. Pigoski. *Practical Software Maintenance*. John Wiley & Sons, Inc., 1996.

[Pre01] Roger S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 5th edition, 2001.