PROCEEDINGS OF THE

Workshop on Learning Software Organizations

June 16 , 1999
Kaiserslautern, Germany

http://www.iese.fhg.de/SEKE99/documents/workshop.html

Edited by
Frank Bomarius, Fraunhofer IESE, Kaiserslautern, Germany

Committee

Workshop Chair

Frank Bomarius
Fraunhofer Institute for Experimental Software Engineering
Sauerwiesen 6
D-67661 Kaiserslautern
Germany

e-mail: bomarius@iese.fhg.de
fax: +49-6301-707-203
phone: +49-6301-707-121

Program Committee

Robert J. Aarts, Nokia Telecommunications, Finland
Andreas Abecker, DFKI, Germany
Klaus-Dieter Althoff, Fraunhofer IESE, Germany
Brigitte Bartsch-Spörl, BSR Consulting, Germany
Frank Bomarius, Fraunhofer IESE, Germany
Barbara Dellen, University of Calgary, Canada
Rose Dieng, INRIA, France
Scott Henninger, University of Nebraska-Lincoln, USA
Knut Hinkelmann, Insiders GmbH, Germany
Matthias Jarke, Aachen University of Technology, Germany
Ralf Klamma, Aachen University of Technology, Germany
Frank Maurer, University of Calgary, Canada
Werner Mellis, University of Cologne, Germany
Wolfgang Müller, Fraunhofer IESE, Germany
Ulrich Reimer, Swiss Life, Switzerland
Kurt Schneider, DaimlerChrysler, Germany
Hideo Shimazu, NEC, Japan
Rudi Studer, University of Karlsruhe, Germany

Table of Contents

Workshop Learning Software Organizations

– Program –

13:30 Welcome, Introduction and Overview

Frank Bomarius (Fraunhofer IESE)

13:40 Invited Talk

❒ Transfer of Experience - Issues Beyond Tool Building

Brigitte Bartsch-Spörl (BSR Consulting)

14:15 Part 1: Organizational Memories (3 x 20 min)

❒ Ontologies for Knowledge Retrieval in Organizational Memories

Minghong Liao, Andreas Abecker, Ansgar Bernardi, Knut Hinkelmann, Michael Sintek (DFKI)

The paper addresses knowledge retrieval in OMs based on ontologies. It identifies three different kinds of ontologies as main contributors to a vocabulary for comprehensive information meta modeling. The paper builds a bridge between more formal logic based approaches on the one side and more pragmatic approaches like CBR on the other side.

❒ Improving Organizational Memories through User Feedback

Klaus-Dieter Althoff, Markus Nick, Carsten Tautz (Fraunhofer IESE)

The paper presents the goal-oriented method OMI for improving an OM incrementally from the user's point of view. OMI consists of a general usage model, a set of indicators for improvement potential represented as protocol cases, and a cause-effect model. At each step of the general usage model, protocol cases are recorded to pinpoint improvement potential for increasing the perceived usefulness. If improvement potential is identified based on the interpretation of the protocol cases (by humans and/or by the software system), the user is asked for specific improvement suggestions.

❒ Developing a Tailored Reuse Repository Structure – Experience and first Results

Raimund Feldmann (University of Kaiserslautern)

Reuse of patterns or code components is demanded in software development. This paper is concerned with the storage of reusable artifacts in a repository, the repository's structure, and a reuse process, which is adaptable to the needs of the organization. Experience from building and evolving over time of an Internet-based reuse repository are presented.

Break

15:30 Part 2: Industrial Experiences in LSOs (3 x 20 min)

❒ Transferring Experience – A practical Approach and its Application on Software Inspections

Frank Houdek (DaimlerChrysler), Christian Bunse (Fraunhofer IESE)

A process for systematic experience transfer covering the activities of acquisition, documentation and evolution, and reuse is presented. The practical use of the proposed process is demonstrated using results from two real projects dealing with experience transfer in software inspections. It is also described how experience can be packaged (documented) in order to both transfer and to improve the technique packaged.

❒ Talk to Paula and Peter – They are Experienced

Conny Johannson, Patrik Hall, Michael Coquard (Ericsson)

When problems occur that cannot be solved experts are sought after. This paper describes how the role of an 'Experience Broker' was defined and implemented in order to facilitate human networking. It is described how the required culture was created and experiences and results from the pilot project performed in 1998 are presented.

❒ Knowledge Management at a Software Engineering Company – An Experience Report

Peter Brössler (sd&m)

This paper describes the introduction of a Web-based knowledge management system at sd&m that is designed according to the 'cafeteria principle'. This principle was applied when the company was small and people did socialize a lot. The presented system stimulates such behavior by organizational and technical means. The history of the system as well as experiences from one year of operational use will be presented.

Break

16:45 Part 3: Process-centered Approaches to LSO (3 x 20 min)

❒ Process Support for Virtual Software Organizations

Frank Maurer, Barbara Dellen (University of Calgary), Harald Holz (University of Kaiserslautern)

This paper describes the MILOS process modeling language and shows how the MILOS system – a flexible, Internet-based process enactment engine to be used to support software process improvement – can be used for process support and explains why the process-centered approach to KM supports a LSO.

❒ Using Software Process to Support Learning Software Organizations

Scott Henninger (University of Nebraska-Lincoln)

This paper investigates how software process can be used as a driving force for collecting and disseminating software development knowledge. An organizational learning meta-process is combined with a rule-based software process engine to create a tool that provides decision support for tailoring software processes to the needs of individual development efforts.

❒ A Knowledge Management Lifecycle for Experience Packages on Software Engineering Technologies

Andreas Birk, Felix Kröschel (Fraunhofer IESE)

This paper proposes a KM lifecycle for experience on SW engineering technologies and how it is packaged for reuse during the planning of software projects and improvement programs. The lifecycle model is substantiated by a tool implementation and lessons learned from an industrial trial application.

until 18:00 Discussion and Conclusion

Invited Talk

Transfer of Experience - Issues Beyond Tool Building

Brigitte Bartsch-Spörl (BSR Consulting)

Transfer of Experience -

Issues Beyond Tool Building

Brigitte Bartsch-Spoerl

BSR Consulting, Munich

Overview

- **Motivation**
- **Some reasons why neither knowledge sharing nor re-use of experience happens without special attention and promotion**
- **Some non-tool-oriented approaches that can help to improve the re-use of existing knowledge and experience**
- **Why it needs holistic approaches to establish a re-use culture**
- **Conclusion: what are the consequences for tool building efforts?**

Motivation

B S R CONSULTING

- **Nowadays big firms are spending money for knowledge management projects because**
  - they have understood that their employees' experience is THE prerequisite for being successful in their business and
  - they believe that the time has come to mould this experience into some form of corporate memory
- **There are only two small problems with this approach:**
  - experience is hard to capture and to represent in a form that is understandable for others and
  - even if you managed to solve the first problem your colleagues may be reluctant to re-use other people's experience...

Motivation

B S R CONSULTING

- **In the following I will NOT focus on the first small problem - but on the second**
- **I will do this by sharing a part of my case knowledge and what I think that can be learned from these cases with you -**
  hoping not all of you will be reluctant to re-use my experiences...

Why are people sometimes not eager to re-use other people's experience?

B S R CONSULTING

- **they don't fully understand it, e.g. because**
  - the description is not sufficient for them (too little context, no support how to map and to adapt the suggestions, not enough time available to go into the details)
- **example: the LUPOS project**
  - part of the team invested time for finding which function modules to re-use, part of the team did not
- **lesson learnt:**
  - the management has to set the right priorities: re-use brings productivity and quality - but not in the first week of the implementation phase...

Why are people sometimes not eager to re-use other people's experience?

B S R CONSULTING

- **it is not creative and you don't become famous for it**
  - especially software developers have a strong tendency to do "their own thing" and do it different from existing solutions (the so-called not-invented-here-syndrome)
- **example: the application within the FABEL project**
  - the project dealt with the case-based re-use of architectural design components and some other architects said: really interesting, but not my style - which means they would never use such a system
- **lesson learnt:**
  - the more time a person had to develop his/her own style the more difficult it is to convince him/her of the re-use idea

Why are people sometimes not eager to re-use other people's experience?

- **it maybe not safe or not perfectly suitable**
  - this means that re-use needs the qualification to judge whether a suggested solution fits or has to modified or should not be taken into account at all
- **example: KAR, my first CBR project for Volkswagen**
  - the users said: we want to know everything about similar situations - but then we decide on our own what has to be done...
- **lesson learnt:**
  - in acceptance critical situations deliver suggestions and let the users decide on their own

The importance of champions

- **the best thing that can happen to a re-use project is a real champion for this idea within the firm**
  - a champion is someone who feels the need to improve the situation and who is convinced of the solution approach and that it is HIS job to field this application right now
- **example: Mr. Busch & SIMATIC Knowledge Manager**
  - such persons are not easy as project partners - but as soon as the application is successfully running a champion will care about its further development and long term survival
- **lesson learnt:**
  - don't start a project without having identified a potential champion - and do everything to make him/her play this role

The importance of showing benefits

- **a system in productive use has to justify the investment made for its development**
  - it is important that the users know that using such a system is a good way of saving their own time, their department's budget and their organisation's money ("re-use culture")
- **example: Siemens says that the SIMATIC KM saves about 0.5 Mio. DM this year**
- **lesson learnt:**
  - do marketing for such a system and try to reach as many users as possible via all possible means (WWW etc.)

Why holistic approaches are necessary

- **convincing the management is a perfect first step but not enough**
- **a champion without budget can't move very much**
- **a project that does not deliver an acceptable solution brings the re-use idea into disrepute for years**
- **a system that is not used brings no benefits**
- **lesson learnt:**
  if you want your system to become a success you cannot avoid dealing with all these non-technical issues...

What does this mean for tool building efforts?

- a tool alone will not change the situation
- you need the management to establish the right goals, priorities and rewards
- you need champions to prove that it works - even in your company and working group
- you have to show clear benefits after a reasonable investment of time and effort
- after the first cycle you have to establish the next cycles as a constant improvement process that deserves less and less special attention because it has become an established working practice

Questions and Discussion

Part 1:

Organizational Memories

Ontologies for Knowledge Retrieval in Organizational Memories

Minghong Liao, Andreas Abecker, Ansgar Bernardi, Knut Hinkelmann, and Michael Sintek

German Research Center for Artificial Intelligence–DFKI GmbH
P.O.Box 2080, D-67608 Kaiserslautern, Germany
Phone:+49 631 205 3467, Fax:+49 631 205 3210
{liao,aabecker,bernardi,hinkelma,sintek}@dfki.uni-kl.de

Abstract. An Organizational Memory (OM) captures, stores and disseminates valuable corporate knowledge and is thus a central prerequisite for enterprise knowledge management. For structuring, accessing, and maintaining large amounts of heterogeneous information, appropriate meta-level descriptions are needed which specify the structure, content, and potential usage of the object-level knowledge. Such meta-level descriptions are provided for data in the form of data models, for formal knowledge as ontologies, and for informal documents as document descriptors. In this paper, we sketch an ontology-based approach for comprehensive meta-modeling and retrieval of heterogeneous data, formal knowledge, and documents. We identify information ontology, domain ontology, and enterprise ontology as main contributors to a vocabulary for comprehensive information meta modeling. We elaborate a bit on the underlying representation formalism, sketch a sample scenario, and present ontology-based heuristic retrieval in the organizational memory.

Keywords: knowledge representation and retrieval, web-based knowledge management

1 Introduction

The systematic management of knowledge has been recognized as a necessity to enhance a company's survival and success in the global market place. To be effective, organizational knowledge management has to improve the capitalization on existing knowledge assets and to facilitate the creation of new knowledge. An Organizational Memory (or Corporate Memory, OM) can be characterized as a comprehensive computer system which captures a company's accumulated know-how and other knowledge assets, and makes them available to enhance the efficiency and effectiveness of knowledge-intensive work processes [Kühn and Abecker, 1997].

A couple of years before the big Knowledge Management hype in the business and management sciences, a very similar concept had already been introduced in the Software Engineering community under the name "Experience Factory" (EF) [Basili *et al.*, 1994]. Recently, [Althoff *et al.*, 1998] showed how the idea of an experience factory can be supported by the use of case-based reasoning (CBR) technology for storage and reuse of documents, designs, code, and other artifacts in the Learning Software Organisation.

A main design issue for an OM (and, consequently, also for the EF) are the respective roles of formal knowledge compared to semi-structured or non-formal documents. Many application-oriented authors agree on the fact that semi-structured and non-formal documents are playing a predominant role in a company's knowledge management (see, e.g., [Choo, 1995,Hartley *et al.*, 1997]). [Abecker *et al.*, 1998b]

propose the OM as sort of a "meta information system" providing a uniform access to a diversity of knowledge and information sources of different degree of formality. Since formalization is costly, error-prone, and requires extensive maintenance efforts later on, they propose to use formalized knowledge mainly for coupling task and retrieval, and for supporting precise-content retrieval to the OM.

Quite a similar point of view can be found in Figure 1 describing Richter's view on the CBR approach (taken from [Althoff *et al.*, 1998]). There, only very stable (vocabulary), useful (similarity measure), or worthful (solution transformation) knowledge is codified into formal representations, here referred to as "compiled knowledge".

The remainder is left in the cases.

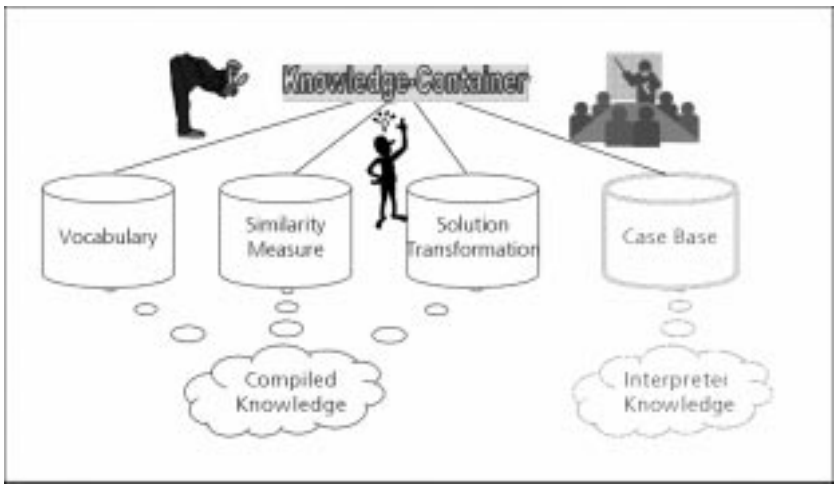

Fig. 1. The knowledge container view on CBR systems (taken from [Althoff *et al.*, 1998], adapted from [Richter, 1995])

Coming from the knowledge-based system point of view, [Benjamins *et al.*, 1998] distinguish two widespread approaches to building knowledge management systems. *Vertical systems* are task-specific performance support systems. They can provide high value in particular business situations because they incorporate much application specific knowledge. Naturally, their usage is restricted to a narrow application environment. *Horizontal systems*, on the other hand, are general frameworks for providing useful corporate knowledge in a wide area of application situations. In practice, this approach essentially amounts to more or less intelligent document management and information retrieval systems.

In their own approach, [Benjamins *et al.*, 1998] propose formal ontologies to allow for comfortable access and knowledge intensive usage of data and information embedded in HTML pages annotated with ontological information. Numerous other approaches propose formal ontologies to ease finding of and access to data and semi-structured information in HTML pages and databases [Luke *et al.*, 1997].

If one understands an information retrieval process as a similarity assessment between query situation and document description, both the CBR and the KBS point of view come together (except for the solution transformation, which is often neglected in practical applications). Further, if one fills the "knowledge containers" of the CBR approach with the heterogeneous information sources available in an enterprise, understands the case retrieval as a logical inference process on the basis

of ontologies and meta data (as at least roughly proposed already by [Kamp, 1996]), and equips the resulting system not only with one similarity measure, but with a library of application programs (consisting of vocabulary, similarity measure, solution transformation), the result may be one step towards a system which encompasses the depth of system services provided by vertical systems and the breadth of usage scenarios applicable to horizontal systems.

Although we will not further elaborate the analogue to the CBR point of view, this is the ultimate aim of the work described here. Essentially, we ask what ontologies are required to have appropriate "containers" and to define useful "similarity measures", how to represent such ontologies, and how to use them.

2 Overall Organizational Memory Design

The organizational memory of the KnowMore project [Abecker *et al.*, 1998a] is an enterprise-internal application-independent information and assistant system. It stores data, information, and knowledge from different sources of an enterprise. They are represented in various forms, such as databases, documents, and formal knowledge-bases. It will be permanently extended to keep it up to date and accessible enterprise-wide through an appropriate network infrastructure. A three-layered model as sketched in Figure 2 [Abecker *et al.*, 1998b].

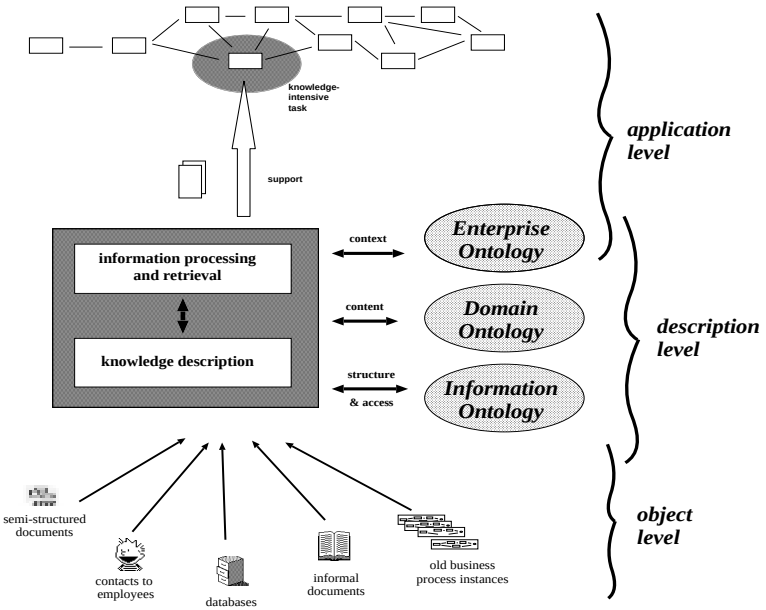

Fig. 2. The Organizational Memory Model

The *object level* comprises manifold information and knowledge sources, ranging from machine-readable formal representations to human-readable informal representations. Crucial parts of corporate knowledge to be processed by the computer must be formalized, whereas other parts that need only be understood by humans might be left informal. The decision whether to formalize or not rests on cost-benefit analyses, stability of knowledge, and the question whether some portion of knowledge can reasonably be formalized at all.

The *knowledge description level* enables a uniform, intelligent access to the diversity of object-level sources. Because legacy information systems must be incorporated without modification, we propose a separate, knowledge-rich information-modeling level, the details of which will be the main focus of this paper.

The OM's *application level* links the information model and the concrete application situation. When a knowledge worker recognizes an information need within the actual flow of work, a query to the OM must be derived. This query is instantiated and constrained as specifically as possible on the basis of the actual work context. In the opposite way, the OM can also store new information created within a given working situation in a contextually enriched form such that subsequent retrieval processes might compare the query situation with the creation situation for estimating context-specific relevance. As one of many possibilities for realizing the application level, we include conventional business-process models and workflow-management systems. Doing so lets us rely on a body of well-understood knowledge already formalized in enterprises and used to guide and support work processes.

3 Construction of the Ontologies

3.1 Three basic ontologies

Every information and knowledge item is described by a number of attributes representing the information metamodel, the information content, and the creation and application context. The concepts necessary for these descriptions form the fundamental ontologies for this modeling. Based on their specific roles in the context of an OM it is useful to distinguish three different ontologies (see Figure 3).

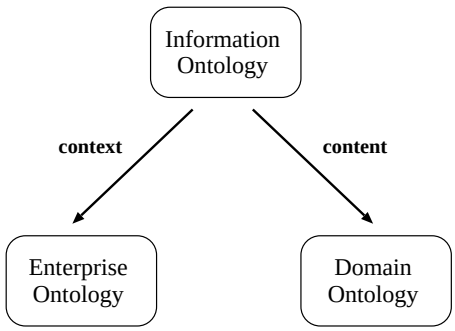

Fig. 3. Three dimensions of knowledge description.

The *information ontology* describes the vocabulary of the information metamodel, which characterizes the different kinds of information sources with their respective structure, access, and format properties. The concepts in the information ontology are stable and domain independent: a book has author, title, etc. as meta data, consists of chapters, and so on. This is always the case regardless of the concrete use of this information source. Specific instances of this ontology, i.e., descriptions of particular information sources, need to refer to concepts from the enterprise and the domain ontologies, respectively.

The *enterprise ontology* is used to describe information context, which is expressed as organizational structure and process models. Both the context of the creation of some information element and the context of its intended use are important contributions when evaluating the relevance of some information element with respect

to a particular task at hand. Thus this context information plays a crucial role within the knowledge handling in an organizational memory. The concepts in the enterprise ontology are expected to be rather independent of an actual company; various projects have provided standard enterprise ontologies which are applicable for most enterprises. However, they are of a different league than the information ontology concepts, thus the distinction between the two ontologies.

The *domain ontology* is used to model the content of the information sources. Typically, the concepts in this ontology are highly specific for a particular application. Thus it has to be taken into account that for the realization of a particular OM that it might be necessary to construct the domain ontology from scratch. However, there have been substantial efforts in the Knowledge Sharing community for providing reusable, task-independent real-world domain ontologies, e.g., in the areas of chemistry [van der Vet and Mars, 1993] or materials science [van der Vet *et al.*, 1995].

In order to emphasize the engineering aspect of ontology research, it would be an interesting exercise to find out how easy it is to (i) take such an "off-the-shelf" ontology which describes the domain of activity of a big company, e.g., in the chemistry sector, (ii) combine it with a preexisting enterprise and information ontology, (iii) put the pieces together, adapt them to the specific needs of the company and assess how much effort it is to configure such an information system from standard ontology modules. Our identification of the basic ontological dimensions information structure and meta data, static and dynamic enterprise context, and application domain should ease this enterprise a bit.

To repeat, a concrete information element is described via concepts taken from the information ontology, where the various attributes are filled with concepts from the enterprise and domain ontology, respectively. This interplay is illustrated in Figure 4 where the various concept types and their interrelations as used in the KnowMore OM are shown.

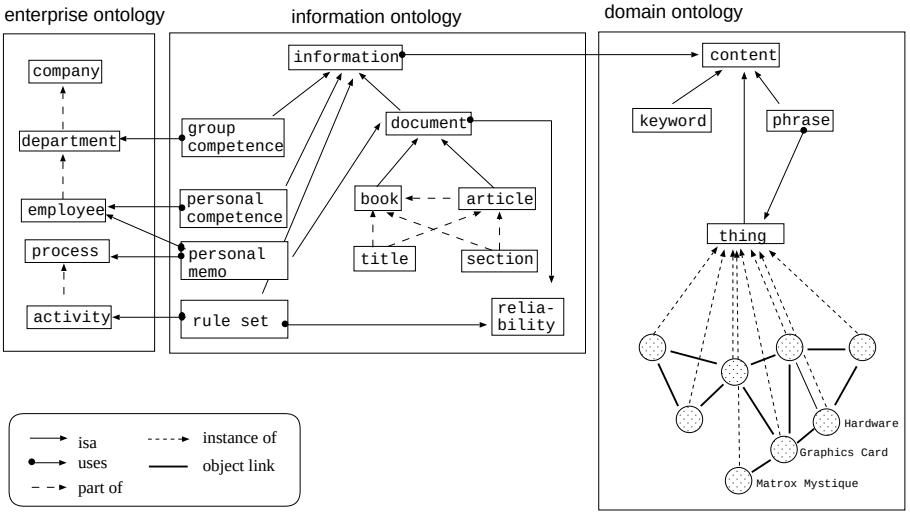

Fig. 4. Sketch of sample ontologies.

3.2 Ontology representation

Although since the very first AI related work on ontologies, there have also been representation languages (of which Ontolingua is the most prominent one

[Gruber, 1993]), most researchers concerned with ontology-based retrieval nevertheless used their own representation formats, e.g., Frame-Logic for the sake of powerful inferences [Fensel *et al.*, 1998], SHOE for the sake of efficient reasoning [Luke *et al.*, 1997], or CKML (the Conceptual Knowledge Markup Language) for the sake of adequate modeling primitives and compliance with de facto standards for the Web and document description [Kent and Neuss, 1996]. It might also be noticed that most of these languages were not originally built for knowledge description or Information Retrieval, but employed in this area. This may indicate that the language debate is far from being finished.

In the following, we will spent few words on the representation requirements we found in our experiments and on our intermediary result, an object-centered relational algebra (OCRA) for knowledge description.

As already the rough sketch of our example (Figure 4) indicates we need some *classification* and *aggregation* (documents consist of parts) mechanism in the information ontology. Further, we need an *instantiation* mechanism to model concrete instances of the respective classes in the concrete information meta models. The same representational means are sufficient for modeling the enterprise ontology. Things get complicated coming to the domain ontology. As, among others, [Lenz, 1998] points out, the weakness of simple keyword-based IR methods is its lack of exploitation of knowledge about domain structures and relationships. Basically, the more domain knowledge I have the more sophisticated retrieval I can get. In the consequence, the aim of modeling the domain of discourse in as much detail as possible leads to the aim of having a maximally powerful, general-purpose knowledge representation language for domain modeling. Consider, for instance, a technical application domain. In the ESB[1] project [Bernardi *et al.*, 1998], we built a sophisticated domain model for intelligent management and retrieval of natural-language based records of maintenance experiences with a highly-complex machine. There, it was not only necessary to have the classification and the aggregation (part-of) hierarchy of the machine in quest, but also very useful for retrieval and analysis of maintenance records to model additionally the hydraulic, electric, and functional connections and relationships in this machine. However, the aim of a very comfortable and expressive knowledge representation language is in conflict with the general requirement of having efficient reasoning mechanisms which can handle huge amounts of documents in the company's archives indexed with respect to complex, large domain ontologies. Hence, we would like to have a simple, efficient core language which could be extended if needed by additional representation primitives provided that there are efficient inferences which can be delivered together with these primitives.

Another problem using off-the-shelf ontology and KR languages for knowledge-based indexing in the OM has extensively be discussed by [Welty, 1998] who identified the semantical and representational issues when talking about subject taxonomies for content description of documents: a subject topic is usually embedded into a hierarchical structure (like a *class* in an object-oriented formalism) but is used as an attribute value (i.e., like an *instance* in an OO formalism). A related problem has been mentioned by a few authors, recently, but was also worked around, up to now (cp. [Schmiedel and Volle, 1996,van der Vet and Mars, 1996]): the higher-order aspects coming into play if one wants to give complex proposition as content descriptions (i.e., technically: assertions as attribute values).

Putting all pieces together we came to a preliminary solution as indicated in the example (Figure 4) and in the language description in Figure 5 (for more details, see [Abecker *et al.*, 1998a]): our main modeling approach is a conventional

[1] ESB = "Elektronisches Störungsbuch" is the German acronym for "Electronic Fault Recording".

object-centered formalism with only classification (subclass-superclass relationship) built-in as first-order means of representation, extended by two important features: (i) set-valued attributes, and (ii) annotated links. I.e., each attribute-value assertion can be associated with one or more annotation objects taken from annotation classes which can also be used to build up an annotation hierarchy. For document representation, content topics are represented as instances of a `concept` class which can be linked by annotated relations. Then, annotation objects can specify several relationships between concepts of the domain ontology, e.g., classification within the topic taxonomy, aggregation of topics, or other, application-specific relationships. In Figure 4, these annotated links are indicated by "object links". Since the evaluation of retrieval conditions formulated with respected to annotations can directly be compiled into efficient database operations of the underlying DBMS, this combines efficient processing and good expressiveness. Furthermore, for special annotation classes, the database procedures can be extended by efficient special-purpose inferences.

Ontologies are represented in an object-oriented manner. The declaration of a *class* looks like this:

$$\text{class}(attribute_1\text{: } type_1, \ldots, attribute_n\text{: } type_n)$$

If a class class1 *extends* a class class2, the following format is used:

$$\text{class1: class2}(attribute_{n+1}\text{: } type_{n+1}, \ldots, attribute_{n+m}\text{: } type_{n+m})$$

Attributes are used to model semantic nets. In order to allow edges in these nets to be labeled with objects, a new complex type constructor is introduced, namely *annotations*. Annotated attributes are declared as follows:

attribute: class1/class2

or (for set-valued attributes)

attribute: {class1}/class2

In both cases, class2 is the annotation class.
An *object* of a class is represented as follows:

$$\text{class}(attribute_1{=}value_1/ann_1, \ldots) \text{ or}$$
$$\text{name: class}(attribute_1{=}value_1/ann_1, \ldots)$$

or (for set-valued attributes)

$$\text{class}(attribute_1{=}\{value_1/ann_1, \ldots\}, \ldots)$$

Fig. 5. A short overview of the object-centered representation formalism.

3.3 Representing the example

Here, we show in some detail how the example introduced in Figure 4 can be implemented with the OCRA formalism.

Information Ontology: Any piece of information has a location (given as an URL) and a content, which is given as a set of content descriptions. Information sources may be documents, data, and rules, as well as references to personal and group competences. Any association of some piece of information with a content de-

scription may be annotated with a strength object, which for personal competences is a capability specification (Figure 6).

```
strength: ann()                  // ann is the annotation super class
                                 // strength is an abstract super class
capability: strength(value: string) // specific subclass of strength

information(                     // information source
  name: string,
  url: string,
  content: {content} / strength // set-valued attribute for content
)                                // description, each content
                                 // identifierannotated with a strength
                                 // specification

personalCompetence: information(
  employee: employee             // defined in the enterprise ontology
)

groupCompetence: information(
  unit: organizationalUnit       // defined in the enterprise ontology
)
```

Fig. 6. A part of the sample information ontology.

Enterprise Ontology: Here, we consider the static organizational structure of a company is modeled, consisting of departments, employees, and their respective roles. In KnowMore, the enterprise ontology is used mainly for two reasons: the employees are actors in business processes, and they have competences and are therefore modeled as information sources in the information ontology (Figure 7).

```
company(
  name:string,
  address: string,
  departments: {department}
)

role: ann(name: string)

department(
  name: string,
  employees: {employee} / role
)
```

```
employee(
   person: person,
   phone: string,
   eMail: string
)

person(
   lastName: string,
   firstName: string,
   dateOfBirth: string
 )
```

Fig. 7. A part of the sample enterprise ontology.

Content Descriptions: Content descriptions as used in the information ontology may either be unstructured, like keywords or concepts of the domain ontology, or they may be complex, structured terms, like a sentence of the form subject-predicate-object (Figure 8).

Domain Ontology: The domain ontology is a complex semantic net, all primitive concepts are instances of class concept where the interrelationships are modeled

```
content()      // abstract super class for information content description

keyword: content(    // the most primitive content identifier: keywords
  name: string       // are just strings
)

concept: content(    // currently used for the KnowMore conceptual retrieval:
  name:string,       // formal concepts have a name and, maybe,
  ld:{ld},           // a number of textual descriptions or representations
  links: {concept}/conceptLink  // all semantic relationships between formal
)                               // concepts are represented by annotated links

conceptLink: ann()   // a conceptLink relates two formal concepts
isco: conceptLink()  // the 'is-subconcept-of' annotation class
                     // marks links which establish the is-a
                     // hierarchy within the domain ontology

ld(                  // a linguistic description is an evidence
  lang: string,      // (representation) of a formal concept in a text;
  name: string,      // several linguistic descriptions may describe occurrence
  abbr: string       // of the same formal concept in different languages
)

relation: concept(        // a complex content description could
  relationship: concept,  // contain a statement
  objects: {concept}
)
```

Fig. 8. Content descriptions as a part of the information ontology.

with the links attribute and some annotation objects like `sub` which is an annotation instance of the `isco` (= is-subconcept-of) annotation class which describes the classification hierarchy between concepts for information content description (Figure 9).

4 Heuristic Retrieval in Organizational Memories

If our knowledge sources are modeled as described above, retrieving information from the OM essentially amounts to a "select" operation on the object-oriented database, performed with appropriate search conditions formulated with respect to (i) the meta data given in the information ontology (e.g., which information sources to consider, or how old information to retrieve), (ii) specific context information (here, sophisticated similarity measures can be employed for comparison of actual query situation and context factors of knowledge sources described in the OM), and (iii) the content searched for. Since at the implementation level, all topic concepts are instances linked via annotated relationships, and the OCRA provides specialized operators for querying with respect to annotations, the retrieval can efficiently be supported by the underlying database system.

Concerning object links, search conditions can be specified (a) with respect to annotation (sub-)classes, (b) with respect to specific annotation objects (e.g., find all links marked with a "strength=strong" annotation), or (c) giving specific attributes with values (i.e., an "ordinary" database selection condition).

If we search for any information source about the concept "database", we can look for "database" in the domain ontology, and then pick out all information

```
...

sub: isco()     // an annotation object instantiates the standard
                // subconcept link annotation class

sw: concept(
  name="software",
  ld={ld(lang="English", name="software")},
  links={cs/sub}    // cs is a subconcept of computer science
)

db: concept(
  name="database",
  ld={ld(lang="English", name="database"),
      ld(lang="German", name=" Datenbank")
     },
  links={sw/sub}         // db is a subconcept of sw
)

...
```

Fig. 9. A part of the domain ontology used for information content descriptions.

sources which have a content characterization containing the "database" concept. The more interesting case is when there is not any information source about "database." Then we should use the links in the domain ontology. Consider, for instance, Figure 10. As we know "deductive database" and "relational database" are subconcepts of database. If no information about "database" is available, material about "deductive database" or "relational database" may also be suitable. Further, if nothing can be found in the subLink of "database", we have to search the upLink of "database", i.e., software. For undirected edges (e.g., some not further specified "has-to-do-with" relationship in a complex domain) we can select any precedent or post node in the domain ontology graph randomly for further retrieval.

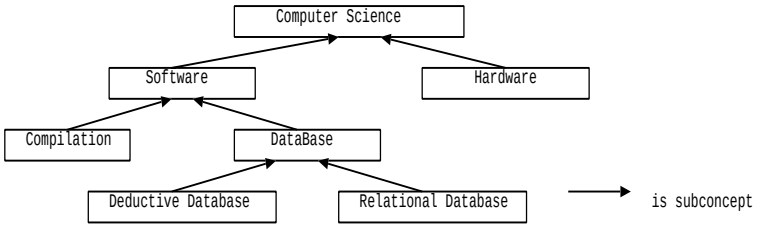

Fig. 10. A part of the domain ontology.

Besides such general search strategies, one can imagine that in each concrete application scenario there may be manifold sophisticated specific search heuristics. For giving to the user a means to specify such domain or application specific search heuristics, we introduced the notion of a *heuristics expression* which is a sequence of formulae of the following form:

$$f_1 \circ f_2 \circ \ldots \circ f_n$$

(denoting the functional composition of the f_i) with

$$f_i \equiv (\lambda)^\gamma$$

where λ is a link or an inverse link (written as $link^{-1}$) and γ is a "partial closure specification", i.e., one of the following path length specifications: n, $n..m$, $\geq n$, * (as abbreviation for ≥ 0), or + (as abbreviation for ≥ 1).

Such a formula takes as input a set of nodes of the directed graph under consideration and, for each node, follows the links specified in the formula in right-to-left order, in each step delivering an intermediary set of nodes as starting point for the next step. "Partial closure" means repeatedly following the same link type (in the case of $\gamma \equiv *$ generating the reflexive and transitive closure of the relation denoted by that link in the ontology). A heuristics formula makes sense if it delivers only information items as result. A sequence of formulae is evaluated in its sequential order with the semantics in mind that less trustworthy heuristics should be denoted last. Some sample retrieval heuristics can be expressed as follows:

1. $(content^{-1})^1$
   "First search for information sources directly linked to a search concept."
2. $(content^{-1})^1 \circ (isco^{-1})^+$
   "Then look for material about any subfield."
   For the sake of clarity, we have denoted two formulae here. An alternative formulation would have been: $(content^{-1})^1 \circ (isco^{-1})^*$
3. $(content^{-1})^1 \circ (isco)^1$
   "If result is empty after step 1 and 2, look for information concerning the direct superconcept of the topic in quest."

5 An Experimental System: CKBS

As a sub-module of the project KnowMore, we designed a competence query system called Competence Knowledge Base System (CKBS), which aims to provide a tool for querying the competence of the employees of an organization. In [Liao *et al.*, 1999] we elaborate in some more detail on the benefits of sophisticated domain models for the formulation of specific search heuristics.

The CKBS is designed as a client-server model, its architecture is shown in Fig. 11, the input and output interface are shown in Fig. 12.

In the input mask, the tool allows to formulate queries over competence fields, project memberships, or (directly) employee names. Complex queries can be composed using "AND", "OR", and "NOT".

The actual knowledge base with persons and their competence indices, as well as the ontological structure of competence fields, project membership, etc., are stored in a conventional relational database (RDB) which is coupled to the JAVA [JAVA, 1998] system code via JDBC [JDBC, 1998]. Details about how to efficiently store and access these object-oriented knowledge structures within the relational paradigm can be found in [Abecker *et al.*, 1998a]. The relational storage approach together with some additional schema information (denoted in the picture as DB-signature) allows to implement an object-centered relational algebra (OCRA, see [Abecker *et al.*, 1998a]) which provides an object-oriented view and access methods with special (weak) deductive capabilities for the underlying data. In detail, the OCRA directly implements the above introduced "partial closure" operator, an essential part of heuristics expressions, which allows to efficiently follow a predefined number of links between objects.

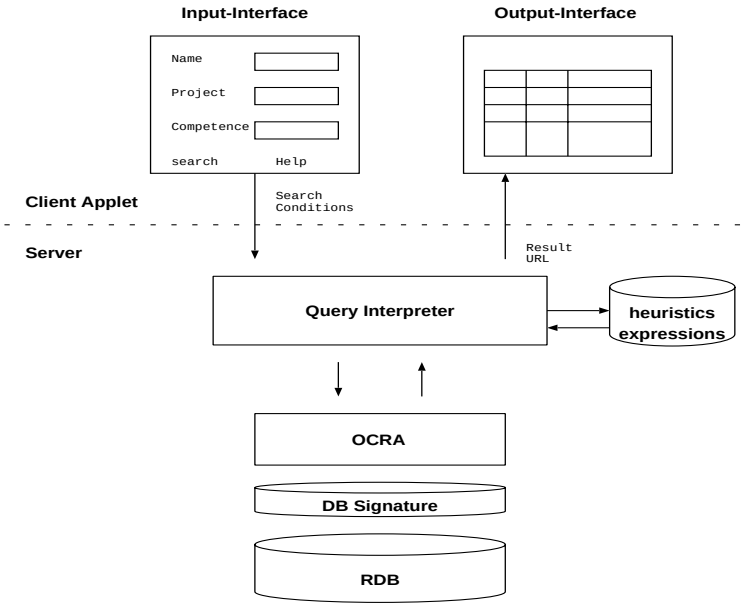

Fig. 11. The architecture of CKBS.

