SUPPORTING KNOWLEDGE REUSE DURING THE SOFTWARE MAINTENANCE PROCESS THROUGH AGENTS

Aurora Vizcaino, Juan Pablo Soto, Mario Piattini
Alarcos Research Group,Computer Science Department, Universidad de Castilla-La Mancha, Ciudad Real, Spain

Keywords: Knowledge management, agents, software maintenance, software engineering

Abstract: Knowledge management has become an important topic as organisations wish to take advantage of the information that they produce and that can be brought to bear on important decisions. This work describes a system to manage and reuse the information (and knowledge) generated during the software maintenance process, which consumes a large part of the software lifecycle costs. The architecture of the system is formed of a set of agent communities. Each community manages different types of knowledge. The communities' agents have the goal of encouraging the reuse of good solutions and taking advantage of information obtained from previous experience. In consequence, the software maintenance is made easier and there are less costs and effort. To achieve this goal, agents use several reasoning techniques such as case based reasoning or decision tree based algorithms which allow them to generate new knowledge from the information that they manage.

1 INTRODUCTION

Software engineering in general, and software maintenance in particular, are activities that generate important amounts of knowledge. This knowledge comes not only from the expertise of the professionals involved in the processes, but is also intrinsic to the product being maintained and in the case of software maintenance, to the reasons that motivate maintenance (new requirements, user complaints, etc.), processes, methodologies and tools used in the organization. Moreover, software maintenance is a constantly changing process since maintenance results from the necessity of adapting software systems to an ever changing environment (Oliveira et al, 2003).

On the other hand, many people are involved in software development and maintenance. These people often work in different phases, activities and projects and nowadays it is common that they act from geographically distributed sub-units. For all these reasons, maintenance organizations frequently have problems identifying their resources, localizations and use of knowledge. As a consequence, companies recurrently spend time, effort and money searching for solutions to problems that have already been solved in their own organization.

On many occasions, organizations had documents or people with the information or knowledge necessary to support or help other colleagues in their activities but either the former did not know what the latter was working on or the latter did not know that other documents or people could have helped them.

Techniques and tools are needed to help software practitioners apply past knowledge to current projects (Henninger, 2003).

A plausible technique is to store good solutions to problems or lessons learned thus avoiding repeating mistakes and increasing productivity and the likelihood of further success (Rus and Lindvall, 2002).

To implement this technique, we used the experience-based approach, which proposes that software maintenance draws on past experiences as a resource for planning and executing software development and maintenance efforts (Basili and Rombach, 1988; Henninger et al. 1995).

An experience-based approach to software maintenance involves using an organization's accumulated knowledge of the development and maintenance processes and application domains as the basis for planning and performing the different types of maintenance. Analysing and structuring the necessary knowledge to achieve this goal is a difficult process that can only be accomplished in well-understood domains (Fischer and Lemke, 1988).

First of all, the experience-based knowledge lifecycle should be considered: Knowledge creation is a spiral where new ideas are built on existing knowledge, made explicit so it can be communicated to others (these are the lessons learned or experience packages), then routinized to become part of everyday practices that serve as the basis for future knowledge creation (Nonaka and Takeychi, 1995). According to this definition, a new problem arises: how to communicate or share knowledge and to foster its reuse and decide who should be in charge of it.

Basili et al. (1994) propose a solution to this problem «the Experience Factory approach». This approach separates the responsibilities of developing projects or maintaining them, (in the case of software maintenance), from capturing experience. The Experience Factory unit is in charge of developing, updating and providing reusable experience that can be utilized by product development teams (Henninger, 2003).

This approach is currently starting to be used in experience management tools (see Seaman et al., 2003) and it is that which is used in our work since it makes knowledge management possible without overloading maintainers with new tasks or responsibilities. Therefore, it is supposed that organizations have an experience factory to carry out all activities necessary to store, manage and share previous experience.

Besides using a software experience based tool we consider how to deal with the different types of knowledge generated during the software maintenance process. Agents were the solution chosen because they can specialize in monitoring specific knowledge. Another advantage of using agents is that they can take advantage of other agents' knowledge by consulting or asking for help from others. Therefore, they reuse and share their own knowledge.

