The ICFAI University Press

PROJECTS & PROFITS

June 2005 www.icfaipress.org Rs. 60

Software Development Projects

PROJECTS & PROFITS

Volume-V: 6 June 2005

EDITOR
E N Murthy

EXECUTIVE EDITOR
Ch Rajeshwer

CONSULTING EDITOR
Y Chandra Sekhar

EDITORIAL TEAM
A Anand, Arshia Anwer, P Ravindra Varma

PRODUCTION MANAGER
A L Subrahmanyam

PRODUCTION TEAM
N Lalitha Devi, Aparna Marthi

CHIEF VISUALIZER
Bangaru Babu A

VISUALIZERS
S Ganesh, PRV Prasad

GRAPHIC DESIGNERS
Md. Nazeer Hussain, KPPS Prasad

ILLUSTRATOR
Anjaneyulu B

ADVERTISEMENT ENQUIRIES
Business Manager, The ICFAI University Press, # 23, Nagarjuna Hills, Panjagutta, Hyderabad - 500 082.
Tel: +91(40) 23430-431 to 436
Fax: +91(40) 5563-9711

SUBSCRIPTION DETAILS

	By Post	By Courier
1 year	Rs. 625	Rs. 750
3 years	Rs. 1650	Rs. 2000
5 years	Rs. 2000	Rs. 2500

Payment to be made by crossed Demand Draft drawn in favor of The ICFAI University Press, Hyderabad.
Visa & MasterCard holders can subscribe online or fax their subscription order indicating their credit card number, name, amount, expiry date, duly signed.
For subscriptions and related enquiries, write to:
Executive (Subscriptions), The ICFAI University Press, # 52, Nagarjuna Hills, Panjagutta, Hyderabad - 500 082.
Tel: +91(40) 23435-368 to 374, Fax: +91(40) 2335-2521
E-Mail: serv@icfaipress.org
Also visit www.icfaipress.org

ISSN 0972-5334

Printed and published by E N Murthy on behalf of The Institute of Chartered Financial Analysts of India, The ICFAI University, # 52, Nagarjuna Hills, Panjagutta, Hyderabad-500 082, Andhra Pradesh. Printed at M/s ICIT Software Center Pvt. Ltd., # 1, Technocrat Industrial Estate, Balanagar 'X' Roads, Hyderabad-500037, Andhra Pradesh and published from The Institute of Chartered Financial Analysts of India, The ICFAI University, # 52, Nagarjuna Hills, Panjagutta, Hyderabad-500 082, Andhra Pradesh.

Editor: E N Murthy

How to Reach Us

Send your feedback to

The Editor, The ICFAI University Press,
6-3-354/1, Stellar Sphinx, Road No. 1, Banjara Hills, Panjagutta, Hyderabad-500 082.
Tel: +91(40) 23430-448, 449, 450, 451
Fax: +91(40) 23430-447
e-mail to info@icfaipress.org, Website: www.icfaipress. org

Advertising

For information and tariff:
E-mail: mkt@icfaipress.org, or Tel: +91(40) 23430-431 to 436
Fax: +91(40) 5563-9711

Articles

Articles may be sent to
info@icfaipress.org

For more information, please visit
www.icfaipress.org

Subscriptions

Please contact your nearest ICFAI Center for details:

Agra (Ph: 0562-2527035); **Ahmedabad** (Ph: 079-6563042, 6562458); **Ajmer** (Ph: 0145-2622707); **Allahabad** (Ph: 0532-2420255); **Anantapur** (Ph: 08554-249334); **Aurangabad** (Ph: 0240-5624774 / 775); **Bangalore** (Ph: 080-2899363 / 262); **Bhopal** (Ph: 0755-5277253, 2576975); **Bhubaneswar** (Ph: 0674-2506203/204); **Chandigarh** (Ph: 0172-381128 749504); **Chennai** (Ph: 044-28235633, 28235688); **Coimbatore** (Ph: 0422-2541190, 5366447); **Dehradun** (Ph: 0135-2654002 / 03); **Guntur** (Ph: 0863-2238958); **Gurgaon** (Ph: 0124-222-3595, 5556); **Gwalior** (Ph: 0751-2322273); **Hubli** (Ph: 0836-2371738); **Hyderabad** (Ph: 040-55663661, 55681180/81); **Indore** (Ph: 0731-5069003, 5068247); **Jabalpur** (Ph: 0761-5066886); **Jaipur** (Ph: 0141-2363695, 2373689); **Jalandhar** (Ph: 0181-5074769); **Jamshedpur** (Ph: 0657-2434957); **Kakinada** (Ph: 0884-2387772); **Kanpur** (Ph: 0512-233-0912 / 1145); **Kochi** (Ph: 0484-2369763, 2382294); **Kolhapur** (Ph: 0231-2655142, 2654491); **Kolkata** (Ph: 033-22873161/ 22817802); **Kurnool** (Ph: 08518-249811); **Lucknow** (Ph: 0522-220-4205/ 4559); **Ludhiana** (Ph: 0161-2772523); **Madurai** (Ph: 0452-2342169); **Mangalore** (Ph: 0824-2432050); **Mumbai** (Ph: 022-22040888, 22823173); **Mysore** (Ph: 0821-543803); **Nagpur** (Ph: 0712-2564314, 2547124); **Nasik** (Ph: 0253-2570413); **Nellore** (Ph: 0861-2301222); **New Delhi** (Ph: 011-23739157, 23739169); **Noida** (Ph: 0120-2592410, 2516024); **Patna** (Ph: 0612-2237942); **Pune** (Ph: 020-4026975, 4026976); **Raipur** (Ph: 0771-5061361); **Rajahmundry** (Ph: 0883-2448813); **Ranchi** (Ph: 0651-2306922); **Thane** (Ph: 022-25375836, 25382659); **Trivandrum** (Ph: 0471-2320853); **Udaipur** (Ph: 0294-5102254); **Vadodara** (Ph: 0265-2341780); **Vijayawada** (Ph: 0866-2473620, 5563620); **Visakhapatnam** (Ph: 0891-2752653, 2598650); **Warangal** (Ph: 0870-2552610)

PROJECTS & PROFITS

Knowledge Management in Software Companies

An Experience Factory Approach

Juan P Soto, Aurora Vizcaino, Mario Piattini and Francisco Ruiz

Appropriate knowledge management would help software companies to improve performance, control costs and decrease effort by taking advantage of previous solutions that could be reused to avoid repeating previous mistakes. The Experience Factory implementation is a good solution for knowledge management since it avoids overloading software employees with extra work. This paper describes the importance of using knowledge management in the software maintenance process and describes how to implement an experience factory to manage the information and knowledge generated during software maintenance, as well as the way in which the information can be explicitly represented in these kind of systems.

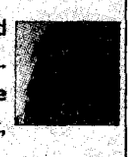
Introduction

Nowadays, topics like knowledge management are used with special interest in organizations worried about employees' learning and about their competitiveness. Knowledge management can be defined as a discipline that enables organizations to take advantage of their intellectual capital in order to reuse it and learn from previous experience [4]. The software industry, encouraged by the idea of improving costs, schedules, quality of their products, and customer satisfaction, is staking its money on adding knowledge management [16]. An important reason to do this is that software engineering is knowledge-intensive work where the main capital is what has been called the "intellectual capital". Unfortunately, the owners of this intellectual capital are often the employees instead of the company as we might expect. Employees, from their experience, obtain tacit knowledge, which is richer and more valuable than explicit knowledge, but this cannot be easily expressed or communicated [17]. Furthermore, software development is a constantly changing process. Many people work in different phases, activities and projects. Knowledge in software engineering is diverse, and its proportions are immense and steadily growing [19]. These issues are also more relevant in software maintenance, where the persons who have to maintain the software usually do not take part in its development. Moreover, the documents related to a specific software system are often scarce and seldom updated as the system evolves. Because of all these circumstances, we have focused our work on the software maintenance stage.

One problem of using knowledge management in these kind of companies is how to implement this process in the company without overloading programmers or maintainers with extra work. This problem can be solved by using an

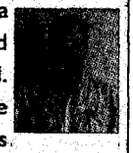
About the Authors

Juan Pablo Soto is a doctoral student in the Computer Science Department of the University of Castilla-La Mancha, Spain. His research interests are focused on applying Knowledge Management to improve Software Engineering Processes by using the Experience Factory approach, especially in the Software Maintenance Process.



Aurora Vizcaíno is an M.Sc. in Computer Science from the University of Granada and she has a European Ph.D. in Computer Science from the University of Castilla-La Mancha. Currently, she is a fulltime Associate Professor in the Computer Science Department of the University of Castilla-La Mancha. Her research interests include Knowledge Management, Agents, Software Maintenance, Groupware and Collaborative Learning.

Mario Piattini is an M.Sc. and Ph.D. in Computer Science from the Politechnical University of Madrid and M.Sc. in Psychology from the UNED. He is a Certified Information System Auditor and Certified Information Security Manager from ISACA (Information System Audit and Control Association) and a Full Professor at the Department of Computer Science at the University of Castilla-La Mancha. Author of several books and papers, he leads the ALARCOS research group that specializes in information system quality. His research interests are software quality, advanced database design, metrics, software maintenance, information system audit and security.



