A Multi-Agent System for Knowledge Management in Software Maintenance

Aurora Vizcaíno[1], Francisco Ruiz[1], Mario Piattini[1], Jesús Favela[2]
[1]Grupo Alarcos, Escuela Superior de Informática, Ciudad Real (Spain)
[2]CICESE, Ensenada (México)

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 present decisions. This work describes a system to manage 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. The agents can learn from previous experience and share their knowledge with other agents, or communities. Different scenarios showing the system's functionality and the convenience of using it during the maintenance process are also described in this paper.

1. Introduction

Knowledge is a crucial resource for organizations [16]. Knowledge allows people and organizations to obtain power, money, and to become more competitive. For this reason, companies are currently researching techniques and methods to manage their knowledge systematically.

Organizations have different types of knowledge that are often related to each other and which must be managed in a consistent way. For instance, software engineering involves the integration of various knowledge sources that are constantly changing. The management of this knowledge and how it can be applied to software development and maintenance efforts has received little attention from the software engineering research community so far [9]. Tools and techniques are necessary to capture and process knowledge in order to facilitate subsequent development and maintenance efforts. This is particularly true for software maintenance, a knowledge intensive activity that depends on information generated during long periods of time and by large numbers of people, many of whom may no longer be in the organisation.

This paper presents a multi-agent system in charge of managing the knowledge that is produced during software maintenance. The contents of this article are organized as follows: section 2 describes the different types of knowledge that are generated during the software Maintenance Process (MP) and presents the advantages of using a Knowledge Management (KM) system in maintenance. Section 3 proposes a multi-agent architecture and describes the roles played by these agents. The ontology that conceptualises the types of information in maintenance is also outlined in this section. Section 4 illustrates the functionality of the system when used in different types of maintenance scenarios. Finally conclusions are presented in section 5.

2. Knowledge in Software Maintenance

Software maintenance consumes a large part of the overall lifecycle costs [13], [1]. The incapacity to change software quickly and reliably causes organizations to lose business opportunities. Thus, in recent years we have seen an important increase in research directed towards addressing these issues.

On the other hand, software maintenance is a knowledge intensive activity. This knowledge comes not only from the expertise of the professionals involved in the process, but it is also intrinsic to the product being maintained, and to the reasons that motivate the maintenance (new requirements, user complaints, etc.) processes, methodologies and tools used in the organization. Moreover, the diverse types of knowledge are produced in different stage of the MP.

During the software maintenance activities different people intervene. Each person has partial information that is necessary to other members of the group. If the knowledge only exists in the software engineers and there is no system in charge of transferring the tacit knowledge (contained in the employees) to explicit knowledge (stored on paper, files, etc) when an employee abandons the organization a significant part of the intellectual capital goes with him/her.

Another well-known issue that complicates the MP is the scarce documentation that exists related to a specific software system, or even if detailed documentation was produced when the original system was developed, it is seldom updated as the system evolves. For example, legacy software written by other units often has little or no documentation describing the features of the software. Using a KM system the diverse kinds of knowledge generated may be stored and shared. Moreover, new knowledge can be produced, obtaining the maximum benefit from the current information. By reusing information and producing relevant knowledge the high costs associated with software maintenance could also be decreased [5].

Another advantage of KM systems is that they help employees build a shared vision, since the same codification is used and misunderstanding in staff communications may be avoided. Several studies have shown that a shared vision may hold together a loosely coupled system and promote the integration of an entire organisation (e. g., [12])

3. A Multi-Agent System to Manage Knowledge in Software Maintenance

The above explained issues motivated us to design a KM system for capturing, managing, and disseminating knowledge in a software maintenance organisation, thus increasing the workers' expertise, the organisation's knowledge and its competitiveness while decreasing the costs associated with the software MP.

The KM system that is described in this paper is an extension of the MANTIS environment [18], for this reason it is called KM-MANTIS. MANTIS is an integrated environment for the management of software maintenance. Its main feature is that it integrates practically all the aspects that must be taken into account for directing, controlling and managing software maintenance projects.

3.1 Ontology

First of all it was necessary to clearly delimit the domain where the system would be used. In order to have a shared vision of the MP it is advisable to define a conceptualisation of the domain. An explicit specification of such conceptualis ation is an ontology [7]. An ontology represents a certain view on an application domain in which the concepts that live in this domain are defined in an unambiguous and explicit way [2]. Besides, as [7] claims an ontology enables knowledge to be shared and reused, precisely what we pretend.