6 Conclusions

With this paper, we made some remarks to the discussion about ontologies for knowledge retrieval in the Organizational Memory. Although many people claim that ontologies would be a natural part of an OM, mostly they do not discuss which ontologies should serve for which purposes, how they are represented, and how they are built and maintained. We identified information source ontology, context, and application domain as the main ontological modeling dimensions to be considered when building practical "knowledge containers" in the sense of Richter. From the engineering point of view it makes sense to separately investigate all three ontological dimensions since they can be developed and reused separately. It is subject to further work to propose a reusable information ontology.

From the knowledge representation point of view, we identified some problems to be tackled, and presented aspects of our "intermediary solution", the object-centered relational algebra. Actually, the OCRA is a pragmatic trade-off between representational ease and efficiency, conceptually near to the Frame-Logic approach. But we will further work in this area. In contrast to most CBR approaches, we relied on a strongly logical basis for describing ontologies. This is not necessarily a contradiction to the CBR point of view, as Kamp showed.

However, it is still open and very interesting to further develop information retrieval inferences at the intersection of logical inference, theoretical foundations of utility and uncertainty processing (like the Dempster-Shafer-Approach, again, cp. [Richter, 1995]) and information theory [Barwise and Seligman, 1997]. In our current system, we employ logical inference only in a very rudimentary way, and merely make use of user-defined search heuristics as shown above. At first hand, this more imperative, navigation-oriented approach for heuristics specification seems to be promising for practical use.

If one wants to really put into use such systems as described here, knowledge acquisition and maintenance with respect to ontology construction as well as ontology use for document description get really crucial for success. Figure 13 shows a

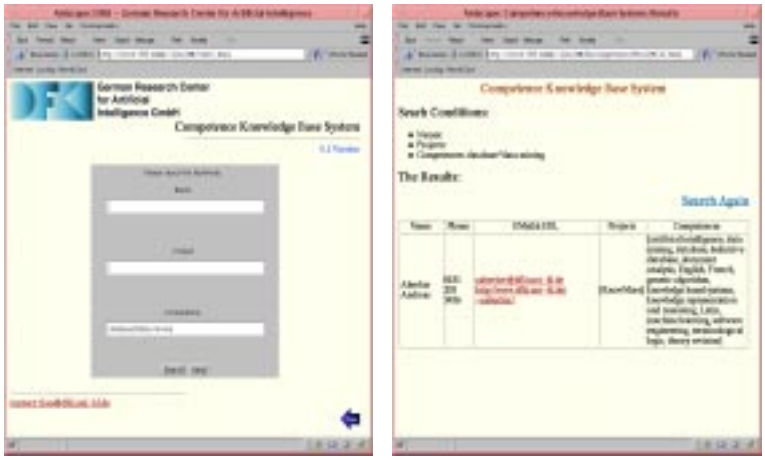

Fig. 12. The input and output interface of CKBS.

screenshot of our KnowMore ontology and knowledge description editor. Currently we investigate comfortable interfaces for end user and knowledge engineer, e.g., by incorporation of learning, automatic text categorization approaches.

References

[Abecker *et al.*, 1998a] A. Abecker, A. Bernardi, K. Hinkelmann, O. Kühn, and M. Sintek. Techniques for Organizational Memory Information Systems. DFKI Document D-98-02, DFKI GmbH, February 1998.

[Abecker *et al.*, 1998b] A. Abecker, A. Bernardi, K. Hinkelmann, O. Kühn, and M. Sintek. Toward a Technology for Organizational Memories. *IEEE Intelligent Systems and Their Applications*, 13(3), May/June 1998.

[Althoff *et al.*, 1998] K.-D. Althoff, F. Bomarius, and C. Tautz. Using case-based reasoning technology to build learning software organizations. In A. Abecker, St. Decker, N. Matta, F. Maurer, and U. Reimer, editors, *ECAI-98 Workshop on Building, Maintaining, and Using Organizational Memories (OM-98)*, August 1998. URL `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-14`.

[Barwise and Seligman, 1997] J. Barwise and J. Seligman. *Information Flow: The Logic of Distributed Systems*. Tracts in Theoretical Computer Science. Cambridge University Press, 1997. To appear.

[Basili *et al.*, 1994] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. *Experience Factory*. John Wiley & Sons, 1994.

[Benjamins *et al.*, 1998] V.R. Benjamins, D. Fensel, and A. Gómez Pérez. Knowledge management through ontologies. In U. Reimer, editor, *Proc. of the 2nd Int. Conference on Practical Aspects of Knowledge Management (PAKM98)*, October 1998. URL `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-13`.

[Bernardi *et al.*, 1998] Ansgar Bernardi, Knut Hinkelmann, and Michael Sintek. Model-based information systems for knowledge management. In *IT& Knows, Conference on Information Technology and Knowledge Systems, a part of the 15th IFIP World Computer Congress*, Vienna & Budapest, August 1998.

[Choo, 1995] C. W. Choo. Information Management for the Intelligent Organization: Roles and Implications for the Information professions. In *Proc. of the 1995 Digital Libraries Conference, Singapore*, March 1995.

[Fensel *et al.*, 1998] Dieter Fensel, Stefan Decker, Michael Erdmann, and Rudi Studer. Ontobroker: The very high idea. In *Proc. 11th Int. Florida AI Research Symposium (FLAIRS-98)*, May 1998.

[Gruber, 1993] T.R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2), 1993.

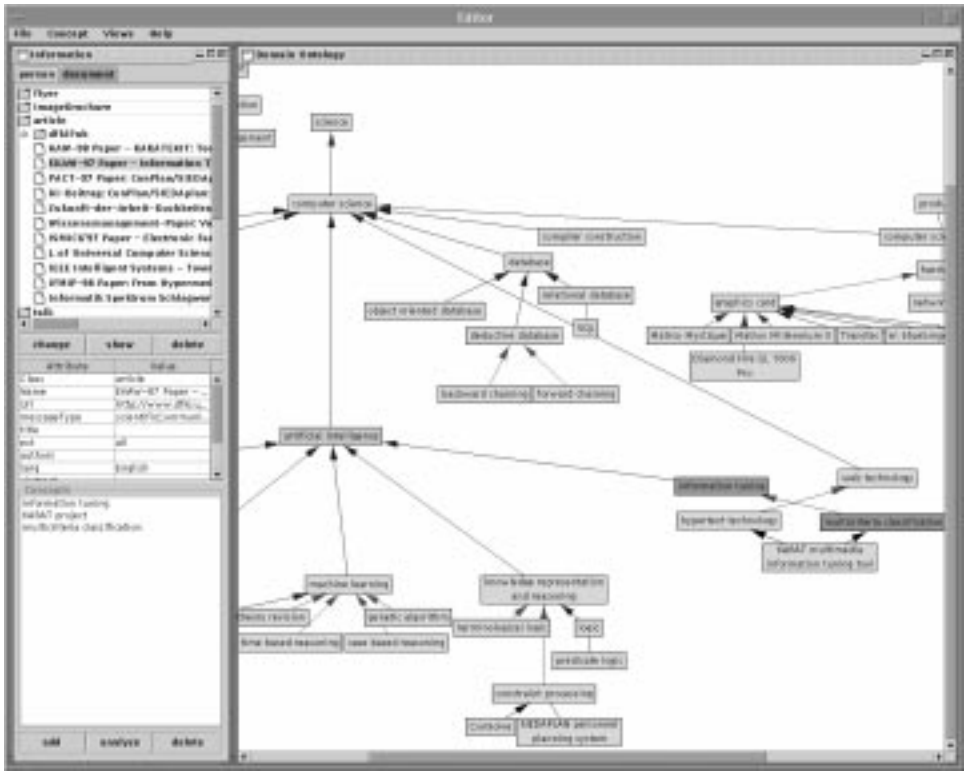

Fig. 13. The KnowMore ontology and document description editor

[Hartley *et al.*, 1997] R. Hartley, R. Kelsey, and R. Webster. Consolidating Multi-Source and Multi-Media Knowledge. In *AAAI Spring Symposium Artificial Intelligence in Knowledge Management*. AAAI, March 1997.

[JAVA, 1998] JAVA. JAVA Hompage at SUN. http://java.sun.com/, 1998.

[JDBC, 1998] JDBC. JDBC Homepage at SUN. http://java.sun.com/products/jdbc/, 1998.

[Kamp, 1996] G. Kamp. Using description logics for knowledge intensive case-based reasoning. Technical Report LKI-M-96/03, Labor für Künstliche Intelligenz, Universität Hamburg, August 1996.

[Kent and Neuss, 1996] R.E. Kent and Ch. Neuss. Web conceptual space. In *Proc. of WebNet'96, the World Conference of the Web Society*, October 1996.

[Kühn and Abecker, 1997] O. Kühn and A. Abecker. Corporate Memories for Knowledge Management in Industrial Practice: Prospects and Challenges. *Journal of Universal Computer Science*, 3(8):929–954, 1997.

[Lenz, 1998] M. Lenz. Managing the knowledge contained in technical documents. In U. Reimer, editor, *Proc. of the 2nd Int. Conference on Practical Aspects of Knowledge Management (PAKM98)*, October 1998. URL `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-13`.

[Liao *et al.*, 1999] Minghong Liao, Knut Hinkelmann, Andreas Abecker, and Michael Sintek. A competence knowledge base system for the organizational memory. In Frank Puppe, editor, *XPS-99 / 5. Deutsche Tagung Wissensbasierte Systeme, Würzburg*. Springer Verlag, LNAI 1570, March 1999.

[Luke *et al.*, 1997] S. Luke, L. Spector, D. Rager, and J. Hendler. Ontology-based web agents. In *Proc. of First Int'l Conf. on Autonomous Agents, AA-97*, 1997.

[Richter, 1995] Michael M. Richter. The knowledge contained in similarity measures. Invited talk at the First Int. Conf. on CBR 1995 (ICCBR-95), 1995.

[Schmiedel and Volle, 1996] Albrecht Schmiedel and Philippe Volle. Using structured topics for managing generalized bookmarks. In *WWW5 Workshop on Artificial*

Intelligence-based tools to help W3 users at the 5th Int'l WWW Conference, Paris, France, 1996. URL `http://www.info.unicaen.fr/~serge/3wia/workshop/`.

[van der Vet and Mars, 1993] P.E. van der Vet and N.J.I. Mars. Structured system of concepts for storing, retrieving, and manipulating chemical information. *Journal of Chemical Information and Computer Sciences*, 33, 1993.

[van der Vet and Mars, 1996] Paul E. van der Vet and Nicolaas J.I. Mars. Coordination recovered. In *Informatiewetenschap 1996*. Delft, 1996.

[van der Vet *et al.*, 1995] P.E. van der Vet, P.H. Speel, and N.J.I. Mars. Ontologies for very large knowledge bases in materials science: a case study. In N.J.I. Mars, editor, *Towards very large knowledge bases: Knowledge Building and Knowledge Sharing*. IOS Press, Amsterdam and Ohmsha, Tokyo, 1995.

[Welty, 1998] Chris Welty. The ontological nature of subject taxonomies. 1998. 1998 Int'l Conference on Formal Ontology in Information Systems (FOIS'98).

Improving Organizational Memories Through User Feedback

Klaus-Dieter Althoff, Markus Nick, Carsten Tautz
Fraunhofer Institute for Experimental Software Engineering (IESE)
Sauerwiesen 6, 67661 Kaiserslautern
Email: {althoff, nick, tautz}@iese.fhg.de

Abstract. The benefits of an organizational memory are ultimately determined by the usefulness of the organizational memory as perceived by its users. Therefore, an improvement of an organizational memory should be measured in the added perceived usefulness. Unfortunately, the perceived usefulness has many impact factors (e.g., the precision of the user query, the urgency with which the user needs information, the coverage of the underlying knowledge base, the quality of the schema used to store knowledge, and the quality of the implementation). Hence, it is difficult to identify good starting points for improvement.
This paper presents the goal-oriented method OMI (Organizational Memory Improvement) for improving an organizational memory incrementally from the user's point of view. It has been developed through several case studies and consists of a general usage model, a set of indicators for improvement potential, and a cause-effect model. At each step of the general usage model of OMI, protocol cases are recorded to pinpoint improvement potential for increasing the perceived usefulness. If improvement potential is identified based on the interpretation of the protocol cases, the user is asked for specific improvement suggestions. Since this is a continuous diagnosis of the usefulness of the organizational memory, this method is able to adapt the organizational memory to the needs of the users – even if the environment, for which the organizational memory was designed, changes.

1 Introduction

The management of knowledge is a critical factor of an enterprise´s success. The objective of knowledge management is the optimal use of the resource »knowledge« for enabling learning from experience, continuous process improvement, and the extension of a company's creativity potential [ADK98]. To support these objectives, a company's knowledge has to be explicitly stored in a so-called organizational memory (OM). While building up such an OM is already a challenging task and involves difficult subtasks like knowledge acquisition, modeling, evaluation, and reuse, the improvement of an OM through user feedback is of essential importance if an OM shall be a continuous source of a company's benefit.
The benefits of an OM are ultimately determined by the usefulness of the OM as perceived by its users. Therefore, an improvement of an OM should be measured in the added perceived usefulness. Unfortunately, the perceived usefulness has many impact factors (e.g., the precision of the user query, the urgency with which the user needs information, the coverage of the underlying knowledge base, the quality of the schema used to store knowledge, and the quality of the implementation). Hence, it is difficult to identify good starting points for improvement.
This paper presents the goal-oriented method OMI (Organizational Memory Improvement) for improving an OM incrementally from the user's point of view. It has been developed through several case studies [ABT98] and consists of a general usage model, a set of indicators for improvement potential, and a cause-effect model. At each step of the general usage model of OMI, data in the form of protocol cases are recorded to pinpoint improvement potential for increasing the perceived usefulness. Such protocol cases include formal parts like the user-specified

query, the retrieval goal of the query (see Section 3), the result of the query, and certain actions the OMI method has invoked (e.g, asking the user a specific question or forwarding the case to the OM maintenance team). The protocol cases also include informal parts like the user´s answers to certain questions, the user´s general feedback at the end of a session, or some explanatory texts that have been written by the OM maintenance team. The purpose of protocol cases is twofold:

- On the one hand they can be used by the OM maintenance team to improve the OM's content, to react on identified user problems by contacting the respective user, or to start any necessary improvement action.
- On the other hand, if a protocol case – or parts of it – have been qualified and published by the OM maintenance team, it can be used by the OM management system (OMMS). Based on the general usage model of OMI and the formal parts of the respective protocol case, the OMMS can automatically interpret the case, i.e., apply case-based reasoning (CBR). This includes asking the user specific questions to collect his feedback or presenting situation-specific lessons learned.

If improvement potential is identified (by the OM maintenance team or the OMMS) based on the interpretation of the protocol cases, the user is asked for specific improvement suggestions. Since this is a continuous diagnosis of the usefulness of the OM, this method is able to adapt the OM to the needs of the users – even if the environment, for which the OM was designed, changes. Thereby the OMI method allows to overcome the general problem that users are often not available during the development of the OM. The basic idea underlying OMI is to start with an »intelligent guess« (not focus of this paper) and to improve the OM/OMMS incrementally by systematically collecting and analyzing focused user feedback (focus of this paper). Basing on the OL-CBR approach (organizational level case-based reasoning: [ABT98]), OMI integrates the manual interpretation of the protocol cases by the OM maintenance team with the automatic interpretation by the OMMS, which actually is based on a CBR tool (see Section 2). The more formal knowledge is recorded as part of the protocol cases (based on OMI's usage model) and/or added through the OM maintenance team´s analysis and revision, the more knowledge can be used by the OMMS to adapt to the users´ needs based on user feedback. Thereby OMI can cope with practical constraints like avoiding user overload through asking too many questions.

In spirit OMI is comparable to evolutionary constructions of repositories (e.g., [Hen97]) and approaches based on Failure Mode and Effect Analysis (FMEA) [JK98, PZ96, Pfe96], which also try to avoid failure situations, i.e. in case of OMI, a decrease of the perceived usefulness. At least partially these approaches base also on CBR [JK98, PZ96].

OMI can also be compared to approaches for diagnostic problem solving, especially case-based diagnosis [AW91, Wes95, Alt97], because a classification problem has to be solved based on collected symptoms. By contrast, OMI is not intended to be fully automated. It is flexible concerning the degree of using fully automated interpretation of cases by the OMMS or manual case analysis and interpretation by the OM maintenance team.

In the next section our approach for implementing an OM is shortly introduced. Section 3 and Section 4 present OMI's cause-effect model for usefulness as perceived by the user and the general usage model, respectively. Section 5 describes OMI's diagnostic process for improving an OM, whereas Section 6 explains the utilization of protocol cases. Finally some conclusions are drawn.

2 Our Implementation: A Case-Based Organizational Memory

To talk about improvement of an OM, we need to state some assumptions about implementation aspects of the OM. We will do so by describing major primitives of the representation formalism REFSENO[1] we use for specifying the ontology (i.e., the schema) underlying an OM [ABT98, TG98, TG99]. REFSENO is implemented in collaboration with a commercial CBR tool provider [Tec99][2].

1. An OM contains different types of experience (e.g., lessons learned, typical effort distribution for

1. REFSENO stands for *Re*presentation *F*ormalism for *S*oftware *En*gineering *O*ntologies.

2. A first prototype implementation can be accessed at http://demolab.iese.fhg.de:8080/.

projects, specific project information such as schedules and milestone plans). Each of these types is specified by a *concept*. The actual contents of the OM are stored as instances of the concepts. Hence, a concept describes a class of experience items. Figure 1 describes the three-layered representation we employ. The meta knowledge (in the form of REFSENO) guides the specification of an ontology for an

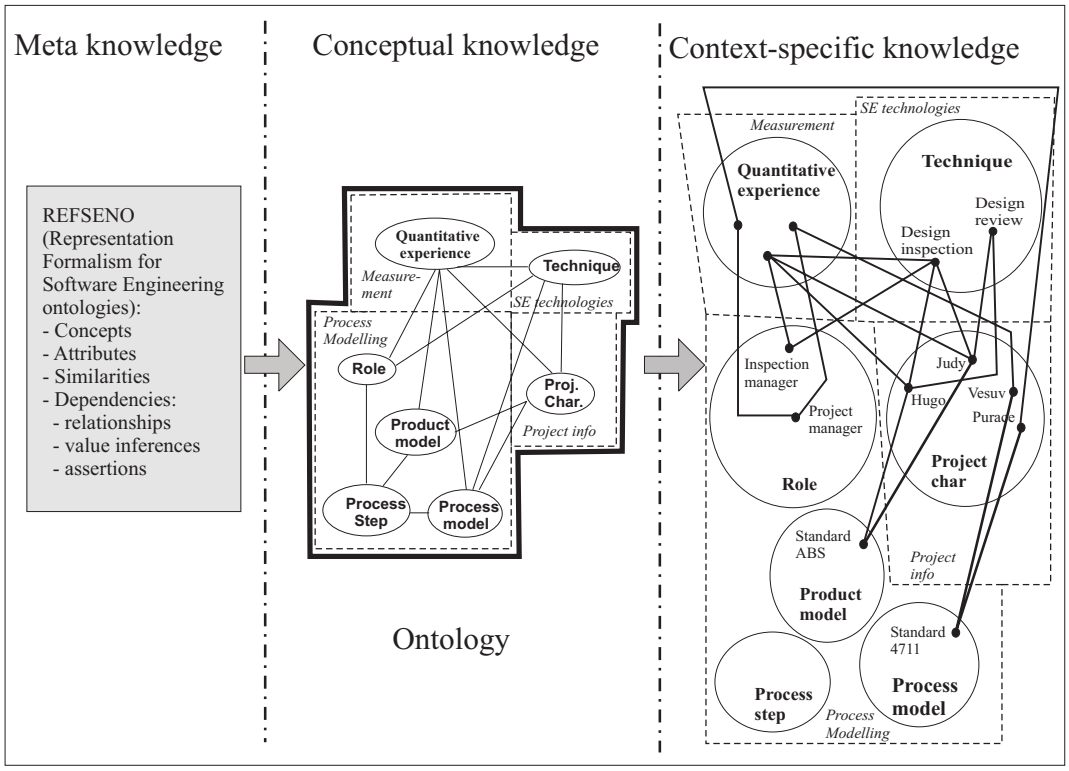

FIGURE 1. Three types of knowledge are used to describe schema and contents of an organizational memory.

OM, whereas the ontology guides the population and exploitation of the OM's contents.

2. The OM's contents are made up of two parts: the artifact itself and its characterization. The artifact itself can be a file or database. Often it has a propriety format, meaning that it is difficult or impossible to access the information contained in it by generic tools (like an OMMS). The characterization part contains all information that is deemed to be necessary for performing some predefined task(s) with the experience item. For instance, a lesson learned may be characterized for the purpose of later application by information about the lesson learned itself (e.g., name, length, descriptions of a problem and one or more solutions for solving the problem), information about its interface (e.g., prerequisites for the application of the lesson learned), and information about the context in which it has been created and used (e.g., project and situation in which the described problem occurred). These characterizations guide both storing and searching for experience items. Information contained in characterizations may overlap with or complement information contained in the artifact [ABT98].

3. It is *partially* known a-priori what information needs to be part of the characterization.[1] This knowledge is captured as part of the ontology in the form of a set of *attributes*. We distinguish two categories of attributes: *terminal* (containing actual information needed for some task) and *nonterminal attributes* (acting as references to related instances which in turn contain more terminal and nonterminal attributes; represented by lines in Figure 1). Terminal attributes can be used for retrieval, whereas nonterminal at-

1. During operation, further information needs will become apparent. Considering them constitutes an improvement of the OM.

tributes can be used for navigational search. Just as in modern programming languages, both categories of attributes can be typed. This gives the OMMS the possibility to perform certain consistency checks on the contents of the OM (e.g., whether a value supplied by the user lies within a predefined value range or whether the value of a nonterminal attribute references an instance of a predefined concept). They also allow to perform type checks for value inferences (formulas that specify how to calculate automatically computable attribute values) and assertions (formulas that specify dependencies between a set of attributes).

4. Exact matches between a query and the instances in the OM can be expected only in rare situations. This leads to the introduction of similarity. If an OMMS supports retrieval based on characterizations, it must be capable of returning characterizations similar to the one used for specifying the needed artifact. The underlying hypothesis is that two artifacts (e.g., milestone plans) are similar if they have similar characterizations. Hence, this assumption allows us to retain experience items as they are gained, that is, as instances of predefined concepts, keeping the effort for storing new experience low. During reuse, stored experience can be retrieved (via the similarity feature) and adapted to the new situation. In the ontology, similarity functions are associated with attributes (local similarity) and concepts (global similarity combining local similarities).

In summary, the structural specification of an OM is made up of a set of concepts that are specified through a set of typed attributes:

$$\begin{aligned} &\text{Spec}_{\text{OM}} = \{\{c_1, \ldots, c_n\}, \{t_1, \ldots, t_l\}\} \\ &\text{with} \quad c_j = \{a_{j,1}, \ldots, a_{j,m_j}\} (j \in \{1, \ldots, n\}) \\ &\text{where} \quad \text{type}(a_{j,k}) \in \{t_1, \ldots, t_l\} (k \in \{1, \ldots, m_j\}) \end{aligned}$$

There is a similarity function associated with the concept and the attributes. The similarity is always computed from an instance i to a query q where i and q are instances of the same concept, i.e.:

$$\text{concept}(i) = \text{concept}(q) = c = \{a_1, \ldots, a_m\} \in \{c_1, \ldots, c_n\}$$

It is computed using the similarity function associated with this concept:

$$\text{sim}(i, q) = \text{sim}_c(i, q) = \text{f}(\text{sim}_{a_1}(\text{val}_{a_1}(i), \text{val}_{a_1}(q)), \ldots, \text{sim}_{a_m}(\text{val}_{a_m}(i), \text{val}_{a_m}(q)))$$

where $\text{val}_a(i)$ stands for the value of attribute a of instance (or query) i.

3 Perceived Usefulness

The success of an OM can be measured in many ways. Examples for specific views on evaluation mainly from the knowledge-based system and related fields are [AA96, Alt97, Coh89, GKP+83, GXG98, Kir94, SW88, vW96]. Also, some evaluation work has been done in the area of software reuse (programs), mainly regarding economic success [BB91, Lim96, HS93]. Many of the economic models for software reuse can also be used to evaluate organizational memories, because OMs are also about reuse. Only in the case of an OM, reuse is not restricted to software development artifacts. Other evaluation criteria, most importantly *recall* and *precision*, come from library and information science [SM83].

However, using a goal-oriented measurement and evaluation approach where experts participate in the definition of a measurement program [BDR96], we found out that the usefulness, as perceived by the user of the OM, is the most important measure for the success of an OMMS [NT99]. This is not surprising since an OM is worthless if it fails to deliver information that is of value to its users. These findings are also supported by Harter [Har92] (there called »psychological relevance«) and Cooper [Coo97] (there called »personal utility«).

Therefore, we will judge whether any change is an actual improvement based on the perceived usefulness *before* and *after* the change. If the usefulness improved, the change is regarded as an improvement.

As pointed out by Cooper, the ideal measurement of the usefulness as perceived by the user is practically and economically impossible [Coo97]. Therefore, we have to simplify the measurement procedure. To do so, we recall

the meaning of similarity. Ideally, the similarity between an experience item in an OM and the needed experience (specified by the query) is an a-priori approximation of the a-posteriori usefulness as experienced by the user [Wes95, AR99]. If the OMMS returns the instances $i_1, \ldots, i_n$ in response to a query q, where i_1 is most similar to q and i_n least similar, the user should select (ideally) i_1 or – if more than one instance is perceived as useful – the set $i_1, \ldots, i_m$ with $m \leq n$. Ideally, $m = n$. This implies also an assignment of the degree of usefulness to the instances on an ordinal scale, that is, i_1 is the most useful instance, i_2 the second most useful, etc.

Since it is difficult to determine a minimal similarity value (this depends among others on the background knowledge of the user), an OMMS could return a fixed number of instances, e.g., $n = 10$. If i_m denotes the last *useful* instance, then the system is optimally useful if an OMMS never returns an instance $i \in \{i_1, \ldots, i_m\}$ that is not useful. The more often an OMMS returns such an instance, the less useful it will be. Using this definition, the user can simply mark all instances $i_1, \ldots, i_n$ as either useful or not useful. Based on this data, the usefulness of an OMMS can be computed (e.g., relative to the number of queries issued by users). Another important aspect of usefulness is that an OMMS returns useful instances at all, i.e., m should be greater than 0.

Unfortunately, there is no single parameter with which the usefulness of an OMMS can be changed. Rather, there is a large number of variation factors. Figure 2 shows the variation factors we have identified so far. However, we do not claim that the list is exhaustive. In the following, we will go into detail for the main branches of the cause-effect model.

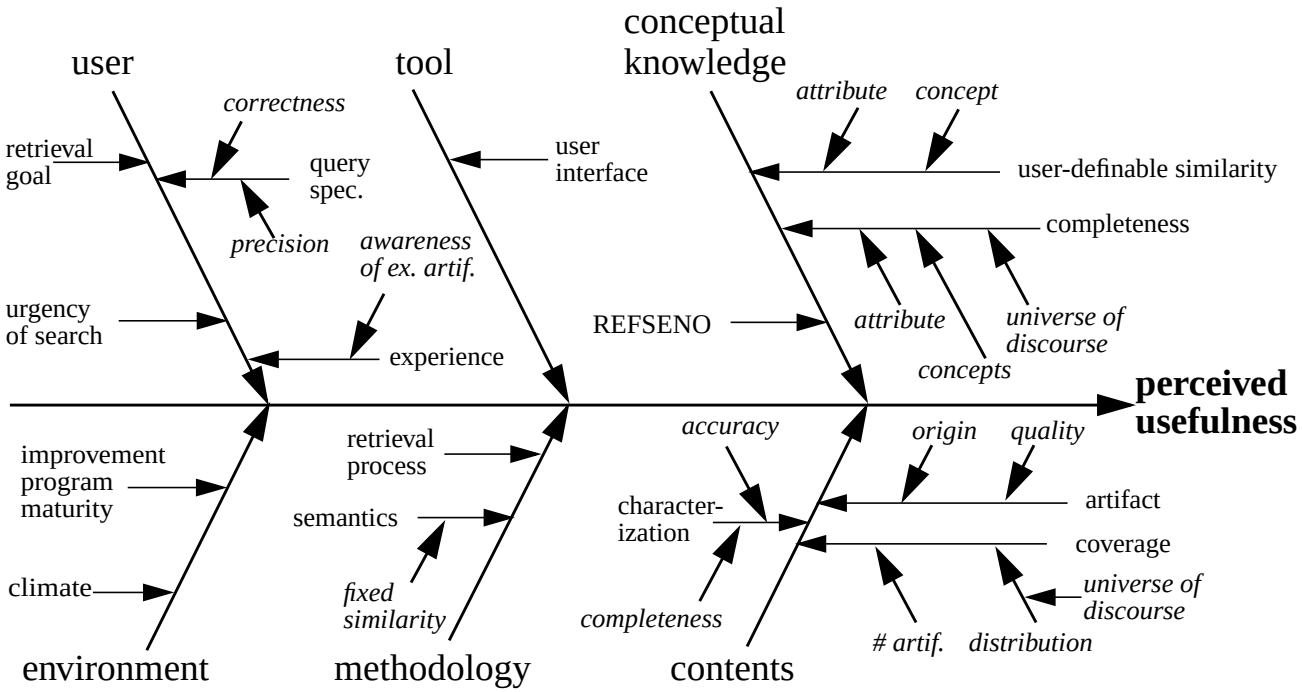

FIGURE 2. OMI's cause-effect model: The usefulness as perceived by the user is influenced by many factors.

- **User.** The intentions and abilities of the user will greatly influence the usefulness of the experience items returned by the OMMS. For instance, if a user does not specify his needs correctly, the OMMS might return experience items adequate for the needed experience as specified by the query, but not for the actual experience needed (which is in the mind of the user). But even if the query is specified correctly, it may be underspecified (user gives too little attribute values to allow a meaningful differentiation among the stored experience items) or overspecified (the user gives too many attribute values – no experience item in the OM can be found which matches the query). An overspecification is only possible in OMMS that are only capable of retrieving exact matches or OMMS computing a cutoff-value for similarities. For a similarity-based OMMS that returns a fixed number of instances, an overspecification is practically impossible.

Also of importance for the usefulness of an experience item is the purpose for which the user retrieves the experience (denoted as »retrieval goal« in Figure 2). A characterization may contain all information that is necessary to apply a lesson learned (thus the lesson learned will be perceived as useful), but may fail to provide hints how to adapt its solution to other situations (thus the lesson learned might be perceived as not useful if it needs to be changed).

Quite interestingly, the urgency of the search [Har92] and the experience of the user (e.g., the user might perceive only new experience, that is, experience he was not aware of, as useful) [Coo97] will also affect the perceived usefulness.

- **Environment.** The environment in which the OMMS operates may be mature. In this case the OM tends to be filled with more experience that has been gained by the organization itself. Typically, own experience is more valuable than experience that is part of textbooks, because it can be tailored more easily to new situations since the contexts between the situation in which it was gained and the situation in which it is applied do not differ as much (or at least, the differences are better known) [BR88, BCR94]. Also, the climate in an organization influences how willingly people are to share their experience with their colleagues over an OMMS. In organizations where mistakes are not viewed as chances to learn, valuable information will remain in the minds of the people – mistakes will be repeated [Dam98].
- **Tool.** For the user, the tool (OMMS) is mainly characterized through its behavior and its user interface. The behavior is determined through all other branches (except for the »environment« branch) and is not considered further at this point. The user interface constitutes a sort of barrier for the usage of a system. If a system is hard or cumbersome to use, people will try avoid using the system [MD97, Nic98, NT98]. And, if a system is not used, it is perceived as not useful.
- **Methodology.** It is the methodology that is supported by the OMMS. If the underlying methodology (e.g., the retrieval process supported by the OMMS or the semantics of REFSENO's primitives) is not optimal, it cannot be expected that the system is perceived as useful as it could be.
- **Conceptual Knowledge.** Clearly, the behavior of an OMMS is not solely determined by its implementation, but also by its contents and the organization of these. As the conceptual knowledge determines what and how experience is stored in the OM, it plays a major role regarding the usefulness of a system. First of all, the concepts and their attributes define a universe of discourse that the OM *can* cover. In contrast to this universe stands the universe of discourse the OM *shall* cover. The concepts and attributes may not cover all kinds of experience to be stored or all information needed to perform certain predefined tasks. In addition, the similarity functions may not approximate the perceived usefulness appropriately. And finally, certain characteristics of the universe of discourse may not be expressible using the primitives of REFSENO.
- **Contents.** Even if the conceptual knowledge is defined optimally regarding the usefulness, the contents of the OM may cause the system to fail. Just as with databases, the information stored in an OM must be accurate and complete [NT98]. Otherwise, users will lose their confidence in the experience provided by the system. This will lower the overall perception of the system's usefulness.

  Also, the universe of discourse the OM shall cover is not covered simply by defining the universe in terms of experience types to be stored. The actual experience has still to be stored! The coverage of a universe of discourse is influenced by two characteristics: the number of artifacts in the OM and their distribution. The more artifacts are stored, the *more likely* it is that the system will return useful information. However, it is possible that many artifacts are stored, but the stored artifacts do not match the users' queries good enough (they are not similar enough). Thus, the distribution of the artifacts must match the distribution of the users' queries.

  The usefulness of experience offered by the OMMS is also determined by the known quality of its artifacts. Quality can be measured in many ways, e.g., in terms of popularity or importance [Coo97]. In addition, the origin of an artifact (e.g., the author) may influence the perceived usefulness. For instance, assume that the user issues a search request on software inspections. Let us further assume that the OMMS returns experience on software inspections whose author is known to the user to be an expert in software testing. The user may

now value the usefulness of this particular experience item very high, because he may suspect a connection between software inspections and testing. In a consecutive query, the user may now also want to include experience regarding software testing.

4 Usage Model

As can be seen from the variation factors presented in the previous section, the usage of an OMMS cannot be described by a sequential process. The result of a query may lead to new insights and thus to additional queries. Queries leading to unsatisfactory results will be changed and issued again. To analyze these situations further, we first describe the »ideal« (i.e., sequential) usage scenario for an OMMS. Later, we describe under which circumstances the user might go back and repeat some of the steps.

Figure 3 shows the sequential usage model. The process starts with the specification the user has in mind for the experience needed. The user formulates a query based on this specification guided by the ontology. This query is input for the OMMS. It identifies potential experience items (e.g., if the user requests a project schedule, all project schedules are potential experience items). The identified experience items are then evaluated by computing a similarity value (as defined by the ontology) for each of the experience items to the query. The resulting list is ordered by decreasing similarity and cut off at a fixed number (e.g., after the tenth experience item). The characterizations of the ten artifacts are displayed to the user who evaluates (manually) the offered experience items. The user selects and retrieves all useful artifacts from the OM based on the provided characterizations. Finally he applies the artifacts.

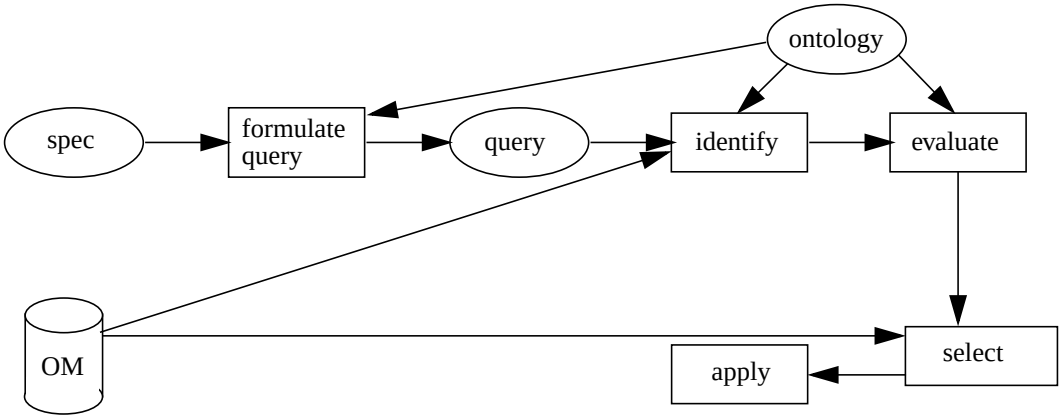

FIGURE 3. The »ideal«, sequential usage model of an OMMS.

In practice, this idealized process does not take place. It starts with the fact that users try to optimize the effort for maximum information gain [Har92]. This means for our usage model that the user will not formulate a query with all known information (from the specification in the user's mind), but rather specify only some attribute values (in interviews, experts stated that at most ten values ought to suffice for a query) [Nic98]. This may result in underspecified queries which in turn lead to an unsatisfactory retrieval result. If the user thinks that the system can do better, he will go back to step »formulate query« and supply additional information and reissue the search request. Also, it is unlikely that a user will solely select artifacts on the basis of their characterizations. Typically, he will examine the artifacts using some editor to make the decision on whether to apply them, or not (e.g., Is the artifact well documented? Can it be easily understood?). However, aim of the characterization knowledge is to limit the number of artifacts that have to be examined in this way as well as to reduce the effort needed to examine the artifacts (by supplying information that can only be extracted using large amounts of effort, e.g., correctness of a technical design with respect to its specification). After the examination of the artifact the rating of the usefulness of the artifacts may change. The deepened understanding may also lead to additional queries, starting a new usage process.

Finally, it may turn out *after* applying the artifact that it was not the best candidate for the purpose. If the task could not be performed by applying a retrieved artifact, the user may want to reissue his (possibly refined) query to find more suitable artifacts.

5 Diagnosing for Improvement

At each step of the usage model presented in the previous section, indicators may be examined to identify improvement potential. The basic idea is to diagnose situations that lead to suboptimal usefulness. These situations can be described using the variation factors of usefulness as they have been presented in Section 3. Based on the diagnosis, changes can be suggested that (hopefully) will lead to improvements of the OMMS. As we have already seen in Section 3, not all of the variation factors can be changed by purely technical means. Although some of these variation factors influence the usefulness of an OMMS substantially, the diagnosis and change of these factors is beyond the scope of this paper.
To ease the understanding of situations that can be changed by tailoring the OMMS, Figure 4 shows the usage model enhanced by change steps to the OMMS. In the following, each of the steps of the usage model is examined in detail.