Francisco Ruiz holds a Ph.D. in Computer Science from the University of Castilla-La Mancha (UCLM) and an M.Sc. in Chemistry-Physics from the Complutense University of Madrid (Spain). He is fulltime Associate Professor of the Department of Computer Science at UCLM. He has been Dean of the Faculty of Computer Science between 1993 and 2000. Previously, he was Computer Services Director (1985-1989) and he has also worked in private companies as an analyst—programmer and project manager. His current research interests include business process modeling, management and measurement, software process technology and modeling, software maintenance, and methodologies to software projects planning and managing.

Experience Factory, which is a separate company in charge of storing and managing knowledge.

The contents of this paper are organized as follows: Section 2 explains the characteristics and problems of software maintenance. Section 3 describes how to implement an Experience Factory for a software maintenance company. In Section 4 we propose different roles that should be played in an Experience Factory, and the interaction among them is also described. Finally, the conclusions are presented in Section 5.

Characteristics and Problems of Software Maintenance

Maintenance is the most costly stage of the software life cycle [8], in some cases representing between 67% and 90% of the total costs [20]. Software maintenance is an activity where different kinds of knowledge is generated from different sources. This knowledge comes from the expertise of the professionals involved in the process, from the product being maintained, and the reasons that motivate the maintenance (new requirements, user complains, etc). Moreover, software maintenance is a constantly changing process since maintenance results from the necessity of adapting software systems to an ever-changing environment [19]. Software maintenance is knowledge intensive work, where maintainers must collaborate with others members of the team, and share their knowledge and experience in order to complete their assignments. Even though maintainers mainly use their own experience, they generally do not have enough knowledge to complete their

work and need to consult other sources of information such as documents or other members of the team in order to finish some assignments [28]. However, this could be a difficult task when those sources are limited, inaccessible, or unknown [24]. Frequently, companies have documents or people with the information or knowledge necessary to support or help the maintainers in doing their activities, but employees often ignore their existence or location. Another problem is that software organizations still lack a culture of reuse and information sharing.

A knowledge management system could help to avoid some of the issues previously commented on. For instance, if organizations store their information and knowledge in a Knowledge Management System (KMS), they would own this intellectual capital. Therefore, even if experts left the organization, their expertise would remain with the companies. Storing knowledge decreases dependency on employees' knowledge because at least some of their expert knowledge has been retained or made explicit. Moreover, storing good solutions to problems or lessons learned avoids repeating mistakes and increases productivity and the likelihood of further success [16]. For these reasons, we decided to develop an Experience Factory (EF) that can be used in the software maintenance process. This factory can supply relevant information, techniques, solutions and lessons learned for the people involved in software maintenance without overloading their work. Moreover, an EF helps to detect and capture employees' expertise and generate knowledge in the organization. Another reason that motivated us to use an EF is that it can use different reasoning techniques depending on the situation. For instance, by using Case-Based Reasoning (CBR) the problem of finding similar information or related problems is simplified. Furthermore, EF systems have already been successfully used in other systems in charge of knowledge management. Therefore, their efficiency in these tasks has been proved [3] [5] [12] [23].

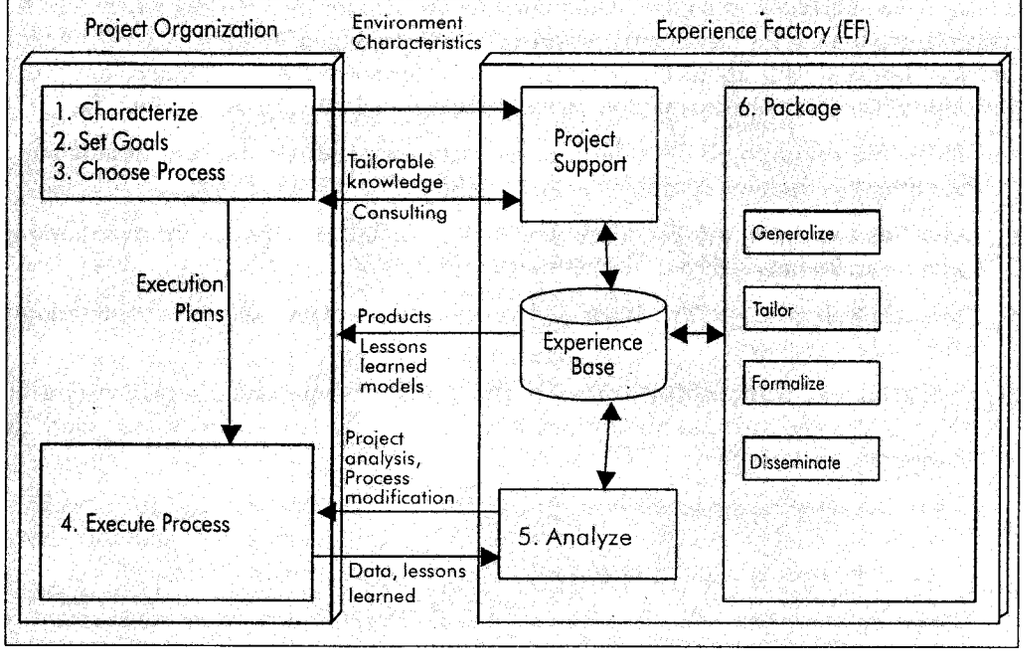
An Experience Factory helps to detect and capture employees' expertise and generate knowledge in the organization