KM-MANTIS uses an ontology based on the one proposed in [11]. This ontology is structured in several partial subontonlogies for software maintenance (Figure 1):

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

Activities ontology: 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*: 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*: 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: 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.

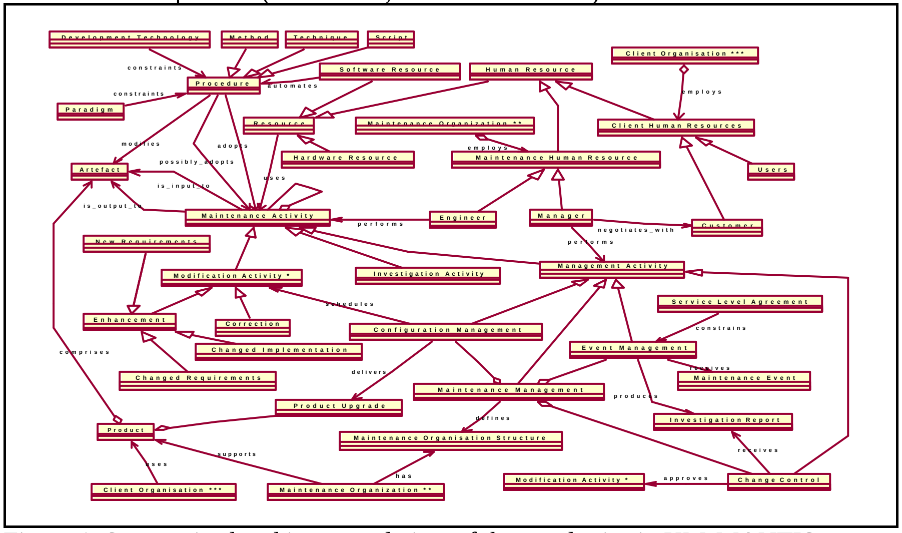

Figure 1. Summarised and integrated view of the ontologies in KM-MANTIS

3.2 KM-MANTIS System Architecture

The system is formed of a set of agent communities which manage the different types of knowledge represented by the subontologies. The system has one community termed "products community" to control information related to products. Another community is in charge of the activities. This is the "activities community". And the last community denoted as "peopleware community" arranges information related to the people involved during the MP. There is no community in charge of the process since the information about the MP itself is divided in all the communities. The process defines how methods and tools should be applied to maintenance activities and which skills are necessary to carry them out [11]. Hence, the information related to these topics are stored in the activities and the peopleware communities.
Products Community: Each product has its own features and follows a specific evolution. For this reason the architecture has one agent per product. The agents have information about the initial requirements, changes made to the product, and about metrics that evaluate features related to the maintainability of the product. Therefore, the agents monitor the product's evolution in order to have up to date information about it at each moment. Besides the information explicitly related to the products each agent has information which is also contained in other communities. For example, the products community also has information about which activity/ies was/were performed each time the product was modified and which person/people (staff member, client, users) was/were involved in that change. Thus, different communities may compare or interchange information and detect inconsistencies. Agents can communicate with each other and benefit from other communities' knowledge. This is one important feature of the architecture since each agent has enough information to be independent and autonomous. Agents can also consult other agents' information as needed.

Each agent only knows the name of the activity and the names of the people involved. The complete infomation about the activities or the staff is contained in their corresponding community.

In addition to information, the agents also have knowledge that they obtain from their learning, experience and their statistics. The knowledge generated by the agents is communicated to a special agent, "the coordinator agent", which stores the knowledge in a central database common to all the agents of this community. In this way the knowledge generated is useful for the entire community. Otherwise, each agent would be the owner of its knowledge and the same would occur as in organisations where employees do not share their knowledge.

The coordinator agent has reasoning techniques, which enables it to infer additional knowledge. We shall describe an example to clarify how this works: When a product agent detects that whenever a requirement A is demanded shortly after a requirement B is also requested, the agent should detect this pattern and communicate this fact to the coordinator. So this knowledge might be utilised by other agents. The coordinator agent, could use its knowledge in order to deduce additional knowledge. For instance, it might estimate that after requirements A and B another requirement C is often demanded. In this case, the coordinator would inform the first agent that a new requirement C will be demanded in brief. Therefore, the agent could prepare itself to perform the requirement C. Moreover, the system would inform (for instance via e-mail) the staff in charge of maintaining the product that a new requirement C should be performed.