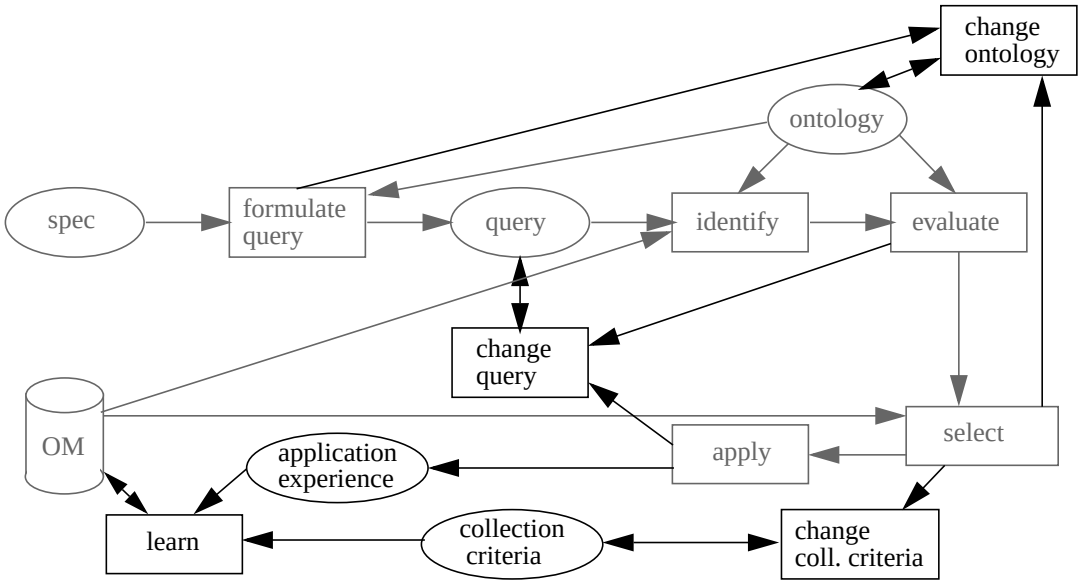

FIGURE 4. The usage model (shown in grey) can be enhanced by change steps to the OMMS.

- **Formulate query.** At this step it may turn out that the universe of discourse to be covered by the OM, is actually not covered. For instance, if no concept for lessons learned is part of the conceptual knowledge, it is not possible to specify lessons learned for retrieval. If lessons learned are needed by the users, they should be part of the universe of discourse of the OM. Therefore, a new concept for lessons learned should be introduced. At the same time, existing lessons learned (e.g., in form of memos and minutes) should be analyzed and characterized for their retainment as artifacts in the OM.
- **Identify.** This step is performed without user interaction. Hence, no situation for improving the usefulness can be identified (by the user) during this step.
- **Evaluate.** This step has an automatic and a manual part. As with the step »identify«, the automatic part cannot be used for identifying improvement potential (by the user). Considering the extended usage model where the user employs some editor to examine candidate artifacts to decide on their usefulness, the manual part of the step »evaluate« is actually a three step process:

1. Decide (based on the characterizations) which artifacts to examine in which order (artifacts that are deemed useless based on the characterizations are neglected).
2. Examine the artifacts in the predetermined order. Stop if only one artifact is needed and an examined artifact meets the needs.
3. Mark all useful artifacts. If not enough useful artifacts have been marked, decide whether to go back to step 1 or to reissue a revised query.

During the first step, the user may not be able to decide whether to examine an artifact or declare an artifact as »useless«. As the examination of an artifact can require considerable effort (imagine, you have to decide whether a textbook actually contains information important to your research work), the user wants to examine only those artifacts that have a high potential of being useful. If the user is not able to make this decision, some information about the artifact is missing. This can be supplied as part of the characterization. Therefore, for all artifacts the user cannot decide on, the user should articulate the missing information. This feedback can be used to improve the conceptual knowledge of the OM in a goal-oriented way by adding new attributes. If the missing information is supplied by the OM maintenance team (i.e., including the values for new attributes), the user will be able to decide next time he issues a similar query. The usefulness of the system will have been improved.

The output of the first step is the order in which potentially useful artifacts are examined. If the examination of the artifacts is invoked under the control of the OMMS, the OMMS can record this order during step 2. Ideally, the order should be the same as the one determined during the automatic part of the step »evaluate« (as pointed out in Section 3). If it is not or if artifacts which are placed high up in the similarity-based ordering although they were deemed useless by the user, the user can be asked why he chose a different ordering. The reasons may be either (a) underspecified queries (the user knows more than he has specified, and he matches the characterizations with the unspecified but known information; in this case nothing needs to be improved) or (b) improper similarity functions (either because of inadequate local or global similarity functions or because of undocumented knowledge known/assumed by the user; in the latter case additional attributes should be defined capturing the undocumented knowledge and the new knowledge should be considered by the similarity functions).

- **Select.** After one or more artifacts have been examined, the best suitable artifact is selected to be applied. If the user shall not be bothered with too many questions (these will arise especially after the initial set-up of an OMMS), the questions can be restricted to the artifact actually applied. This alternative will not be able to identify situations in which artifacts were originally judged to be useful (based on the characterization), but later judged to be useless (based on the artifact's examination).

  At any rate, if the selection step yields no selected artifact, the user should be asked to give a reason why the most similar artifact was not chosen. In this case, a hole in the coverage of the OM has been identified. The artifact needed to fill this hole is both specified by the original query supplied by the user and the reason why the most similar artifact does not cover the requirements. Based on this feedback, the collection criteria for the type of the requested experience should be analyzed. Do they allow the collection of an artifact similar enough to the requested one? If not, the collection criteria should be changed. If the missing artifact is deemed to have a high application potential in the future by other users, either a separate project for creating such an artifact may be started or – if the artifact is created as part of the project the user belongs to – the artifact may be stored after the project has been completed.

- **Apply.** During the application of an artifact, experience such as effort for understanding and modifying the artifact as well as how the artifact should (not) be changed should be recorded. Such experience can then be attached (in form of an extended characterization) to the artifact after its application. In this way, the applicability information is improved continuously with each application of an artifact.

  During the artifact's application, it may turn out that the artifact is not as useful as originally estimated. If this is the case, one of four choices can be made:

1. Ignore the fact and continue applying.
2. Stop applying and do nothing more (e.g., if it turns out that a lesson learned is not applicable in the current situation).
3. Stop applying and create the needed artifact from scratch (e.g., a project schedule).
4. Stop applying and retrieve another (hopefully more useful) artifact (e.g., a project schedule).

In the latter case, the old query may serve as an entry point.

The descriptions above show how improvements in the conceptual knowledge and contents of an OM can be pinpointed by automatically collecting data in the form of protocol cases and asking the user for feedback if the system behaves not as expected (based on the usage model). Changes regarding other branches of Figure 2 (e.g., user interface and missing experience items the user is aware of) can be suggested based on answers to questions posed to the user after each (or after each n-th) retrieval attempt at the end of step »select«. However, in contrast to the situations outlined above, these questions cannot be posed based on some (automatic) analysis and, thus, must be posed unconditionally. This might lead to an overburden on the user who has to devote his time in answering the posed questions.
Finally, the variation factors listed in the »user« branch deserve a closer look. While the precision and the correctness of the query has been taken care of by allowing the user to reissue a revised query, the variation factor »retrieval goal« may not be underestimated. Different retrieval goals have different information needs and the usefulness of the retrieval result is measured (at least partly) in terms of how well the presented experience items are suited to perform some predefined task. Consequently, both the information need (represented by a set of attributes) and the similarity functions (estimating the usefulness) may differ. Therefore, all optimizations of the OMMS must be performed with respect to the retrieval goal. Otherwise, an improvement done for one retrieval goal may result in a change to the worse for another retrieval goal.
As a consequence, an OMMS should ask the user for his retrieval goal at the beginning of the session. Based on the retrieval goal, the query attributes can be restricted to the relevant ones. Thus, the user is guided more effectively which in turn also helps to reduce the risk of incorrect and/or meaningless queries. In addition, optimal similarity functions can be selected. Finally, the retrieval goal can be used to tailor the OM for specific retrieval goals – the optimizations do not influence the behavior of the OMMS for other retrieval goals. However, besides the (marginal) additional effort on part of the user to specify the retrieval goal, a new error source is introduced. Now, not only the query may be incorrect but also the specification of the retrieval goal. Hence, it may not be enough for the user to change his query in case the system does not behave as wanted, but perhaps the user must correct his retrieval goal as well. An empirical investigation is needed to find out whether an average user is capable of recognizing what to change.

6 Improvement With Protocol Cases – An Example

The representation of all the information being collected during the OM usage process in the form of protocol cases (see Figure 5) helps the OM maintenance team in systematically analyzing the information, because they can apply CBR based on the formally described parts of a protocol case. During this analysis user problems may be identified and the OM maintenance team could directly contact the respective user. Also based on the analysis the contents of the OM may be improved. In addition, the behavior of the OMMS may be improved (see Figure 6)

- manually by the OM maintenance team and/or
- by adding revised, qualified protocol cases to the system such that the user can access them and/or

- by applying CBR for the automatic interpretation of these cases by the OMMS.

Protocol case:
- Retrieval goal
- 1
- 2
- 3
- …

Query
- Result
- OMMS action/user reaction (*text of user reaction*)
- *Explanatory texts (OM maintenance team)*
- Actions to be carried out by OMMS
- *Suggestions to the user*

formal knowledge
informal text

FIGURE 5. A protocol case contains a log of the human-computer interaction as well as actions aiming at the improvement of the organizational memory.

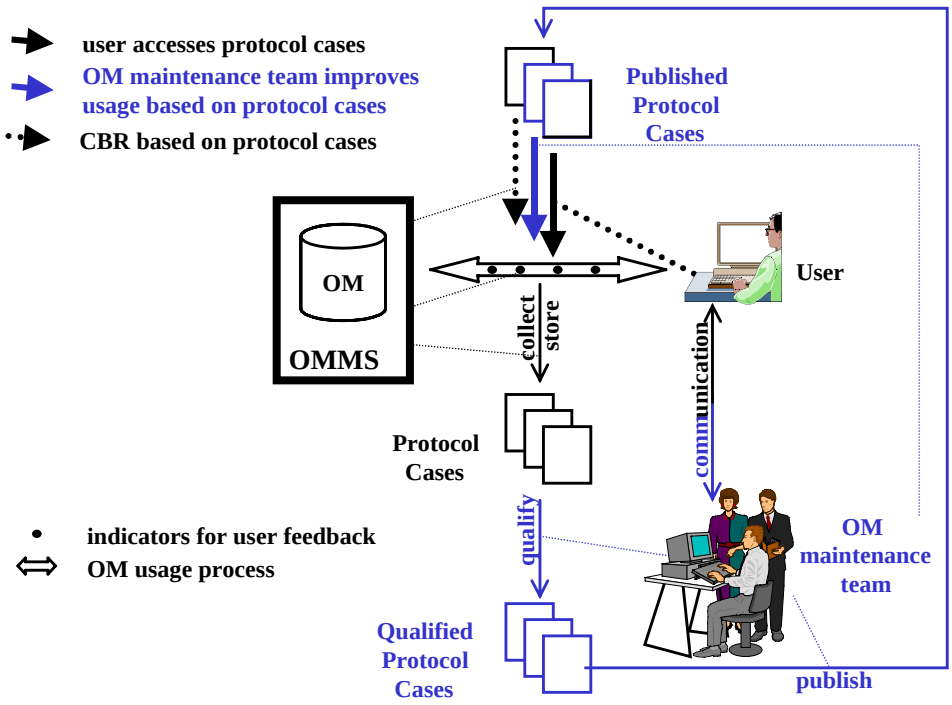

FIGURE 6. Protocol cases are the basis for improving the organizational memory.

In the following, we will demonstrate the utilization of protocol cases using GQM plans as an example. GQM (Goal/Question/Metric Paradigm) [BDR96] is an innovative technology for goal-oriented software engineering measurement [GB97]. GQM helps defining and implementing operational and measurable software improvement goals. It has been successfully applied in several companies, such as NASA-SEL, Bosch, Digital, and Schlumberger [CEM96]. In GQM programs, the analysis task of measurement is specified precisely and explicitly by detailed measurement goals, called GQM goals. Relevant measures are derived in a top-down fashion based on the goals via a set of questions and quality/resource models. This refinement is precisely documented in a GQM plan, providing an explicit rationale for the selection of the underlying measures. The data collected is interpreted in a bottom-up fashion considering the limitations and assumptions underlying each measure. The process of planning GQM programs can be substantially supported through the reuse of measurement experiences [GB97, GABT98]. The GQM plan is a principal element of the planning of a GQM-based measurement program (see Figure 7).

GQM-Based Measurement: GQM Plan
A GQM plan is developed based on a measurement goal consisting of the following components [BDR96]:
- a goal, defining the object, purpose, quality focus, viewpoint and the context of the measurement program,
- a set of questions, operationalizing the goal,
- a set of models, specifying how to answer the questions,
- a set of measures, operationally defining the data to be collected to feed the models.

GQM Goal: Analyze the Software process in order to characterize relaibility from the viewpoint of the software developer in the company Y.

Q_1What is the overall number of failures reported before delivery?

M_1.1count of failure reports turned in before delivery [ratio: integer][1]

Q_2What is the distribution of failures reported before delivery by criticality level?

Model: Distribution = (# critical failures/ total # failures, # uncritical failures/ total # failures)
critical: complete breakdown of system; uncritical: unable to perform one or more of the functions F1-F6

M_2.1classification by criticality [ordinal:uncritical;critical]

M_2.2count of failure reports before delivery [ratio: integer]

Q_3What is the distribution of faults by life cycle phase of detection before delivery?

Model: Distribution = (# faults in REQ/ total # faults,# faults in HLD/ total # faults, # faults in LLD/IMP/ total # faults)

M_3.1count of fault per life cycle phase where the fault was introduced [nominal: REQ, HLD, LLD/IMP]

Q_4What is the total rework effort?

Model: rework effort = (effort to isolate fault + effort to correct fault)

M_4.1for all failures reported before delivery: effort to isolate the faults that caused the failures (person-hours) [ratio: integer]

M_4.2for each fault detected before delivery: effort to correct the fault (person-hours) [ratio:integer]

1. [scale: range]

FIGURE 7. Simplified example of a GQM plan (taken from [TG99])

The contents of a GQM plan can be (partially) characterized by the measurement goal as indicated in Figure 7. For our example, we will assume that the organizational memory consists of the GQM plans listed in Table 1 and the project characterizations listed in Table 2.

TABLE 1: GQM plans of the organizational memory

Name	MObject	MPurpose	Quality focus	MViewpoint	Context
A:X - Adaptability 1	Design document	Characterization	Adaptability	Software developer	A: X
A:X - Adaptability 2	Requirements document	Characterization	Adaptability	System engineer	A: X
A:X - Completeness	Requirements document	Characterization	Completeness	System engineer	A: X
A:X - Efficiency 1	Design inspection	Characterization	Efficiency	Project supporter	A: X
A:X - Efficiency 2	Code inspection	Characterization	Efficiency	Project supporter	A: X
A:X - Effort	Development process	Characterization	Effort	Technical leader	A: X
A:X - Reliability	Software product	Characterization	Reliability	Quality assurer	A: X
B: Y - Communication efficiency	Communication	Characterization	Communica-tion efficiency	Software developer	B: Y

TABLE 1: GQM plans of the organizational memory

Name	MObject	MPurpose	Quality focus	MViewpoint	Context
B: Y - Comprehensibility	Development document	Characterization	Comprehensibility	Software developer	B: Y
B: Y - Effectiveness	Quality assurance activities	Characterization	Effectiveness	Department head	B: Y
B: Y - Reliability	Software product	Characterization	Reliability	Software developer	B: Y
C: Z - Effort	Development process	Characterization	Effort distribution	Technical leader	C: Z
C: Z - Usability	Specification	Characterization	Usability	Software developer	C: Z
C: Inspection - Effort	Inspection process	Characterization	Effort	Department head	C: Z
D: W - Effort 1	Development process	Characterization	Effort	Technical leader	D: W
D: W - Effort 2	Development process	Characterization	Effort	Department head	D: W
D: W - Effort 3	Development process	Characterization	Effort	Software developer	D: W
D: W - Reliability 1	Development document	Characterization	Reliability	Software developer	D: W
D: W - Reliability 2	Development document	Characterization	Reliability	Technical leader	D: W
D: W - Reliability 3	Development document	Characterization	Reliability	Department head	D: W

TABLE 2: Project characterizations of the organizational memory

Project	Duration	Team size	…
A: X	24	100	
B: Y	<unknown>	200	…
C: Z	18	20	…
D: W	8	7	…

A query for a GQM plan that describes the measurement of the effort of a software process (includes both the development process and the quality assurance activities) from the viewpoint of a project manager for a project with a duration of 6 months and a team size of 3 would result in a protocol case as shown in Table 3. The protocol case reveals:

- The user changed the query once. He extended it by a specification of the context. Apparently, the query was too imprecise.

- »Text of user reaction (1)« shows two problems:
  1. The user has a hard time to decide which viewpoint (department head or technical leader) is appropriate for the needed viewpoint
  2. Information about the completeness of GQM plans is missing in the characterization of GQM plans. As it is unclear what is meant exactly by »completeness of GQM plans« the maintenance team will have to talk to the user before adding a new attribute to the characterization schema of GQM plans.
- Because there is no GQM plan on the software process, but only on its parts (development process and quality assurance activities), the user decides to merge two existing GQM plans (positions 1 and 4 are checked out). This fact is emphasized by »Text of user reaction (3)«.
- The deviation from the expected user reactions (viewing position 4 before position 6 and checking out positions 1 and 2) does not require a change of the OMMS because the user grouped the returned positions by their objects (see »Text of user reactions« (2) and (3)).
- The last three sections are empty because they are filled out by the maintenance team.

TABLE 3: Exemplary protocol case produced by a query

Section	Dimension	Value
User	Name	U. Ser
Retrieval goal	Object	GQM plan
	Purpose	Modification
	Viewpoint	Project supporter
Query (1)	MObject	Software process
	MPurpose	<undefined>
	Quality focus	Effort
	MViewpoint	Project manager
	Context	<undefined>
Query (2)	MObject	Software process
	MPurpose	<undefined>
	Quality focus	Effort
	MViewpoint	Project manager
	Context: Duration	6
	Context: Team size	3
	Context: …	<undefined>
Result	Position 1	D: W - Effort 1 (0.976)
	Position 2	D: W - Effort 2 (0.976)
	Position 3	C: Z - Effort (0.967)
	Position 4	C: Inspection - Effort (0.967)
	Position 5	D: W - Effort 3 (0.965)

TABLE 3: Exemplary protocol case produced by a query

Section	Dimension	Value
	Position 6	A: X - Effort (0.962)
	Position 7	A: X - Efficiency 1 (0.896)
	Position 8	A: X - Efficiency 2 (0.896)
	Position 9	D: W - Reliability 2 (0.889)
	Position 10	D: W - Reliability 3 (0.889)
OMMS action/ user reaction	User reaction (1)	View position 1
	User reaction (2)	View position 2
	User reaction (3)	View position 3
	User reaction (4)	View position 6
	User reaction (5)	View position 4
	User reaction (6)	View position 7
	OMMS action (1)	Ask why user could not decide on usefulness without viewing the GQM plans
	Text of user reaction (1)	Answer: »(a) it was not clear whether the department head or the technical leader viewpoint matches the needed project manager perspective better; (b) there was no information on how complete the GQM plans were«
	OMMS action (2)	Ask why position 6 is preferred over position 4
	Text of user reaction (2)	Answer: »Positions 1-3 and 6 are about the development process, whereas positions 4 and 7 are about the inspection process.«
	User reaction (8)	Check out position 1
	User reaction (9)	Check out position 4
	OMMS action (3)	Ask why position 4 is preferred over positions 2 and 3
	Text of user reaction (3)	Answer: »In contrast to position 4, positions 2 and 3 do not cover quality assurance activities.«
Explanatory texts	–	–
Actions to be carried out by OMMS	–	–
Suggestions to the user	–	–

The maintenance team reviews this protocol case as part of its regular analysis activities. The team:

1. asks U. Ser what he means by completeness of GQM plans and adds a new attribute »completeness« for GQM plans. The »completeness« values of existing GQM plans are set to <unknown>; however, the at-

tribute value will be determined for all future GQM plans stored in the OM.

2. seeks a solution for deciding which viewpoint is best for the project manager. It does so by modifying the protocol case shown in Table 3:
   - Removing query 1
   - Defining an »=« filter for »MViewpoint« (meaning that this protocol case will only be retrieved if the user needs a GQM plan for the project manager; however, »MObject« and »Quality focus« may be similar).
   - Adding »There was no GQM plan available for the project manager« as explanatory text
   - Adding »A GQM plan for the technical leader was used which corresponded better to the needs of the project manager than the ones for the department head« as suggestions to the user.
   - Adding the action »ask user« as an action to be carried out by the OMMS with the following prompt: »Hypothesis: Whether a GQM plan from the viewpoint of the department head should be preferred over a GQM plan from the viewpoint of the technical leader depends on whether the department head typically acts as the project manager. For organizations in which the line and project managers are not the same, the project manager is more similar to the technical leader than the department head.
     (a) Would you agree with this?
     (b) If not: why not?
     (c) What other factors influence the similarity between the project manager and the department head/ technical leader?«

The latter OMMS action exemplifies how the users can be involved *actively* in the improvement of the OMMS while they are trying to solve a similar problem themselves. Alternatively, the users could be called and asked for their opinion (e.g., as members of a steering committee). However, in this case they would be asked without having a concrete problem at hand. Therefore, the quality of the answer is likely to be lower than if they have to solve a similar problem anyhow. In the particular example presented, the user will be asked only if he looks for a GQM plan for a project manager that has a similar measurement object, quality focus and context. The OMMS can ensure that each user is asked each question at most once.

7 Conclusion

In this paper we have presented a method for improving the perceived usefulness of an organizational memory incrementally through user feedback. It is based on a general usage model, a cause-effect-model for usefulness as perceived by the user, a set of indicators for improvement potential, and an organizational level case-based reasoning approach (based on protocol cases) that is flexible concerning the degree of using fully automated interpretation of the cases by the organizational memory management system or manual case analysis and interpretation by the organizational memory maintenance team. The organizational memory improvement method considers the practical constraints typically encountered in industrial environments, e.g., limited time of users. Currently both our general organizational memory approach as well as the improvement method are validated within a number of industrial and in-house projects. It is expected that the method will lead to quick improvements after the introduction or extension of an organizational memory. Later on, the change rate will decrease. Thus, the additional effort requested from the users for their feedback will be limited in time.

Acknowledgements

The authors would like to thank Christiane Gresse von Wangenheim for providing the exemplary GQM plan.

References

[AA96] Klaus-Dieter Althoff and Agnar Aamodt. Relating case-based problem solving and learning methods to task and domain characteristics: Towards an analytic framework. *AICom - Artificial Intelligence Communications*, 9(3):109–116, September 1996.

[ABT98] Klaus-Dieter Althoff, Frank Bomarius, and Carsten Tautz. Using case-based reasoning technology to build learning organizations. In *Proceedings of the the Workshop on Organizational Memories at the European Conference on Artificial Intelligence '98*, Brighton, England, August 1998.

[ADK98] Andreas Abecker, Stefan Decker, and Otto Kühn. Organizational memory (in German). *Informatik-Spektrum*, 21(4):213–214, August 1998.

[Alt97] Klaus-Dieter Althoff. Evaluating case-based reasoning systems: The Inreca case study. Postdoctoral thesis (Habilitationsschrift), University of Kaiserslautern, 1997.

[AR99] Klaus-Dieter Althoff and Michael M. Richter. Similarity and utility in non-numerical domains. In W. Gaul and M. Schader, editors, *Mathematische Methoden der Wirtschaftswissenschaften*. Physica-Verlag, Heidelberg, Germany, 1999.

[AW91] K.-D. Althoff and S. Wess. Case-based knowledge acquisition, learning and problem solving in diagnostic real world tasks. *Proceedings of the Fifth European Knowledge Acquisition for Knowledge-Based Systems Workshop*, pages 48–67, 1991.

[BB91] Bruce H. Barnes and Terry B. Bollinger. Making reuse cost effective. *IEEE Software*, 8(1):13–24, January 1991.

[BCR94] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. Experience Factory. In John J. Marciniak, editor, *Encyclopedia of Software Engineering*, volume 1, pages 469–476. John Wiley & Sons, 1994.

[BDR96] Lionel C. Briand, Christiane M. Differding, and H. Dieter Rombach. Practical guidelines for measurement-based process improvement. *Software Process*, 2(4):253–280, December 1996.

[BR88] Victor R. Basili and H. Dieter Rombach. The TAME Project: Towards improvement–oriented software environments. *IEEE Transactions on Software Engineering*, SE-14(6):758–773, June 1988.

[CEM96] The CEMP Consortium. Customized establishment of measurement programs. Final report, ESSI Project Nr. 10358, Germany, 1996.

[Coh89] P. R. Cohen. Evaluation and case-based reasoning. In K. Hammond, editor, *Proceedings of the Second DARPA Workshop on Case-Based Reasoning*, pages 168–172. Morgan Kaufman, 1989.

[Coo97] William S. Cooper. On selecting a measure of retrieval effectiveness. In K.S. Jones and P. Willet, editors, *Readings in Information Retrieval*, pages 191–204. Morgan Kaufmann Publishers, 1997.

[Dam98] Leela Damodaran. Development of a user-centered IT strategy: A case study. *Behavior and Information Technology*, 17(3):127–134, 1998.

[GABT98] Christiane Gresse von Wangenheim, Klaus-Dieter Althoff, Ricardo M. Barcia, and Carsten Tautz. Evaluation of technologies for packaging and reusing software engineering experiences. Technical Report IESE-Report No. 055.98/E, Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern (Germany), 1998.

[GB97] Christiane Gresse and Lionel Briand. Requirements for the Knowledge-Based Support of Software Engineering Measurement Plans. In *Proceedings of the Ninth International Software Engineering and Knowledge Engineering Conference (SEKE'97)*, pages 559–568, Madrid, Spain, June 1997.

[GKP+83] J. Gaschnig, P. Klahr, H. Pople, E. Shortliffe, and A. Terry. Evaluation of expert systems: Issues and case studies. In F. Hayes-Roth, D.A. Waterman, and D.B. Lenat, editors, *Building Expert Systems*, pages 241–282. Addison-Wesley, Reading, Mass., USA, 1983.

[GXG98] Avelino Gonzales, Lingli Xu, and Uma Gupta. Validation techniques for case-based reasoning systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 28(4):465–477, July 1998. Part A: Systems and Humans.

[Har92] Stephen P. Harter. Psychological relevance and information science. *Journal of the American Society for Information Science*, 43(9):602–615, October 1992.

[Hen97] Scott Henninger. An evolutionary approach to constructing effective software reuse repositories. *ACM Transactions on Software Engineering and Methodology*, 6(2):111–140, April 1997.

[HS93] B. Henderson-Sellers. The economics of reusing library classes. *Journal of Object-Oriented Programming*, (4):43–50, 1993.

[JK98] Matthias Jarke and Ralf Klamma. Innovation based on computer-aided failure management: Results of the BMBF project FOQUS (in German). In *Proc. Wirtschaftsinformatik*, 1998.

[Kir94] S. Kirchhoff. *Mapping Quality of Knowledge-Bases Systems: A Methodology for Evaluation (in German)*. Josef Eul Verlag, Bergisch-Gladbach, Germany, 1994.

[Lim96] Wayne C. Lim. Reuse economics: A comparison of seventeen models and directions for future research. In Murali Sitaraman, editor, *Proceedings of the Fourth International Conference on Software Reuse*, pages 41–50, Orlando, Florida, USA, April 1996. IEEE Computer Society Press.

[MD97] Michael G. Morris and Andrew Dillon. How user perceptions influence software use. *IEEE Software*, 14(4):58–64, July/August 1997.

[Nic98] Markus Nick. Implementation and Evaluation of an Experience Base. Diploma thesis, Fraunhofer IESE, University of Kaiserslautern, 1998.

[NT98] Markus Nick and Carsten Tautz. Practical evaluation of an organizational memory using the goal-question-metric technique. Technical Report IESE-Report No. 063.98/E, Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern (Germany), 1998.

[NT99] Markus Nick and Carsten Tautz. Practical evaluation of an organizational memory using the goal-question-metric technique. In *Proceedings of the Workshop on Knowledge Management, Organizational Memory and Knowledge Reuse during Expert Systems '99 (XPS-99)*, Würzburg, Germany, March 1999.

[Pfe96] T. Pfeifer, editor. *Knowledge-Based Systems in Quality Management (in German)*. Springer-Verlag, 1996.

[PZ96] T. Pfeifer and T. Zenner. Using experience during failure analysis - the application of case-based techniques (in German). In R. Grob and J. Spiekermann, editors, *Workshop at the Third German Conference on Expert Systems*, pages V1–V12. Technical Report LSA-95-04, University of Kaiserslautern, 1996.

[SM83] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill Book Co., New York, 1983.

[SW88] J. R. Slagle and M. R. Wick. A method for evaluating candidate expert system applications. *AI Magazine*, 9(4):44–53, 1988.

[Tec99] CBR-Works. URL http://www.tecinno.de/english/products/cbrw_main.htm, 1999. tecInno GmbH, Germany.

[TG98] Carsten Tautz and Christiane Gresse von Wangenheim. REFSENO: A representation formalism for software engineering ontologies. Technical Report IESE-Report No. 015.98/E, Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern (Germany), 1998.

[TG99] Carsten Tautz and Christiane Gresse von Wangenheim. A representation formalism for supporting reuse of software engineering knowledge. In *Proceedings of the Workshop on Knowledge Management, Organizational Memory and Knowledge Reuse during Expert Systems '99 (XPS-99)*, Würzburg, Germany, March 1999. http://www.aifb.uni-karlsruhe.de/WBS/dfe/xps99.proc.htm.

[vW96] Bert van Wegen. *Impacts of KBS on Cost and Structure of Production*. PhD thesis, 1996.

[Wes95] S. Wess. *Case-Based Reasoning in Knowledge-Based Systems for Decision Support and Diagnostics (in German)*. PhD thesis, University of Kaiserslautern, 1995.

Developing a Tailored Reuse Repository Structure
-Experience and First Results-

Raimund L. Feldmann

University of Kaiserslautern
Department of Computer Science
Software Engineering Group
Postfach 3049
67653 Kaiserslautern, Germany
r.feldmann@computer.org

ABSTRACT

Learning from experience gained in past projects is seen as a promising way to improve software quality in upcoming projects. Thus, the reuse of patterns or code components to support software development is widely accepted in research and industry. Furthermore, some approaches require the comprehensive reuses of other forms of knowledge that go far beyond mere software artifacts. This includes, for instance, the reuse of techniques, methods, tools, processes, or even metrics that have already been successfully used in projects. However, the different reusable elements must be stored somewhere, to be easily accessible for later reuse. This calls for a repository that organizes the reusable elements and offers them, on demand, to the (re-)user. In accordance with the applied learning and reuse processes, such a repository must be individually adapted and tailored to the special needs of the organization. It can not be expected that a solution that holds for one organization works for another. This paper describes how a comprehensive reuse repository is developed and tailored for a research organization at the University of Kaiserslautern. First, a HTML-based prototype was installed, to define the repository structure. Details of the resulting current structure are discussed. It is sketched how the structure changed over time, before it was stable enough to serve as a basis for a database-supported implementation.

Keywords

reuse repository, repository structures, comprehensive reuse, process improvement

1 INTRODUCTION

Learning from experience gained in past projects is seen as a promising way to improve software quality in upcoming projects. Here, learning is not only limited to successful projects. Failures, especially, that occurred in past projects have to be carefully analyzed and documented in order to be avoided in the future. As a result, (anti-)patterns, frameworks, and code fragments are being developed to capture the gained experience of already developed software [15]. But experience is not only represented in the form of (directly) reusable software artifacts. To allow comprehensive reuse, as proposed in [6], a large variety of different reusable elements exists, fixing success factors and reasons for failures. For example, process descriptions are used to precisely describe how to develop software in a certain domain, or lessons learned [7] can be used to store qualitative experiences. Consequently, every kind of (software engineering) experience, independent of its type of documentation, is regarded as *experience element* [12] in the remainder of this paper.

To be able to (re-)use the experience elements, they have to be systematically collected and stored. A system for their search and retrieval has to be defined. As a result, reuse repositories are suggested to face this problems. Unfortunately, there exists no universal reuse repository that is suitable for all kinds of organizations. The reason for this is that a reuse repository has to be tailored to the applied reuse process. It should also support the learning process, which, of course, has to be defined individually for each organization. In the following, an approach for the development of such a tailored reuse repository is described.

At the University of Kaiserslautern the Sonderforschungsbereich 501 (SFB 501) ‘Development of large Systems with generic Methods’, a long-term strategic research activity funded by the Deutsche Forschungsgemeinschaft (DFG) was established. Its goal is the development and evaluation of a set of techniques, methods, and tools that support fast reliable customization of complex domain-specific software systems. Hence, several reuse approaches are being developed and tested in different *exper-*

iments[1]. To store the gained knowledge, captured in all kinds of experience elements, the SE-Laboratory of the SFB 501 develops and operates a reuse repository that is integrated in the basic learning cycle of the SFB 501. A simple, HTML-based prototype implementation of the repository was used to define the repository structure and requirements, for the final implementation. Currently, the gained structure is transferred onto an object-relational database management system (ORDBMS), which will replace the prototype in the near future and offers an extended functionality.

The remainder of this paper is organized as follows: Section 2 motivates the development of the reuse repository by describing its integration in a learning cycle. A short usage scenario presents the intended functionality. In Section 3 it is presented, how the repository structure was defined and tailored with the help of a prototype implementation of the repository. How the structure evolved in time, before it was suitable to be used for the database-supported implementation of the repository (Section 4), is discussed in detail. Finally, related work is described in Section 5 and the results are summarized in Section 6.

2 INTEGRATING THE REUSE REPOSITORY IN A BASIC REUSE CYCLE

This section shortly describes the basic learning cycle in which the reuse repository is integrated within the SFB 501 organization, and presents a usage scenario to motivate the development of the repository structure.

On the one hand, a learning cycle has to support continuous improvement by: a) transferring experience from former experiments into new ones, b) systematically analyzing each experiment to gain new or validate existing experience, and c) packaging (i. e., (re-)structuring and storing) the results from the analysis, so that they can be used as valuable input for future experiments. This requires precise and clear definition of the learning cycle. But on the other hand, it has to be easily adjustable to the different experiments that are performed in the SFB 501, and therefore, can not be too restrictive. A basic learning cycle, which meets these requirements, is an adaptation and generalization of the Quality Improvement Paradigm (QIP) as described by Basili et. al. [5], where each experiment is conducted in four steps: plan, execute, analyze, and package. At the beginning, a new experiment is carefully planned, and its (learning) goals are defined before it is executed. During these first two steps, already existing experience can be reused (a). In the next step, the result of the experiment is analyzed (b), before the (hopefully) newly-gained experience is fixed in the last step (c). The experience can then serve as input for the planning of new experiments, and the learning cycle starts again.

Fig. 1 illustrates how the reuse repository is integrated into this basic learning cycle. A distinction is made between two (logically, not physically) different sections of the repository: the *experiment-specific section* and the *organization-wide section*. The experiment-specific section stores all information concerning single experiments, and therefore, is filled while an experiment is planned, executed, and analyzed. It can be compared to a project-database and serves as a basis for the last step, which packages the gained experience into the organization-wide section. The organization-wide section stores the packed experience elements that are relevant to several experiments of the organization and, on demand, offers them as input for new projects in the planning and executing steps. Note that both sections together form the reuse repository and are not physically disjunct. The following scenario motivates the developed reuse repository structure, which includes the experiment-specific section.

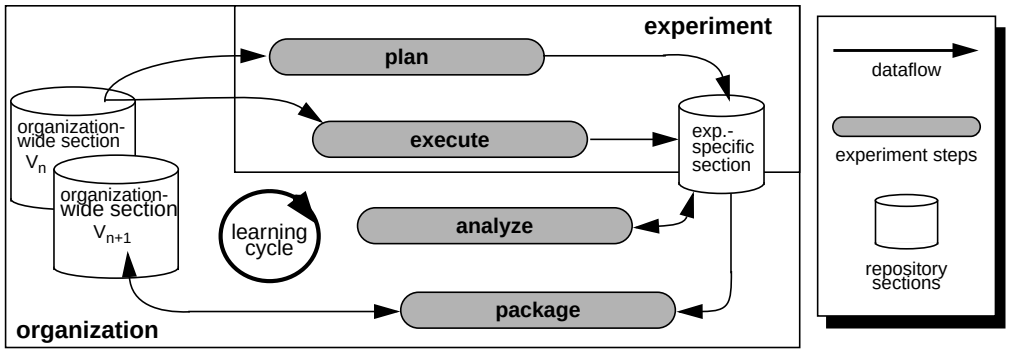

Fig. 1: Integration of the reuse repository in a basic learning cycle

Let us assume that a new experiment (remember, in the SFB 501 each project is seen as an experiment) is planned according to the four steps of the described learning cycle. As in common project environments, the new experiment first has to be charac-

1 All SFB 501 projects are regarded as *experiments* because their main focus is to learn about, and improve, the new approaches (i. e., the developed techniques, methods, or tools), and their minor focus is on the (software) product development itself.

terized by means of "*What are the deliverables?*", "*In which environment is it to be conducted?*", and "*Are there any time restrictions?*". This characterization serves as a basis for searching the organization-wide section of the repository for reusable experience elements. Hence, each experience element should be stored with additional information, to allow easy formulation of such search queries. Most of the time, there will be no exact match between the formulated search request and experience elements in the organization-wide section. Therefore, similarity-based search and retrieval should be offered by the repository, to retrieve experience elements that are relatively close to the requested ones. To gain more information about the retrieved reuse candidates, it would be useful to see experiments in which they have been successfully used, or from which they have been derived in the package step. A closer look at the old experiments, which are stored completely in the experiment-specific section of the repository, could even result in further reusable experience elements. As an example, one could find information about tools that have been successfully used together with a process model that is going to be reused in the new experiment.

According to the underlying reuse process, more reusable experience elements (e. g., patterns or code fragments) are retrieved and used during the execution of the new experiment. The complete experiment documentation is recorded and stored together with the results of the analyzing step in the experiment-specific section. The analyzed data can then be used to systematically improve and/or adapt the (re-)used technologies, process models, or components that were used in the actual experiment. Even if no problems occurred and everything worked out fine, the results are useful for future experiments: this is because the used experience elements can be trusted more, since they have been successfully tested in practice, and therefore, can guarantee a minimum of quality assurance in an experiment, if they are selected for reuse.

3 DEVELOPMENT OF THE REPOSITORY STRUCTURE

Subsequently, it is reported how a suitable reuse repository structure, tailored to the needs of the SFB 501, was defined. The used strategy aims to quickly run a prototype of the repository, without offering the complete final intended functionality. Because the SFB 501 was already successfully running an intranet for information exchange between the participating research groups, and most of the requested requirements for the prototype were met by web technologies, it was decided to use HTML-pages for the prototype implementation[2]. In the following sections, the modular repository structure (Section 3.1), relations and attributes used to characterize the stored experience elements (Section 3.2), and elements of the HTML-based prototype implementation (Section 3.3), are described.