Experience Factory

The Experience Factory (EF) approach was initially designed for software organizations and takes into account the software discipline's experimental, evolutionary, and non-repetitive characteristics [2]. The EF organizes a software development enterprise into two distinct organizations: The Project Organization, which uses packaged experience to deliver software products, and the Experience Factory, which supports software development by providing tailored experience, each one specializing in its own primary goals. Therefore, the Project Organization focuses on delivering the software product and the EF focuses on learning from experience and improving software development practices in the organization. Although the roles of the Project Organization and the EF are separate, they interact to support each other's objectives. The feedback between the two parts of the organization flows along well-defined channels for specific purposes, as is illustrated in Figure 1. These flows may be summarized as the Project Organization informs the EF what features the new project has, so the EF searches in the experience base for what previously learned lessons can be reused in the new project. After executing the plans that help to reach the goals previously established, the Project Organization sends the EF the new

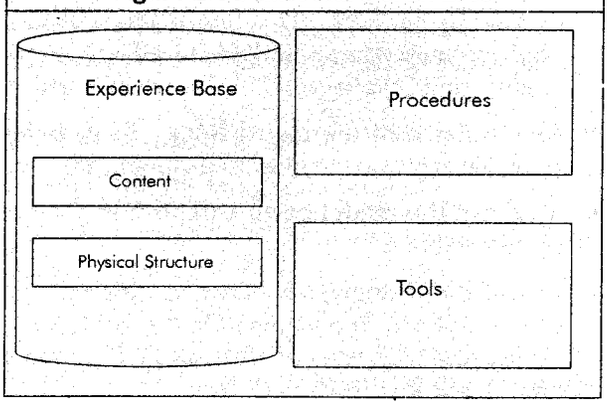
information generated. This information (experience, lessons learned, data of project, and technology reports) is analyzed by the EF, which decides if it should be stored in the experience base.

Figure 1: Experience Factory Structure (Figure in [25])



A physical implementation of the EF in an organization is called the Experience Management System (EMS). The EMS is composed of content, structure, procedures and tools (see Figure 2). The content may be data, information, knowledge or experience. The structure defines how the content is organized. The content plus the structure are often considered as the Experience Base. Procedures are instructions that indicate how to manage the experience base, including how to use, package, delete, integrate and update the experience. Tools support managing the content and the structure, and carrying out the procedures, as well as helping to capture, store, integrate, analyze, synthesize and retrieve experience. In the following section, the content of the experience base that we propose for an EF for maintenance is described.

Figure 2: Architecture of an EMS



A. Modeling and Implementing Knowledge in an Experience Base

Before constructing a knowledge management system for software maintenance, it is vitally important to model, structure and generalize the information that is generated during the software maintenance process. In order to achieve this goal, we have developed a common conceptualization of the domain related to the software maintenance process, where objects, concepts, entities and their relationships are explicitly represented. As a result, a *Maintenance Ontology* (more details about the ontology in [22]) for software maintenance has been developed. The maintenance ontology is formed of several subontologies. These are:

- **Products Subontology:** This defines the software products that are maintained, their internal structure, composition and existing versions of each product.
- **Activities Subontology:** This describes how to organize activities for maintaining software and what kinds of activities they may be.
- **Processes Subontology:** This is divided into two different focuses, and a subontology is defined for each one:
  - **Procedures Subontology:** This defines how the methods, techniques and tools can be applied to the activities and how the sources are used in order to carry out these activities.
  - **Process Organization Subontology:** This focuses on how the support and organizational processes are related to the software maintenance activities, how the maintainer is organized, and what his contractual obligations are.
- **Agents Subontology:** 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.