The rest of the paper is structured as follows: Section two describes what type of knowledge should be taken into account to develop an experience-based knowledge management tool for the software maintenance process. Section three is focused on describing the multi-agent architecture, designed to foster knowledge reuse in software maintenance companies. Finally, conclusions are outlined.

2 SOFTWARE MAINTENANCE KNOWLEDGE

There are several proposals of mental models to describe how software engineers go about carrying out maintenance (Rugaber and Tisdale, 2000; Briand et al, 1994). However, these works focus on the process of doing maintenance rather than on the knowledge generated or used during this process.

Kitchenham et al (1999) designed an ontology of software maintenance. In this ontology all the concepts relevant to the classification of empirical studies in software maintenance were identified.

Kitchenham's ontology is structured in several partial subontonlogies:

Products ontology: This represents how the software product is maintained and how it evolves over time.

Activities ontology: This describes how to organise activities for maintaining software and what kinds of activities they may be.

Processes ontology: This is divided into two different focuses, defining a sub-ontology for each one:

- Procedures sub-ontology: This defines how the methods, techniques and tools can be applied to the activities and how the resources are used in order to carry out these activities.
- Process Organization sub-ontology: This focuses on how the support and organizational processes are related to the software maintenance activities, how the maintainer is organized, and what his/her contractual obligations are.

Peopleware ontology: This describes what skills and roles are necessary in order to carry out the activities, what the responsibilities of each person are, and how the organizations that intervene in the process (maintainer, customer and user) relate to each other.

Oliveira et al. (2003) present an ontology where the knowledge useful to software maintenance is determined. Their ontology is inspired by Kitchenham's ontology. They propose five aspects instead of the four described above, although four of them are similar to the sub-ontologies previously cited. The aspects considered by Oliveira et al (2003) are: Knowledge about the software system itsef which corresponds to the product ontology. Knowledge about the maintainer's skills, which corresponds to the peopleware ontology. Knowledge about the maintenance activity which corresponds to the activities ontology. Knowledge about the organization structure which corresponds to the Process Organization sub-ontology. And knowledge about the application domain which is not considered in Kitchenham's ontology.

Ruíz et al. (2003) propose a semi-formalised ontology where Kitchenham's ontology has been extended and focused on the management of Software Maintenance Projects. This is the ontology that has been used in the design of our experience based tool since it is that which is best adapted to our necessities in exactly defining the software maintainance projects domain.

Our experience-based tool is currently focused on knowledge directly related to the maintenance problems. Therefore, storing the application domain was not considered convenient, at least in the first step. Moreover, it is more likely that organizations obtain experience about what activities are most performed, or about the processes that should be followed to carry them out than about the application domain itself.

Therefore, the tool manages information related to the products to be maintained, the activities that are performed during the software maintenance process, and of course, the methods, techniques and tools used to carry out the different activities. Moreover, the tool monitors the work that each person has done and in which project and activity s/he is working at this moment. Thus, different kinds of knowledge should be taken into account and it is for this reason that we have designed a multiagent architecture.

3 THE MULTI-AGENT ARCHITECTURE

There are several reasons why agents are recommendable for managing knowledge (see Tacla and Barthès, 2002). First of all, agents are proactive. This means they act automatically when it is necessary. Moreover, agents can manage both distribute and local information. This is an important feature since the software maintenance information is generated by different sources and often from different places.

Another important issue is that agents can learn from their own experience. Consequently, the system is expected to become more efficient with time since the agents have learnt from their previous mistakes and successes.

On the other hand, each agent may utilize different reasoning techniques depending on the situation. For instance, they can use ID3 algorithms to learn from previous experiences and use case-based reasoning to advise a client how to solve a problem.

Having explained the convenience of using agents let us describe the multi-agent architecture. We followed MESSAGE, a Methodology for Engineering Systems of Software Agents (Evan et al., 2001) to design the architecture.