3.1 Defining the Modular Repository Structure

Right from the beginning, it was decided to further subdivide and structure the two repository sections. The idea was to cluster similar types of experience elements together into different logical *areas* [12] that are utterly disjunct. This kind of modularization was chosen mainly for three reasons: First, to allow an iterative implementation and growth of the repository, by simply adding one area after the other to the already implemented parts. Second, to give (re-)users of the repository a first orientation on which types of experience element can already be found in the repository, and where they can be found, since it was not planned to develop and offer an advanced search and retrieval mechanism for this first prototype implementation. And third, to support the definition and development of the repository structure. The last point needs some more explanation: Before an experience element is added to the repository, it can be classified with the help of the areas (since all areas are disjunct and store only experience elements of a similar type). When an experience element does not fit exactly into one of the already defined/implemented areas, there are two solutions:

- If the experience element does not fit into one area because it contains characteristics of more than one area, it must be split into smaller pieces that belong to the different areas.
- If the experience element does not fit into one area at all, possibly a new area is found that must be added to the repository structure and might be implemented in the prototype later.

Both solutions help by either finding the right granularity of an experience element, or by extending and completing the structure of the repository.

At the beginning, the experiment-specific section was subdivided into two areas called *case studies* and *controlled experiments*, which store the experiments of the respective types[3]. For each area a template was defined in accordance with the first three steps of the learning cycle, providing a structure for documenting complete experiments. These predefined templates are

2 The complete list of the requirements and the HTML-page style guide for the prototype installation of the repository is documented in [14].

models	**technologies**	**qualitative experience**	**SE-glossary**	**literature**	**domain-specific knowledge**		
process modeling -process models -product models -resource models	**technologies** -techniques -methods -tools			**literature** -research group 1 : -research group n			
			glossaries -SE -SDL -GQM	**literature** -references -on-line documents -contact addresses		**component repositories** -code -SDL patterns	
process modeling -domain-specific models -domain-independent models	**technologies** -techniques -methods -tools	**qualitative experience**	**background knowledge** -glossaries -literature -domain-specific knowledge			**component repositories** -code -SDL patterns	**measure-ment**

Tab. 1: Changes in the repository's module structure of the organization-wide section

filled out while an experiment is conducted (i. e., planned, executed, and analyzed). The current structure of the experiment-specific section remains more or less the same. The only changes over time occurred in the template structure. Here, three more specific templates replaced the initial one for the case studies area [10]. This fact allowed an improved structuring of the area by further subdividing it into three (sub-)parts. One for each kind of experiment documentation according to the new templates. As will be seen later, such *area refinements* could be introduced not only in the case studies area.

As illustrated in Tab. 1, the structure of the organization-wide section was initially subdivided into six different areas, each without area refinements (First row in Tab. 1).

The *models area* was defined to store all kinds of models that are developed or used in the SFB 501. Soon it turned out that this definition could be redefined more precisely, since only experience elements related to process modeling topics were stored in this area. First, renaming the complete area was avoided, because other types of models were still expected to occur over time. Therefore, the area was only further subdivided into three area refinements called *process models*, *product models*, and *resource models*. This would still allow to store other models by simply adding another area refinement to the structure. But finally, after two major structure changes, and the request to store the (process / product / resource) models according to domain-specific and domain-independent models, the area was renamed into *process modeling area* subdivided into two area refinements. Note that each row in Tab. 1 indicates another major version of the structure. The last row represents the current structure that serves for the database supported implementation. Columns represent the different areas. Each definition of an area is valid for the rows below, unless it is replaced by an other definition.

Different descriptions of techniques, methods, and tools are stored in the *technologies area*. Therefore, the area is subdivided into three area refinements. Experience elements stored in this area contain basic information about the technologies and help to select the appropriate techniques when setting up a new experiment. The SDT (SDL Design Tool) package, for instance, helps newcomers get into the SDL development environment and is therefore stored in the area refinement that is called *tools.*

All experiences that were not planned to be made in an experiment, but turned out to be useful, are represented in the *qualitative experience area*. This is the only area of the organization-wide section that remained stabile right from the beginning of the prototype implementation. Several attempts have been made to further refine the area, but no satisfactory solution has been developed until now.

The three areas *SE-glossary, literature*, and *domain-specific knowledge* were defined and implemented separately in the first version of the repository's prototype. They where meant to provide all kinds of knowledge about our domain: real-time house automation and the technologies used in the SFB 501. Knowledge concerning, for instance, methods or techniques used, consistent definitions of (software engineering) terms, or expert knowledge, such as a collection of thermodynamic formulas for heating systems, are stored here. When it turned out that changes in the structure of these areas occurred in almost every new version because the definition of these areas and their refinements were too specific, it was decided to pack them together into one big area called *background knowledge*.

3 SFB 501 experiments are either *case studies*, as defined in [20], or *controlled experiments* (sometimes also denoted as 'formal experiments').

One area that was not part of the first version of the structure is the *component repositories area*. It contains directly reusable components and was added soon after the first prototype was running. Currently, it is subdivided into two area refinements for *code* and *SDL patterns*.

The newest area in the organization-wide section is called *measurement area*. It stores predefined measurement goals and product measures that can be easily adapted to new experiments. Currently, this area is not further refined.

As described, areas are the main building blocks of the repository structure. It turned out that this kind of modularization was a good choice, not only in combination with the HTML-based prototype implementation. Furthermore, the module structure can be used to build specialized repositories inside the global one, simply by using only some areas or area refinements [13,16].

3.2 Characterization of Experience Elements

Just storing the different experience elements in the areas and sections would not allow the definition of an advanced search and retrieval system as required for a reuse repository. This is especially true when the experience elements are stored in the repository in heterogeneous formats (e. g., PostScript-files, GIFs, or word processor binary-files) that can not easily be searched for keywords. Furthermore, additional information describing the environment in which an experience element can be (re-)used must be provided.

As described in [14], the prototype implementation of the repository uses a Unix file system to store the experience elements. Basic describing attributes, like creation date, last modification date, or the owner/author of an experience element, were used directly from the file system. Therefore, at the beginning, links to the copies of the experience elements in the file system, inside the repository's HTML-pages were only accompanied by a short description and a *context vector* [12]. The context vector characterizes the environment / type of experiment in which the experience element is valid (i. e., can be reused). It simply consists of a collection of attribute-value-pairs, for example, ⟨(project effort, 50.000 h), (programming language, JAVA), …⟩. Soon it became obvious that additional information must be associated and stored with each experience element, to provide the basis for an advanced search and retrieval system. Relations, for instance, indicating how experience elements in different areas interrelate, must be added (e. g., when an experience element was split into smaller pieces), or attributes that not only describe the context of the experience element, but the experience element itself.

As a result, the concept of the *characterization vector* was developed. Characterization vectors (CV) are used to store all information concerning experience elements. Therefore, they are a major concept in the schema of the repository. The information contained in a CV can be divided into three segments. One for references to the different representations of the experience element, one with attributes that describe the experience element and its validity, and one holding information about relations from the experience element to other experience elements.

References
CV references are a *set of links* to different possible representations of one experience element. For instance, if the experience element is a requirements document, one link could reference the original word processor binary-file in which the document was written. Another one could be a reference to a HTML version or graphical representation, to allow direct viewing of the experience element. A third one might be a PostScript version to allow easy printouts of the document. In the prototype implementation of the repository, all CV references are realized by HTML-Links to the experience elements in the file system.

Attributes
The CV attributes serve as the basis for all search functions offered by the repository. They integrate the attributes of the context vectors as well as all other describing attributes, including the ones that are offered by the standard file systems. Additionally, attributes that describe to which section, area, or area refinement an experience element belongs have to be included, too. This is because of the intended final implementation of the repository with the help of an database system. Here all attributes have to be integrated into the database schema. Hence, they have to be defined in the CV as well, since per definition they store *all* information concerning an experience element! A detailed list with all CV attributes can be found in [11].

Relations
CV relations are attributes that are used to express relationships from a single experience element to other experience elements. Since the CV relations connect the different areas of the repository structure, they are discussed in detail. Currently, the repository structure defines six different types of CV relations:

1. `measures/measured_by:` This type of relation is defined between experience elements of the measurement area and experience elements stored in the process modeling, technologies, qualitative experience, or component repositories areas. It is used to identify the experience elements that are measured (e. g., by a standardized questionnaire that is stored in the

measurement area). In the other direction, it helps the planner of an experiment to find a (measurement) experience element that measures a certain element s/he is reusing.

2. `is_about/has_part`: This type of relation is defined between lessons learned stored in the qualitative experience area and experience elements of the process modeling, technologies, component repositories, or measurement areas. It is used to identify the experience element(s) that the lesson learned is concerned with. In the other direction it helps to find lessons learned that deal with a certain experience element.
3. `used_in/uses:` This type of relation is defined between experience elements of the organization-wide section and the experiment documentations in the experiment-specific section (i. e., the case studies area and the controlled experiments area). It allows to identify the experiments in which an experience element has been used, or which experience elements have been used in a certain experiment. Note that since it can be assumed that an experience element that has been (successfully) reused often is quite valid. Therefore, the number of used_in relations for an experience element gives a first impression about its validity.
4. `gained_in/gains:` This type of relation is defined between a lesson learned in the qualitative experience area and the experiment documentations in the experiment-specific section. It identifies in which experiment a lesson learned was gained, or which lessons learned have been gained in a certain experiment.
5. `uses/used_with:` This type of relation is defined between experience elements of the process modeling area and the component repositories area. It documents that a certain experience element is used in the description of a process model. The opposite direction gives hints as to which process descriptions a certain experience element is used or its usage is suggested. This type of relation is further defined between experience elements of the process modeling area and the technologies area, and between experience elements of the technologies area and the component repositories area.
6. `explains/explained_in`: This type of relation is defined between an experience element of the background knowledge area and experience elements of all other areas. It describes where background knowledge about a certain experience element can be found, or helps to find examples where the background knowledge is already used.

Fig. 2 summarizes the framework defined by the different areas and CV relations. With these defined relations it is now possible to offer another search and retrieval functionality. Let us assume that a first search, based on the CV attributes, retrieved

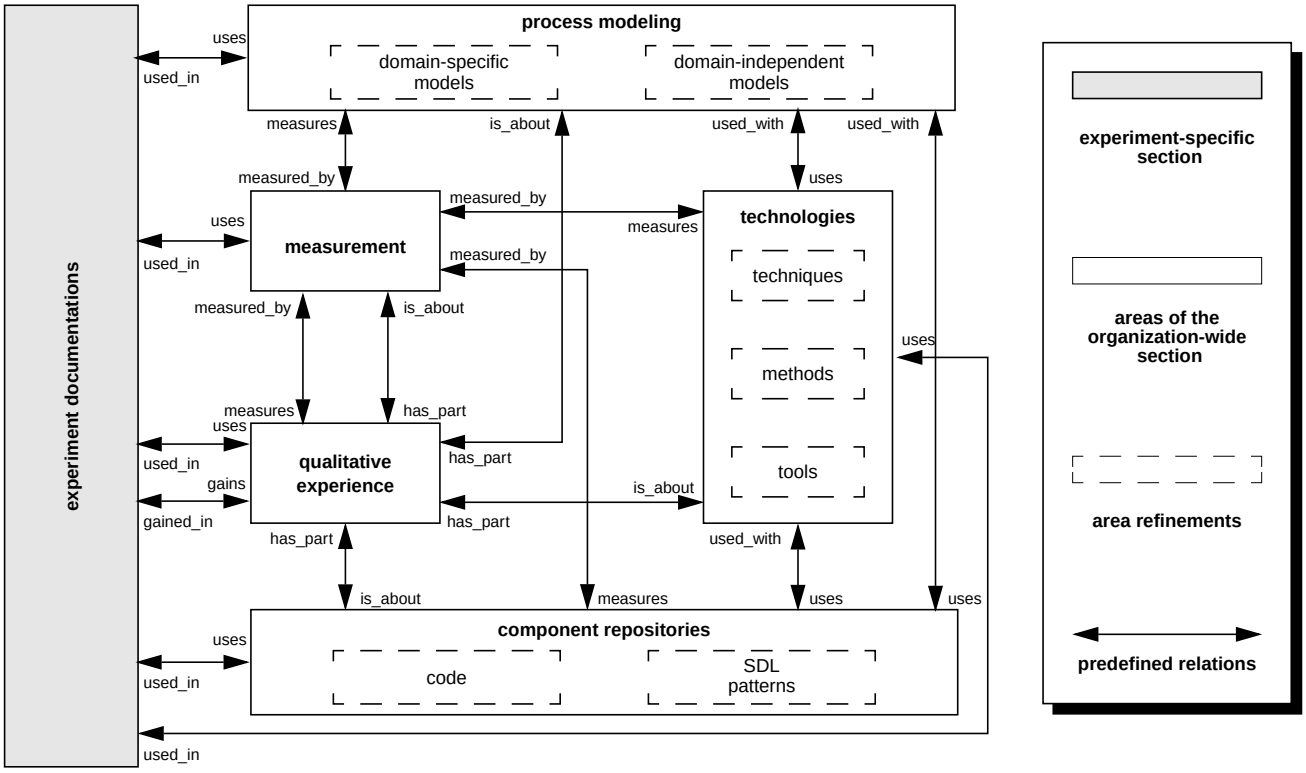

Fig. 2: CV relations between the different sections/areas of the repository (background knowledge area not included)

several experience elements that all meet the search requirements to exactly the same extend. Now one can follow the CV relations to get further information that help to decide which experience element should be reused.

Overall, the HTML-based prototype implementation of the repository positively influenced the development of the characterization vector concept. This is especially true for the CV references and CV relations. On the other hand, the HTML-based implementation showed some shortcomings in defining the complete list of attributes that have to be included into the CV for a database implementation. But this shortcoming might also be influenced by the fact that it was never intended to provide an advanced search and retrieval system functionality with the prototype implementation. Some examples of implemented prototype functions are given in the next section.

3.3 Elements of the HTML-based Prototype Implementation

To define a tailored reuse repository structure, it is important to receive fast feedback on how parts of the repository structure are accepted (i. e., used). Here again, the HTML-based prototype implementation turned out to be extremely useful. First, because all changes to the repository structure are accessible to all repository users as soon as they are implemented, without any further actions. This is due to the fact that the changes must only be made centrally in the file system on which the web server is operating. They will be automatically distributed to all users who are using the repository (i. e., requesting information from the web server). Second, because all requests for experience elements stored in the repository can be logged by the web server that provides the repository. With the help of this information it is possible to create usage statistics of the repository. The access to newly-structured or added areas of the repository, especially can be visualized. Consequently, a tool was developed that automatically generates the usage and access statistics to the repository [17]. Fig. 3 shows two of the plots generated by this tool. The first plot illustrates the usage of the repository (right grey bar) and other tools, like compilers or editors, which are installed in the SE-Laboratory of the SFB 501. Here it turned out that the repository is widely accepted by the users, if compared to other tools. The second plot shows how many experience elements have been requested during the year 1998. Again, it can be seen that the repository is frequently accessed throughout the year and its usage is not only limited to a special period. The relatively high number of accesses in March can be explained, because during this time a major change in the repository structure was implemented and therefore, many users just accessed the repository to get an idea of its new structure. Additional plots can be generated that illustrate, for example, the top 20 accessed experience elements of the repository in a month or over the complete year.

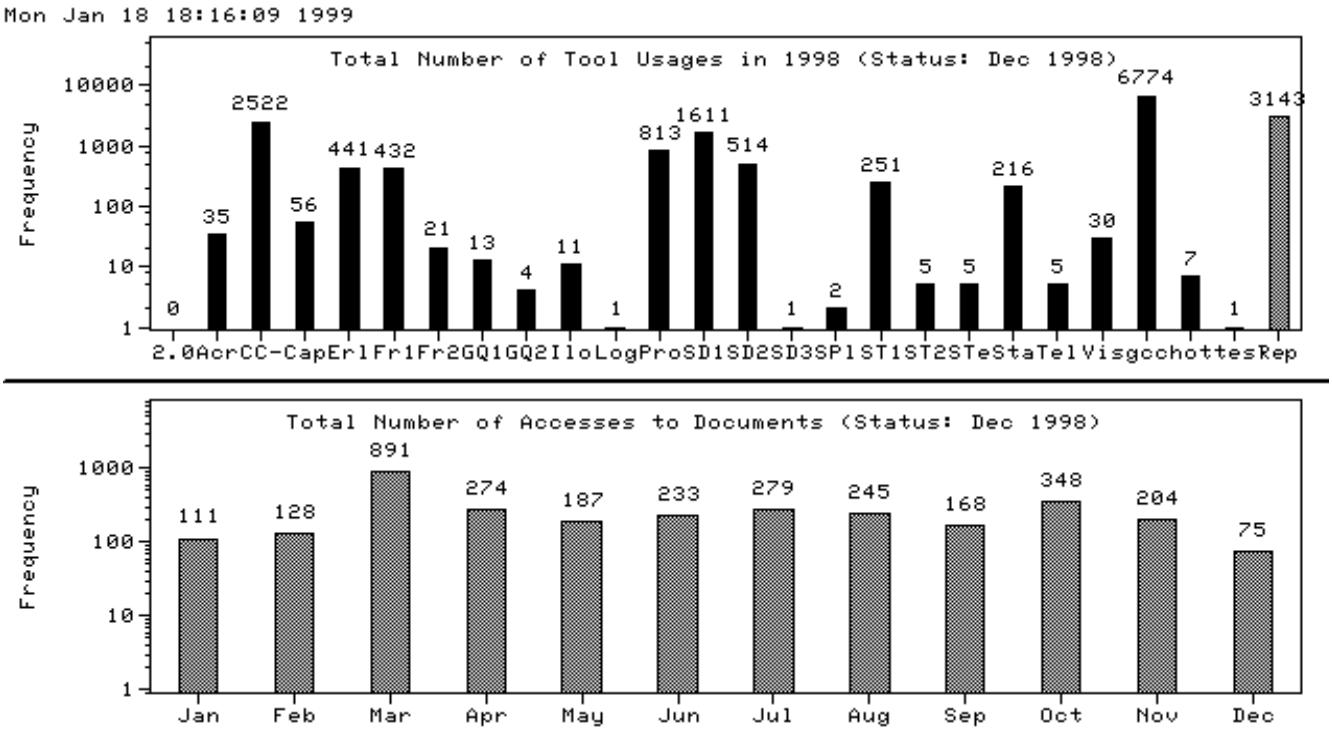

Fig. 3: Usage of the HTML-based prototype of the reuse repository

A look at the usage statistics show that not a problem was caused by the fact that only a prototype of the reuse repository, offering only a simple usage functionality, was provided. The interest of the users in the repository remains stable over time and does not get less because some requested functionality was not implemented. For example, no similarity-based search and retrieval of the experience elements was provided, although it was requested by some users. The users basically have to browse through the repository to find the experience elements they are looking for. Only a simple full text search is supported, as provided for many home pages in the WWW.

4 ORDBMS-BASED IMPLEMENTATION OF THE REPOSITORY

It always was intended to use a database management system (DBMS) to store the experience elements in the final implementation of the reuse repository. When the repository structure (i. e., areas, area refinements, and CV's) became quite stable, it was time to think about a suitable DBMS. In cooperation with the database research group that is part of the SFB 501 project, it was evaluated which kind of DBMS would best serve our purpose [11, 21]. Both relational database systems (RDBMS) and object-oriented database systems (OODBMS) offer features that are useful for the implementation of the reuse repository. Transactions, queries, and views, for instance, as offered by RDBMS are needed to support the search and retrieval of experience elements. Complex objects and user-defined data types, on the other hand, as offered by OODBMS are helpful to implement the repository structure. Therefore, it was decided to use an object-relational database system (ORDBMS) [23]. ORDBMS integrate features of both, OODBMS and RDBMS, and are currently considered to be the most successful trend in DBMS technology. To avoid the cost of new construction from scratch, it was further decided to use one of the commercially available ORDBMS and extend its functionality to meet our needs. Next it is described how some of the features of ORDBMS can be directly exploited to transfer the HTML-based prototype to the final DBMS-based implementation.

In the ORDBMS-based implementation of the repository the different types of experience elements (e. g., PostScript-files, GIFs, or word processor binary-files) are considered as large objects, which are stored in the database by using the management features for CLOBs (character large objects) or BLOBs (binary large objects). To be more precise, each experience element is regarded as a set of CLOBs or BLOBs. This is because of the fact that there can be many CV references for one experience element to allow easy access to its different available representations stored in the repository.

During the execution of an experiment, different tools are used to create, extend or change experience elements that all to be stored in the corresponding experiment documentation of the experiment-specific section. Most of these tools store data in proprietary formats by using the file system and can not directly access the DBMS. While this was no problem for the HTML-based prototype implementation where the experience elements are stored in a Unix file system, problems might occur in a DBMS-based implementation. The problem of the proprietary formats is already met by storing the experience elements as BLOBs or CLOBs. Luckily, ORDBMS offer several mechanisms to extend database processing to data that is not directly stored in the database, but, for example, is held in the file system. These mechanisms are bridging the gap between the database storage system and the (external) file system. Hence, the tools can store experience elements in files, but, simultaneously, they can be accessed and controlled via the database interface of the repository. This concept offers referential integrity, access control, and coordinated backup and recovery for database related file system data. A file system filter intercepts file system requests to the linked experience elements and, thereby, controls access (w. r. t. the database access rights). This means that the different tools can further access experience elements in the file system, but they are closer integrated with the experience elements in the database. In cases where the experience elements written by tools are not in a binary format, it is, furthermore, possible to extend the full-text search offered by the database to this 'external' experience elements. Nevertheless, a 'simple' full-text search is not always possible. Therefore, a similarity-based search system is implemented that operates on the CV attributes associated with each experience element. Here, again, ORDBMS render a substantial contribution. Similarity functions can be realized as so-called *user-defined functions*.

Another advantage of ORDBMS, which is extensively used for the reuse repository, is the predefined infrastructure for connecting a database to the WWW. Thus, appropriate interfaces for a distributed, heterogeneous system environment can be offered as before with the HTML-based prototype. For the user of the repository there is not much of a change in the user interface of the new ORDBMS-based implementation and the former HTML-based implementation of the repository. It is still possible to brows through the repository (i. e., the framework spanned by the areas and CV relations) by simply clicking some links. The query used to access the database behind the link is transparent for the user.

5 RELATED WORK

The description of related work is twofold: First, we take a look at the structure of existing reuse repositories in Section 5.1. Then, we will compare the technical realization of other repositories with our final implementation, based on an ORDBS

(Section 5.2). Due to space restrictions, a comparison with AI systems (like Case-based reasoning systems [1]), which could be used to support the intelligent search and retrieval of experience elements, is beyond the scope of this paper. The same holds for approaches that focus on integrating reuse repositories into a certain (learning) process, in order to build up a corporate memory.

5.1 Repository Structures

In [19] Henninger discusses a ***repository for reusable software components***. The paper focuses on the indexing structure of such a repository. A method is suggested of how the index of a repository can be implemented "*with a minimal up-front structuring effort*" and be incrementally refined while the components are reused. The evolutionary construction of such repositories starts with the *"Repository Seeding"*. That is, a rudimental set of reusable components is stored in the repository with a simple, basic index structure to get the reuse activities started. Then, while the components are being reused, the index structure, which is used for finding components in the repository, is incrementally improved, according to the practical needs of the repository users. - The idea of starting the reuse process in a state of incompleteness to gain practical results as the basis for incremental improvement is similar to our approach. The information stored in our repository becomes the more detailed, the more experience elements are being reused, caused by the growing number of relations (e. g., the `used_in/uses` relation) between the experience elements. However, Henninger offers no predefined relation structure between the components, which is an essential part of our repository structure.

The ***Experience Factory (EF)*** approach by Basili et. al. [4] describes an organizational approach for building up software competencies in a company and transferring them systematically into projects. As a main part of the approach, a repository called ***Experience Base*** stores all relevant knowledge and offers it to new projects of the organization on demand. The idea of comprehensive reuse of all kinds of software engineering experience that is manifested by the experience base is also realized in our approach. In contrast to Basili, our reuse repository includes the single project databases in the form of complete experiment documentations in the experiment-specific section. Nevertheless, our repository structure can be seen as a specialized instantiation of an experience base tailored to the SFB 501 environment. However, the main concern of an experience factory is the establishment of a global organizational improvement process, thus, no fixed structure for the reuse repository (i. e., the experience base) is given. Our focus, on the other hand, is on the repository itself.

Finally, the ***ASSET Reuse library WSRD*** [22] is an example of a web-based implementation of an object repository. The WSRD reuse library is a domain-oriented reuse repository that contains more than 1,000 experience elements, dealing with topics such as software reuse practice or the Y2K problem. The repository is organized according to certain domains and collections that offer a mixture of different experience elements, like lessons learned, process models, or code fragments. Therefore, a certain domain (or collection) that stores experience elements from similar, but yet different projects in one entry can be compared to an experiment documentation of our experiment-specific section. However, complete project documentations are not an integral part of WSRD. Cross-references that interrelate the entries in the collections are offered, but they are much more general and unstructured than the relations defined in our repository.

5.2 Technical Realizations of Repositories

Here, we first compare our approach with a repository of the Arcadia project [25], in which several object management systems [18,24,27] have been developed to support the various needs of their process-centered software engineering environment. Triton [18] is one of those object managers. The whole system is based on the Exodus [8] database system toolkit to avoid the cost of new construction from scratch, just like we are using the standard functionality of a commercially available ORDBMS and extend it, in accordance with our needs. Heimbigner describes Triton in [18] as follows: "*It is a serverized repository providing persistent storage for typed objects, plus functions for manipulating those objects*". Whereas we, on demand, link the stored experience elements from the ORDBMS to the file system, so that different (commercial) tools can use them, Triton uses Remote Procedure Calls (RPC) for communication between clients programs (tools) and the repository. Therefore, the server offers a procedural interface to its clients, acting as a kind of library of stored procedures. They are accessible from programs written in a variety of programming languages (such as Ada, C++, or Lisp). But this is a limitation that we can not accept, since most of the (commercial) tools that we are using in our environment are not offering the needed functionality to use RPC's and can not be modified because their source code is mostly not available. To overcome this problem, we could have introduced mediators as described in [26] to bridge the gap between a repository like Triton and our tools. But this would call for a wide variety of mediators, basically one mediator for each tool of our environment. For the same reason, we consider experience elements as CLOBs and BLOBs to keep the original storage formats required by the different tools. Triton, on the other hand, uses a homogeneous storage schema to provide efficient representation for the wide variety of software artifacts.

Consequently all data that is shared by two or more tools has to be converted to the Triton schema, even if a direct data exchange via a format like RTF is available between the tools. Again, this is acceptable in an environment like the Arcadia project, where most of the tools have been developed from scratch, but it is not suitable for an environment like ours' where commercially available tools are heavily in use.

In [2] a hybrid system for delivering marketing information in heterogeneous formats via the Internet is described. An object-oriented client/server document management system based on a RDBMS is used for storing the documents of the repository. Standard attributes, like creation date, author, or a version number are automatically assigned to the repository entries by the document management system. Additional attributes, which further characterize a document, have to be included as HTML metatags. Web servers are running search engines to index the repository with the help of these tags. Since we are not using HTML metatags in our repository, all describing attributes are summarized in the characterization vectors assigned to each experience element. Based on these characterization vectors, we offer tailored search mechanisms, including similarity-based search functions, for the different tasks, whereas the system described in [2] offers the standard web functionality for search and retrieval.

6 CONCLUSION

In this paper an approach for developing a tailored reuse repository for a University research project has been presented. A learning cycle that integrates the repository as one of its central parts has been introduced. It was described, how the repository structure was defined and tailored using a simple prototype. Some of the (technical) features of the web-based prototype implementation have been sketched. Details of the current repository structure, and how it evolved in time, are discussed. The characterization vector concept was presented, and how it can be used to transfer the repository onto a database system in order to effectively support the search and retrieval of different, heterogeneous, experience elements. Finally, reasons for choosing an ORDBMS as a basis for the final implementation of the repository with the resulting structure are given.

As future work, we intend to integrate additional services, like versioning of experience elements and complete experiments, into the ORDBMS-based repository. Furthermore, it is intended to use the repository as a basis for the implementation of different tools that support the software development process. For example, a process modeling tool as described in [3], or a specialized SDL pattern editor [9] could use the repository to directly (re-)use existing experience elements from the organization-wide section in new projects, and store the newly-created experience elements in the corresponding areas of the experiment-specific section. A more detailed evaluation of access and usage statistics could be used to improve the reuse process and the stored experience elements.

ACKNOWLEDGMENT

Part of this work has been conducted in the context of the Sonderforschungsbereich 501 'Development of Large Systems with Generic Methods' (SFB 501) funded by the Deutsche Forschungsgemeinschaft (DFG). Thanks go to my colleagues from the SFB 501, who have been involved in realizing this project and to my students, who implemented large parts of the web-based prototype. Especially Norbert Ritter and Wolfgang Mahnke from the database research group at the University of Kaiserslautern gave valuable insights into the different database systems. My gratitude is extended to our project leader Prof. Dr. H. D. Rombach. Last but not least, I would like to thank Sonnhild Namingha from the Fraunhofer Institute for Experimental Software Engineering (IESE) for reviewing the first version of this paper.

REFERENCES

[1] Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AICom - Artificial Intelligence Communications*, 7(1):39–59, March 1994.

[2] V. Balasubramanian and Alf Bashian. Document Management and Web Technologies: Alice Marries the Mad Hatter. *Communications of the ACM*, 41(7):107–115, July 1998.

[3] Jürgen Münch, Barbara Dellen, Frank Maurer and Martin Verlage. Enriching software process support by knowledge-based techniques. *Special issue of International Journal of Software Engineering & Knowledge Engineering*, 1997.

[4] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. Experience Factory. In John J. Marciniak, editor, *Encyclopedia of Software Engineering*, volume 1, pages 469–476. John Wiley & Sons, 1994.

[5] Victor R. Basili and H. Dieter Rombach. The TAME Project: Towards improvement–oriented software environments. *IEEE Transactions on Software Engineering*, SE-14(6):758–773, June 1988.

[6] Victor R. Basili and H. Dieter Rombach. Support for comprehensive reuse. *IEE Software Engineering Journal*, 6(5):303–316, September 1991.

[7] Andreas Birk and Carsten Tautz. Knowledge Management of Software Engineering Lessons Learned. In *Proceedings of the Tenth International Conference on Software Engineering and Knowledge Engineering (SEKE'98)*, pages 116–119, San Francisco Bay, CA, USA, June 1998. Knowledge Systems Institute, Skokie, Illinois, USA.

[8] Michael Carey, Dave Dewitt, Goetz Graefe, Doug Haight, Joel Richardson, David Schuh, E. Shekita, and S. Vandenberg. The EXODUS Extensible DBMS Project: an Overview. In Stan Zdonik and David Maier, editors, *Readings in Object-Oriented Databases*. Morgan Kaufman, 1990.

[9] Daniel Cisowski, Birgit Geppert, Frank Rößler, and Markus Schwaiger. Tool Support for SDL Patterns. In *Proceedings of the 1st Workshop of the SDL Forum Society on SDL and MSC (SAM98)*, pages 107–115, Berlin, Gemany, 1998. ISSN: 0863-095.

[10] Marion Fechtig. Fixing the case studies' structure for the access and storage system of the experiment-specific section in the SFB 501 Experience Base (in German). Projektarbeit, Dept. of Computer Science, University of Kaiserslautern, Germany, 67653 Kaiserslautern, Germany, January 1998.

[11] Raimund L. Feldmann, Wolfgang Mahnke, and Norbert Ritter. (OR)DBMS-Support for the SFB 501 Experience Base. Technical Report 12/98, Sonderforschungsbereich 501, Dept. of Computer Science, University of Kaiserslautern, 67653 Kaiserslautern, Germany, 1998.

[12] Raimund L. Feldmann, Jürgen Münch, and Stefan Vorwieger. Towards Goal-Oriented Organizational Learning: Representing and Maintaining Knowledge in an Experience Base. In *Proceedings of the Tenth International Conference on Software Engineering and Knowledge Engineering (SEKE'98)*, pages 236–245, San Francisco Bay, CA, USA, June 1998. Knowledge Systems Institute, Skokie, Illinois, USA.

[13] Raimund L. Feldmann and Carsten Tautz. Improving Best Practices Through Explicit Documentation of Experience About Software Technologies. In C. Hawkins, M. Ross, G. Staples, and J. B. Thompson, editors, *INSPIRE III Process Improvement Through Training and Education*, pages 43–57. The British Computer Society, September 1998. Proceedings of the Third International Conference on Software Process Improvement Research, Education and Training (INSPIRE'98).

[14] Raimund L. Feldmann and Stefan Vorwieger. Providing an Experience Base in a research Context via the Internet. In *Proceedings of the ICSE 98 Workshop on "Software Engineering over the Internet"*, http://sern.cpsc.ucalgary.ca/ maurer/ICSE98WS/ICSE98WS.html, April 1998.

[15] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[16] B. Geppert, F. Rößler, R. L. Feldmann, and S. Vorwiger. Combining SDL Patterns with Continuous Quality Improvement: An Experience Factory Tailored to SDL Patterns. In *Proceedings of the 1st Workshop of the SDL Forum Society on SDL and MSC (SAM98)*, pages 97–106, Berlin, Gemany, 1998. ISSN: 0863-095.

[17] Marco Habetz. Tools for supporting the Software Engineering Laboratory of the SFB 501. Technical Report 04/99, Sonderforschungsbereich 501, Dept. of Computer Science, University of Kaiserslautern, 67653 Kaiserslautern, Germany, 1999.

[18] Dennis Heimbigner. Experiences with an object manager for a process-centered environment. In *Proceedings of the Eighteenth VLDB Conference, Vancouver, British Columbia, Canada*, August 1992.

[19] Scott Henninger. Supporting the Construction and Evolution of Component Repositories. In *Proceedings of the Eighteenth International Conference on Software Engineering*, pages 279–288. IEEE Computer Society Press, March 1996.

[20] Barbara Ann Kitchenham. Evaluating software engineering methods and tools, part 1: The evaluation context and evaluation methods. *ACM SIGSOFT Software Engineering Notes*, 21(1):11–15, January 1996.

[21] Norbert Ritter, Hans-Peter Steiert, Wolfgang Mahnke, and Raimund L. Feldmann. An Object-Relational SE-Repository with Generated Services. In *Proceedings of the 1999 Information Resources Management Association International Conference (IRMA99)*, Hershey, Pennsylvania, USA, May 1999.

[22] The ASSET staff. Reuse Library, December 1997. http://www.asset.com/WSRD/indices/domains/REUSE_LIBRARY.html.

[23] M. Stonebraker, P. Brown, and D. Moore. *Object-Relational DBMSs*. Morgan Kaufman Series in Data Management Systems, second edition, September 1998.

[24] Peri Tarr and Lori A. Clark. PLEIADES: An object management system for software engineering environments. In David Notkin, editor, *Proceedings of the First ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 56–70. ACM Press, December 1993. Published as ACM SIGSOFT Software Engineering Notes 18(5), December 1993.

[25] Richard N. Taylor, Frank C. Belz, Lori A. Clarke, Leon Osterweil, Richard W. Selby, Jack C. Wileden, Alexander L. Wolf, and Michal Young. Foundations for the arcadia environment architecture. In Peter Henderson, editor, *Proceedings of the Third ACM SIGSOFT/SIGPLAN Symposium on Practical Software Development Environments*, pages 1–13, November 1988. Appeared as ACM SIGSOFT Software Engineering Notes 13(5), November 1988.

[26] Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, March 1992.

[27] Jack C. Wileden, Alexander L. Wolf, Charles D. Fisher, and Peri L. Tarr. PGraphite: An Experiment in Persistent Typed Object Management. In *Third Symposium on Software Development Environments (SDE3)*, 1988.

Part 2:

Industrial Experiences in LSOs

Transferring Experience: A Practical Approach and its Application on Software Inspections

Frank Houdek[(1)], Christian Bunse[(2)]

[(1)]DaimlerChrysler AG
Research and Technology
P.O. Box 23 60
D-89013 Ulm, Germany
frank.houdek@daimlerchrysler.com

[(2)]Fraunhofer Institute for
Experimental Software Engineering (IESE)
Sauerwiesen 6
D-67661 Kaiserslautern, Germany
christian.bunse@iese.fhg.de

Abstract

Experience and knowledge management are seen as key capabilities for systematic software development and process improvement. However, it is still not quite clear, how to get this vision to work.

In this paper, a process for systematic experience transfer is presented. It covers the activities of experience acquisition, experience documentation and evolution, and experience reuse. This process is a result of the German publicly-funded project SoftQuali, and its practical use is demonstrated by two real project examples, dealing with experience transfer for software inspections. In general it is described how experience can be packaged, both to transfer the technique and to improve it.

Keywords

Experience transfer, inspections, quality pattern, organizational learning, experience management.

1 Introduction

Software is becoming more and more important. This importance leads to an increase in the demand for high quality software products. Software as well as its development processes have to be improved continuously. To do so, best practices have to be identified, maintained, and transferred systematically from one project to another. A continuous, company-wide learning cycle must be established to transfer knowledge, avoid mistakes, and thereby improve both processes and products.

Everyone is willing to accept this high level view. But what does this mean? Up to now, it is not quite clear yet how to make this vision work. What are experiences? Humans always use this word intuitively without defining its meaning. However, there are several possible definitions. But each definition will have a different impact on the derived activities. In the context of this paper experience is understood as *'collection of witnessing and insights gained by a human'* [1]. Strictly speaking, this implies that not experience itself (tuple of witnessing and insight) but experience knowledge (the insight) can be transferred. For the sake of simplicity the term 'experience' is used instead of 'experience knowledge' throughout this paper.

The next question will be 'How can experience be captured, documented, stored?' Experience has to be externalized to go beyond individual and towards organizational learning. To do so, a set of interacting mechanisms for experience acquisition, experience packaging, experience evolution, and experience reuse is needed.

In this paper, an implementation of these activities originating from the project SoftQuali is presented. Its main idea is an aggregation of knowledge from concrete *descriptive* observations to generalized *prescriptive* descriptions. We start the experience transfer process by capturing real-life observations in *observation packages.* An observation packages contains the description of a real-life fact together with a description of the context the observation was made in. In a subsequent step, we can use these observation packages to build more mature *practice packages*. Unlike observation packages, practice packages provide constructive descriptions of processes (e.g. how to perform inspections) or products (e.g. software architecture for ATMs). These descriptions are build upon the concrete observations provided by the observation packages and (potential) external knowledge. By this, the descriptions are tailored towards one environments needs and restrictions. On request, practice packages are provided with or without experts help to all advice-searching people in one enterprise. Figure 1 depicts the entire process graphically.