In order to represent all the information described in the ontology in a computer we used the methodology called REFSENO (Representation Formalism for Software Engineering Ontologies) [27]. The reasons why this methodology was chosen are:

- As the name of the methodology itself indicates, it was specifically designed to develop software engineering ontologies.
- It allows the modeling of software engineering knowledge by using alternate representations.
- REFSENO distinguishes between different levels of knowledge: Conceptual and context-specific knowledge. On the contrary, other approaches such as [9,11, 26] represent a high-level of abstraction. Consequently, they represent a lesser level of granularity than REFSENO does.
- The methodology proposes different techniques to check the consistency of the ontology and, what is more, has methods of controlling the consistency of the instances to an implementation level, a feature that other methodologies do not consider.

In order to explain how the information is stored in our database some concepts of REFSENO should be introduced. REFSENO provides constructs to describe concepts where

each concept represents a class of experience items. Besides concepts, its properties (called terminal attributes) and relationships (non-terminal attributes) are represented. The first step to implement the ontology by using REFSENO is to define a concept glossary which provides a general description and the purpose of the concepts (products, activities, process organization, agents involved in the software maintenance process). Each row of the table corresponds to one concept, for example in Table 1 the products subontology concept is presented.

Table 1: Products Subontology: Concept Glossary

Concept	Super-Concept	Description	Purpose
Artifact	Element	This is a Software Product, part of which is created or modified by the activities. It may be a document (text or graphic), or code module. Examples: Requirement specification documents, quality plan, class module, routine, test report, user manual. Synonymous: Software element, work product, product item.	To define the internal structure and software composition.
Product	Concept[1]	Software application which is being maintained. It is a conglomerate of different artifacts. Synonymous with Software.	Maintenance
Version	Concept	This is a change in the base line of the product. It could be an upgrade, release or actualization.	To implant the configuration management process.

The second step is to construct a terminal attribute table for each concept defined in the glossary table. Terminal concept attributes are described by a tuple formed from the following items:

- **Name:** The name is used for reference purposes.
- **Description:** A narrative text which defines the meaning of the attribute.
- **Cardinality:** A range specifying the minimum and maximum number of values the attribute may have.
- **Type:** Each terminal concept attribute is given a type, and the types are viewed as an epistemistic primitive. REFSENO has some predefined types such as Boolean, Integer, Real, Text, Identifier or Date. New types can be described by users.
- **Mandatory:** This is also related to new instances. It indicates whether an attribute value of an instance has to be specified.
- **Inferred Attributes:** This component lists all the attributes whose value is inferred using a value of this attribute. There is a mutual dependence between value

[1] The super-concept "Concept" is the root.

inferences and inferred attributes, thus inferred attributes can automatically be derived from the value inferences.

- **Standard Weight:** This weight may be used by the similarity functions (explained later) of the concept this attribute belongs to. A weight of 0 denotes an attribute whose value will not be used for querying.

In this paper, only the product terminal attribute table is shown due to space constraints. Table 2 shows some of the types which we have defined, such as Maturity Type, which defines a range of types of maturity that a software product might have.

REFSENO distinguishes three layers to which attributes may belong. These are artifact, interface and context. The attributes of the artifact layer characterize the instances themselves. Attributes of the interface layer characterize how a particular instance can be integrated into the system. Attributes of the context layer characterize the environment in which the instance has been applied and the quality of the instance in the specified environment.

In Table 2, the attributes preceded by (I/F) mean attributes of the interface layer. In the case of the "Product", there is only one interface layer attribute which indicates in which of the artifacts the product may be decomposed. Obviously, if the product has a relation with the concept "artifact", this concept must have the opposite relationship Decomposition-of indicating the product which it is a decomposition of.

Table 2: Products Subontology: Attributes Table for the Concepts "Product"

Name	Description	Cardinality	Type	Mandatory	Stand.Weight
Maturity	Phase of the product life cycle	1	MaturityType	Yes	1
Size	Qualitative measurement of the size	1	SizeMeasuerment	Yes	1
Composition	Abstract level of the artifacts that form it	1	CompositionType	Yes	1
Application Type	Type of application. For instance: Management, scientist, etc.	1	ApplicationType	Yes	1
Quality	Qualitative measure of the quality, basically of the documentation	1	MeasurementQ	Yes	1
Age	Number of years from when the first version was obtained	1	Integer	Yes	1
(I/F) Component	Artifact into which the product is divided.	0..*	Has-decomposition (artifact)	No	1