MESSAGE proposes different levels of analysis. At level 0 the system to be developed is considered as a black box focusing on its relationships with the entities in its environment (eg. users, stakeholders, and resources).

Figure 1 shows the level 0 of our design, where there are three entities: the experience based tool, the maintainers' team and the experience factory organization. As Figure 1 indicates the experience based tool is updated by the members of the experience factory. However, they need to obtain experience from the maintainers' teams. This means that the activities of the experience factory and those of the maintainers' team should be perfectly integrated (Rus and Lindvall, 2002). A feedback between them is indispensable to assure the success of the experience factory approach.

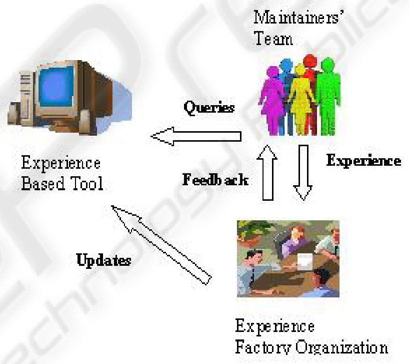

Figure 1: Organisation Diagram

Moving from level 0 to level 1, analysis focuses on the system itself, identifying the types of agents and roles. The tool has different types of agents. They are groups in three communities. The *product community* is formed of product agents in charge of monitoring information related to the products to be maintained. The *activity community* has activity agents, which control all information, related to a specific activity such as the process where it is used, the tool/s necessary to perform the activity and the best methods to carry it out.

The third community is called the *peopleware community*. This community has three different agents, one per type of profile involved in the software maintenance process (Polo et al., 1999). One is the "staff agent" in charge of managing information related to the members of the staff. Another type of agent, called client agent, is in charge of the information received from the clients and of their data. And the last agent, the user agent, manages users' information. Following the MESSAGE methodology we have developed an Agent/Role schema for each agent, which as its name suggests, describes the features and roles of each agent. Following the product agent/role schema is presented:

Table 1: Product agent/role schema

Role Schema	Product Agent
Goals	Predicting future changes. Looking for similar features in other products.
Capability	Case-Based reasoning
Knowledge	Initial requirements of the product. Product's features. Changes performed in the product.
Agent requirements	This role is played by the agent that each product has.

Table 1 summarizes the fact that by using case-based reasoning the product agents try to predict future changes, since similar software projects often require similar maintenance demands. This role is very relevant as studies show that the incorporation of new requirements is the core problem for software evolution and maintenance and supposes, along with the adaptive maintenance, around 75% of the maintenance effort. As Bennet and Rajlich (2000) claim, if changes can be anticipated they can be built in by some form of parameterisation. In this way costs and efforts are decreased.

The activity agent/role schema is now shown and commented on.

Table 2: Activity agent/role schema

Role Schema	Activity Agent
Goals	To advise the best way to perform an activity.
Capability	Induction and decision trees based algorithms to learn from previous experience.
Knowledge	Methods, techniques and resources to use for performing an activity. Lessons learned.
Agent requirements	This role is played by the agent that represents each activity.

In order to carry out suitable maintenance it is advisable to follow a specific methodology. In our case MANTEMA (Polo et al., 1999), a complete methodology designed for software maintenance, is used.

When an organization uses our tool for the first time, the activity agents contain MANTEMA's indications about when to use an activity, or what methods and resources to use. However, while the tool is used agents are learning what techniques are the most appropriate for each activity and also learning which mistakes are often made in each activity. Thus, the more the tool is used the better it will work. With this role the aim is to reuse lessons learned and avoid the repetition of mistakes.

The following paragraphs describe the agent/role schema for the agents belonging to the peopleware community. They are the staff agent, the client agent and the user agent.

The staff agent is in charge of monitoring the employees' work in order to determine in which task each person has more experience or where a person obtained better performance. Therefore, the staff agent has enough information to be able to recommend the most suitable person to perform a task attempting to decrease time, cost or effort or to obtain the best performance.