Activities community: Each new change demanded implies performing one or more activities. This community, which has one agent per activity, is in charge of managing the knowledge related to the different activities.

In order to carry out an activity, as the procedures sub-ontology indicates, methods, and techniques can be applied and different resources could be used as well. We had considered adding a new community to the architecture in charge of controlling this information. However, the fact that this information is so narrowly related to the activities changed our minds and we now consider that the information related to methods, techniques, tools and resources should be managed by each agent belonging to the activity community.

As with the previous community, in this community the agents have information related to other communities. For instance, each agent knows for what requirement and in what product/s each activity is carried out. The agents also have references to the people involved in that activity. Furthermore, the agents have knowledge obtained from their experience and learning. For instance, an activity agent can learn what resources are always used in order to carry out a task or which method gives better results.

The activity community also has a coordinator agent that is informed about all the knowledge generated. As in the previous case, it has reasoning techniques that help produce new knowledge.

Peopleware Community: An analysis was performed with the objective of indicating which people are involved in the MP. This showed that three profiles could be clearly differenced: the maintainer, the customer and the user. The result of the analysis and the description of the roles of each profile can be found in [14]. We have designed three agents, one per each profile detected. One agent is in charge of the information related to staff (maintainers). This is the staff agent. Another manages information related to the clients (customers) and is called the client agent. The last one is in charge of the users and is termed the user agent.

The staff agent knows the personal data of the employees, in what activities they have worked, and what product they have maintained. Of course, the agent also has current information about each member of staff. Therefore it knows where each person is working at each moment.

As happened in the previous community, there is common information (such as, name of products and activities) which may be used in a similar way to a foreign key in databases, enabling the communities to be connected.

The agent utilises the information that has to generate knowledge. For instance, it calculates statistics that indicate the time that an employee took to perform a task or calculates the performance graph of each member.

The client agent stores the information of each client, their requirements (even the initial requirements if they are available) and the name of the product which should be modified.

The client agent also tries to obtain new knowledge. For instance, it tries to guess future requirements depending on previous requirements or it estimates the costs of changes that the client wants to make, warning him for instance of the high costs associated to a specific change request.

The user agent is in charge of knowing the necessities of the users of each product, their background and also their complaints and comments about the products. New knowledge could be generated from this information, for example by testing to what degree the users' characteristics influence the maintenance of the product.

As will be described in more detail later, the agents use different information and generate new knowledge through different artificial intelligent techniques. This is one of the most important advantages of using agents, since they contain the capacity to manage information and reasoning. Another feature associated with agents is that they are proactive: they act when they consider that it is convenient to do so and nobody has to indicate to them when and how to act [19].

The architecture has certain information duplicated, and although this has the problem of having to control the consistency of information, it allows agents to act autonomously and independently since they have complete information. It also increases the robustness of the system.

The architecture presented enables two types of collaboration. Collaboration between agents belonging to the same community and between agents of different communities. An example of the former occurs when an agent of the product community asks another agent about the costs of performing a concrete activity. Moreover, the agents belonging to the product and activity collaborate with their coordinator agent sharing their knowledge with it so that the knowledge is accessible to all the agents.

The second type of collaboration is produced between agents of different communities. An activity agent may check, by asking the staff member, data about a member who is performing a specific activity or verify whether the information that it has is correct.

Both types of collaboration allow agents to take advantage of the information and knowledge that other agents have, besides controlling the consistency of the information.

3.3 General Roles of the Agents

Although each agent has specific roles, in a very general way agents should:

Compare new information that they receive with that which has already been stored in order to detect inconsistencies between old and new information. If an inconsistency is detected, the agent must consult those agents which contain related information in order to discover where the inconsistency is and why it has occurred.

Inform other agents about changes produced. For example if the staff agent was informed that a member in charge of the maintenance of a specific product was substituted for another member, it will inform the product agent (and the activity agent if necessary) about this change, since the product agent must know which people are working on the product at each moment.

Predict new clients' demands. This role is played mainly by the products agents since similar software projects often require similar maintenance demands. What a company has done before tends to predict what it can do in the future [8]. This role is very important, 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 [1] claims, if changes can be anticipated they can be built in by some form of parameterisation. In this way costs and efforts are decreased.