The knowledge stored in a system should be reused. In order to do this, we used Case-Based Reasoning (CBR), which is often used to find the best solution to problems dealing with selecting a solution from many existing ones [18]. Fortunately, REFSENO provides

constructs such as the similarity function that facilitated the use of CBR in our system. Taking advantage of the information shown in Table 2, we are going to illustrate how our system calculates the similarity function when it is necessary to compare two instances of product, for example, i and q. First of all, the similarity functions for each layer, artifact and I/F (the context layer is omitted because in this case there are no attributes of this layer) should be calculated. They are stressed in the formula below.

$$\text{Sim (product) (i, q)} = W_{artif} * sim_{artif}(\text{product})(i, q) + W_{I/F} * sim_{I/F}(\text{product})\ (i, q)$$

The local similarity functions are calculated by computing the sum of the similarity function of each type of attribute belonging to this layer. Finally, each local similarity function is normalized resulting in a value in the range [0,1]. Thus, in the case of the artifact layer of the concept product, it is necessary to know the similarity function of the types "Maturity Type", "SizeMeasurement", "CompositionType", "ApplicationType", "MeasurementQ" and "Integer" in order to obtain their sum.

REFSENO provides several predefined types and their similarity functions. For instance, the "Integer" type has the following similarity function to compare two instances i and q:

$$\text{Sim}(i, q) = 1 - \frac{|i - q|}{(\text{max value} - \text{min value})}$$

where minvalue and maxvalue are respectively the lower and upper bound of the value range.

In the case of using types defined by the user, such as "MaturityType", their similarity types should also be described. For instance, Maturity Type is a taxonomy formed of four labels: Initial, evolution, service and retired and its similarity function is the following:

Sim(i, q):	1	if i= q
	0.5	if i= initial and q=evolution or vice versa
	0.25	if i= initial and q=service or vice versa
	0	if i= initial and q= retired or vice versa
	0.5	if i= evolution and q= service or vice versa
	0	if i= evolution and q= retired or vice versa

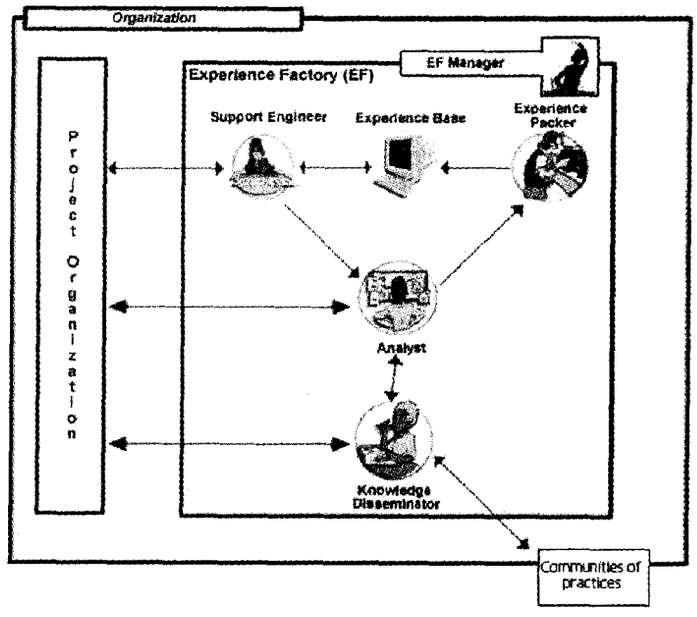
After calculating the local similarity functions $sini_{artif}(product)(i, q)$ and $sim_{I/F}(product)\ (i, q)$ the global similarity function should be calculated by assigning values to W_{artif}, and $W_{I/F}$, depending on what the user's needs are. For instance, if the system wants to compare the similarity between two products according to their own features, the value of W_{artif} should be maximized and $W_{I/F}$ decreased since the sum of the weights is always 1. Therefore, the system adapts the weights according to the convenience of giving more priority to one layer or to another. With these similarity functions the system can compare software products, and maintenance requests. One goal of comparing products is to predict new clients' demands since what a company has done before tends to predict what it can do in the future [10]. Therefore, products with similar features often demand the same modifications.

Now that we have explained how the information is stored by using REFSENO tables and consulted in the experience base by using CBR, we are going to explain an aspect seldom dealt with in literature. This is: What roles should exist in an EF for software maintenance and how the different roles interact each others. This issue is very important to help a company to implant the EF.

Roles in an EF

Taking into account the architecture of an Experience Factory, as well as the needs of knowledge management in software engineering, we propose the following roles to analyze, process, pack and disseminate the experiences generated in an organization. Of course, in small organizations these roles can be played by the same person. The communication existing between both is graphically represented in Figure 3.