Table 3: Staff agent/role schema

Role Schema	Staff Agent
Goals	To follow the performance of each employee in order to recommend the most suitable person to carry out a task.
Capability	Statistics techniques that indicate the time that an employee took to perform a task or calculate the performance graph of each member.
Knowledge	Personal data of the employees, in which activities they have worked, and which product they have maintained.
Agent requirements	This role is played by one agent in charge of all the members of the staff.

On the other hand, as was mentioned before, similar products often demand similar changes. The same often happens with the clients. Clients with frequent analogous features have similar needs. Thus, the client agent searches for analogies among clients in order to predict future demands or to use the knowledge obtained from previous experience to help clients to make a decision.

When more knowledge and experience is managed it is easier to identify problems, to develop different alternative solutions and to select the best option (Gnyawali et al., 1997).

For example, let us consider that a client wants to change two modules of the same product. However, he wonders whether it is more suitable to carry out the two changes at the same time or order one change first, and after a time request the second change. The client agent can check whether similar changes were demanded previously and calculate statistics in order to advise him/her which method is most suitable to follow in order to obtain the best performance and price.

Table 4: Client agent/role schema

Role Schema	**Client Agent**
Goals	To help to make decisions.
Capability	Analogy reasoning techniques.
Knowledge	Profile of each client and their requirements (including the initial requirements if they are available).
Agent requirements	This role is played by one agent in charge of monitoring clients' demands.

Finally, the user agent schema shows that the user agent plays a role similar to that of the client agent. The reason for having two different agents is that the clients are frequently not the users of the software to be maintained. For this reason it is necessary to monitor the information of both parts. For a software maintenance organization it is very useful to know the users' opinion of the software and of the changes that have been performed.
Organizations can use this information to evaluate their work and to study how the users' characteristics influence the maintenance of the product. In order to illustrate how the user agent works let us imagine that an accounting department requests a change in its software because a new tax has appeared.

Table 5: User agent/role schema

Role Schema	**User Agent**
Goals	To predict new requirements
Capability	Analogy reasoning techniques
Knowledge	Necessities of the users of each product, their background and also their complaints and comments about the products.
Agent requirements	This role is played by one agent in charge of monitoring users' features.

The user agent sends an email to other users, whose jobs are related to accounting, warning them of the possibility that their software should be modified and updated because of the new tax. A budget of the cost of the change could even be attached to the email. Thus, users can plan the changes in advance.

To conclude this section some considerations related to the implementation of the system are explained. The platform chosen to implement the multiagent system is JADE which is an FIPA compliant agent platform (Bellifemine et al., 2001), implemented in Java and developed as an open source project.

This platform provides a Java API which simplifies the development of agents that run in the environment of the platform. The language used for the agents' communication is ACL.

Moreover, the experience and information managed by the system are represented in the experience repository as XML documents which are managed by TAMINO, a database created especially for XML documents.

4 CONCLUSIONS

Software maintenance is one of the most important stages of the software life cycle. This process involves a lot of time, effort, and costs. It also generates a huge amount of different kinds of knowledge that must be suitably managed. This fact is more visible in big companies since the larger the product the more likely it is that product knowledge will be spread among the maintenance staff, making it more difficult to find the cause of problems. Furthermore, the more people working together the more opportunities there are for misunderstandings that may lead to quality problems.

This paper presents a multiagent system, based on the experience-based approach, which stores information and generates knowledge with the finality of encouraging the reuse of previous information and knowledge in software maintenance organizations. Thus costs and effort are decreased.

ACKNOWLEDGEMENTS

This work is partially supported by the TAMANSI project (grant number PBC-02-001) financed by the Consejería de Ciencia y Tecnología of the Junta de Comunidades de Castilla-La Mancha and the MAS project (grant number TIC2003-02737-C02-02, Ministerio de Ciencia y Tecnología, SPAIN).

REFERENCES

Basili, V. R. , Caldiera, G., and Rombach, H. D. (1994). The Experience Factory. Encyclopedia of Software Engineering (pp. 469-476). Marciniak, J.J.; Wiley, J. (Eds.).

Basili, V. R., and Rombach, H.D. (1988) The TAME Project: Towards Improvement-Oriented Software Environments. IEEE Transactions on Software Engineering, 14 (6), pp 758-773.