To document experience itself, we applied the Quality Pattern approach [2]. Quality Pattern support structuring and documenting of experience. Its main idea is a presentation of experience in pairs of problem and solution assuming the following experts' behavior: Every time an

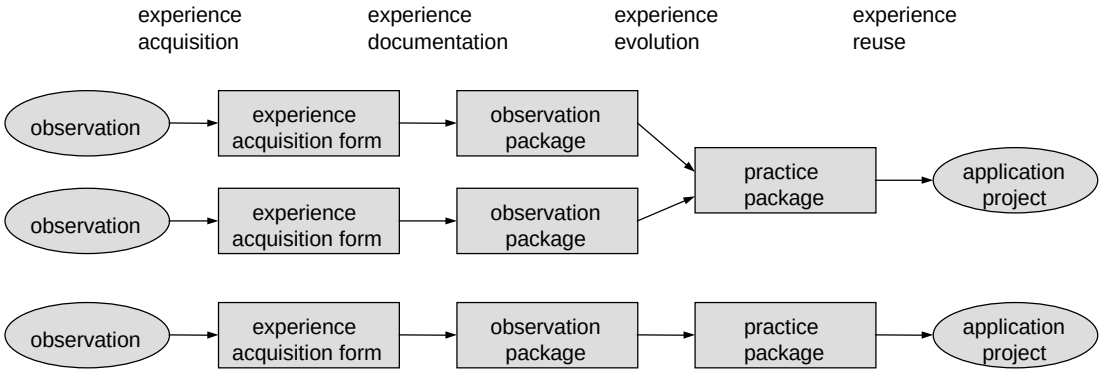

Figure 1: Process of experience acquisition, documentation, evolution, and reuse.

expert has to solve a new problem, she remembers similar, previously solved problems and adapts and combines those solutions to a new one.[1]

In the subsequent sections, the elements of this process are described in more detail. To make the process more comprehensible, two real-life examples are presented, too. Both examples are concerned with process improvement in the domain of software inspections. In the first one, documented experience was used to transfer this technology from one environment to another. In the second example, documented experience was used to improve inspections across several sites.

The main findings of the work presented here can be summarized as follows:

- Quality Patterns and their subtypes are suitable for documenting various types of experience (observations, lessons learned, proposed best practice)
- The evolutionary quality pattern model (which is the maturation from observations to practice packages) supports the evolution of knowledge over time. It helps make the complex area of experience management both more understandable and more usable.
- The approach is applicable in real-life, as shown in the two application examples.

Knowledge and experience are increasingly becoming the main assets of software developing companies. In order to keep or improve competitive advantage, establishment of successful experience transfer mechanisms is becoming crucial. The approach presented in this paper is the first step in this direction

The remainder of this paper is structured as follows. In Section 2, the quality improvement paradigm (QIP) and the results of the PERFECT project are briefly introduced as basis for the work presented in this paper. Additionally the project SoftQuali, its objectives, structure, and partners are described. In Section 3, the question of how to structure and document experience to make it reusable is emphasized. Sections 4 describes the task of experience acquisition. Section 5 gives a short glance at tool support for these activities. In Section 6, the application of the approach, in the domain of software inspections, is presented. Finally, we conclude in Section 7.

2 Context

2.1 QIP and PERFECT

Reuse of experience is a key element of the quality improvement paradigm (QIP) of Basili et al. [4]. In their work, Basili et al. describe a high-level process performed continuously. There are several steps in this process: First, an organization is characterized to determined their strengths and weaknesses. Based on these findings, improvement goals are defined and augmented by improvement procedures. During project work, these procedures are performed and measured. At the end of one cycle, the measured data is analyzed according to the defined goal and the findings are packaged for future usage. In their work, Basili et al. emphasize the importance of experience packaging. But they do not give any concrete guidelines for this task. Instead, they give some ideas on what experience packages should look like. For instance, they emphasize the importance of context descriptions, or introduce several types of experience packages, like *process packages* (experience packages providing experience on processes), *product packages* (e.g. software architectures), or *data packages* (providing quantitative materials, like fault distributions or cost estimation models).

The project PERFECT, founded by the European community, tried to make the idea of QIP more concrete [5]. Therefore, they were also concerned with the activities of experience documentation, evolution, and reuse.

[1] In this sense, this approach is quite similar to case-based techniques in artificial intelligence, see e.g. [3]

One result of the PERFECT project was a structure for experience packages consisting of the following parts:

- a process model, describing a process for a particular task,
- process quality models, providing quantitative models on, e.g., effort, fault density, or size, and
- process experiences, providing lessons learned regarding the proposed process.

The main deficit of this structure is that it is limited to process experience only and there exist no reports on its application. So it is unclear whether this approach really works.

2.2 The SoftQuali Project

The experience transfer approach presented in this paper is embedded in a larger structure within the SoftQuali project [6, 7]. However, there are interdependencies between the various elements of this project. To make the presentation more understandable, first a short overview on SoftQuali is given.

The SoftQuali project (full title: systematic *soft*ware *quali*ty improvement by goal-oriented measurement and explicit reuse of experience knowledge) aims to apply the experimental paradigm (see, e.g., [8]) in the domain of software development and improvement. Figure 2 gives a graphical overview of this idea.

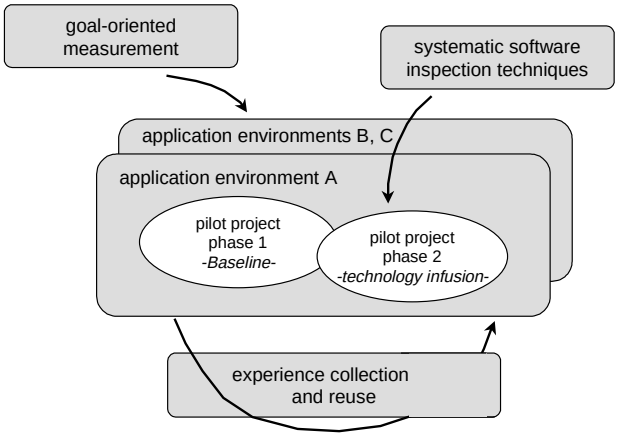

Figure 2: Structure of the SoftQuali project.

In each of the application environments (depicted as A, B, C) the first activity was to build a baseline of the actual situation using goal-oriented measurement (GQM, [9]). On top of this baseline systematic software inspection techniques were tailored towards the needs of each application environment and introduced. Then, the new situation was measured as well. This whole process is augmented by experience collection and reuse to make the findings persistent and applicable both for new application environments within one partner and across the partners.

The partners in the SoftQuali project are

- Allianz Life AG
- ALSTOM Energy Technology
- Daimler-Chrysler AG
- Fraunhofer Institute for Experimental Software Engineering (IESE)
- Siemens AG

3 Experience Documentation and Evolution

3.1 Types of Experience

Documentation is essential to externalize knowledge and experience in order to enable people-independent transfer of experience. Especially in environments with a high turnover-rate (as in the software business, see, e.g., [10]) or in distributed development, documenting experience becomes a crucial activity.

Using the definition of experience given in the introduction, we can package experience in pairs of observations and conclusions. But this can be unsatisfyingly, as the subsequent example shows.

Example: To document the requirements for ATM switches, company XYZ uses the SA/RT according to Harel [11]. The project manager of this project observes, that this method is not successful in her project.

Here the question arises, whether the observations and conclusions should be described (e.g., 'SA/RT according to Harel is not suitable for analyzing ATM switches in company XZY because ...') or the proposals for improvement (e.g., 'I suggest to document requirements for ATM switches in company XYZ by ...'). Both alternatives are, in a way, problematic: In the first case, they document 'real' experiences, which may be of little use for future users ('Ok, I know what to avoid, but not, what to do'). In the second case, the proposed solution has not been *experienced.* We observed this dilemma quite often in our early work on documenting experiences. To deal with this problem, we build an evolutionary model of experience documentation including several types of experience packages:

- *Observation packages* are used to package experience originating from real-life observations. They may, but need not, include conclusions on these observations. By this, observation packages are not necessarily useful for someone looking for a solution for his or her problem. But they are valuable input for subsequent activities, in which more mature experience packages are build.
- *Practice packages* provide applicable solutions for particular problems. These solutions must not nec-

essarily have been *experienced*. They may originate from subjective conclusions or are derived from positive observations. The described solution is tailored for a particular environment, reflecting its situation, limitations, and special needs. The mission of a practice package is to provide the best[2] available solution for a problem in the given environment.

Another aspect which cannot be handled sufficiently using these experience packages concerns *external experience*. External experience is defined as knowledge that was gained in external environments and that is often generalized. Typical examples for external knowledge are the SA/RT technique and inspection guidebooks. These documents provide valuable information, but it is unclear what results the application of these descriptions will cause in a particular environment. To cope with this issue, theory packages are introduced.

- *Theory packages* provide external knowledge (i.e., experience gained in other environments). Typically, the information provided is very general and not adopted to the given environment.

3.2 Evolution of documented experience

Using these types of experience packages, the evolution of documented experience can be illustrated quite easily (see Figure 3). Typically, it starts with external knowledge. Then this knowledge is tailored towards the special characteristics, constraints, and demands of the

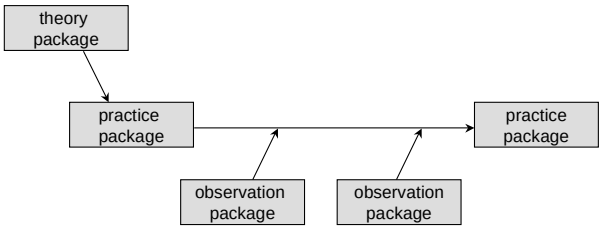

Figure 3: Evolution of documented experience.

environment, building a first practice package. The application of this 'best practice description' will result in observations, which are packaged in observation packages. These observations may provide both positive and negative experience. If deficits are found in the previous solution, new insights can be used to improve practice, building a new, more mature practice package. This evolution of practice packages is going on continuously, using new observation packages as well as new theory packages.

3.3 Quality Pattern

Given the model of evolution of documented experience, the question of documentation has to be dealt with. However, plain documentation is not sufficient enough. Documented experience must be processed to be easily accessible, readable, and understandable. Aiming towards those requirements, the Quality Pattern approach [2, 12] was developed. The main concepts of a Quality Pattern are problem-solution patterns, domain descriptions, explicit rationales, and the pyramid thinking principle.

The idea of problem-solution patterns was first published by C. Alexander [13] and has found wide adaptation in the area of design patterns. It is based on the observation that experts, who have to solve a new problem, remember similar problems and adapt and adopt their solutions to the new situation. They do not start from scratch every time. Quality Patterns use this idea by structuring information in a problem-oriented way. By this, the content of a Quality Pattern is aimed to fit a problem an experience-searching person will probably have (e.g., performing an inspection meeting, starting a new project, estimating project costs).

The description of the domain the experience was gained from is another crucial element in each Quality Pattern. The main reason for this is the understanding that no solution can be universally valid but is limited to a particular domain. The assumption is that applying a solution that worked for a *similar* problem in a *similar* domain will produce *similar* results.

By providing a rationale for the described problem-solution pattern, the reader can understand the *why* of the proposed solution and relate it to its own situation.

The application of the pyramid thinking principle [14] produces documents which are structured in several layers. In the top layer, only the most important information is presented. The layers below cover more and more details without providing totally different information than the layers above. The advantage of this principle is that the main content of a document can be understood quite fast. There is no need to read a whole document to see that is does not provide any interesting information. Figure 4 gives a graphical overview of the different parts of a Quality Pattern.

The upper part contains summary information which help a reader to get a first impression of the Quality Pattern's content. Then follows the main part providing problem, solution, and context (domain description). The lower part contains more detailed information like an example, the rationales, and administrative information.

[2] It is important to see that the best available solution is time-dependent, i.e., the solution is not globally optimal, but optimal with respect to the current understanding. At a later point in time, another solution may be the best available solution.

According to the evolutionary model mentioned above (Section 3.1), there are three subtypes of this generic quality pattern structure, called observation pattern, practice pattern, and theory pattern. They differ somehow in the various sections of a quality pattern. For instance, the section 'solution' in an observation pattern can be refined to 'observation' (what did really happen) and 'hypothesis' (what may be the reasons for it).

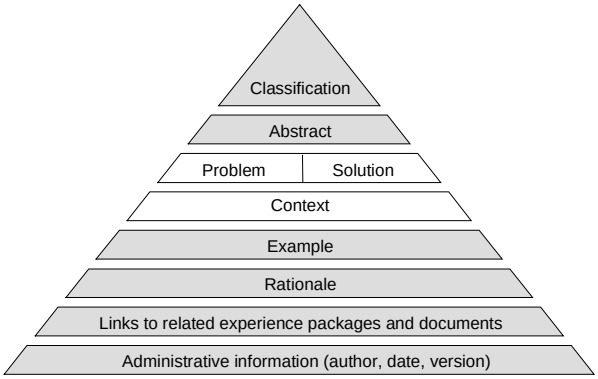

Figure 4: Structure of a Quality Pattern.

Figure 5 and 6 provide examples of a practice and an observation pattern, respectively. The next step using these pattern could be the creation of a new practice pattern describing the process of Figure 5 augmented with the strong recommendation to distribute the artifact under inspection via paper in the kick-off meeting. The new practice pattern will have at least two links to other experience packages: one to the old practice pattern (to show the evolution of experience) and one to the observation pattern (to give a justification for the refined solution).

Classification	Experience package type: Practice pack. Provided experience: Process Object: Reviews and inspections Problem: Kick-off meeting User: Inspections manager
Abstract	An inspection kick-off meeting consists of three steps: (1) distribution of review material, (2) explanation of inspection process, and (3) introduction of review material.
Problem	How do I perform an inspections kick-off meeting?
Solution	There are three steps in a kick-off meeting, with the third step being optional. (1) The moderator distributes the review material, i.e., the artifact under inspection (AUI), reference documents, checklists, and forms for defect detection. (2) The moderator explains the goals of the inspection process. (3) (Optional) The AUI author explains his product and technical details.
Context	Environment XYZ
Example	Here is an agenda of an inspection kick-off: 5 min, greeting 5 min, distribution of documents 10 min, explanation of goals and process 10 min, presentation of the AUI 10 min, discussion
Rationale	Mainly IEEE 1028; step (2) is non-optional, because inspections are performed rarely in this environment so there is the need to recapitulate the inspection process each time.
Links	Exp. Package: inspections form
Admin. info.	Author: W.C.Linton, December 18th, 1998

Figure 5: Practice pattern example.

Classification	Experience package type: Observation pack. Provided experience: Process Object: Reviews and inspections Problem: Kick-off meeting User: Inspections manager
Abstract	Consistent printouts of the artifacts under inspection (AUI) are an essential criteria for successful inspection meetings.
Problem	How do I perform an inspections kick-off meeting, even with time-pressure
Observation	Event: Inspection meeting Dec. 20th, 1998 Due to a rapidly performed kick-off meeting, the AUI (C-Code) was distributed after the kick-off meeting electronically. Later, in the inspection meeting, every inspector had a different listing (due to different line counting in the various mail systems and different printers). This caused a lot of confusion, because the different line numbers the inspectors reported had to be reassigned.
Hypothesis	- Time pressure - Inexperienced moderator and inspectors - Electronic distribution Documents must be distributed via paper in the kick-off meeting
Links	Exp. package: kick-off process (Fig. 5)
Admin. info.	Author: W.C.Linton, December, 21st 1998

Figure 6: Observation pattern example.

4 Experience Acquisition

We used experience packages according to the evolutionary quality patterns within various application projects (see also Section 6). Here, we made several observations:

- People like experience documented in quality patterns. Typically, these experience packages are quite short but very concrete. They provide immediately applicable solutions.
- The incorporation of new experience, which leads to maintenance activities in the 'experience base', is well manageable due to the evolutionary model.

- It is hard to write good quality patterns. Writing a quality pattern is time consuming due to the preparation of information towards the quality pattern structure. Writing the redundant parts (e.g., the abstract) is boring, too.
- Quality patterns cannot be written spontaneously. Instead, a writer has to think about the particular needs a future reader may have.

Despite – or especially because of – the advantages of quality patterns in reading, the inhibition threshold to write a quality pattern is high. To lower both the writing efforts and the inhibition threshold, we defined an 'experience acquisition form', implemented both on paper and using web technology [15]. The main idea of this form is that it can be used spontaneously. The sequence in which information is gathered differs from the sequence information is provided in a quality pattern. In these forms, the writer is asked for information she is able to provide quite easily, like a concrete observation or the description of her particular context. Redundant information (like the abstract) is not covered at all.

The experience acquisition form is used by project members. Every time they observe something remarkable, they can fill out such a form. In a later step, the information gathered is reprocessed by experts towards quality patterns as described above. By this, a kind of quality assurance process for quality patterns is established, as several people are concerned with writing quality patterns and reviewing other people's work.

Experience acquisition forms are mainly used for capturing observations. Capturing practice packages in this way is not very useful, since practice packages are seldom built by one person, but rather by groups of several people (like software engineering process groups, SEPGs, or quality circles).

Figure 7 presents an experience acquisition form. The Arial-typed text and the marked boxes show how this form can be filled in. The example text was used to build the observation pattern is shown in Figure 6.

In the context of the transferred technology used (i.e., software inspections), we applied paper-based experience acquisition forms. Overall 24 observations within several weeks only were collected. The amount of time to fill in one form was only a few minutes.

It is important to see, that experience acquisition forms do not replace experience documentation using quality patterns, but help to build quality patterns. They help to lower the threshold for building observations patterns. In the later step 'experience reuse', quality patterns are most preferable (see also discussion at the beginning of this section).

Observation on ☒ process, ☐ product, ☐ data	
Date: August 20th, 1998	Observer: Calvin Klein
Project: XYZ	Role of observer: Insp.-moderator
Particularity of the environment: Inspections have been introduced right before	
Observation: The object under review (C-Code) was distributed electronically. Every inspector was asked to print it out by himself. In the inspection meeting there was a lot of confusion, because the line numbering in the various printouts differed.	
Hypothesis (assigned with the observation) ☒ Cause: High time pressure (no time for previous printout) ☒ Consequence/solution: Distribution of printed material is mandatory for every kick-off meeting	
Task (in which somebody may be interested in this experience): Planning and performing an inspection kick-off meeting	

Figure 7: Experience acquisition form.

5 Reuse of Experience and Tool Support

The main goal of the experience transfer process is, of course, the reuse of previously captured and packaged experiences. Users of experience (and experience packages) are, in general, people requiring advice. This may be

- a project manager looking for experience on how to estimate a new project,
- a test responsible looking for information on using a particular test strategy,
- a developer who has to select a case tool for her new project and is looking for experience upon their utility in former projects, or
- a quality manager looking for quality assurance plans.

Benefits of (re-)using (documented) experience are quite obvious. To reuse experience, it has to be delivered to the potential users. In our approach, this is done in two way. First, experience packages can be provided 'as is' by, e.g., electronic channels, like web or mail. This kind of experience transfer is mainly suitable for quite general information, like benefits of inspections in this company.

Second, experience packages can be provided in junction with human experts. In this case the expert brings the experience packages with him. They are present during the whole time the expert is working together with the project people. The expert himself uses them to do his job. Also, packages are improved or created during the interaction of experts and project people. This kind of interaction is preferable for more sophisticated material, like introduction of inspection processes. The main benefit of using experience packages during the interaction of experts and project people is, that after the departure of

the expert the experience persists. And, it is not new to the project people (as e.g. a experts final report). Project people have seen how to use the information packages in the experience packages.

In addition, for electronic distribution we developed several prototypes [2]. They are all web-based in order to support different platforms in several sites. Figure 8 shows a screenshot of one prototype. It contains a search-mask for quality patterns. Attributes, which are left blank, are don't care attribute. If this form is submitted, beyond others, the pattern depicted in Figure 5 is received.

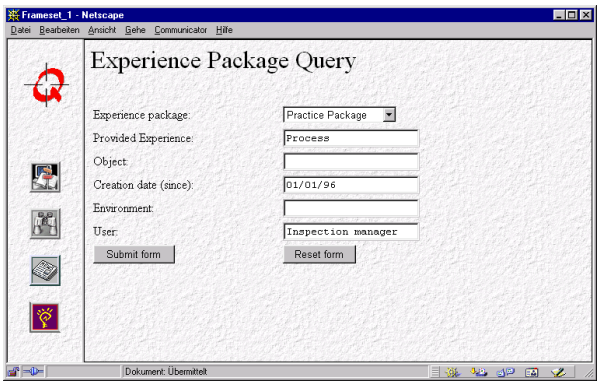

Figure 8: Screenshot of experience base prototype.

6 Application of the SoftQuali Experience Transfer Approach

In this section, two applications of experience transfer are presented. The first one is about transferring experience between several companies, the second one is about improving a technology by explicit documentation of experience.

6.1 Transferred Technology: Software Inspection

Both examples deal with inspections as transferred technology. Inspections have shown to be a very effective way of detecting defects early on in the development process. Thus, substantial effort can be saved, since defects detected later are known to be significantly more expensive.

However, there is not only one inspection technique, but several modifications of the idea of assessing a product by static examination. The implementations may differ, e.g., in the processes used, the roles involved, the artefacts inspected, and the reading technique used. In both examples the IEEE1028 definition of inspection is used, with one difference. In the first example, perspective-based reading (PBR), a new reading technique [16], based on the idea of TQM (Total Quality Management) [17] that each product has its own customers and that each customer views a product from his own point of view, was applied in the individual preparation phase and in the second example, some kind of checklist-based reading was used.

6.2 Experience Transfer between Fraunhofer IESE and Allianz

Situation. Development of high quality software satisfying cost, schedule, and resource requirements is an essential prerequisite for improved competitiveness of life assurance companies. Allianz Life, the market leader of life assurance companies in Germany, is part of the Allianz Group - one of the largest insurance groups with subsidiaries all over the world. The IT department of Allianz Life has more than 500 employees, with 350 of them being application developers. The Fraunhofer Institute for Experimental Software Engineering (FhG IESE) is acting as coach providing most up-to-date knowledge and experience on innovative reading techniques as the essential background technology.

Recent measurement programs have shown that currently quality assurance is focused on testing. At Allianz Life, about 30% of development effort are devoted to testing and about 50% of detected defects have their origin in early phases of software development. There, a technique is sought which improves productivity by finding defects when they are cheaper to correct, reduces the number of defects which originate in phases prior to the ones they are detected in, and reduces both test and overall project effort.

Use of documented experience. FhG IESE supported the introduction of PBR as a scenario-based reading technique at Allianz Life based on its wide range of experience in the field of software inspections. This step was accompanied by goal-oriented measurement to quantitatively demonstrate the proper usage of PBR as well as its cost/benefits ratio [18]. Additionally, experiences and measurement results are reused in the follow-up projects.

A post-mortem analysis of using PBR as defect detection technique at Allianz Life showed that not only the technique involved, or the organizational context, are the main driving factors for successfully introducing state-of-the-art technology. Another important factor were the experiences with that specific technique in different contexts. On the one side, this experience influenced the introduction process (what, when, and how to do) and gave confidence in the results to expect. On the other side, these experiences allowed inspectors to learn from others and use that newly gathered knowledge to improve their personal performance.

In order to support the introduction of PBR systematically, it is not sufficient to use that kind of experience described above on an ad-hoc basis, meaning the IESE-

PBR expert reuses the knowledge stored in his memory. Instead, experience packages documenting different types of experiences, at different levels of abstractions, were used for planning and communication. Both observations packages and practice packages were created using the quality pattern structure. Consequently, each experience package contains a detailed description of the influencing factors which lead to that experience. A typical example for knowledge stored in an experience package is that 'PBR-Scenarios for a specific organisation can only be defined with the knowledge of a domain expert from that organisation'.

Experiences on experience transfer. At Allianz Life, the use of documented experience[3] was found to be beneficial for technology transfer in general and for the introduction of software inspections in particular. This was mainly due to the fact that typical problems could be avoided (e.g., involving too many people in an inspection meeting) or solved in advance (e.g., scenarios, the driving factor of PBR, were developed at the Allianz Life site). Furthermore, in comparison to technology transfer at other organisations without the help of experience packages, the effort was reduced significantly. This is also reflected by the experiences made in the HYPER project [18]. In Figure 9, for each inspection the total inspection effort is compared with effort savings in terms of 'person hours'. It can be observed that for each inspection run the savings in effort are larger than the invested effort. We, FhG IESE and Allianz, believe that the reuse of documented, inspection experience is one (other possible influences are discussed in [18]) reason for the positive influence on this

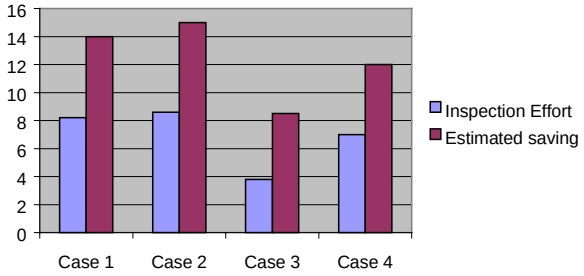

Figure 9: Primary Inspection Benefits [18]

cost/benefit ratio.

[3] Documented experience may be of different forms. At the Allianz site, the knowledge of domain experts, described in the form of scenarios, was used to guide defect detection. In addition FhG IESE used experience made with transferring inspection technology, documented with quality patterns, to guide the implementation of software inspections at Allianz Life.

6.3 Improvement of Inspections at DaimlerChrysler

Situation. Building mission-critical software at high quality within a tight schedule is increasingly becoming a core competency at Daimler-Chrysler. There are several divisions within Daimler-Chrysler (e.g., trucks or airplanes). And each division has its own software development unit, according to their particular needs. These units range from several dozens to several hundreds of software

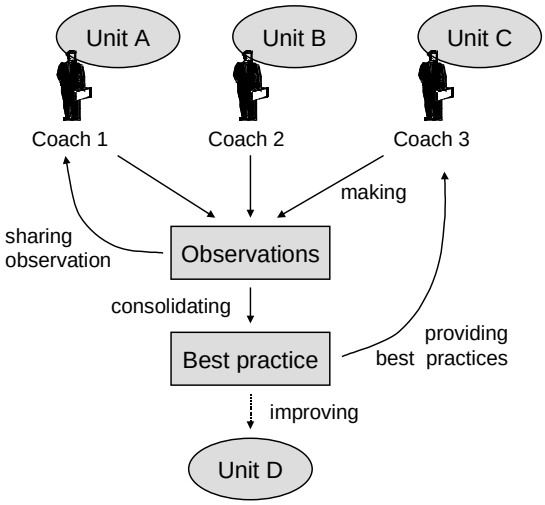

Figure 10: Using observations and experience to improve software inspections at several sites.

developers.

To improve the quality of their software development processes, the company's research division, introduced software inspections at several places. Unlike the Allianz-IESE approach, this happened mainly in the traditional way by providing teaching classes and coaching the inspection processes. This happened in several units quite simultaneously, with several researchers involved as coaches.

Use of documented experience. To improve both the introduction and the inspection process, and to avoid multiple pitfalls, experience documentation was incorporated in activities, starting in very early phases.

During their work in the application projects, the coaches documented their observations regarding inspections and the inspection introduction process. For this, they used experience acquisition forms as illustrated above. At regular meetings, the coaches discussed their observations and used them to build more mature practice packages which in turn they used in their daily work (see Figure 10). In the following the content of these practice packages is illustrated through some examples:

- Inspection process: In one unit, it is now mandatory to inform the inspectors about the goals of an

inspection in each kick-off meeting, because there are quite long delays between single inspections)

- Forms: Inspections are now supplemented by role-specific inspections folders containing all material the various roles need (e.g., inspector: AUI, preparation form, inspection guidelines, checklist, experience acquisition form)
- Entry criteria: It is now mandatory in one unit to distribute the AUI on paper in the kick-off meeting (see also Figure 6 and 7).

Experiences on experience transfer. The usage of observations packages (build by using experience acquisition forms) was found to be very useful. Without them, a lot of those typically small, but valuable insights in the inspection process would have been lost. This is especially important for mistakes everyone makes in the beginning when using inspections. If they would not have been packaged, some pitfalls could have not been avoided.

Another benefit is aggregating observations to practice packages. They provide not only applicable solutions but also rationales (in Section 'explanation' of the quality pattern structure). For smaller process changes, especially, the risk of reinventing the wheel twice (i.e., a change A, introduced first, is skied after a while by change B, which is replaced by change A, and so forth) was observed several times.

As in every technology transfer activity, inspection know-how is handed over gradually to the involved units. There, we see a benefit of the experience packages, too. Experiences described in such packages is:

- *very concrete*. The content of the experience packages is tailored to the specific needs and restrictions of one domain.
- *built on experience in the user's environment.* By this, the practitioners in the units are willing to accept them more than external knowledge, as they were involved in building the experience packages.
- *quite short.* Due to high time-pressure no one is willing/able to read huge textbooks or conference proceedings.
- *maintainable.* Each unit can improve their processes using the experience evolution process described above (Section 3.1).
- *transferable.* They can both be used to make new employees familiar with the established processes and to exchange experience with other units in order to improve their overall understanding.

The coaches will use the documented experiences both to train new researchers and to establish inspections in new units (as depicted by unit D in Figure 10). Starting on a more mature foundation of knowledge and experience, many troubles and pitfalls can hopefully be avoided.

7 Summary and Future Work

With the rapid rate of innovation in and around software technology, one might have expected to have seen significant improvements in the area of organizational learning and reuse of experiences. In practice, however, this has only started to begin due to the problems in capturing and reusing knowledge. The recent advent of experience factory technology [4] and its accompanying technology seems to be a promising approach to overcome those problems. In this paper, we have presented the experiences made with that technology in the SoftQuali project.

We described the basic technology of experience documentation and evolution. We introduced a crisp classification of experiences in three different types, providing a generic pattern (i.e., Quality Pattern) to document such types of experiences, and presented a practical approach for acquiring experiences. Finally, we presented the results of using this technology in two real-world examples, both related to introducing inspection technology. In general, the reuse of documented knowledge/experience to plan and control the software technology transfer process was found to be useful in both projects.

Our next activities cover several directions. First, we see the need for gaining more practical experience in using our approach in several domains. Then, we plan to build more sophisticated tools to support humans in their experience maintaining activities. Last, but not least, we need to evaluate the quality pattern structure more thoroughly. Especially our assumption that quality patterns are more comprehensible and written towards a readers needs, has to be validated empirically. As a first step here, we are currently performing a long-term study in a laboratory environment at the University of Ulm. There, we have asked students to write both quality patterns and reports on particular topics (namely inspections and modeling reactive systems). At a later point in time, we will give these documents to other students, asking them to perform these particular tasks and observe the time and depth of understanding. Subsequently, we plan to use these findings to start such an investigation in an industrial setting.

Another interesting question concerns the reinfusion of documented experience in development processes. Section 5 covered some technical aspects of this activity. However, there are also social aspects to cope with. Literature on reuse initiatives (e.g. [19]) often mention the problem, that people are not willing to externalize their knowledge as they fear to get unimportant. In this case, often incentives are proposed as a solution. In our environments, we observed another interesting phenomenon: people are willing to share their experience, but they are less willing to accept advice by others. As a long term goal, we are interested to investigate this contradiction to

see, whether there are (social) patterns explaining this fact.

Acknowledgements

The authors would like to acknowledge the contributions of all SoftQuali partners. Thanks are also extended to Sonnhild Namingha for her efforts in editing this paper.

This work was founded by the German Federal Ministry of Education, Science, Research, and Technology under contract number 01 IS 518. It does not necessarily reflect the position or the policy of the government and no official endorsement should be inferred. The responsibility for the content of this paper remains with the authors.

References

[1] Lexikon-Institut Bertelsmann, *Dictionary*, Bertelsmann, Gütersloh, 1979 (in German).

[2] F. Houdek and H. Kempter. Quality Pattern – An approach to packaging software engineering experience. In M. Harandi (ed.): Proceedings of the 1997 Symposium on Software Reusability (SSR'97), Software Engineering Notes, ACM, Mai 1997, pp. 81-88.

[3] M.Y. Jona and J.L. Kolodner. *Case-based reasoning*. In S.C. Shapiro (ed.): *Encyclopedia of Artificial Intelligence*. John Wiley & Sons, 1987, pp. 1265-1279.

[4] V.R. Basili, G. Caldiera, and H.D. Rombach. *Experience Factory*. In J.J. Marciniak (ed.): *Encyclopedia of Software Engineering*. John Wiley & Sons, New York, 1994, pp. 469-476.

[5] *The PEF model*. A booklet from the PERFECT EPSRIT project 9090, 1996.

[6] The SoftQuali Consortium. *SoftQuali Project Documentation*. Available at http://www.iese.fhg.de/Proj/SOFTQUALI/index.htm.

[7] H. Kempter and F. Leippert. *Systematic Software Quality Improvement by Goal-Oriented Measurement and Explicit Reuse of Experience knowledge*. BMBF-Statusseminar 1996, pp. 281-297, DLR (German Center for Aerospace, in German).

[8] H.D. Rombach, V.R. Basili, and R.W. Selby. *Experimental Software Engineering Issues: critical assessment and future directions*. Lecture Notes in Computer Science 706, Springer, Berlin, 1993.

[9] V.R. Basili, G. Caldiera, and H.D. Rombach. *Goal question metric paradigm*. In J.J. Marciniak (ed.): *Encyclopedia of Software Engineering*. John Wiley & Sons, New York, 1994, pp. 528-532.

[10] T. DeMarco and T. Lister. *Peopleware*. Prentice Hall, 1995.

[11] D. Harel. Statecharts: A visual formalism for complex systems. Journal of Science of Computer Programming, vol. 8, 1987, pp. 231-274.

[12] D. Landes, K. Schneider, and F. Houdek. *Organizational learning and experience documentation in industrial software projects*. In Proceedings of the 1st Interdisciplinary Workshop on Building, Maintaining, and Using Organizational Memories (OM-98), Brighton, UK, August 1998, pp. 47-63.

[13] C. Alexander. *The Timeless Way of Building*. Oxford University Press, Oxford, 1979.

[14] B. Minto. *The Pyramid Principle - Logic in Writing and Thinking*. Minto International, London, 3rd edition, 1987.

[15] W. Bierer. *A Process for Building Quality Patterns*. Masters thesis, University of Stuttgart, May, 1997 (in German).

[16] V.R. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S, Sorumgard and M.V. Zelkowitz. *The Empirical Investigation of Perspective-Based Reading*. Journal of Empirical Software Engineering, 1997.

[17] Richard E. Zultner. *TQM for Technical Teams*. Communications of the ACM, 36(10), October 1993, pp79-91.

[18] L. Briand, B. Freimut, O. Laitenberger, G. Ruhe, and B. Klein. *Quality Assurance Technologies for the EURO Conversion – Industrial Experience at Allianz Life Assurance*. Proceedings of the Software Quality Week Europe, Brussels, 1998.

[19] C. McClure. Extending the software process to include reuse. Tutorial given at the Symposium on Software Reusability, May, 1997, Boston, MA.

Talk to Paula and Peter - They are Experienced

Conny Johansson, Patrik Hall, Michael Coquard
Ericsson Software Technology AB
P. O. Box 518, SE-371 23 Karlskrona, Sweden
Conny.Johansson@epk.ericsson.se, Patrik.Hall@epk.ericsson.se, Michael.Coquard@epk.ericsson.se

Abstract

People must solve problems every day to get their work done. For successful problem solving, they need to find others with specific experience. To help individuals with this, we created an 'Experience Broker' role in the organization. The broker must be officially recognized, and function as a facilitator for the internal human network in the organization. Our approach to the 'Experience Factory', what we call the 'Experience Engine', focuses on mediating referrals to sources holding the correct expertise—that are usually human sources.

We claim that the most valuable experience is stored on the individual level. The most valuable experience transfer comes while on the job when the person holding the experience verbally relates to the learner directly. Getting the right context for this experience transfer is difficult without direct contact between the person with the experience and the recipient. Experience is transferred at the right time when the learner is actually working on something that concerns it.

Values must be rooted in the organization, where individuals are willing to spend the necessary (most often short) time to spread valuable experiences to others in the organization. Our pilot project showed that a measurement and database approach is less valuable than individual face-to-face meetings, to transfer experience.

1. Introduction

The 'Experience Factory' [1] is an important approach for software companies in their struggle for survival. We felt it important to relate this approach to the current situation at the company, and to its organizational learning processes. We also believe that gradually integrating new ideas, concepts and approaches is the smoothest and best way for our company to apply such an extensive approach as the experience factory.

Ericsson Software Technology AB has several business centers to develop software products, nodes—which establish product requirements and have primary customer interface as well as research and corporate functions. Each business centre is divided into several business units, each normally working on one or up to three products. The business units have 20 to 30 employees each, for a company total of approximately 800 employees. Together, they develop a wide range of software applications, including several types of telephone switches, and mobile phone management systems.

This paper starts by describing the similarities and differences between the experience engine and the experience factory. The pilot run in 1998 is presented, followed by a section describing the narrower scope and focus of our measurements. The Experience Practice describes the basic values for experience transfer, then the role of experience broker and communicator are defined, and finally we describe the different experience occasions which we identified.

2. The Experience Engine Vs. the Experience Factory

We believe that when applying the experience factory approach, the same thing is true as with software development–it is experimental and there is no one true way to do it [2]. We realized rather quickly that we could not implement the entire approach, so we renamed our application of it to 'The Experience Engine'. We wanted to create a engine that would continuously work to recycle the collective experience within the company.

The experience factory focuses on process improvement. And it should not be a part of any project organization, since its focus does not normally coincide with that of any other part of the organization. The experience factory should be an independent part of the organization that collects, analyzes, generalizes, formalizes, packages, stores, retrieves and reuses collective experience [1]. It can provide a project organization with two kinds of experience: one that hints at what to do and the other that guides in doing it. Further, it functions to investigate and introduce new technology to the company, is the skills center for software engineering, and has operative responsibility for process improvements.

On the other hand, the experience engine does not take this broad approach. That would imply a larger reorganiza-

tion requiring redefining established concepts and groups at our company. The experience engine focuses on creating and managing experience transfer, and the role of the individuals involved when this takes place. It also supports such activities as analyzing measurements and experiences.

The experience engine organization is, in this comparison, not as large as the experience factory organization. The considerable difference in focus on measurements and quantification, is one reason for this. In our approach, generalization and customization are handled when the transfer occurs, face-to-face. This as opposed to documenting the generalizations in writing, as required in the experience factory approach (as we have interpreted it). Another reason for not focusing on more extensive documentation is that our company primarily provides customized products. We venture to guess that companies providing standardized products can benefit more from focusing on generalization and formalization (which we have not done), as opposed to companies like ours.

Other activities also not included involve new technology infusion, skills management, and process improvement. They certainly exist in some form, but are organizationally considered on the business unit level, and so were excluded from the study. For example, process improvement activities are normally initiated by improvement proposals reported in intermediate or final reports. Our experience is that the best way to carry out process improvement is to include it as an activity in each project. These are coordinated and followed up at the business center level by a manager, with the results communicated to the rest of the organization primarily through informal channels.

Taking the entire experience factory approach would have been revolutionary at our company. As well, and we thought it would not bring any improvement. First, it would require an extensive re-organization involving definition of an experience factory organization. Second, it would require unnecessary changes to work routines that are perceived to work well already. Finally, it would involve introducing a company-wide approach affecting all activities in the organization without any easily identified cost benefits. Further, it would probably hinder the success of our other more important efforts where we found shortcomings and the need for improvement.