Figure 3: Experience Factory Roles



First of all, a person in charge of controlling that everything works appropriately in the Experience Factory is necessary. We called this role *Experience Factory Manager*. Therefore, this role coordinates the work of the other team members. The Manager usually interacts with the executives in charge of the projects to determine the goals that the Experience Factory should achieve. The manager also evaluates the contribution of the experience factory to the company and determines future improvements. Using computer science terminology, one could say that this is the interface between the managing team and the Experience Factory.

In order to foster knowledge reuse, it must be disseminated to the person or group of people who may need it. This task is carried out by the *Knowledge Disseminator*. He must detect the group of people or communities who generate and use similar information. For instance, the people who maintain the same product or those who use the same programming language. Therefore, this role fosters the idea of a community of practice in which each person shares knowledge and learns thanks to the knowledge of other community members [29]. An appropriate knowledge management linked to communities of practice helps to improve the organization's performance [15]. Disseminated information may be of different types; it may be information linked to the company's philosophy or

specific information about a determined programming language. This focus can help to create communities of practice of people who might not be directly involved in the development of projects, such as the management team. In this way, the Experience Factory is intended to help manage the different sorts of knowledge within a company (organizational, process-linked, activity-linked, etc.) thus fostering knowledge sharing at all levels. To spot communities of practice, the Knowledge Disseminator needs to know exactly what kind of work each member of the organization is in charge of and the knowledge flows linked to their jobs. The detection of such flows may be carried out following the KOFI methodology proposed by [21] in which knowledge sources, the kind of knowledge that can be obtained from each source and the knowledge flows can be detected.

A third role is that of the *Support Engineer* who is in charge of helping software developers, and receives all the information and features of the projects. He compares current information with that of past projects and proposes development choices. Moreover, the Support Engineer helps to establish costs and project deadlines. This person has access to experience package stored in the repository. Therefore, this person has the responsibility of recovering the experience packages related to the current project and detecting the best methods for its implementation. When the Support Engineer does not find packages related to the problem to be solved, then he needs to warn the Analyst so he can retrieve the necessary information to create experience packages that can be used in current and future projects (see Figure 3). For the retrieval of information Visual Query Interface can be used (VQI)[7]. This tool allows the support engineer to graphically visualize the content of the experience base, making it easier to carry the search out.

A critical role is that of the *Analyst* who is in charge of analyzing the processes used during the software development, as well as of detecting the lessons learned during the implementation of the project. He should be able to decide which of the new lessons must be stored for reuse in future projects. To evaluate the knowledge gained or learnt in each project, the Postmorten Analysis (PMA) [6] can be used. To do this, the Experience Factory Analyst must communicate with the projects Analysts and project programmers with the purpose of deciding the lessons learned from each project. The interaction between the analyst and the knowledge disseminator allows a greater profit of information and of organizational learning.

The last role proposed is that of the *Experience Packer.* This is the person in charge of giving an appropriate electronic format to the experiences obtained during the development of projects so that it can be stored in a database (repository) for retrieval. The Packer receives information considered by the analyst as being appropriate for storage in experience packages. He is also responsible for updating experience packages so that information does not become obsolete or inconsistent.

Other proposals which define different roles related to knowledge management exist [14],[13],[1]. However, they do not describe in detail the tasks that each role should play and they do not indicate either the relationship between the different roles and the software organization.

Conclusions

As software maintenance is an knowledge-intensive task, in this paper we propose to use an EF to manage this knowledge with the goal of sharing and reusing it. Moreover, the

paper explains how the information related to maintenance has been formalized by using an ontology and represented in a database using REFSENO tables. An example of how to reuse the information stored in the EF has also given. In addition, we have tried to help companies' managers to implant an EF since we have explained the roles that an EF should have and how these interact.❖