Bellifemine, A., Poggi, G., and Rimassa, G. (2001). Developing multi agent systems with a FIPA-compliant agent framework. Software Practise & Experience, 31: 103-128.

Bennet K.H., and Rajlich V.T.(2000). Software Maintenance and Evolution: a Roadmap, in Finkelstein, A. (Ed.), The Future of Software Engineering, ICSE 2000, June 4-11, Limerick, Ireland, pp 75-87.

Briand, L. C., Basili, V., Kim, Y., Squier, D. R. (1994). A Change Analysis Process to Characterize Software Maintenance Projects. In Proc. of The International Conference on Software Maintenance. ICSM'94, pp. 38-49, September 1994, Victoria, British Colombia, Canada.

Evans, R., Kearney, P., Stara, J., Caire, G., Garijo, F.J., Gomez Sanz, J.J., Pavon, J., Leal, F., Chainho, P., Massonet, P. (2001). MESSAGE: Methodology for Engineering Systems of Software Agents. http://www.eurescom.de/~pub-deliverables/P900-series/P907/TI2/p907ti2.pdf. Consulted on 6th August 2003.

Fischer, G., Lemke, A.C., (1988). Construction Kits and Design Environments: Steps Toward Human Problem-Domain Communications. Human Computer Interaction, 3 (3), pp 179-222.

Gnyawali, D.R., Stewart, A.C., and Grant J.H. (1997). Creating and Utilization of Organizational Knowledge: An Empirical Study of the Roles of Organizational Learning on Strategic Decision Making. Academy of Management Best Paper Proceedings, pp. 16-20.

Henninger, S. (2003). Tool Support for Experience-Based Software Development Methodologies. _To Appear in Advances in Computer, 59, 29-82.

Henninger, S. (1995) Supporting the Domain Lifecycle. IEE Seventh International Workshop on Computer-Aided Software Engineering-CASE’95, Toronto, Canada. IEEE Computer Society Press, pp 10-19.

Kitchenham, B.A., Travassos, G.H., Mayrhauser, A., Niessink, F., Schneidewind, N.F., Singer, J., Takada, S., Vehvilainen, R. and Yang, H. (1999). Towards an Ontology of Software Maintenance. Journal of Software Maintenance: Research and Practice. 11, pp. 365-389.

Nonaka, I., and Takeychi, H. (1995). The Knowledge-Creation Company: How Hapanese Companies Create the Dynamics of Innovation. Oxford Univ. Press, New York

Oliveira, K. M., Anquetil, N., Dias M.G, Ramal, M., & Meneses, R. (2003). Knowledge for Software Maintenance. Fifteenth International Conf. on Software Engineering and Knowledge Engineering (SEKE'03) 61-68

Polo, M., Piattini, M., Ruiz, F., and Calero, C. (1999): MANTEMA: A complete rigorous methodology for supporting maintenance based on the ISO/IEC 12207 Standard. *Euromicro Conf. on Software Maintenance and Reengineering* (CSMR’99). Amsterdam (Netherland). IEEE Computer Society, , pp. 178-181.

Rugaber, S., and Tisdale, V.G. (1992). Software Psychology Requirements for Software Maintenance Activities. Software Engineering Research Center, Georgia Institute of Technology.

Ruiz, F., Vizcaíno, A., Piattini, M. y García, F. (2003). An Ontology for the Management of Software Maintenance Projects. Sent to the International Journal of Software Engineering and Knowledge Engineering.

Rus, I., and Lindvall, M. (2002). Knowledge Management in Software Engineering. IEEE Software, May/June, 26-38.

Seaman, C., Mendonca, M. G., Basili, V. R., and Kim, Y-M. (2003). User Interface Evaluation and Empirically-Based Evolution of a Prototype Experience Management Tool. IEEE Transactions on Software Engineering, Vol. 29, No. 9.

Tacla, C., and Barthès, J-P. (2002). A Multi-Agent Architecture for Knowledge Management System. Second IEEE International Symposium on Advanced Distributed Computing Systems. ISADS.