3. The pilot project

The pilot project we set up was limited to a few business centers, and was designed to test our ideas and evaluate their cost benefits. We continuously refined, removed, and added new guidelines and working practices as we gained experience and feedback.

This pilot empirically evaluated over 100 experience transfer occasions in the organization. These involved a wide range of experiences related to management, technical, as well as business related issues. We felt these studies should be related to their context. The experience occasions were evaluated continuously using attitude surveys, both written questionnaires and interviews. We also generalized these experience occasions to determine common patterns, further developing our approach. The project consisted of a core team with four persons working part time. All project members have a Software Engineering or Computer Science degree and several years of professional experience in software development.

Another empirical study, related to our pilot, reports that the information system at our company is rather well developed. Formal communication channels such as electronic mail and documentation (written reports) have been well established for several years. But, this might be at the expense of verbal communication. Establishing a corporate culture that includes a coordinating mechanism for communication (to transfer knowledge and experience) is essential for employees' knowledge-creating activities [3].

4. The measurement approach

Measurement is the process of assigning numbers or symbols to attributes of entities in the real world so they can be described using clearly defined rules [4]. Measurements in software engineering can be direct (as in cost or number of faults) or indirect, by combining different measurements (such as for productivity or fault density). The usefulness of a measurement depends largely on the control status of the process when it is collected [5]. This is the cause of variances observed during the process. The problem is that the causes of variance are rarely analyzed. The reasons we found for this are:

- When an organization starts measuring, its approach to measurement is too comprehensive so too many measurements are defined,
- The necessary activities for analyzing the causes for variances are not completed,
- Feedback from analysis of the causes of variances is poor,
- Improvement activities initiated because of the measured variances are not followed.

Today, some companies still think they can put all their corporate knowledge on a server to form a kind of giant hyperlinked encyclopedia [6]. The real value of information systems is connecting people to people, so they can share their expertise and knowledge on the spot. We started with a significant effort to determine how measurements should be collected, packaged and stored. While the original purpose of the experience factory does not lend itself to creating a large database, this trap is easy to fall

into. The trap is set by the descriptive illustrations of the experience factory, which often indicate storage using the now old-fashioned, disk storage device [7]––one easily visualizes physically storing experiences on a computer device. Our first approach included a large database, but after interviews, some research, and several working sessions, we concluded that this was not a good solution for the company.

In fact, we spent nearly 1000 hours on database specification before we realized this was not right for us. We determined this from the fact that over the last years we have gathered and stored large amounts of data in a large corporate database. This includes hundreds of measurements like lead time, planning vs. actual, productivity, and more. These data have rarely been analyzed for use in process improvement. Some of our time was even used to identify additional data to collect. When we presented yet another database specification, most everyone sighed at our approach. In the end, we found it more important for analysis and feedback efforts to gather a small number of different measurements. The time spent on these activities can at first be seen as wasted, but it was actually well worth finding the right direction to proceed.

We decided to narrow the scope of the measurement program. At the company level we focused on fault data and defining collection routines, storing principles, analysis methods (including statistical methods), activity definition (including responsibilities for monitoring the activities), and simple tips on how to provide feedback on the data and fault measurement results. We focused on fault data at the company level because this is the only general measurement we saw as worth collecting now and in future. This approach is also supported by Demarco [8]. Additional measurements and local adaptations to the corporate approach were dealt with by the local organization.

Measurements were designed to help develop a method and support tool for variance analysis and forecasting. These primarily concerned forecasting the number of faults in operations. We chose the standard method of correlation of metrics using simple linear regression technique [4]. This simple correlation technique takes a mean squared fit of the data and calculates the differences between the actual data and the calculated line. If these residuals are within an area predicted from a normal distribution, then the correlation has a reasonable probability to be valid. Further, significance tests were done to investigate the practical significance and calculate the likelihood of finding the correlation by chance. The correlation calculation procedure and the significance of a calculation are described by Humphrey [9], among others.

Interesting correlations were identified using this method, of which, the most interesting are:

- The number of faults in operations decreased as the number of faults in inspection increased (this is considered an adequate correlation based on Humphrey [9])
- The number of faults in operations increased as the number of faults in function test (the integration and use case test of the smallest building blocks) increased. (We considered this a strong correlation based on Humphreys [9])

We also developed a process to support the measurement cause analysis. This simple process describes how to conduct the analysis of measurement deviations reported in a project. The process includes team member interviews and evaluation sessions to determine the cause(s) of the deviation. We noticed that even though a measurement deviates from the expected, this is not necessarily unacceptable. Special conditions in the project can cause deviations but these should not always be seen as indicators for action or other decision.

5. The Experience Practice

Our company values are based on a concept we call Guiding Lights. The five Guiding Lights, short phrases symbolized as lighthouses, show us the best course to follow and help us in our voyage into the future. One Guiding Light is ‘We learn from each other’. This concept is emphasized to a greater extent in our organization since the ability to share knowledge and experience is most often an important aspect in salary negotiations between individual employees and their managers.

In defining Experience Practice, we used the basic value contained in the concept ‘We learn from each other’:

The most useful experiences are stored in every individual’s brain. This is the primary storing media, and verbal communication (preferably in face-to-face meetings) is by far the best way to communicate and spread these experiences.

Information and knowledge are sometimes distinguished [10]. Information expresses knowledge but is unreliable in transferring it. Knowledge should be transferred when the receiver is actually doing something related to the experience being transferred. Knowledge can be defined as: “a capacity to act” [10]. We consider experience as: “knowledge that has been perceived by acting”. The main focus is to increase competency, and so transfer knowledge––a transfer mechanism which we consider equivalent to transferring experience. This paper does not discuss in detail the difference between knowledge and experience. It is impossible to transfer knowledge precisely, but if the knowledge is merged with experience(s), it will most probably enhance competence.

We use only the word 'experience' further, and do not consider the possible differences in the knowledge transfer and transferring experience.

Orr [11] states that experience is a socially distributed resource, stored and spread primarily through an oral culture. Interpreting raw measurement data is difficult without extensive routines to classify data and other context related definitions and restrictions. Written and stored information is barely recognizable to outsiders, so the author usually has to be contacted to understand the context of the experience [11]. In a face-to-face meeting with the source, the receiver can get immediate response to any questions and clarification concerning relevance, context and detail––whatever seems appropriate for the situation. In rank of importance, face-to-face meetings are the most valuable media for transferring experience––followed by telephone, video conferences, email, personally addressed mail, and finally impersonal mail such as formal reports [12].

Useful experiences are normally stored on the individual level. The benefit and gain from spending time to write down these experiences is hard to measure. Our conclusion from interviews of professionals is that this time does not, most often, pay off. Time must be taken by the author to interpret their experience, write everything down and then formulate the context. The person reading about this experience, in turn, has to spend the time to read and interpret the experience related. Several links must be performed in a uniform way to get the same result at both ends. We judge this unlikely in most cases. But of course, do not extend this concept into absurdity. Writing down specialist literature, documentation for standardized products or methods, and explicit (or apparently explicit) knowledge still serves a purpose.

Another important aspect is based on the axiom demonstrated by the postman game, which Swedes call the whispering game. If one person tells another an experience, who, in turn, tells it to another, and so on, the original experience related most probably differs greatly from the final version. This is analogous to the example of the written material described. Our finding in this case is that the most valuable and useful experience transfer comes when told by those who have actually gained the experienced themselves.

If experience is not stored in writing, it will disappear when the person who actually gained it leaves the company. This is, of course, negative, but we consider that:

- Experiences which can be unambiguously interpreted are (hopefully) stored in final reports,
- The life span of the experience in software engineering is short, and
- There is a fair chance that when our experience engine approach is established, the valuable experience has been communicated to another person, and it has been merged into their own experience, so it can be shared (in its new form).

To address this last point, we made a qualitative study of the company learning process. In order to understand the organizational learning process, the entire company must be considered, not just temporary projects [13]. Our study indicated that individuals review and observe experiences from different perspectives, and so, reflect on it to a greater extent. Further, they interpret and use their observations for problem-solving and decision-making. These are used as a base for acquiring more concrete experiences, which in turn forms the basis for more learning. These practical learning processes are similar to the cyclic learning process described by Kolb [14]. We think this indicates that experience is not as tied to specific individuals during its (longer) life span, as we initially thought. As well, people do not depend on older experiences since these are merged and refined into new, fresher understanding.

Strong management commitment is necessary to succeed with our approach. Senior and mid-level management must actively and continually support the experience engine and the principle it builds upon. Management support should not only include communicating information, but also action by continuous facilitation in using the experience engine.

6. Experience occasions

An important question was: 'When does experience transfer take place?' Our study indicates that much experience transfer takes place during informal communication. We believe common values can be developed to support such informal experience transferring. Shrivastava [13], among others, supports this. The transfer is dependent on the work environment, time, co-workers, and the type of transfer involved. We believe that direct communication is more suited for informal experience transfer than written text. What happens when two people meet and talk depends on their interpretations, language, personalities, and the context. The basic values for communicating experiences already exists in the organization and instances of experience transfer occur rather frequently. When introducing the experience engine, we saw that experience occasions could be identified in three groups, which we named 'Flashes', 'Learning Situations' and 'Education/Training' to more easily differentiate them. Flashes and Learning Situations are described in this section. Education/Training is when the learner lacks formal training and/or professional experience in a specific area.

The experience engine is not designed for education/training situations.

Of course, whether an experience transfer belongs in one of these categories or the other is always debatable. We created these groups simply as practical guides to clarify them, but they are definitely not intended to formalize any relationships or be used as rules. In our pilot study we estimated there were 20 learning situations and 80 flashes in 1998. These are only estimates since we could not register some flashes and learning situations before being fully aware they had occurred.

6.1. Flashes

Flashes are the most common experience occasion. We identified two types of flashes: 'Broker Flashes', where the referral to an experience source is mediated; and 'Communicator Flashes', where experience is actually communicated. Most flashes were internal to the organization, involving employees. However, we see no limitation that others outside the organization can be involved in this type of experience occasion. We see this as a matter of mutual needs where there is no direct competition between the organizations. If we help you this time, you help us next.

Experiences are transferred most effectively when the receiver is performing in the process concerning the experience actually required [10]. Experiences are valuable only when they can be contextualized, decontextualized and recontextualized at the proper time [15]. Broker Flashes do happen through written messages (mostly email), but occur more often in telephone or face-to-face meetings. Face-to-face meetings are short occasions that usually happen by chance resulting from actual needs. In listening to people as they discussed everyday problems in corridors, on breaks, and at lunch, brokers can act as agents for those needing extra, experienced advice. It is at these short (usually only a few minutes), recurring occasions where we acted as middleman in the human network. However, to reduce relying on chance, we could create broker flashes by actively seeking them. These flashes could be created by the broker:

- Walking around - dedicating time to just walking around specifically to talk and listen to others,
- Participating in reviews and inspections – these are designed for concrete active help to the receiver so this activity is therefore easy to adapt,
- Participating in project risk analyses,
- Participating in different kinds of cross-section teams and networks.

We defined communicator flashes as taking less than 10 minutes. They have occurred previously, but only in limited size groups, such as teams or units of less than 30. Communicator flashes do not cost the receiver anything. Examples are questions and problems regarding specific programming language constructs, product storage handling, and test case specifications.

6.2. Learning Situations

Learning situations are when the receiver has formal training but lacks professional experience. This situation normally requires planned time and takes more than 10 minutes. Learning situations involve on-the-job training where the receiver learns from doing in actual conditions, but in a session with an experienced communicator. This is also known as experience transfer by tradition. And is reported as being up to seven times more effective in transferring experience as the most common method––the lecture [10].

Learning Situations can be when the communicator:

- Moderates a work-shop,
- Participates in developing a project plan with the receiver (for the receiver's project),
- Guides a quality coordinator in using company software quality assurance processes.

We found calculating the cost benefits of learning situations difficult. The best approach we found for this was simple attitude surveys on a case-by-case basis. These attitude surveys were done with either interviews or questionnaires. One example of a learning situation is where the communicator participated in a three hour walk-through of a preliminary study. The receiver reported the walk-through (involving only the communicator and the receiver) as successful, estimating a savings of several hours and a considerable quality improvement gain.

This highlights our feeling that we are not only interested in actual cost benefits, but also other organizational benefits. This is often difficult to measure in numbers so we felt the approach of estimating benefit using these attitude surveys could provide evidence of success.

Regarding another aspect, when companies pay for a service, they have to evaluate the results of the work performed. They are keen to optimize the use of the time paid for, whether they buy it internally or externally. This argues strongly for the receiver to want to pay for the learning situations. Also, from the perspective of the communicator (and the financial unit they belong to), it is important that the learning situation does not affect their budget negatively. Follow-up for both costs and benefit would therefore be simplified if these situations are budgeted.

Again, we do not see any limitation for involving external personnel in learning situations. With the learning situation financed by the receiver, this is easier done than for flashes. However, the problem of allocating the necessary time for experienced communicators remains, whether this is acquired internally or externally to the learning situa-

tion. This kind of resource is scarce, especially in software engineering, so these busy people have difficulty allocating the necessary time.

7. Experience Broker and Experience Communicator

McDonald and Ackerman [16] state that people in organizations must solve problems to get their work done every day. For this, they must often find expertise with this experience. We use the term expertise as the embodiment of experiences within individuals. Individuals have different levels of expertise about different topics and in different contexts. To solve the problem of identifying the expertise required and selecting who to acquire it from [16], two new roles were instituted in the organization, that of 'Experience Broker' and 'Experience Communicator'. The experience broker has a visible, appointed role within the organization, similar to the liaison position described by Mintzberg [17]. These must be visible and accessible to facilitate helping individuals looking for expertise, when they need an experience transfer. Experience communicators, on the other hand, have until now a more informal role in the organization.

Those lacking the necessary experience just contact the experience broker who helps them find the right expertise, or experience communicators, holding the necessary experience. For this, there will be many experience communicators but only a handful of specifically appointed experience brokers, connecting them all in the experience engine.

7.1. The Experience Broker

Experience brokers add some extra power to the engine to make it run better. They function as the hub of the experience engine network. Their primary responsibility is to maintain, extend, and facilitate using the experience engine network and its experience communicators. These are generalist with wider experience using tools and methods–both within and outside the company. So they know a little about many different things, but can easily understand the company structure (products) and its processes.

In this sense, the experience broker is a loaf about––someone who listens, sometimes even eavesdropping, but who also talks with many people, referring those who need it to the right experience communicator(s). This is not to say they spy on employees, but they invest their time trying to find out what others' needs really are. This can be thought of as taking the organization's temperature, finding out what is going on, and solving networking problems. The broker therefore knows the projects currently running, even participating in formal activities for selected projects (including meetings) to extract what experience is needed.

The broker knows the arenas where people interact and should also try to find or create additional arenas for transferring experience [18]. This role is essential in creating a company learning climate.

The experience broker's role is to:

- Identify several experience communicators who can provide others with help/experience,
- Facilitate contacts with experience communicators,
- Strive to connect people across organizational boundaries,
- Be perceived as always available,
- Follow up on the benefits from flashes and learning situations.

This role must be fully recognized in the company for it to succeed. This means that it is both well known, and appreciated. The person acting as a broker must have a large contact network, and be respected as a generalist in the company. This can be achieved by communicating the concept as well as success stories through both informal and formal communication, including departmental and project presentations. An important consideration for such presentations is to have advance information about the current status of the projects or units the (potential) receivers represent. This way, the broker can find concrete applications for flashes and learning situations.

The restaurant is a useful arena for mediating referrals to experience sources. In the extreme case, the broker should always try to have lunch with different people, take breaks at different locations, and participate in activities with employees outside work. In these situations the broker always works when others have breaks, lunch or spare time. This may not be fully attainable from the broker's social perspective, but these arenas for mediation and experience transfer are very good opportunities.

The experience broker must have good communication and interactive skills with others. Some people are naturally this way, but these skills can also be learned. Experience brokers must have a good sense of when to interact with a project or a person, and when not to. They should not, for example, disturb a person who is deeply occupied and they have to know where to find the right person with the necessary experience and then mediate the contact. And even more important, project members and others must see their needs met when using the experience broker.

The length of time for anyone to be experience broker was also discussed. Though we have not come to any final determination, we believe this depends largely on the individual involved. While some should stay brokers for shorter periods, others have the qualities necessary to hold this role longer. Some are suited to do this full time, while others are more suitable as part-time brokers. One benefit

with part-time brokers is that they can maintain their role as a generalist by participating in different projects with their remaining time. It is, however, of utmost importance that the organization has full confidence and trust in the individuals acting as brokers.

There are too many specialists and too few generalists at the company today. To ensure that the experience broker stays a generalist, they must be known to the various parts of the organization and different projects. One way to achieve this, is known as walk around management, which has successfully been used by great companies around the world. We feel this principle can be used for experience brokers, though they are not managers. This should be an independent position that is not assigned to any one branch or specialty.

7.2. The Experience Communicator

Experience communicators are those with the experience and willing to share it with others. They should have some pedagogical skills in addition to their willingness. We do not think everyone with vast experience can become a communicator, rather it is equally important, as with brokers, to have a human understanding and social skills.

We found it important for experience communicators to try to give educational answers. These answers are focused on teaching receivers how to solve related issues by themselves in future. Lao Tzu is credited with coining a useful proverb: "Give the man a fish and you feed him for a day, teach the man how to fish and you will feed him for a life time". Providing educational answers means placing the answer in a broader context, or even just hinting at where to find the answer. For example, an answer may be easily available on the web or in a report. Otherwise, providing receivers with clues on how to solve the question themselves may be the best approach without further describing the detailed context.

Experience communicators should not to do others' work for them. Rather, their main task is to help others solve their own problems. This is different than the role of regular project resource. While using the communicator as a resource in a project may help the project, it would damage the entire idea behind the experience engine. Little or no experience transferring would take place in such situations.

The role of experience communicator has, until now, been an informal role in the organization. However, we are discussing how this role should be defined and more visible in the organization. The benefit for being communicators should be that they would be perceived as experts, attaining a higher status in the company.

It is also important for the experience communicator to give feedback to the experience broker. This not only ensures that processes can be reviewed, but also that the communicator actively takes part in the experience engine. It is equally important for the broker to know which communicators are currently active. For example feedback from the communicator to the broker could be: "I have had many people calling me about the same type of problem. I think we need some courses in this field," or "I was asked a difficult problem that I could not resolve. I referred them to another person I know who is good in that field." This fits the goal of the experience engine: to get people to build human networks.

It is important to realize that everyone is not suited to be an experience communicator. Everyone does not have the aptitude and the social skills necessary to transfer their experience pedagogically. We found it difficult to evaluate whether a person is suited to be a communicator, and are still looking for a model or guideline. However, we have on very rare occasions run into this problem and believe that this can be handled as a normal procedure within the organization.

8. For the Future

We will continue to refine our approach focusing on mediating referrals to others as a source of collective experience. We think that our approach was a success. Increasing the number of mediations five-fold during a three month period indicates this. We plan to consider integrating more of the key features described in the 'Experience Factory' approach into our experience engine. In particular, process improvements will be a key issue for this.

Currently, we are incrementally extending use of the experience engine in the organization. Some spontaneous mediation and experience transfer takes place throughout the company, of course. But we plan to formally introduce it step-by-step. Right now, there are only a few individuals working as brokers part-time. We plan to allocate more resources to this role. The (informal) list of communicators also has weekly additions. We also need to define and resolve the issue of visibility for the communicator role.

As well, we need more success stories focusing especially on cost or lead time improvements. Success stories are one of the most important incentives to apply any approach. For our part, we want more stories that emphasize using project managers and team members, as well as success stories useful for senior management.

We found it very difficult to find reliable methods to calculate cost benefits. For this, we found the best solution was learning about the benefit through attitude surveys, while not concentrating only on cost.

9. Conclusions

Using our 'Experience Engine' approach, we aim to eliminate the attitude that managing collective company experience involves huge databases and processing large amounts of measurements. Our approach favors verbally communicating valuable experience and de-emphasizes the practice of collecting lots of data. This relies heavily on values of willingness to share experiences, and to mediate referrals to others who hold the necessary experiences. It also relies on two important roles, experience brokers to mediate the referrals, and experience communicators to transfer the actual experience. We use the term 'broker flashes' for when mediation occurs, 'communicator flashes' for experience occasions, and 'learning situations' for education/training. The experience engine handles communicator flashes––short experience transfer occasions which take only a few minutes, and learning situations––longer sessions with experienced people, when the receiver has formal training but lacks professional experience.

The experience broker is a generalist who helps others in finding the right expertise with valuable experience. The broker is visible within the organization, and has a wide, general knowledge of methods and tools. The broker is active in arenas for mediating referrals to sources holding experience, and also tries to find and create new similar arenas.

The main task of the experience communicator is to help other people to solve problems. The communicator focuses on giving pedagogical (educational) answers to teach the receiver how to solve related issues on their own. The broker and the communicator must have an aptitude for interacting, and knowing when to interact, with others.

One drawback with our approach is its heavy reliance on senior, highly experienced and respected individuals, to act as brokers and communicators. These individuals are the most attractive ones on the market today increasing the difficulty in keeping them as brokers or communicators––or even keeping them in the organization at all. But, knowing that old experience is not as valuable as new, and that previous experiences are, to some extent, retained and combined with new experiences, we feel that reasonable staff turnover will not have a significant negative affect.

We have a vision that a formal experience broker role will not be needed in an optimal learning organization. When the practises outlined in this paper mature and reach higher company levels, many individuals will be able to act as brokers. Everyone can adopt the company values, and obtain the skills needed to act as brokers or communicators. Experience occasions will occur often, simply when people meet in the corridors, at the copier, or in the lunch queue. So before they buy their meal, they will have been served either an experience they need or the name of a human source with that experience.

10. References

[1] V. Basili, G. Caldiera, H. D. Rombach, "*Experience Factory*", Encyclopedia of Software Engineering Vol. 2 Editor: J. Marciniak, John Wiley and Sons Inc., 1994, pp 528-532

[2] V. Basili, "*The Experimental Paradigm in Software Engineering*", Experimental Software Engineering: Critical Assessment and Future Directives, Springer-Verlag, 1992

[3] I. Nonaka, "*A Dynamic Theory of Organizational Knowledge Creation*", Organization Science Vol 5 No 1 1994, pp 14-37

[4] N. E. Fenton, S. L. Pfleeger, "*Software Metrics, A Rigorous & Practical Approach Second Edition*", International Thomson Computer Press, 1997

[5] A. Burr, M. Owen, "*Statistical Methods for Software Quality*", International Thomson Computer Press, 1996

[6] T. A. Stewart, "*Intellectual Capital: The New Wealth of Organizations*", Brealey Publishing, 1998

[7] PERFECT Consortium, *"The PERFECT Experience Factory Model*", PERFECT Esprit-III Project 9090, 1996

[8] T. Demarco, "*Why does Software Cost so Much? And Other Puzzles of the Information Age*", Dorset House, 1995

[9] W. S. Humphrey, "*A Discipline for Software Engineering*", Addison-Wesley Publishing Company, 1995

[10] K. E. Sveiby, "*The New Organizational Wealth*", Berrett-Koehler Publishers Inc., 1997

[11] J. E. Orr, "*Talking about Machines - an Ethnography of a Modern Job*", IRL Cornell Press, 1996

[12] R. L. Daft, G. P. Huber, "*How Organisations Learn: A Communication Framework*", Research in the Sociology of Organizations Vol 5, pp 1-36

[13] P. Shrivastava, *"A Typology of Organizational Learning Systems*", Journal of Management Studies Vol 20 No 1, pp 7-28

[14] D. A. Kolb, J. Osland, I. M. Rubin, "*Organizational Behavior: An Experiential Approach*", Prentice Hall, 1995

[15] M. S. Ackerman, C. Halverson, "*Considering an Organization's Memory*", Proceedings CSCW 98, 1998, pp 39-48

[16] D. W. McDonald, M. S. Ackerman, "*Just Talk to Me: A Field Study of Expertise Location*", Proceedings CSCW 98, 1998, pp 315-324

[17] H. Mintzberg, "*Structures in Fives, Designing Effective Organizations*", Prentice-Hall International Inc., 1983

[18] I. Nonaka, H. Takeuchi, "*The Knowledge-Creating Company*", Oxford University Press, 1995

Knowledge Management at a Software House

A Progress Report

Peter Brössler
sd&m GmbH & Co. KG, Munich, Germany
Peter.Broessler@sdm.de
http://www.sdm.de

Abstract

This progress report describes the introduction of a knowledge management system at sd&m - a large software house in Germany. It explains the circumstances leading to the implementation of sd&m's knowledge management, the realization process as well as its pros and cons. Today, after nearly two years in operation, we would like to present a first evaluation and some thoughts and ideas for the future.

1. Introduction

Founded in 1982, sd&m has grown to employ nearly 700 employees, most of them software developers. The company specializes in designing and implementing large business information systems tailored to the specific needs of the client, while not restricting itself to a certain technology. Projects at sd&m may range in size and scope, involving from one to 100 team members.

2. Problems encountered

The company's rapid growth (more than 50% in some years) and its size (close to 700 employees today) made it increasingly difficult to maintain a consistently high quality standard for all projects. To remedy this problem, the company introduced an ISO-9001 certified quality management system in 1995. By the end of 1996, though this system helped avoid serious quality problems in our projects, it was felt that this form of quality improvement governed by rules and controls (audits) did not provide the necessary flexibility to really boost our projects.

Problems encountered:

- Acquiring the experience and skills needed in our projects (for instance, C++ programming or project management) takes a very long time.
- PCs, C/S, OO, Internet, etc. are great technologies compared to mainframes, but at the same time they are every project manager's worst nightmare. Just consider how many new versions of various Internet development tools are released within one year!
- Over a short period of time, a significant number of highly qualified and experienced people left the company. Some of them were the *only* experts in specific areas, leaving severe gaps in their projects and in the company.

- When using a technology (i.e. programming language, database system, middleware, CASE tool, operating system, ...) that was new to all team members of a given project, the engineers all too often resorted to the "learning by doing" approach. This approach sometimes led to serious time slips and/or quality problems.

- Often, the knowledge and insight gained with respect to tools, database systems, etc. were garnered in one project but not applied to others. So, instead of actually learning from mistakes, other projects using the same technology did not benefit from the existing experience. Consequently, they ran into the same problems over again, and in the end gathered the same, already available, experience.

- A large number of cases showed a lack of knowledge in the specific project, while this knowledge was actually available in the company! This problem usually occurred either because the project manager and the expert on the respective topic were not aware of each other, or the specialist was so involved in his or her own project that there was no time to support the other project.

- sd&m's projects are generally very challenging and sophisticated, both with respect to requirements and technology. Large corporations with their own extensive IT department (e.g. one of the major German banks) often contract sd&m to take over a particular project, mainly because they do not believe they will be able to successfully implement this project on their own. Thus, some of sd&m's projects could be considered "Red Adair projects". Unfortunately, such projects require the best resources, but not every employee in a company can be the best.

- Since the average timeframe for a typical sd&m project is quite long, a number of bids have been lost due to a proposed delivery date that was too late for the customer.

These problems led to many discussions among management and software development staff. The central question was how to improve the situation. The quality management system as a corrective measure can only detect the consequences of these problems, but is unable to prevent them in the first place.

3. Initial Ideas and Approaches

The first idea was to drastically limit the growth of the company, increase the average experience level and to prevent the worst case scenario, which is having few or no experienced software developers and project managers in a project team. Management eventually decided to limit the growth per year to a reasonable figure (around 25%). Although this decision was an important step in the right direction, it was not even close to solving all of the problems described above.

Another idea was to form an internal "Red Adair group" that could be called upon by troubled projects to help them tackle burning issues. The idea, however, was rejected because it was believed that this approach would not have actually solved the real problem.

It was then discussed to create the position of a "chief technologist", someone who could oversee all problems and offer possible solutions. It was quickly realized, though, that one individual could not possible fulfil such a complex and comprehensive task.

Many other ideas and solutions were offered, mostly directed at improving the educational

situation at sd&m. Some of them have been successfully implemented and are still in place today, such as a lecture series on about twenty important topics given to newcomers by experienced sd&m staff. A better internal training program was an integral part of the solution, but it was felt still that more was needed.

In order to find the right solution, it was necessary to go back to company's roots, to the days when there was a staff size of about 50 employees. They all worked at the same location, the company's headquarters in Munich, and – most importantly – they all shared one single coffee room. And that was it. Sooner or later, every problem that could not be solved by the project team was discussed in the coffee room. It usually didn't take very long before the specialist for this topic was found and the problem was solved. When we compare the situation then with the situation today, the staff size of 50 versus 700 today is not the only difference: Instead of one coffee room, there are now ten, and while the 50staff had to deal with mainframe-based systems only, the number of technical topics today is not only higher, but the topics themselves are much more complex.

Eventually, we came up with the idea that would be successful in solving the sd&m's worst problems. The concept was to utilize as much of the company's knowledge as possible (which is mainly "produced" during project work) by identifying, collecting and spreading it. Furthermore, it was necessary to find a way to ensure that this knowledge was actually used in new projects (i.e. to fight the "not invented here syndrome") and also to promote the reuse of software components. Last but not least, it was crucial to supplement the knowledge and components developed in the course of projects for individual client solutions with more information and experience on a general level. All this was identified by the author as the need for a "knowledge management organization" [1],[3] to help the company to become a "learning organization" [2]. In mid 1997, the author was offered the position of "Technology Manager". The job description was roughly along the lines of a "chief knowledge officer", with a strong focus on technology.

4. Building a Knowledge Management Organization

In early 1997, when the management board of sd&m decided to ask the author to form a technology management group, a major difficulty had to be overcome. The company has a long tradition of avoiding unnecessary overhead activities, it believes in a lean organization that follows the development and the needs of the projects. With this in mind, it was obvious that not everybody welcomed the idea of a central technology group – especially in middle management.

The author approached this challenge by developing a customized knowledge management concept for sd&m, which was then discussed with the entire management in great detail. He knew he could count on the approval of senior management, but also recognized that it was crucial to get as much support as possible from all managers. Many of the specific rules (see below) were developed during these discussions. The overall concept was described in a single overhead presentation, which the author then used to inform the whole company. In a series of about ten presentations, the knowledge management concept was introduced to all the different divisions and branch offices of the company. Besides informing everybody, these presentations also provided valuable additional input for the concept[1]. Last but not least, this was a first chance to attract interested colleagues to join the new group.

---

[1] Many of the discussions took place via email or in newsgroups as follow-ups to the presentations.

At that time, the group consisted of the author only. The goal was to have an operating group in place by early 1997.

What are the most important elements of sd&m's knowledge management[2]?

- The knowledge management group consists of so-called knowledge brokers. Most of them are employed as full-time knowledge brokers, so the group is not just a virtual organization.
- Each knowledge broker is responsible for one of sd&m's core topics[3]. The list of core topics was suggested in the initial concept and has been extended during the course of the following discussions.
- Knowledge brokers report to the technology manager. They work as knowledge brokers for a limited period of time, usually for one to two years. Then, they return back to their unit to work in projects as before.
- As a general rule, each unit contributes one knowledge broker. This eases the process of identifying candidates and also helps to generate a demanding attitude from all units: "We contributed an important consultant, what do we get back?".
- Knowledge brokers act as internal consultants, they support the projects with their knowledge and identify possible internal and external knowledge resources.
- Knowledge brokers also develop specific sd&m knowledge related to their topic. They describe best practice and maintain a knowledge store related to their specific area of expertise.
- The whole group (called "sd&m Technology management" or short: sTm) works as an internal service unit. Their clients are the projects themselves.
- sTm is also responsible for
  - the library,
  - technical education,
  - co-operation projects with partners (vendors, universities, ...)
  - research enquiries,
  - dealing with information providers (such as Gartner Group or Giga).
- Besides approximately 10 knowledge brokers, the technology manager, the secretary, and the librarian, a number of developers are needed to build and maintain technical elements

---

[2] Because the important knowledge at sd&m is mainly technology-oriented, the terms "Knowledge Management" and "Technology Management" were and are used interchangeably.

[3] Current topics encompass: Requirements Engineering, Specification, Software Architecture, Database Systems, Quality Management, Testing, Internet Technology, Middleware Technology, Reengineering and Reuse.

of knowledge management.

5. Integrating the Processes

One of the greatest challenges for a successful introduction of a knowledge management system is its integration into the core processes. In our case, it is crucial that sTm and the knowledge brokers are as closely related to the projects as possible.

Every project starts with a “kick-off” phase, which includes a kick-off meeting, and ends with a “touch-down” phase with a touch-down meeting. This is part of sd&m’s quality management system. With the introduction of sTm, a new rule was added to this standard procedure: sTm has to take part in every kick-off and touch-down meeting. This ensures that a project learns about possible sources as early as possible and gets in touch with sTm, so the project team knows who to contact in case of problems. On the other hand, and most importantly, by attending the touch-down meetings sTm learns about experience gathered during the project, including potentially reusable components. The results of these touch-down meetings become part of sd&m’s knowledge base.

sTm is needed most when a projects runs into a technical problem or when it needs general consulting support in technical matters. Furthermore, sTm can also provide technical input for a bid during the acquisition phase of a project. In all these cases, it is vital to establish a common understanding of the actual problem and how sTm can support the project or the acquisition. For this purpose, a checklist has been developed.

6. Benefits for the Customer

As described so far, our knowledge management supports the projects. This also results in benefits for our customers:

- As more available experience is utilized by a project, fewer lessons have to be learned. For example, a project team does not have to evaluate the market of test tools itself, but can rely on sTm's advice. This cuts down delivery time and sometimes even leads to a lower price for the customer.
- Leading edge technology in particular can introduce many risks to projects. By providing projects with the right technology and consulting service, these risks can be reduced.
- sd&m leaves technology and experience with the customer after completing a project. More often than not, project teams also consist of the client's staff. Therefore, sd&m shares its knowledge with its customers.
- By having knowledge brokers in place, customers can talk to subject matter experts during important project phases. Without knowledge brokers, this would hardly be possible.

7. Technology to Assist Knowledge Management

This is the most obvious part of sd&m’s knowledge management system. Besides organization and processes, the storage and retrieval of knowledge must be supported by appropriate technology. Everybody in the company must be able to retrieve stored knowledge – not just the knowledge brokers.

What we have done (in brief):

- Each knowledge broker is responsible for one or more knowledge stores related to his or her specific topic. All knowledge stores share the same structure and presentational and navigational style. They are part of sd&m's Intranet.
- The technical quality and contents of our databases for employees, customers, partners, projects and acquisitions have been considerably improved. They are Lotus Notes databases.
- These databases have been integrated into our Intranet. Now, every employee has access to basic information on all customers, projects and employees. All relationships between customers, projects, employees and organizational units are hyperlinked in the Intranet.
- A skill database has been developed to store the skills of all employees. The skill database is an experiment in co-operation, because every employee assesses himself or herself and can introduce new skills as part of a skill tree.
- External resources have been integrated into the Intranet, such as reports from Gartner Group, Giga and Forrester.
- A central search engine makes it possible to search the whole Intranet to find all projects, people, documents, research results, books, etc. related to a specific subject[4].
- All of this is called the sd&m Knowledge Web (KWEB).

8. Cultural Aspects

As in probably all organizations that introduce knowledge management, cultural aspects proved to be the greatest challenge. Fortunately, sd&m had a good starting position: being asked something by a colleague is entirely welcomed – "all doors are open". To achieve and maintain a good knowledge management, however, much more is needed: people must be encouraged to behave proactively. This means they should share what they have learned with others, in our case especially with sTm. Very often, though, this just doesn't happen. There are many reasons for this, the most prominent one being lack of time. Therefore, sTm must monitor all traces of useful knowledge very carefully and follow them up accordingly. To this end, the touch-down meetings are an important source of information.

In terms of supporting and encouraging the reuse of software components we have developed specific procedures, which would require a separate report to describe.

The most difficult job was (and to a lesser extent still is) finding knowledge brokers. People come to sd&m to work full-time on projects. The job of a knowledge broker is very different from that. Most notably, every knowledge broker has to be able to deal with many questions and topics at the same time. Not everybody is up to that challenge. As a knowledge broker, the fruits of labour are not as apparent as they are when working on projects. Last but not least, it's a personal change ... Nevertheless, we are happy to report that so far, all positions for knowledge

[4] We are using technology from Verity for this, but have greatly enhanced the search engine with own extensions.

brokers have eventually been filled.

9. Evaluation

In January 1998, sTm was under full steam, with about 15 people actively working in this new group. Considering the staff at that time was around 550 people, this was quite a large number. A minority of these 15 people worked on systems to support the knowledge management (as described earlier). The management of sd&m set the evaluation date for the entire knowledge management for the end of 1998. At that time, there was no doubt that sTm would continue to exist, as there was hardly anyone in the company who could imagine what sd&m would be like without sTm. In the end, sTm was proud to present a long list of satisfied projects.

Although it is very difficult to "prove" a financial pay-off, it can be seen very clearly that the problems described in Chapter 2 do not occur nearly as often as before – despite continuing double-digit growth.

10. Ideas for the Future

Here is a list of some plans and ideas for the future:

- Increase the number of situations for which a "self-service" from a knowledge store is sufficient and no knowledge broker is needed personally.
- Increase the number of useful documents in the Intranet.
- Move away from a pure cost-center by introducing charges.
- Further increase the willingness for sharing and reusing components.
- Offer more than just consulting to projects, e.g. training and even the full responsibility for software development environments.
- Carefully extend the consulting service to customers without neglecting the primary goal.
- Build knowledge management systems for customers.

11. References

[1] **Davenport, Thomas H. and Laurence Prusak:** Working Knowledge: How Organizations Manage What They Know (Harvard Business School Press, 1998)

[2] **Senge, P**. (1990). The Fifth Discipline: The Art and Practice of the Learning Organization, Doubleday Currency, New York NY

[3] **Stewart, Thomas A. Intellectual Capital**: The New Wealth of Organizations (Currency/Doubleday, 1997)

Part 3:

Process-centered Approaches to LSO

Process Support for Virtual Software Organizations[1]

Frank Maurer & Barbara Dellen
University of Calgary
{maurer, dellen}@cpsc.ucalgary.ca

Harald Holz
University of Kaiserslautern
holz@informatik.uni-kl.de

Abstract

In this paper, we describe how a flexible, Internet-based process enactment engine can be used to support software process improvement. We present the MILOS process modeling language, show how the MILOS system can be used for process support and explain why the process-centered approach to knowledge management supports a learning software organization.

1 Introduction

In many companies, software development still is more of an art than an engineering discipline. To overcome the resulting problems, companies are trying to improve their software processes following, for example, the capability maturity model (CMM) or similar approaches. One key issue here is to transform the company to become a learning organization that uses past experiences to improve future development activities. A core task in this transformation is process modeling and related activities: the development of explicit descriptions of how software is created and maintained by the company. These descriptions are needed to analyze process maturity as well as a baseline and starting point for any improvement initiatives. Most often, these descriptions are textual: Companies create process manuals and rely on their employees to interpret the contents of these documents while executing development processes.