Predict possible mistakes by using historic knowledge. As stated in [9], KM avoids the repetition of common mistakes. In KM-MANTIS, when the activity community coordinator agent is informed of a mistake which occurred in the development of an activity, all the activities agents will be informed about this in order to prevent the same mistake from being repeated.

Suggest solutions to problems. Storing solutions that have worked correctly in previous maintenance situations helps to avoid the effect document by [20] in which, due to the limited transfer of knowledge, companies are forced to reinvent new practices resulting in costly duplication of effort. The best practices often linger in companies for years unrecognised and unshared. The coordinator of the product and the activity communities should know what solutions were the best (quicker and cheaper) for problems.

Help to make decisions. This is one of the most important goals of the KM systems. When knowledge is enhanced it is easier to improve problem identification, development of alternative solutions and the selection of the best solution [6]. In the system the agents of the activities community can advise, for instance, whether it is convenient to outsource certain maintenance activities.

Advise certain employee to do a specific job. The staff agent has information about each employee's skills, their performance metrics, and the projects they have worked on. For example, the staff agent may process this information to suggest which person is most suitable to carry out a task.

Estimate the cost of future interventions. Information available may be used to make statistical analyses that help predict maintenance effort and costs. This is an important issue since sometimes planned changes can not be performed because of lack resources [1].

The agents use different types of information in order to play their different roles:

Data that the agents receive explicitly. An example of this type of information is the name of the member of the staff or the products that should be maintained.

Information and knowledge that agents generate from the data obtained. The agents may infer new knowledge through different reasoning techniques. For example analogy, they compare new types of knowledge with previous ones. Similarities and differences are analysed in order to reach conclusions. Imitating experts' reasoning in this way, experts seem to employ a form of analogical reasoning where effort is estimated by comparing the problem at hand with cases attempted earlier [3]. One advantage of using an multi-agent architecture is that each agent can have its own reasoning technique depending on the type of task they perform. For instance, genetic algorithms can be used in optimisation tasks, while neural networks will be appropriate for agents that need to find complex mappings between patterns of data. The type of reasoning used is hidden in the agent and can be updated as needed without affecting the rest of the agents. This architecture also allows to incorporate additional agents to perform new tasks.

Knowledge generated from learning. One of the most important features of the agents is that they can learn. Agents can use neuronal nets or other learning algorithms such as ID3 to learn from previous information. One important advantage of the system is that besides managing knowledge it also generates knowledge.

Knowledge shared by other agents. Agents collaborate with the agents of their own community and with those of other communities

4. Using KM-MANTIS in Software Maintenance

In order to illustrate the system's functionality two scenarios are described. Each scenario corresponds to a type of maintenance. The goal of this section is to indicate how KM-MANTIS would work in each situation and what benefits would be obtained from its use.

There exist different classifications of maintenance. In our work we use the classification considered in MANTEMA [17], this classification is based on the norm ISO 12207 [10]. MANTEMA is a complete methodology designed for software maintenance [15].

Experience has shown that the process of maintenance is different when it is urgent and when it is not. When there is urgent planning is practically non-existent. For this reason, the maintenance is divided into plannable and non-plannable. Plannable maintenance would be the perfective, the adaptive, the preventive and the corrective non-urgent. Non-plannable maintenance is the urgent corrective.

The first scenario that is described illustrates how the system would act in a case of maintenance plannable. The second example shows a case of maintenance non-plannable.

4.1 Plannable Maintenance

When, for instance, a change to improve the quality of a product is demanded and performed, the coordinator of the products community would check whether products with very similar functions are being maintained. In this case, KM-MANTIS could predict that the same change might be required for these products in the future and inform the staff in charge or their maintenance. This might be the case, for instance, of changes in the tax law that originate changes in collection of software systems.

The system, besides predicting new changes, could check in what moment it is advisable to perform them. Sometimes it is better to delay the modification tasks thus reducing the possibilities of introducing new errors. Apart from checking whether a change is suitable for other products, the system would inform the staff about the predictions of possible changes or the convenience of carry out them.

4.2 Non-plannable Maintenance

A common source of maintenance tasks is generated when a user informs the staff that the program that he is using no longer works and it shows a strange error message. In this case the staff must act immediately: there is no time to plan the activities to perform. The staff could consult the system in order to obtain information about the error and the causes that could have motivated it. The system might also indicate the questions that should be asked to the user to obtain additional information that may help deduce the origin of the problem.