References

1. Basili, V, Caldiera, G, and Cantone, G, "A Reference Architecture for the Component Factory". *ACM Transactions on Software Engineering and Methodology* 1,1992.1 pp. 53-80.
2. Basili, V, Lindvall, M, Costa, P, "Implementing the Experience Factory Concepts as a Set of Experience Bases". Proceedings of 13th International Conference on Software Engineering and Knowledge Engineering, 2001 pp. 102-109.
3. Basili, VR, Caldiera, G, McGarry, R, Pajerski, R, Page, G and Waligora, S, "The Software Engineering Laboratory – An Operational Software Experience Factory". Proceedings International Conference Software Engineering, 1992. May pp. 370-381.
4. Basili, VR, Costa, P, Lindvall, M, de Mendoca Neto, MG, Seaman, C, Tesoriero, R, and Zelkowitz, MV, "An Experience Management System for a Software Engineering Research Organization". The 26th Annual NASA Goddard Software Engineering Workshop, 2001.
5. Basili, VR, Daskalantonakis, M, and Yacobellis, R, "Technology Transfer at Motorola". *IEEE Software*, 1994. March pp. 70-76.
6. Birk, A, Dingsoyr, T, Stalhane, T, "Postmortem: Never Leave a Project Without It". *IEEE Software*, 2002.19(3) pp. 43-45.
7. Brassier, P, "Knowledge Management at a Software Engineering Company – An Experience Report". Workshop on Learning Software Organizations (LSO '99), 1999pp. 163-170.
8. Card, DN, Glass, RL, *Measuring Software Design Quality*. 1990, Englewood Cliffs, USA. Prentice Hall.
9. Farquhar, A, Fikes, R, Rice, J, "The Ontolingua Server: A Tool for Collaborative Ontology Construction". *International Journal of Human-Computer Studies*, 1997. 46pp. 707-728.
10. Gupta, A, Govindarajan, V, "Knowledge Flows within Multinational Corporations". *Strategic Management Journal*, 2000. 21(4) pp. 473-449.
11. Hikita, T, Matsumoto, MJ, "Business Process Modeling Based on the Ontology and First-Order Logic". Proceedings Third International Conference on Enterprise Information Systems (ICEIS' 2001), 2001. 7-10 July pp. *111-123*.
12. Houdek, F, Schneider, K, and Wieser E, *Establishing Experience Factories at Daimler-Benz: An Experience Report*. Proceedings 20th International Conference on Software Engineering, 1998. April pp. 443-447.
13. Huang, K-T, Lee, WY, Wang YR, *Quality Information and Knowledge Management.*, ed. Prentice-Hall. 1998.

14. Jorgensen, M, Sjorgerg, DIK, Conradi, R, "Reuse of Software Development Experience at Telenor Telecom Software". European Software. Process Improvement Conference (EuroSPI '98), 1998.

15. Lesser, E, Storck, J, "Communities of Practice and Organizational Performance". *IBM Systems Journal*. Vol. 40. 2001, Germany, pp. 831-841.

18. Niknafs, A, Shiri, M, Javidi, M. "A Case-Based Reasoning Approach in E-Tourism: Tour Itinery Planning". In Proc. 4th International Workshop on Theory and Applications of Knowledge Management. During DEXA'03. 2003. Prague, Czech Republic, pp. 818-822.

19. Oliveira, KM, Anquetil, N, Dias, MG, Ramal, M, Meneses, R, "Knowledge for Software Maintenance". Fifteenth International Conference on Software Engineering and Knowledge Engineering (SEKE '03), 2003. 1-3 July pp. 61-68.

20. Polo, M, Piattini, M, Ruiz F, Calero, C, "Roles in the Maintenance Process". *ACM Software Engineering Notes*, 1999. 24(4) pp. 84-86.

21. Rodriguez, OM, Martinez, AI, Vizcaino, A, Favela, J, Piattini, M, "Identifying Knowledge Flows in Communities of Practice". Sent to the *Encyclopedia of Communities of Practice in Information and Knowledge Management*, 2004.

22. Ruiz, F, Vizcaino, A, Piattini, M, Garcia, F, "An Ontology for the Management of Software Maintenance Projects". *International Journal on Software Engineering and Knowledge Engineering*, 2004. 14(3). June pp. 323-346.

23. Schneider, K, Von Hunnius, J, and Basili, VR, "Experience in Implementing a Learning Software Organization". *IEEE Software*, 2002. June pp. 46-49.

24. Seaman, C, "The Information Gathering Strategies of Software Mainteners". Proceedings of the International Conference on Software Maintenance, 2002 pp.141-149.

25. Seaman, C, Mendonca, M, Basili, V, Kim, Yong-Mi, "User Interface Evaluation and Empirically-Based Evolution of a Prototype Experience Management Tool". *IEEE Transactions on Software Engineering*, 2003. 29(9). September pp. 838-850.

26. Staab, S, Schnurr, H-P, Sure, Y, "Knowledge Process and Ontologies". *IEEE Intelligent Systems*, 2001.16(1).

27. Tautz, CG, Von Wangenheim, "REFSENO: A Representation Formalism for Software Engineering Ontologies", in FraunhoferIESE-Report. 1998.

28. Walz, DB, Elam, J J, Curtis, B, "Inside a Software Design Team: Knowledge Acquisition, Sharing, and Integration". Communications of the ACM, 1993. 36(10) pp.63-67.

29. Wenger, E, *Communities of Practice: Learning Meaning, and Identity*. 1998, Cambridge UK Cambridge University Press.

Reference # 17M-2005-06-10-01