To reduce the problems of textual descriptions (e.g. ambiguity, missing descriptions of some aspects of the software engineering process), various formal or semi-formal process modeling languages were developed [Ost-87, CK-92, AK-94, Ver-94]. (Semi)-formal software process models describe the activities carried out in software development as well as the products to be created and the resources & tools used. However, even (semi-)formal process models cannot ensure that the „real“ development process follows the prescribed model; they still rely on humans to interpret and follow the model.

In order to overcome this problem and to provide active guidance during process execution, enactment engines have been proposed. Enactment engines use process models as the actual basis for the coordination and the management of software engineering activities. They improve the efficiency of software processes and increase the chances that the process model and the real-world process stay in sync.

---

[1] This paper is an extended version of the paper “An Internet Based Software Process Management Environment” presented to a small audience at the ICSE 98 workshop on “Software Engineering over the Internet”.

Basically, an enactment (or workflow) engine is an interpreter that operationalizes process descriptions and guides its human users in their software development task. The workflow management coalition defines workflow management as:

"The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules." [WFMC-96].

Today, workflow management approaches are basically only used to support repetitive administrative tasks whereas knowledge-intensive tasks are not supported. There are several reasons for this:

- The knowledge needed for executing the process is not explicitly described in a workflow model.
- Current workflow approaches are not flexible enough to adapt on the fly to changing processes.[2] Our work in the past years was addressing this problem.

Process support includes improved communication, guiding people when performing processes, improving both processes and their results, and automating process steps [RV-95]. Process models are the basis for enactment engines and can be reused for future processes. They form, at least in our opinion, one key asset of a learning software organization. Software process modeling and enactment is one of the main areas in software engineering research. Several frameworks have been developed (e.g. procedural [SOH-95], rule-based [KFP-88, TKP-94, PSW-92], Petri net based [BFG-93a], object-oriented [CHL-94]). Process-sensitive software engineering environments which support evolution of executed process models [CFF-93] are a focal point of current research, but the results are still immature [MP-93]. Several approaches to support the evolution and flexibility of software processes were developed [BFG-93b, BK-93, CNG-95, PEM-95]. None of these approaches is supporting globally distributed software processes and their evolution (although work in this direction is starting [Kai-97, Con-97]).

Published approaches on process evolution mainly deal with changing the enacted process model so that it reflects the changed real world process. Automatically sending notifications about the changed process/products over the Internet to the appropriate members of the global team is not supported. An approach that recognized the necessity to allow the project members to express interest in certain changes is Serendepity [GH-98]. Unfortunately the user has to explicitly state about which changes he wants to be notified. There is now way to make use of abstraction.

Automatically deducing potential receivers of change messages requires managing knowledge about change dependencies which typically occur between processes and products. A research area dealing with change dependencies between and in products is software change impact analysis [BA-96].

Managing software process knowledge is also the goal of the experience factory approach [Bas-89, BCR-94]. They distinguish between the organizational structure to manage the software knowledge (the experience factory department) and the activities that have to be carried out to build an experience factory.

Another problem in software development projects is the shortage of skilled workers. It is often impossible to find appropriate people locally. This fact forces companies to create virtual teams (or even virtual enterprises) with members distributed all over the world. Process models are a means that can be used to educate members of the virtual team in how the (customer) company wants the software to be developed.

In this paper, we present a process-oriented view on knowledge-intensive tasks in software development. We describe the MILOS system, a Web-based process support system that improves the coordination and information exchange of virtual teams. We believe that such an approach is

2 This also holds for business process modeling tools like, for example, ARIS. They are not targeted to supported process changes on the fly but follow a two phase model: (phase 1) modeling and code generation (phase 2) process execution

necessary to create virtual software organizations that learn and improve. MILOS' flexible workflow engine allows to create, refine, and adapt project plans during enactment; therefore, it is suited for the highly dynamic, distributed environment of virtual teams. Its traceability and change notification mechanism supports team members in coping with changes. The MILOS system will support the reuse of process knowledge by allowing a project planer to import partial plans from an experience base into the current project and tailor it to the project's specific needs.

In Section 2, we discuss our process modeling language and explain how knowledge is linked to generic software processes. The next section gives an example. The architecture of our system is described in Section 4. The last section discusses the results of the paper.

2 The Process Modeling Language MILOS

This section explains the main concepts of the process modeling language MILOS are presented, as far as it is necessary to understand this paper omitting syntactical details. MILOS was developed in cooperation with the working group of Prof. Rombach and integrates basic process modeling [Ver-94] and knowledge management concepts. The rationales for the introduced concepts were already presented in a variety of publications which describe MVP-L[Ver-94] and CoMo-Kit [DMP-97]. Moreover, a recent publication [DMMV-97] explains the requirements for the language concepts from a common point of understanding. However, the process modeling concepts in MILOS reflect common process modeling concepts.

The core notion of MILOS is the *process*. Any other information is grouped around this structure, providing a *process-centered structuring* of knowledge: Products are related to processes by input or output relationships. Furthermore factual knowledge, like guidelines, business rules, studies etc. is linked to processes. Resources are assigned to processes by roles or qualifications.

2.1 Product models

MILOS allows the specification of hierarchical, object-oriented product models. A *product type* defines the structure of a set of products with the same behavior. A product type is either basic or complex. Basic types are predefined and are external references. Currently, we support URLs, MS Word documents, Rational Rose files, Java and C source code.

A complex type consists of one or more typed subproducts and attributes. Complex product types define hierarchical type structures. Complex product types can be specialized (IS-A relation). The current MILOS prototype does not yet implement complex product types.

2.2 Process Models & Project Plans

Process models specify processes and relate process specifications by control flow and product flow relationships. A process is defined by a description of the process goal, a set of conditions, products needed to execute the process, a set of contexts (URLs pointing to background knowledge), a set of alternative methods to reach the process goal, the products to be produced and resource allocations.

Inputs of processes may be either products that are produced by other processes during project enactment or predefined contexts taken from generic process models. Here the process and the knowledge view in MILOS is integrated. Outputs of process represent the products to be created when the process is actually executed.

Every process is associated with a set of roles and qualifications which are needed to perform the task (e.g. the process "Implement Class" is associated with the qualifications "Java knowledge available" and the role "Programmer").

Methods may be either *complex* or *atomic*. Complex methods refine a process into one ore more subprocesses while atomic methods are used to state that a process is a leaf in the process hierarchy. The reason why we represent atomic methods explicitly is to allow prescribing specific ways to solve

a leaf process. Complex methods furthermore specify the product flow between their subprocesses.

Process models are generic descriptions of the project's processes, and their interdependencies. The models have to be specialized and customized within the projects in form of *project plans*. Project plans in MILOS are very similar to process models. They describe process decomposition and information flow. We integrated MS Project into the MILOS system for planning and scheduling purposes. We extended MS Project to allow the definition of information flows.

2.3 Resource Models

Resource models allow for assigning roles and qualifications to project team members. Roles and qualifications can be specialized. Resource models can be used to map qualifications and roles required for process execution (see section 2.2) to concrete persons.

3 Example scenario

In the following, we want to illustrate our system's functionality with the help of an example scenario describing our own project's software development process.

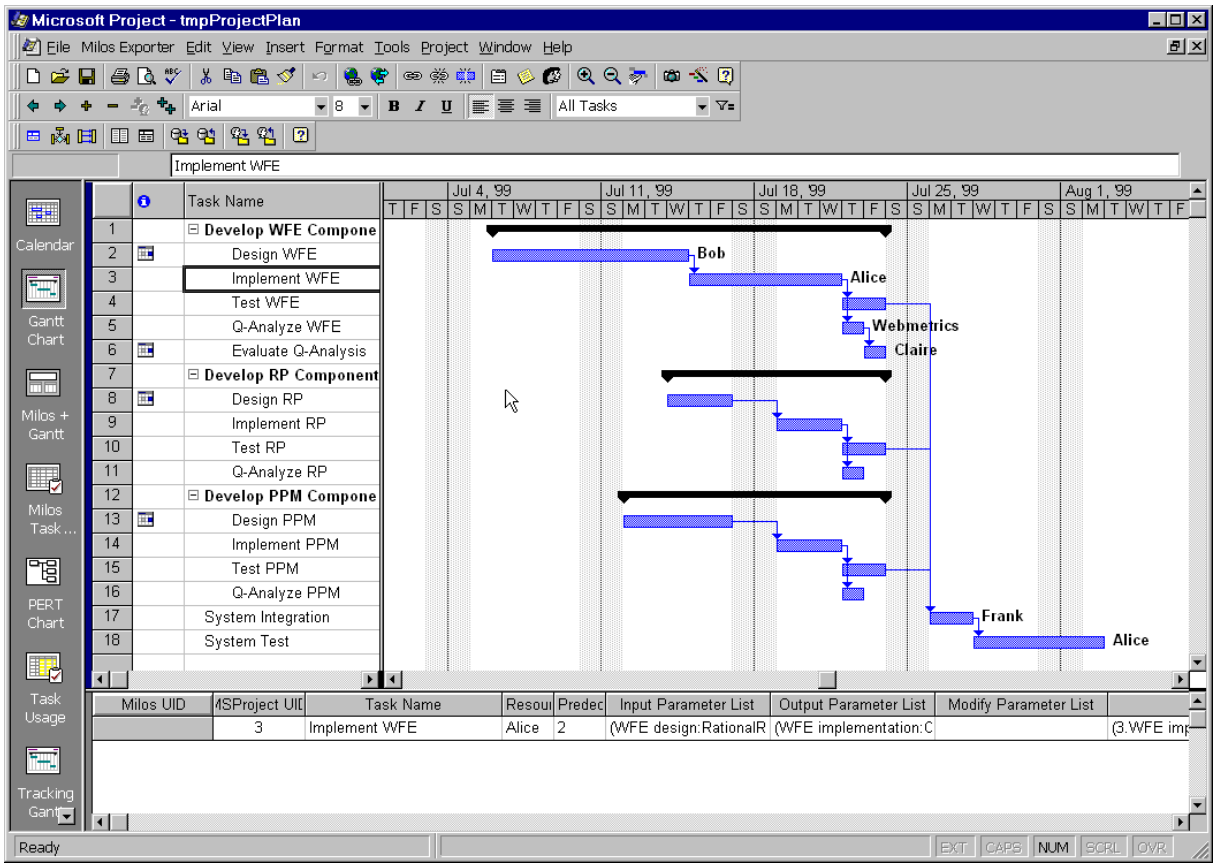

Fig. 1: Example plan created with MS-Project.

Based on the system's architecture, the project planner creates an initial plan using some standard software. The screenshot taken from MS-Project in Fig. 1 shows a simplified example plan for developing the MILOS architecture. For each of the three components *Project Plan Management* (PPM), *Workflow Engine* (WFE) and *Resource Pool* (RP), the plan contains a task[3] describing the component's development process. These tasks are complex, i.e. they are refined by a set of subtasks that describe the activities required to perform the task in more detail. As Fig. 1 shows, the complex

3 In the following, task, activity and process are used synonymously.

task *Develop WFE component* consists of the subtasks *Design WFE*, *Implement WFE*, *Test WFE*, *Q-Analyze WFE*[4] and *Evaluate Q-Analysis*. Also shown is some scheduling information, i.e. planned start and finish times, duration as well as team members assigned to each task. In addition to the information shown in Fig. 1, the plan also contains a loop from *Test WFE* back to *Implement WFE*. This loop is modeled by specifying a product flow between those two processes.

For plan enactment, the project planner exports the plan from MS-Project into MILOS. From now on, team members, regardless of their geographical location, can log into MILOS via standard Web browsers and are provided with individual workspaces. Figure 2 shows the current workspace of team member Alice. According to the project plan, she is responsible for the tasks *Implement WFE* and *System Test*. Consequently, these task appears on her to-do list.

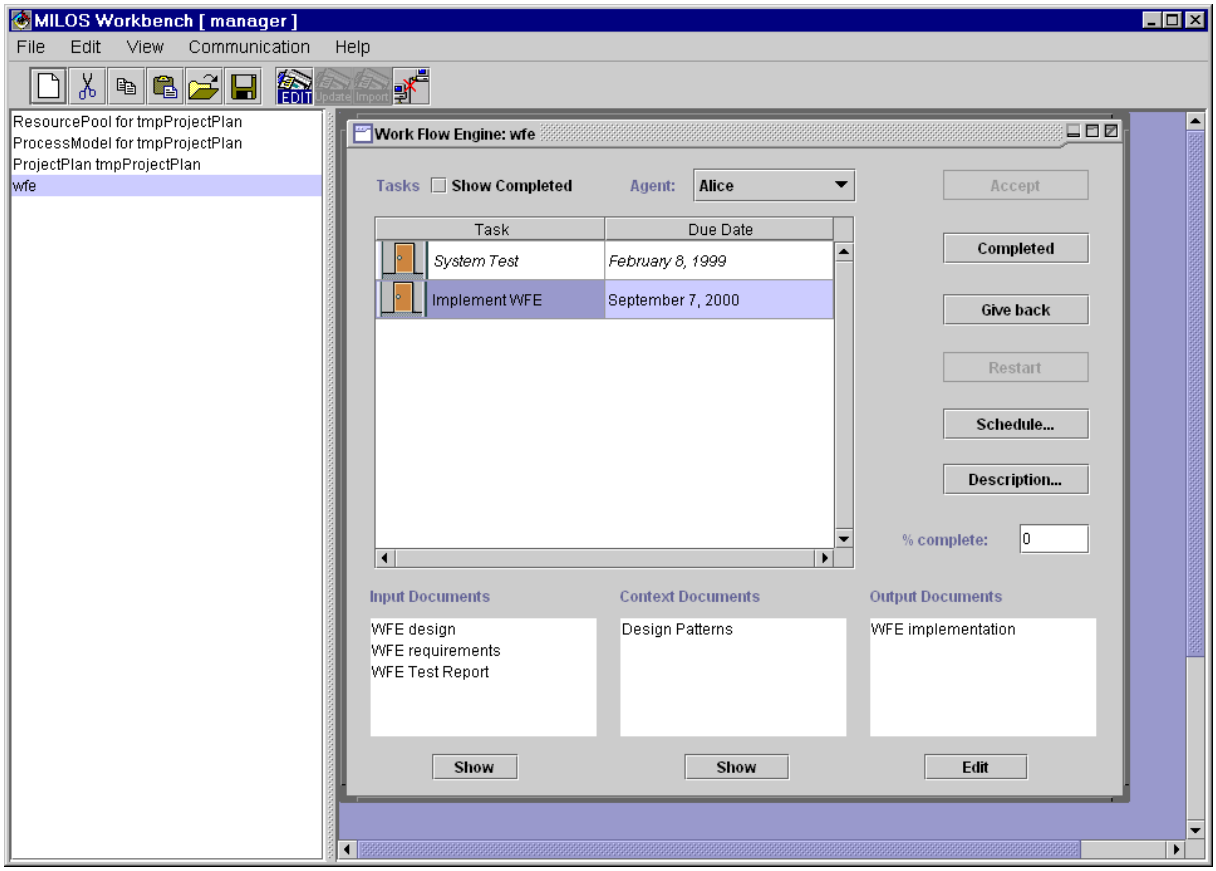

Fig. 2: MILOS workspace for Alice.

The workspace allows Alice to browse the information associated with each task, e.g., a more detailed task description and scheduling information. In particular, she is given access to any documents needed to execute the task. Hence, the list of input documents (WFE design, WFE requirements, WFE Test Report) as well as the list of output documents (WFE implementation) is displayed for *Implement WFE*. In addition, the context document lists contains background information (here: a link to a Design pattern URL). The corresponding documents can be opened it with the appropriate tools (in this case: Rational Rose for the WFE design and MS Word for WFE requirements).

As soon as the required input document *WFE design* has been released by Bob, the team member who is responsible for task *Design WFE* (see Fig. 1), Alice is notified that the inputs of task *Implement WFE* are now available (Fig 3).

4 Quality analysis of code.

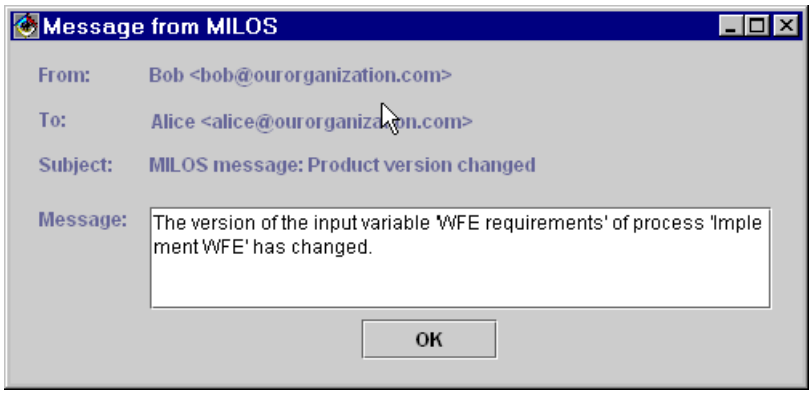

Fig. 3: Input changed notification

After having inspected the design document, she forecasts her start and finish times for the implementation. In the case that her forecast violates the project schedule, the planner receives an automatically generated notification about this problem. However, since her forecast conforms to the schedule, no notification is sent.

After selecting the output document *WFE implementation*, a click on “Edit” starts the appropriate Java implementation environment for Alice. At the end of each day that she is working on this task, she can save her work and specify a "percentage complete" value for it. This value will be exported from MILOS back to MS-Project in order to provide the planner with up-to-date information on the project.

When she completes the task, it will be removed from her to-do list. In addition, the document *WFE implementation* is released, to the effect that the two succeeding tasks *Test WFE* and *Q-Analyze WFE* become executable. According to the plan (see Fig. 1), the former has not been assigned to any team member yet, while the latter has been assigned to a software agent (Webmetrics) that is a wrapper around a specific software metric tool. The software agent performs the task automatically as soon as it becomes executable.

Depending on the metrics gathered, the planner refines the task *Test WFE* to white-box testing. He defines two new subtasks *Write WFE Test Cases* and *Run WFE Test Cases* (Fig. 4).

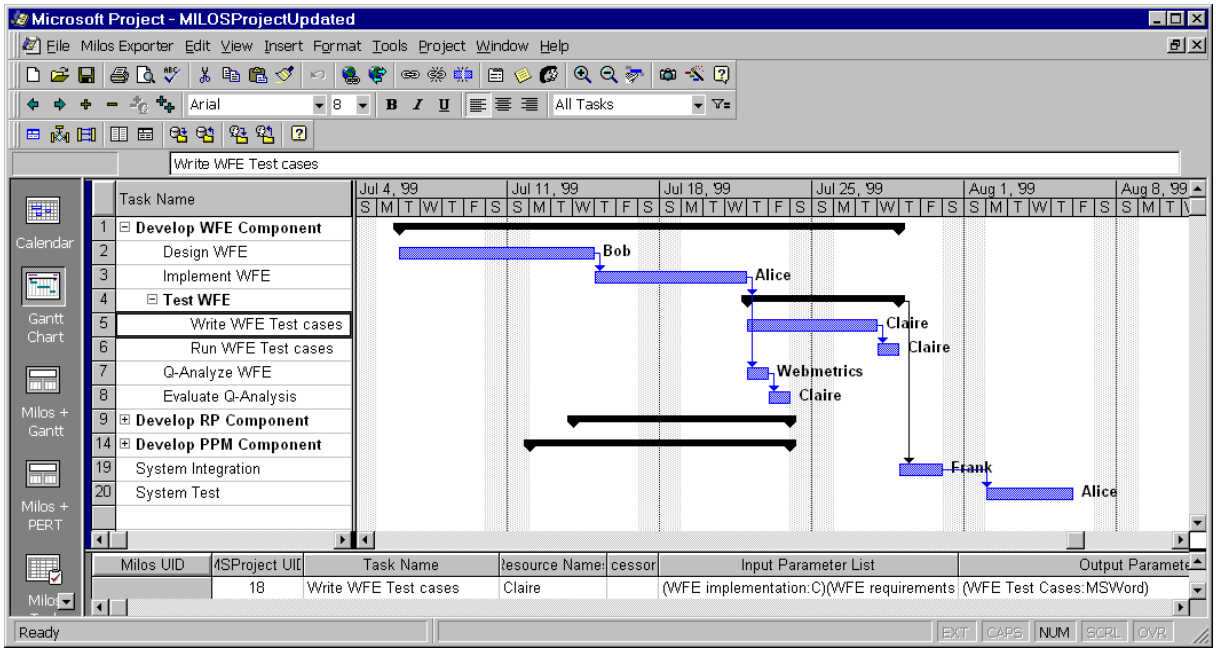

Fig. 4: Updated project plan

After finishing the plan update, the planner exports it again into MILOS. This causes the two newly created tasks to appear on the to-do lists of team members the tasks were assigned to (here: Claire).

If Claire encounters problems with Alice's code during task *Run WFE Test Cases*, an MS-Word document containing a description of the problems will be created as output. The presence of this document will cause a notification to Alice stating that a new test report is available. Alice will then check this report and may then restart her implementation task. Alice will later release a new version of the document *WFE implementation* as soon as she has corrected the code. Analogous to the restart of task *Implement WFE*, the release of a new *WFE implementation* document version will cause a restart of the succeeding tasks *Test WFE* and *Q-Analyze WFE*. That way, a single restart might cause a "restart-cascade" that reaches all tasks affected by a change in a document.

4 Architecture of the System

We now want to discuss the MILOS architecture from two perspectives. First, we illustrate the architecture from a point of view showing where specific kinds of knowledge reside. Then, we concentrate on the technical perspective.

4.1 Knowledge Structuring Perspective

Process models are not only basis for understanding, and improving the project's processes, but may serve as knowledge bases for supporting project enactment. Project enactment support systems should guide developers and managers through the project, by organizing their to do lists, and providing knowledge required to execute the tasks properly. We distinguish between three different views on project knowledge (see Fig. 5).

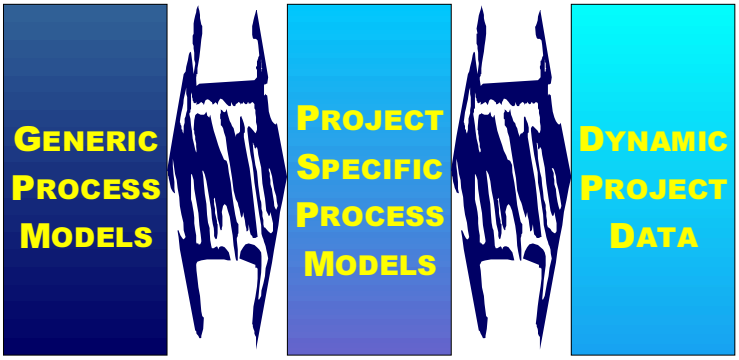

Fig. 5: Three-tier architecture

This three-tier architecture allows to distinguish between

- reusable/generic process model knowledge,
- project plans (representing knowledge required for a specific project only, e.g., schedule information), and
- data created during project execution.

Within each tier, the information is structured in a process-oriented fashion: information is linked to processes to be carried out in the course of the project.

Instead of searching in the whole body of knowledge available in the system, the user working on a specific task sees direct links to the knowledge associated with this task. This approach provides in-context access to knowledge needed for specific tasks.

4.1.1 Generic Process Models

The first tier handles generic and reusable process models and associates generic knowledge to the entities of the process model. The level of abstraction may reach from general proceedings to domain specific processes models. Process models should be stored in experience bases for further reuse [Bas-89]. Knowledge may be stored in several forms:

- Concrete knowledge chunks for human use, are stored URL references pointing to HTML pages or other files containing information in specific document formats. Access to this can be guaranteed using standard web mechanisms of binding file types to specific applications. Portability and cross-platform access is supported as long as the mapping of file types to applications are valid and the application software is available on the client platform.
- Predefined queries are used when the knowledge chunk can not be defined explicitly. The query describes the knowledge needed intentionally. We can at least distinguish between several kinds of queries:
    - Database queries: To access knowledge stored in a relational database, the process model has to contain SQL queries that return the appropriate result. Example: “determine the average time required in the past for the design phase of a medium-sized project”
    - Information retrieval requests: Unstructured information can be accessed using information retrieval technology. The request may be posted to a special IR server or to one of the public web search engines. Example: “Find information on JINI from the Javasoft Web site and from Yahoo”
    - CBR requests: Case-based reasoning technology integrates structured and unstructured queries. It allows for similarity-based queries on structured data whereas relational databases mainly support Boolean queries and IR systems work with unstructured data. Example: “Find all projects in our organization with similar characteristics”

During project enactment, the access to generic process models and the associated knowledge is restricted to read only.

4.1.2 Project Specific Process Models (Project Plans)

The second tier contains project plans that contain knowledge about the tasks to be done and the knowledge related to them. They are a basis for project enactment and coordination. By incorporating a workflow engine, a project plan serves as basis for *actively guiding* human users in their work. Using the single representation trick (known from machine learning), the mapping of generic process models to project specific process descriptions is easy: We use the same representation for both tiers and are able to copy generic models to a project. Then the generic descriptions are manually tailored to be used in the specific project.

A project plan can be made up of several different process models. For instance, the test processes of a project plan are taken over from a model for testing, whereas the general course of action is copied from a waterfall model.

Using generic process models as basis for project planning, even inexperienced project managers are able to come up with a plan according to the company’s quality standards and procedures. In general, the project starts using an initial plan that defines the general course of action and the first project steps. Customizing extends over the project start. Based on the results of early project activities (for example information gathering activities) parts of the plan are further customized during project execution. So far, our approach does not provide automatic tools (beside the access to all process models) for the tailoring step.

From the technical point of view copying models from the first tier to the second tier is relatively easy but conceptual questions remain:

- Which parts of a process model shall be copied (selection)?
- How to determine that a generic model can be reused in a given situation?
- How to support the customization and adaptation process?

4.1.3 Dynamic Project Data

The third tier handles dynamic data which is the core of a flexible process engine[5]: The state of the work process and its tasks, do-do lists for its users, the products created during process enactment, causal relationships between process entities, etc.

The information stored in this tier is created during process execution: it is the output of the work processes and scheduling information. In software development processes, this includes e.g., requirements specifications, design documents, rationales, source code, measurement results, etc.

The third tier provides a *process-oriented view* on the data created during task enactment. Users are able to access information based on the processes carried out. Furthermore they can trace the information flow in the project (thereby tracing where and by whom a specific information was used).

4.2 Change Management Support

Process changes may occur in every tier during project enactment. Supporting changes requires coping with possible impacts on related knowledge chunks located in the same tier as well as in different tiers. Process model changes, for example, may affect the state of the processes in execution. On the other hand, a changed process outcome may affect planning decisions. Part of our past, and current work addresses this problem [DMP-97]. The resulting system provides mechanisms that support to

- change project information within each tier on the fly,
- identify affected entities (processes, products and resources)
- inform affected developers and managers about the change,
- trace the reasons for the change, and
- (automatically) return to a consistent project state.

The next section discusses the technical realization of these mechanisms.

4.3 Technical Perspective

The architecture of the MILOS system is shown in Figure 6. It uses Java RMI technology for communication between client and server.

Interactive thin clients run as Java Applets and use helper applications to integrate software development tools. This requires that the tools are installed on the client side and that the browser is configured to start the appropriate tools. We integrated MS Project as the primary project planning client. We also developed a software agent wrapper that is used to run tools that need no user interaction (e.g., we currently integrated a software metrics tool using the agent wrapper).

5 In this paper, we do not discuss the design and implementation of our workflow engine and its unique features.

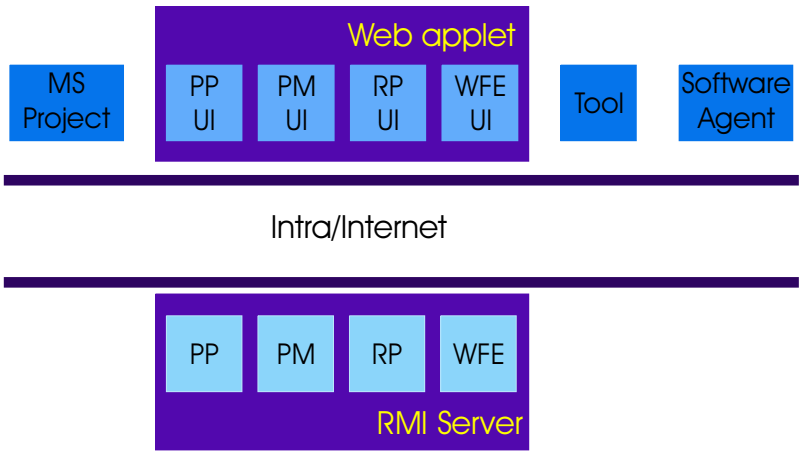

Fig. 6: MILOS system architecture

The MILOS system consists of four components on the server side (with user interfaces on the client side).

- One problem companies have is finding the right people for a specific job: although somebody with the "right" qualifications may work for the enterprise, the company has not yet learned to assign her to the task. The Resource Pool (RP) overcomes these problems by managing information about resources. It allows describing the qualifications, roles, and responsibilities of agents as well as their place in the organization. The RP supports a partial match to determine a set of agents that match specific criteria. Using this component, a user is, for example, able to find an agent that works in Calgary, has Java experience and is a system architect.
- The process model component (PM) stores process descriptions and (alternative) methods describing the information flow. It represents generic process models and, hence, is an experience base.
- A project plan (PP) enriches process model information with scheduling data as well as with method decisions.
- The workflow engine (WFE) operationalizes the project plan and manages all information that is created during enactment. Later includes the states of all processes and versions of products. It also contains a component for handling causal relationships and change notifications (based on event-condition-action (ECA) rules).

All components are connected using event notifications. Whenever a change occurs (e.g. a new task is added to the project plan or an output is deleted from a process description), a corresponding event is propagated to all objects interested in this specific change. The change events are used for two purposes:

(1) To keep the internal data structures of all components consistent and

(2) To trigger ECA rules that are used for generating change notifications to users

ECA rules are automatically generated by the MILOS system to represent information flow dependencies and scheduling dependencies. The first is based on the heuristic that the decisions made in a task needs to be reconsidered when the task inputs are changed. The later used to detect inconsistencies in the schedule. The change management component is based on ideas from [DKM-96].

5 Discussion

In this paper we described an Internet-based process-centered knowledge management environment for software development. The environment structures knowledge around work processes: In the center of our representation are processes. Linked to a process, the user can find methods (describing ways how to perform the process to reach its goals), products (input and outputs to the process),

factual knowledge (implemented as web references), and knowledge about the qualifications needed to perform the process.

The process-centered structure of the system has the following advantages:

- Processes are „natural“ entities for managers and team members: they are well used to thinking about the process (e.g., for project planning).
- For their daily work, people don’t need knowledge per-se but the knowledge they need for performing specific task. A process-centered knowledge management system associates explicitly the task with the knowledge needed for it.
- By linking web references to task, the lost in hyperspace problem is reduced because the user immediately finds the knowledge needed instead of being forced to browse to relevant pages.
- Our approach provides a technical basis for creating a closed loop of process modeling, project planning, and project enactment. Having this loop is essential for any learning software organization because it can learn from the experiences gathered in the execution of projects by updating the process models. We provide an operational environment for this loop. A process engine that guides the human users in their daily work interprets the explicit description of processes. This guidance is especially useful for new employees because they lack the knowledge about the standard procedures of a company.
- Process guidance is important for creating a learning organization: it provides a way to transfer knowledge and best practices about software development processes to inexperienced team members. And it does so in an non-intrusive manner: people are guided by the process engine in their daily work and can access additional information about the current task (e.g., programming guidelines or manuals) when they think it is necessary.

A prototype of the MILOS system is completely implemented and will be made available as open-source by the end of Summer 1999. The restrictions of the prototype are so far:

- It supports a limited number product types
- It has to user interface for uploading a process model to MS project and for transferring process models back to the experience base.

There is no industrial experience with the approach so far. To gather initial experience with the system, we will start using the MILOS system for supporting our own development process (with people in Kaiserslautern, Germany, and in Calgary, Canada) by the end of June 1999.

Acknowledgements

We thank our colleagues Boris Koetting and Sigrid Goldmann for fruitful discussions of the system design and their help in the implementation. We thank Fawsy Bendeck for the design and implementation of the interface to MS project. The work described in this paper is sponsored by Nortel, NSERC and the University of Calgary as well as by the German Research Foundation DFG and the University of Kaiserslautern.

References

[AK-94] J. Armitage, M. Kellner. A conceptual schema for process definitions and models. In D. E. Perry, editor, Proceedings of the Third International Conference on the Software Process, p. 153–165. IEEE Computer Society Press, 1994.

[BA- 96] S. Bohner, R. Arnold: Software Change Impact Analysis, IEEE Computer Society Press, 1996

[Bas-89] V. R. Basili, "The Experience Factory: packaging software experience," in Proceedings of the Fourteenth Annual Software Engineering Workshop, NASA Goddard Space Flight Center, Greenbelt MD 20771, 1989.

[BCR-94] V. R. Basili, Gianluigi Caldiera, and H. Dieter Rombach, "Experience Factory," in Encyclopedia of Software Engineering (John J. Marciniak, ed.), vol. 1, pp. 469--476, John Wiley Sons, 1994.

[BFG-93a] S. Bandinelli, A. Fuggetta, S. Grigolli. Process Modeling-in-the-large with SLANG. In IEEE

Proceedings of the 2nd International Conference on the Software Process, Berlin (Germany).

[BFG-93b] S. Bandinelli, A. Fuggetta, C. Ghezzi. Process Model Evolution in the SPADE Environment. IEEE Transactions on Software Engineering. Special Issue on Process Evolution, December 1993.

[BK-95] Douglas P. Bogia ad Simon M. Kaplan. Flexibility and Control for Dynamic Workflows in the wOrlds Environment. Proceedings of the Conference on Organizational Computing Systems, November 1995

[BK-93] I. Ben-Shaul, G. Kaiser: Process Evolution in the Marvel Environment, 8th International Software Process Workshop : State of the Practice in Process Technology, March 1993.

[CFF-93] R. Conradi, C. Fernström, A. Fuggetta. A conceptual framework for evolving software processes. ACM SIGSOFT Software Engineering Notes, 18(4): 26–35, October 1993.

[CHL-94] R. Conradi, M. Hagaseth, J.O. Larsen, M. Nguyen, G. Munch, P. Westby, W. Zhu: EPOS: Object-Oriented and Cooperative Process Modeling. In PROMOTER book: Anthony Finkelstein, Jeff Kramer and Bashar A. Nuseibeh (Eds.): Software Process Modeling and Technology, 1994, p. 33-70.Advanced Software Development Series, Research Studies Press Ltd. (John Wiley).

[CK-92] B. Curtis, M. Kellner, J. Over: Process modeling. Communications of the ACM, 35(9): 75–90, Sep 1992.

[CNG-95] G. Cugola, E. Di Nitto, C. Ghezzi, M. Mantione. How to deal with deviations during process model enactment. In Proceedings of the 17th Int. Conf. on Software Engineering (ICSE 17), 1995.

[Con-97] R. Conradi: Cooperative Agents in Global Information Space, http://www.idi.ntnu.no/~cagis/

[DKM-96] B. Dellen, K. Kohler, F. Maurer: Integrating Software Process Models and Design Rationales, Proceedings Knowledge-Based Software Engineering KBSE-96, IEEE press, 1996.

[DMMV-97] B. Dellen, F. Maurer, J. Muench, M. Verlage Enriching Software Process Support by Knowledge-based Techniques, Int. Journal of Software Engineering and Knowledge Engineering, Volume 7, No. 2, pp. 185-215, 1997.

[DMP-97] B. Dellen, F. Maurer, G. Pews: Knowledge-based techniques to increase the flexibility of workflow management, Data & Knowledge Engineering Journal, Vol. 23 No. 3, page 269-295, September 1997.

[GH-98] J.C. Grundy and J.G. Hosking: Serendipity: integrated environment support for process modelling, enactment and work coordination, Automated Software Engineering: Special Issue on Process Technology *5(1)*, January 1998, Kluwer Academic Publishers, pp. 27-60.

[Kai-97] G. Kaiser: OzWEB, http://www.psl.cs.columbia.edu/ozweb.html, 1997.

[KFP-88]G.E. Kaiser P.H. Feiler, S.S. Popovich: Intelligent Assistance for Software Development and Maintenance (IEEE Software, May 1988).

[MP-93] N. Madhavji, M. Penedo. Guest editor's introduction. IEEE Transactions on Software Engineering, 19(12):1125–1127, December 1993. Special Section on the Evolution of Software Processes.

[Ost-87] L. Osterweil, Software Processes are Software Too, Proceedings of the Ninth International Conference of Software Engineering, Monterey CA, March 1987, pp. 2-13.

[PEM-95] G. Pérez, K. Emam, N. Madhavji: Customizing Software Process Models. In Proceedings of the 4th European Workshop on Software Process Technology, pp. 70 -78, Springer, 1995.

[PSW-92] B. Peuschel, W. Schäfer, S. Wolf: A Knowledge-based Software Development Environment Supporting Cooperative Work. In International Journal on Software Engineering and Knowledge Engineering, 2(1): 79-106,1992.

[RV-95] H.-D. Rombach, M. Verlage. Directions in software process research. In M. V. Zelkowitz, editor, Advances in Computers, vol.41, pages 1–63. Academic Press, 1995.

[SOH-95] S. Sutton, L. Osterweil, D. Heimbigner: APPL/A: a language for software process programming, IEEE Transactions on SE and Methodology, Vol. 4, No. 3, p. 221-286, 1995.

[TKP-94] A. Tong, G. Kaiser, S. Popovich, A Flexible Rule-Chaining Engine for process Based Software Engineering, 9th Knowledge-Based Software Engineering Conference, September 1994

[Ver-94] M.Verlage, Multi-view modeling of software processes, in: B. C. Warboys, ed., Proc. Third European Workshop on Software Process Technology, (Springer Verlag, 1994) 123-127.

[WFMC-96] Workflow Management Coalition: Terminology & Glossary, http://www.aiai.ed.ac.uk/WfMC/DOCS/glossary/glossary.html

Using Software Process to Support Learning Software Organizations

Scott Henninger
Department of Computer Science & Engineering
University of Nebraska-Lincoln
Lincoln, NE 68588-0115
scotth@cse.unl.edu

Abstract

An initial and key ingredient to creating a learning software organization is to create a repository containing software development experiences. These repositories have been the subject of much research, but the strategy of using the repository or integrating it into the development process has thus far been limited to providing search facilities to find relevant information. While search engines will always be an integral part of repository design, our experiences indicate that it is insufficient to lead to satisfactory results. This had led us to begin investigating how software process can be used as a driving force for collecting and disseminating software development knowledge. In this regard, we have combined an organizational learning meta-process with a rule-based software process engine to create a tool that provides decision support for tailoring software processes to the needs of individual development efforts.

1. An Organizational Learning Approach to Software Development

An organizational learning approach to software development captures project-related information during the creation of individual software products [Henninger et al. 1995]. This information can then be disseminated to subsequent projects to provide experience-based knowledge of development issues encountered at the organization. In the past few years, we have demonstrated through this approach an exploratory prototype, named BORE (Building an Organizational Repository of Experiences) [Henninger 1997]. In BORE, cases represent project issues and contain information about how issues were resolved and the resources used to resolve the issues [Henninger 1996a]. Cases are stored in a repository that can be searched and browsed to find issues with similar characteristics [Henninger 1996b; Henninger 1997; Henninger et al. 1995].