The coordinator agent of the products community would test whether a similar situation had already occurred and if so, retrieve the solution given to that problem. If the answer was positive, the coordinator of the product community would contact the coordinator of the activity community asking for the appropriate method or technique to solve the problem and their associated costs. The staff agent could also be asked to suggest who can carry out the changes and how to contact them.

If the error have not occurred before, the product agent would process the information that it had and the new information incorporated by the indications of the user, with the goal of trying to predict how the error could be solved.

Once the solution is found (by the system or by the staff) all the information related to the mis take is stored to be used in similar circumstances in the future. Thus, all new situations offer new opportunity to learn.

5. Conclusions

Software maintenance is one of the most important stages of the software life cycle. This process takes 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.

We have presented a multiagent system in charge of managing this knowledge in order to improve the MP. The advantage of using agents is that, apart from managing information they also learn and generate new knowledge

The scenarios described have illustrated the use of KM-MANTIS in plannable and non-plannable maintenance. They have also helped to show how the system would assist maintenance engineers perform their jobs. In addition, costs and efforts would decrease by using KM-MANTIS because solutions that worked in the past are reused and good decis ions are made. New changes could also be predicted thus increasing the organisation's competitiveness.

References

1. 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.
2. Card, D.N., and Glass, R.L .(1990) Measuring Software Design Quality. Prentice Hall, Englewood Cliffs, NJ.
3. Carr, M., and Wagner, C. (2002). A Study of Reasoning Processes in Software Maintenance Management. Journal of Information Technology & Management. Pirkul, H., and Vargheses .J (Eds.). Vol. 03. Kluwer Academic Publichers, pp 181-203.
4. Deridder, D., (2002). A Concept-Oriented Approach to Support Software Maintenance and Reuse Activities. KBOSSE Workshop. ECOOP, Málaga. 11 June.
5. De Looff, L.A., Information Systems Outsourcing Decision Making: a Managerial Approach. Hershey, PA: Idea Group Publishing, 1990.
6. 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.

7. Gruber, T. (1995). Towards Principles for the Design of Ontologies used for Knowledge Sharing. International Journal of Human-Computer Studies, 43(5/6), pp 907-928.
8. Gupta, A., and Govindarajan, V. (2000). Knowledge Flows within Multinational Corporations. Strategic Management Journal, 21(4), pp. 473-496.
9. Henninger, S., and Schlabach, J. (2001). A Tool for Managing Software Development Knowledge, 3ª International Conf. on Product Focused Software Process Improvement. PROFES 2001, Lecture Notes in Computer Science, Kaiserslautern, Germany, pp 182-195.
10. ISO/IEC, 1995. International Standard Organization, ISO 12207: *Information Technology-Software Life Cycle Processes*. Switzerland.
11. 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.
12. Orton, J.D., and Weick, K.E. (1990) Loosely coupled systems: A reconceputalization. Academy of Management Review, 15(2), pp 203-223.
13. Pigoski, T.M. (1997). Practical Software Maintenance. Best Practices for Managing Your Investment. Ed. John Wiley & Sons, USA, 1997.
14. Polo, M., Piattini, M., Ruiz, F. and Calero, C. (1999): Roles in the Maintenance Process. *Software Engineering Notes*; vol 24, nº 4, pp. 84-86. ACM.
15. 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. *Third Euromicro Conf. on Software Maintenance and Reengineering* (CSMR'99). Amsterdam (Netherland). IEEE Computer Society, , pp. 178-181.
16. Ruhe, G. (2002) Learning Software Organizations. Handbook of Software Engineering and Knowledge Engineering. Chang, S.K. (Ed.) Vol.1, World Scientific Publishing.
17. Ruiz, F., Piattini, M., Polo, M., and Calero, C. (1999). Maintenance types in the MANTEMA methodology. *International Conf. on Enterprise Information Systems* (ICEIS'99). Setubal (Portugal). Filipe and Cordeiro (Eds.), pp 192-202.
18. Ruiz, F., García, F, Piattini, M., Polo, M. (2002) Environment for Managing Software Maintenance Projects. In *Advances in Software Maintenance*: Technologies and Solutions. Idea Group Publishing, USA (in press).
19. Russell, S. J. and Norvig, P (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ.
20. Zell, D. (2001) Overcoming Barriers to Work Innovations: Lessons Learned at Hewlet-Packard. Organizational Dynamics, 30 (1), pp 77-86.