The case-based technology is further supported by a process that captures information such as "lessons learned," tips and techniques for accomplishing specific tasks, and reusable source code. As cases accumulate in the organization-wide repository, the knowledge contained in the repository becomes increasingly tailored to the kinds of design problems that frequently occur in the organization. This information can be analyzed and generalized in a domain analysis [Arango 1989] or software factory [Basili et al. 1992] setting to create generalized knowledge with broader applicability than the context-specific cases. Furthermore, we can begin embedding this knowledge in design environments that automate parts of the development

process. We have referred to this as the "domain lifecycle," which refers to the progression of knowledge within an application or technical domain [Henninger 1996b].

The repository therefore serves not only as a means to disseminate design knowledge, but also helps an organization *learn* what does and does not work for their development context. This is where we distinguish between organizational *memory* systems that many have advocated [Terveen et al. 1995; Walsh, Ungson 1991] and our notion of organizational *learning*, where the emphasis is placed on learning from previous experiences, which has also been applied to the organizational level as “learning organizations [Senge 1990].

Recent work has begun to emphasize establishing continuous improvement processes that improve product quality and developer productivity, while recognizing past experiences as a catalyst for the learning process. While it is desirable to systematically institutionalize knowledge as it is generated by people in the organization, recent experiences with BORE have indicated that this must be balanced with procedures that control the content, structure and validity of the knowledge. In the following, we first elaborate on the domain lifecycle, then describe some of our experiences with the BORE system and then describe how we have begun to couple the organizational learning approach with a rule-based software process environment to further support software development efforts.

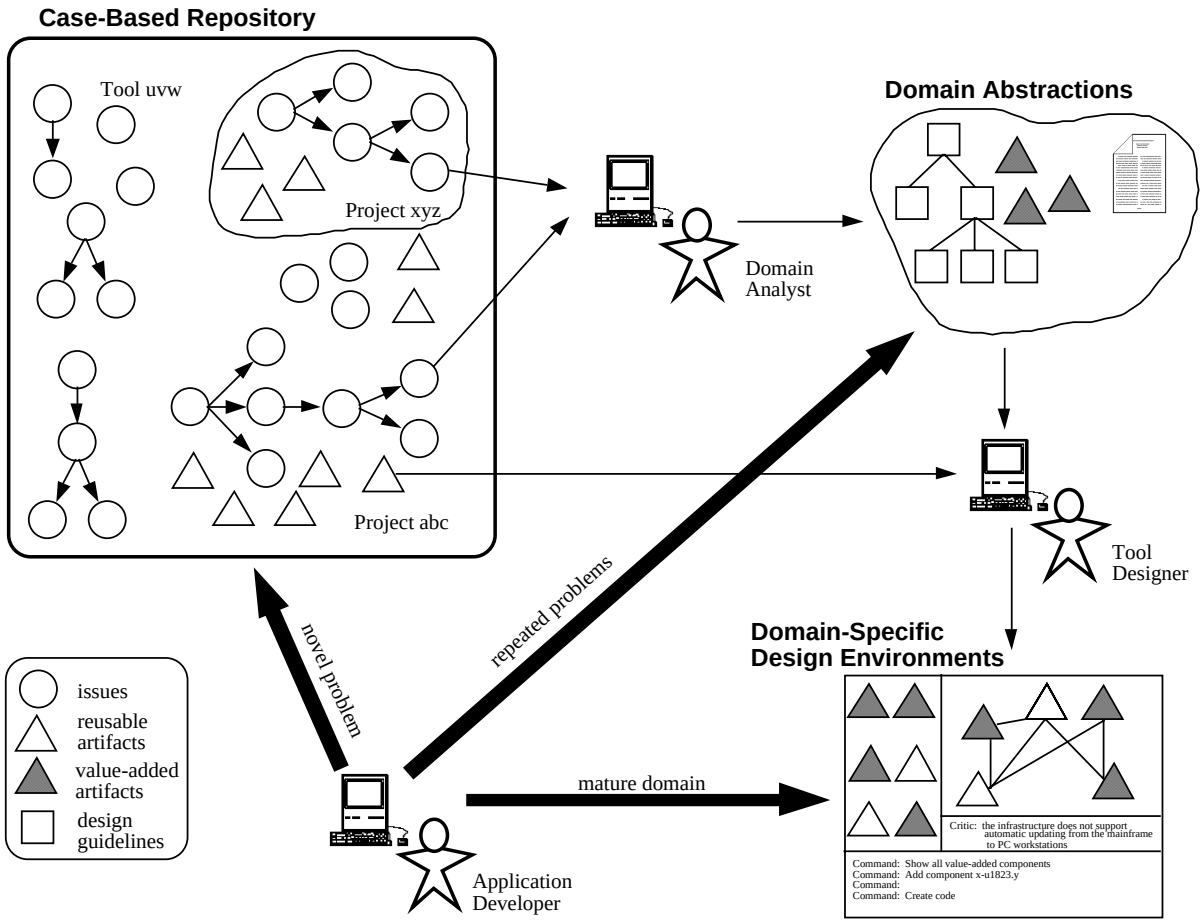

Figure 1: The Domain Lifecycle.

2. The Domain Lifecycle

Figure 1 depicts how the domain lifecycle supports an organizational learning approach to software development. As frequently encountered problem domains mature from novel problems to repeated problems to a fully mature domain, support is provided in increasing levels of automation. Three levels of knowledge are identified in the domain lifecycle:

- **A case-based repository** collects experiences from individual projects, tool experiences, and other artifacts generated to support the development infrastructure of the organization. Project experiences and design artifacts are collected through a process of issue management and design rationale that describes the problems that are addressed while creating an application. Tool experiences are how-to advice and descriptions of problems encountered while developers are using a tool to develop software. This provides solutions to organization-specific problems that are not found in manuals. Reusable artifacts can be in the form of procedures for approaching a problem (process models), software modules, specifications, requirement documents, algorithms, designs, test suites, and other items generated in the course of a project.
- **Domain abstractions** are domain-specific models of design problems, including domain models, design guidelines, value-added reusable artifacts, domain-specific handbooks,

process models, design checklists, analyzing similarities and differences between systems, and other forms of knowledge. Domain-specific knowledge is created by a domain analyst refining knowledge contained in the case-based repository into forms that are more generally applicable to problems frequently encountered in the organization.

- **Domain-specific** design environments automate or provide knowledge-based support for the development of systems within well-established domains. The environments are created by tool designers using accumulated knowledge from the domain models, specific cases, and reusable artifacts from the case-based repository.

These steps define increasing levels of automation and support mirroring domain maturity. Problems that have yet to be analyzed are supported by searching for similar problems in a case-based repository of project and tool experiences. The cases will contain information that is specific to the original project and may need work to apply to the current context, but at least there are some prior experiences to help guide design decisions. As activities are repeated, case-based technology is employed to identify recurring development issues and support the process of generalizing from individual cases to domain-specific abstractions such as design guidelines, domain models and other formal structures. Using handbooks, guidelines, and domain models provides a generalized level of support as the knowledge has been processed by domain analysts into a form that is applicable to general problems in the domain. As the domain matures, tool designers can use the synthesized knowledge and components to construct design environments that automate development tasks and provide intelligent support for mature domains repeatedly encountered in the organization.

The domain lifecycle acknowledges that knowledge for software development cannot always be universally applicable. Unlike homogeneous design domains such as hardware design or chemical engineering, software must be designed for a wide variety of disciplines, user populations, and business contexts. Knowledge will therefore tend to fall in domain-specific islands. Technical disciplines, such as databases, human-computer interaction, CASE tools, networking, and others cross the application domains, creating a complex network of interconnections with many possible solutions to similar problems. The ways in which these domains intersect are largely organization-specific. Hence, the domain lifecycle acknowledges that while general, top-down, knowledge has its place, particularly in organizing information, much of the knowledge that impacts software development comes from more of a bottom-up process in which context-specific information is generalized within specific domains of knowledge.

3. Evaluation of BORE Prototypes

Our work thus far has largely focused on developing tools to support the creation of case-based repositories. Early BORE prototypes were evaluated in two separate contexts. In the first, a pilot project at UPRR evaluated BORE by documenting some cases and providing comments of how BORE can be improved to better meet their specific needs. The second evaluation context was a Software Engineering course consisting of seniors and first-year graduate students. Students in this class developed small to medium scale software systems for clients external to the university system (one of the projects worked with UPRR). Project assignments for requirements, design, implementation, and formal testing all required the use of BORE. The scope of the projects were rather small, with less than a semester to develop the software and 4-6 people on the project. Nonetheless, the six projects generated over 150 cases.

In both instances we were disappointed in the detail and amount of information provided by BORE users. Cases were not described adequately, solutions were often left blank, and seldom contained adequate information for subsequent users to re-use the knowledge. There was clearly a need to encourage users to provide higher quality information. It could be argued that better results would have been achieved in more structured settings where more emphasis was given on project documentation. While this argument has merit, our observations also indicate that any activity deemed as ancillary to the immediate goal of producing a working software product was often given short shrift. Because BORE was not intimately tied to these goals, people regarded using the system as supplemental and not part of the critical path.

On the other hand, we have observed developers taking time to document activities in various Lotus Notes databases [Henninger et al. 1995], demonstrating that capturing key information was given some emphasis in the organization. During one session, we observed a project manager spending the better part of an hour translating handwritten notes to a Notes database, explaining that "this is a procedure that we know other projects will need, and I've been asked to document our experiences for future reference." One problem with their approach, though, was that the information was spread throughout over 2,000 Notes databases with few formal mechanisms to let people know what information existed and where it could be found. Our own explorations of the repository demonstrated that useful databases were found through personal contacts and "folklore." While personal networking is a useful mechanism for knowledge dissemination, it is a fragile and error-prone system [Terveen et al. 1995] that needs better support than search engines alone can provide.

Our paramount conclusion from the pilot studies was that procedures for using a system such as BORE are necessary to gain maximum benefit from an organizational learning approach to software development. Furthermore, these procedures must provide motivation for contributing and using the repository in a manner that satisfies the pragmatic concerns of software developers striving to accomplish project goals.

4. Capturing and Disseminating Best Practices in an Organizational Setting

Through our studies, it became clear that if the potential benefits of an organizational learning approach [Henninger 1996c; Henninger 1997] was to be realized, people needed guidelines on the level and quality of information they needed to capture through support tools, such as BORE. Use of an organizational learning system needs to be *formalized* in the sense that its use is mandated in a documented process to ensure that proper information is collected. In addition, this requirement must be carefully balanced with adding documentation tasks to already burdened development staffs. Instead of treating the repository as a supplemental obligation with little to no immediate benefit, mechanisms are needed to turn the repository into the central planning and documentation repository for a project. This would both avoid duplication of effort and provide one place for projects to coordinate and document their work.

In analyzing our less than stellar success, a theme began to emerge that the problem was not one of providing better search engines or knowledge acquisition methodologies, but one of designing work practices to fit the organizational learning approach. Based on our experiences, a set of requirements for such an approach include:

- use of the organizational learning repository must become an integral part of the development process,
- use of the repository cannot be discretionary, it must be obligatory,
- using the repository must provide easily perceived benefits for the development effort at hand -- i.e. the repository must become a repository of best practices,
- the repository must contain up-to-date information of relevance to development efforts.

We began to conceptualize our solution in terms of a dynamic technology for Standard Development Methodology (SDM) documents that serve as a focal point for an organizational learning process. While software process modeling techniques, such as software process programming [Ambriola et al. 1997; Curtis et al. 1992; Osterweil 1987], have received a great deal of attention in the research community, most software development organizations still practice the use of Software Development Methodology (SDM) documents. These are usually found in a monolithic manual, although hypertext versions are becoming popular. While we have begun to investigate using a software process programming approach [Henninger 1999], this paper will focus on turning current SDMs into a *living document* designed to evolve and improve through use, drawing on the collective knowledge of the organization.

4.1 Towards A Defined, Extensible, SDM

Most traditional SDMs created by software organizations eventually fall into disuse for two reasons: 1) To accommodate diverse needs, the SDM is stated at a very high level and lacks the detail that can truly guide a project. Stating the need for "detailed design" is insufficient. There also needs to be specific guidelines, examples, and detailed procedures that help people create the design. I.e., to be successful, the SDM must provide valuable resources for development efforts. 2) The SDM quickly falls behind the rapid pace of technology and business needs. Even if SDM curators are appointed, it is difficult for a few people to keep pace with changes in large development organizations.

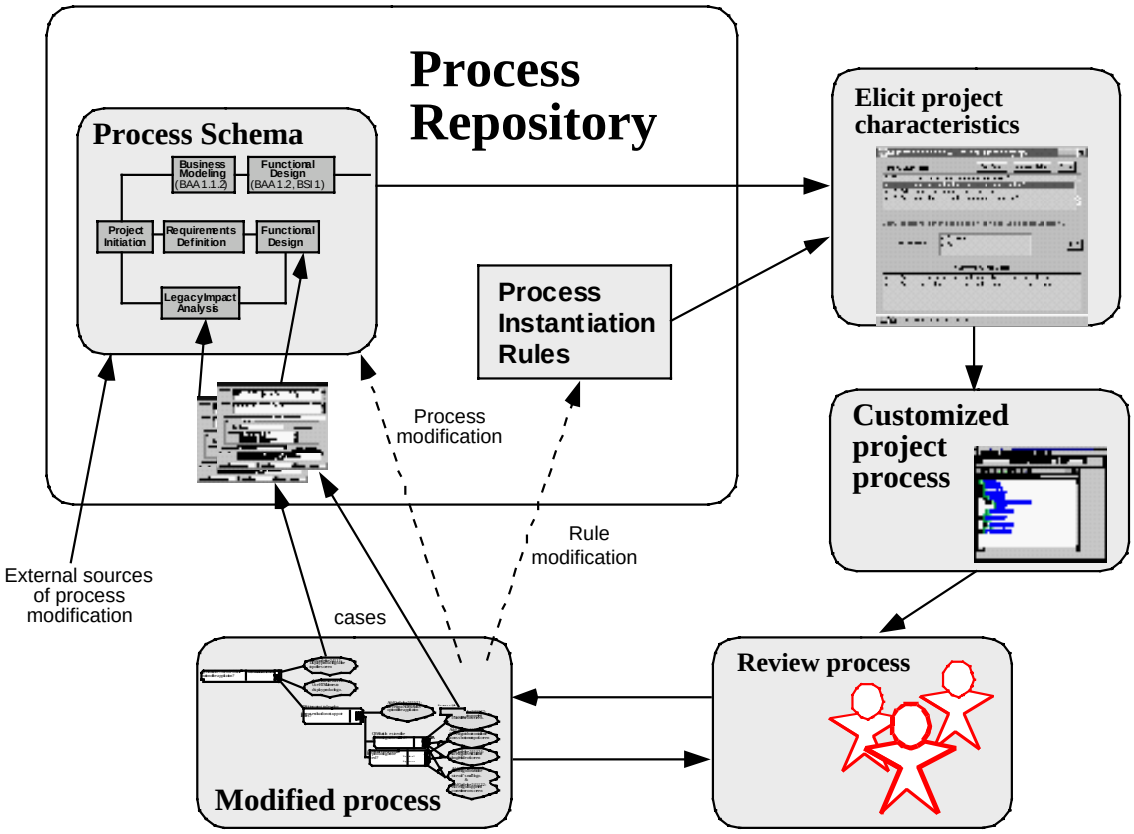

Figure 2: Tailoring and Refining the SDM.

To reduce these problems, our process begins with a defined on-line SDM. In addition, multiple paths through the SDM are designed to accommodate the different kinds of projects typically encountered in the organization [Osterweil 1987]. Figure 1 shows a diagram of the overall process of using and evolving the SDM. In this figure, the SDM (shown in the box marked "Process Schema" in the BORE repository) has three different paths. More complex structures are possible, and it is not expected that everyone should understand the SDM in its entirety. Instead, a decision support system is used to reduce the complexity by eliciting project characteristics, such as end-user, data and security requirements, whether an iterative or waterfall methodology should be used, etc.

The decision support system steps through questions designed to elicit project characteristics. These characteristics are matched against requirements in the SDM to propose a customized process for the project. Developers can then modify this process to meet the project's needs. The end result is subjected to an approval review in which the answers provided to the decision support system and any modifications made to the resulting process are carefully examined. The outcome of the review can be one of three actions: 1) the project is required to re-visit some of the questions posed by the decision support system, 2) modifications are rejected and/or new modifications are proposed, and 3) the process is approved.

Upon approval of the process, a BORE project is created and new cases are created for each process element. These cases are associated with SDM process elements and can be used as an information resource for that element. In addition, any exceptions to the SDM, such as adding new process elements, must become part of the SDM. This means modifying the SDM and the decision support system so that projects with similar characteristics are guided toward the SDM

modifications made to accommodate the project. The overall process is designed to retain a degree of process rigor while being able to support a dynamic development context.

Over time, the SDM will evolve to meet the needs of issues that recur in the organization [Henninger 1996a], positioning the organization to take advantage of previous experiences and improve. But, particularly in the face of diverse applications and interface design requirements, this process can create a large and potentially cumbersome repository.[1] Its complexity will exceed any individual's ability to understand all the issues involved in the guideline repository. Appropriateness is ensured through the conformance and modification, but without proper technology, the organization has no hope of ensuring that the guidelines lead to quality applications.

5. Tools for Supporting SDM Tailoring and Modification

The process outlined in the previous section is designed to turn an organization's SDM into a living document that evolves and improves with use. The following sections use a BORE prototype to demonstrate how this is accomplished. There are two major components of this process: 1) The standard SDM is *tailored* to specific project requirements, and 2) if necessary, the SDM is *modified* to meet the emerging needs of development projects. The BORE prototype is a Web-enabled application using a three tiered architecture consisting of HTML and Javascript for rendering the interface, Java for application logic and JDBC access, and relational database back-end. It can be accessed through a Web browser (Navigator 3.1 or greater or Internet Explorer 4.0 or greater) at http://cse-ferg40.unl.edu/bore.html.

[1] Note that there is a metaphor with American Case Law which, while sometimes maligned for being cumbersome, has stood the test of time under the constantly evolving forces of human culture and diverse circumstances. While some of these problems are inherent to human activities, which become embodied in the interfaces and inner workings of our systems, we hope to create technology that can lead to effective and timely decision making.

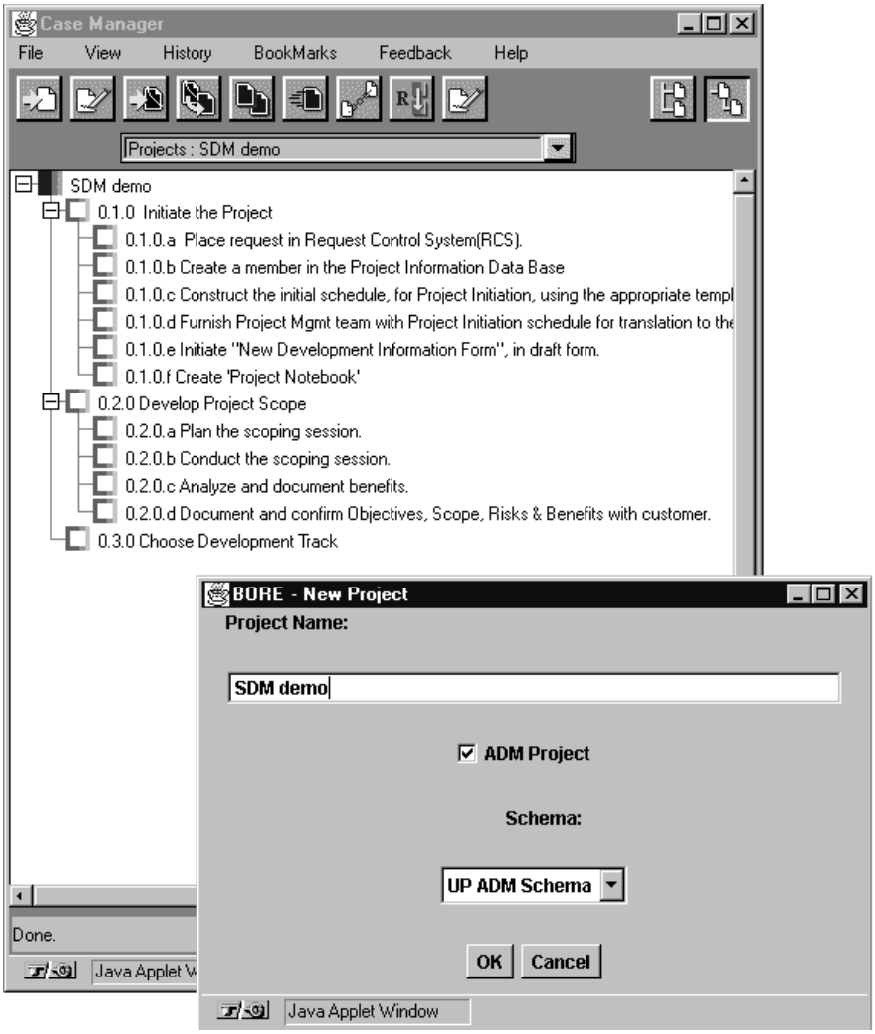

Figure 3: The Case Manager view of the Project Initiation Process.

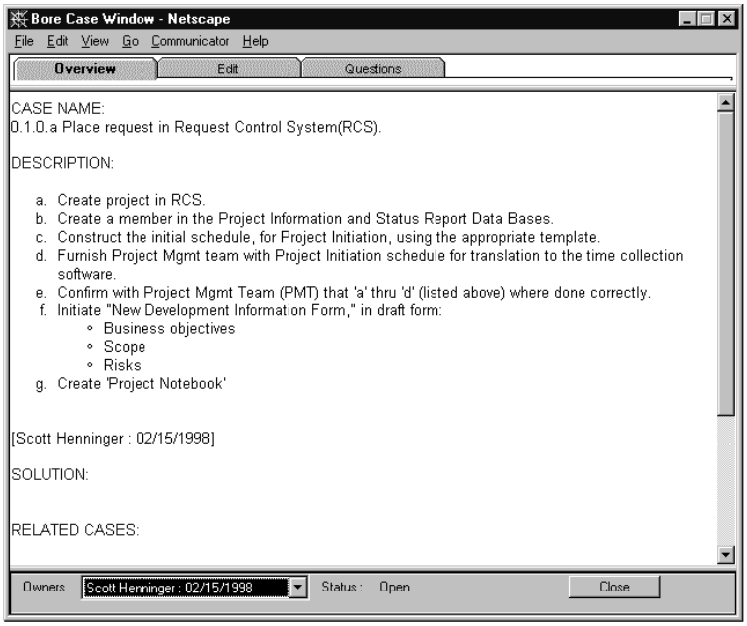

Figure 4: A Case Window.

5.1 The SDM Schema

One of the primary interfaces of the BORE prototype is the Case Manager, as shown in Figure 3. This interface displays a hierarchical arrangement of cases that define the tasks in a project's development process. For example, in Figure 3, A project named "SDM demo" has been chosen from the list of resources in the drop-down menu that displays "Projects:". A set of cases are displayed in a hierarchical arrangement that represent a set of tasks in a simple software development project. Each of these items are cases that contain project-specific information as shown in window in Figure 4, which was obtained by double-clicking on the case named "0.1.0.a Place a Request in Request Control System (RCS)."

This is essentially the case-based organizational memory paradigm of BORE [Henninger 1997]. Each case represents project-specific information that is used to help coordinate and disseminate project experiences. The significant difference is the addition of project "schemas" that begin to instantiate the tailoring and modification process shown in Figure 2. The SDM "schema" is a set of cases and relationships between cases that define the software development process for all kinds of projects, which can range from throwaway prototypes to in-house applications to shrink-wrap or custom-built software systems. Projects choose an appropriate subset of the schema cases to define its development process. We have chosen to call it a "schema" in the sense of data model schema, where the SDM schema is the logical data model that is instantiated by the physical model and data contained in the database.

To tailor the SDM schema to project-specific needs requires a process by which developers and managers decide which processes are necessary and/or relevant for the requirements of the given project. While there are many valid approaches to this decision process, we have been experimenting with a semi-formal approach that utilizes decision trees such as the one shown in Figure 5 that shows a partial decision tree for choosing a development strategy. These decision trees can be used at an arbitrary level of project detail, from high-level development processes such as whether business modeling or requirements definition is needed (see the 'Process Schema' box in Figure 2) to specific corporate standards, such as which login screen should be used.

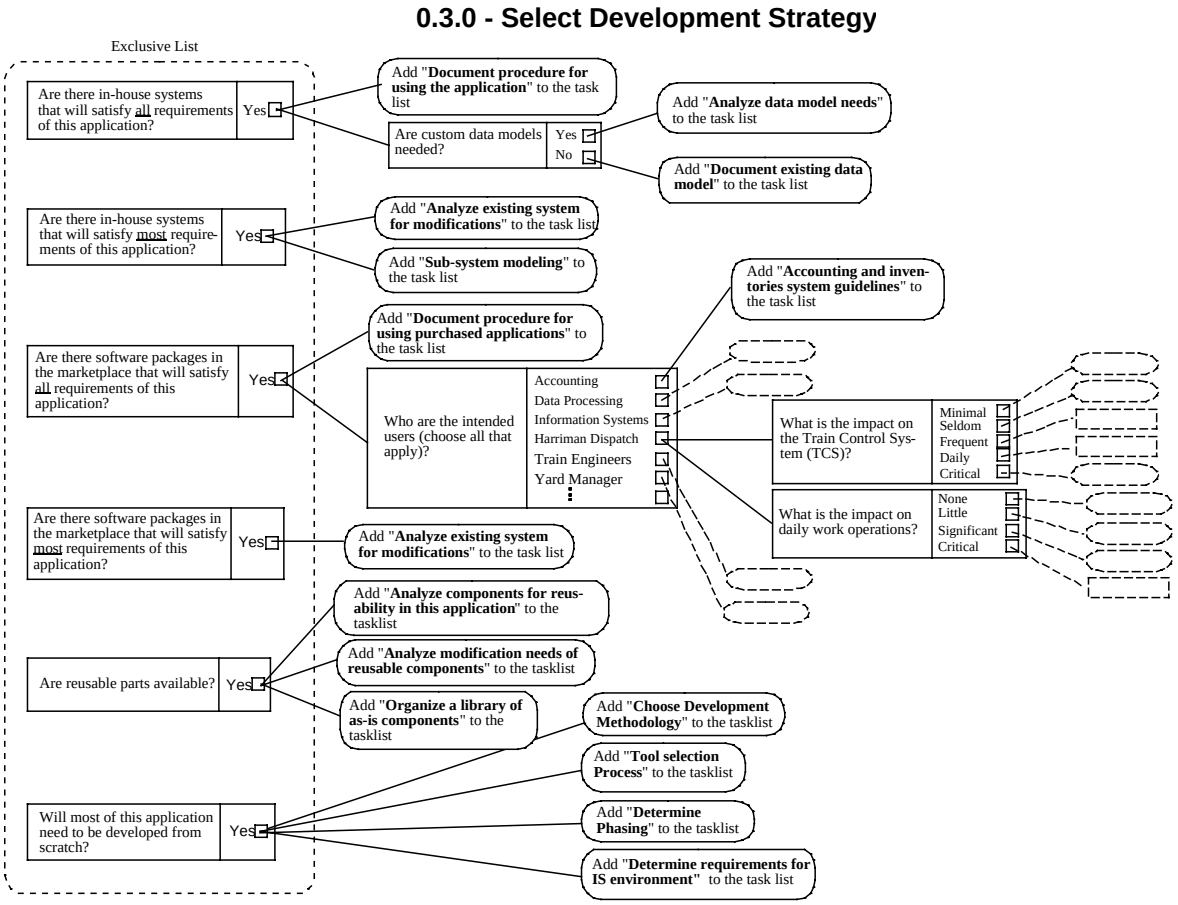

Figure 5: A Partial Decision Tree for Development Strategies.

When creating a new project, the window in the lower right of Figure 3 is used to choose the schema from which the project's cases will be derived. In our example, the three major tasks of Figure 3 are assigned to the project in a partial development process that details project initiation steps. These cases can serve as a checklist of project tasks that document progress and coordinate efforts. BORE supports a color coding of the glyph to the left of the case name in the Case Manager to provide a visual means of identifying the status of cases -- whether the case is open, actively being worked on, resolved, and etc. The purpose of the case is to document how a given task was addressed by the project. The Overview, Resources, Attachments, and Related Cases tabs are used to document project-specific issues that arose when addressing the task (Figure 4).

This is the essence of a case-based approach -- general principles (the SDM schema) are used as a guide for decision makers, and cases describe situation-specific problems that may arise in certain contexts [Kolodner 1993]. In our version of this case-based paradigm, the SDM schema defines the canonical development process under a wide range of situations encountered in the organization. Cases describe some of the problems or necessary steps that must be taken to ensure that the process is properly followed. For example, a process step may dictate that the corporate standard login screen is used for an application. A project trying to follow this step may find that using the password database required a couple of work-arounds to fit their specific architecture. The project's case would reflect these problems and document how the problems were solved in the "Solution" field of BORE cases.

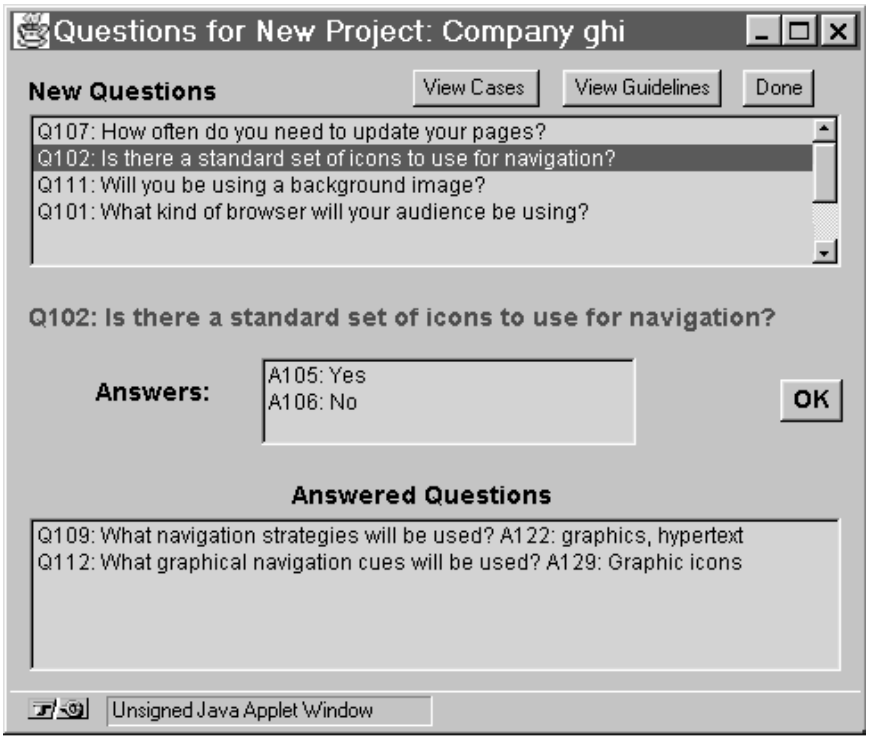

Figure 6: A BORE Case Showing Questions for Tailoring the SDM to an Individual Project.

5.2 Tailoring the SDM to Specific Projects

In BORE, the process of tailoring the SDM schema to the specific characteristics of a project involves answering questions from a rule-based system. In the spirit of traditional Expert Systems, one may visualize this as consisting of a long question-answer session at the beginning of a project that would construct a plan from beginning to end. We found this kind of approach to be unappealing for a number of reasons. First, we want BORE to support a number of different levels of decision making, from choosing whether the project is enterprise-wide or local to what kind of database should be used all the way to choosing reusable code, such as logon screens and associated logic or back-up and recovery schemes. Secondly, if a system spans these levels of support, and indeed it must to be called an organizational memory, it would be impossible to anticipate the answers to all quests up-front.

Therefore, BORE uses a more iterative strategy. When a project begins, an initial set of cases is instantiated, an example of which is shown in Figure 3. The cases copied from the SDM schema can contain questions that are used to further refine and expand the project's process. These questions appear in the "Question" tab (see Figure 6) and are intended to elicit information about a project's characteristics so that the SDM can be appropriately tailored to the project. Any case from the SDM schema can have questions associated with it, allowing an organization to define and refine their process to whatever level of detail they wish at any desired point in the development process.

This multi-step process allows projects to incrementally expand each part of the project's process as they gain the knowledge necessary to address process issues [Osterweil 1987]. In this way, the project tailors the overall SDM to its needs when it is able to address the issues

involved. Each of the cases shown in the hierarchy may have questions that will further refine the tasks and provide resources for the project. Since cases control this process, any level of detail can be supported.

5.3 The Rule-Based System

Questions are represented in a rule-based production system. Each rule has a set of preconditions consisting of one or more question/answer pairs. When all preconditions evaluate to true, the rule "fires," causing a set of actions to be executed. Actions in BORE can take one of three forms: 1) A question can be removed from the question stack (displayed in the Question window of a case), 2) a question can be added to the question stack, and 3) a case or set thereof that is be added to the project in the proper places in the hierarchy as defined by the SDM schema (note that if a case is added to a project at some arbitrary level of the hierarchy, the system will iteratively add parents of the case until a parent is found that already exists in the project). We are in the process of adding a fourth consequent type that will allow function calls to be evaluated. This will provide flexibility for extending the system to include other actions necesary for process enactment .

The production system provides the flexibility to implement complex decision trees. While AND relationships are inherent in the precondition/action structure of the rules, OR relationships are easily represented by having two rules with different preconditions and the same action. Combinations can be created by mixing these strategies as needed. This flexibility comes at the price of complexity, though, and future efforts will look at improving interfaces and checks on the system to ensure the integrity of the rule base. To facilitate modification of rules (see next section), questions and answers are represented in a MS Access database back-end.

5.4 Modifying the SDM

As shown in Figure 2, the SDM is refined in two ways. First, a new case is instantiated for each process element assigned to the project. This case will document how the project met the requirements of the given process element. This will also provide a resource for subsequent development efforts because one can look at all cases for a given process element. In addition to ensuring a measure of conformance to the SDM, this provides an opportunity to disseminate best practices. Developers assigned a given process element can easily check the repository to see cases documenting how other projects accomplished the process goals and potentially reuse their solutions.

The second modification to the SDM changes to SDM tasks and the production system that assign SDM tasks to individual projects. This involves creating new decision points and process elements in the SDM and integrating new questions and answers to existing questions into BORE. Rules are edited through an interface (not shown) which displays preconditions and actions for a given rule. Tasks are added to the SDM Schema through the Case Manager, which supports a view for editing the SDM schema in the same way that BORE cases can be edited.

There are some security considerations associated with changing the schema and rules. Whereas editing cases can and should be the responsibility of individual projects, editing the SDM involves some policy decisions on whether individual projects should be allowed to make changes or whether some curator or administrative body controls the SDM. Note that in Figure 2 changes are made to the SDM repository only after undergoing a review process. Following this kind of process will ensure that decisions are appropriate for future development efforts.

There is also the issue of creating rules so they are not inconsistent, contradictory, or incomplete. While we will strive to make tools to reduce these errors [Terveen, Wroblewski 1990], it is recommended that personnel with expertise in rule-based systems create and test new rules before they are placed in the repository.

Modifying the SDM consists of either or both adding tasks to the SDM and editing the rules for adding tasks to a project. For example, suppose a project is amongst the first to create Java applets, and designs a method to use JDBC to access Oracle tables. To do this means deviating from the SDM because certain issues of creating a Web-based database server haven't been addressed before in this organization. The project goes through the review process, is approved, and engages in the extra documentation mandated to pave a path for this kind of project. The project creates a number to tasks and subtasks that must now be added to the SDM. A process expert is assigned to add these tasks and insert new questions into the repository that ensures that future Web-based project can take advantage of, and improve on, the procedures created by this ground breaking project.

6. Open Issues and Future Directions

The general goal of this approach is to try to split the gap between overly-restrictive procedures and not following a defined process. Currently, the industry standard is to define a SDM and then ignore it in its entirety or follow it enough to justify it to certification authorities. We wish to turn SDMs into something that truly supports the development process as it is actually practiced, while adding necessary degrees of formal procedures to ensure high-quality products. This involves not only defining a process, but also using feedback from projects using the SDM to refine and improve its procedures.

One of the greatest dangers of this approach is that the resulting SDM will become unwieldy and incomprehensible. It remains an empirical question as to whether BORE's production system approach is sufficient to help developers make decisions about which process elements are necessary and how the SDM should be followed. There is a need for higher-level views and models to help people understand the SDM's structure and support a principled evolution of the processes it contains. Nonetheless, the complexity of the resulting SDM reflects the true diversity that lie in the intersection of business needs, development tools, and developer skill sets that defines modern software development. Without this, the process will settle at the ineffective least common denominator that current SDMs suffer from.

More tools are needed that help SDM administrators insert new process and the associated question/answer pairs into the SDM. This is essentially a knowledge acquisition problem, and we are looking into using agents and critics to flag knowledge base inconsistencies and gaps [Terveen, Wroblewski 1990] and ensure that complete and consistent processes are assigned to projects.

Some of the research questions we will continue to explore with BORE include designing work practices centered around case-based organizational learning processes. Design tools must reward users for use in the current project as well as leaving information for subsequent development and maintenance efforts. We believe these are compatible goals, as long-term projects need to track their progress and decision making process. Some major research questions include: designing methodologies based on standards set by previous projects yet flexible enough to accommodate changes in business needs and technology; creating a development process that begins by using the resources supplied by the repository and updates

the repository as the project progresses; finding the "right" level of documentation that doesn't overwhelm developers yet leaves a trace for subsequent efforts; and organizational changes needed to make such techniques work in the organization (i.e. the technology transfer and tool adoption problems).

7. Conclusions

Centering the information around a single repository through an ongoing process of capturing project experiences can prevent the duplication of efforts, avoid repeating common mistakes, and help streamline the development process. Similar projects have shown that such a system will be used by development personnel, provided it contains relevant, useful and up-to-date information [Terveen et al. 1995]. This mandates a strong tie between technology and process in which using the technology must become part of routine work activities. Such an approach will succeed to the extent that people are rewarded in the short term for their efforts.

To achieve these goals, we have coupled process and technology to turn SDM documents into dynamic "living" resources that can be extended and improved as new project needs and application requirements emerge. As the repository accumulates through principled evolution of the SDM, it improves to able to handle a wider range of circumstances [Kolodner 1993], while evolving toward answers to problems that fit the organization's technical and business context. The real question is not whether the repository is "correct" in some objective sense, but rather whether less mistakes are repeated and better solutions adopted when using the repository.

References

[Ambriola et al. 1997] Ambriola, V., Conradi, R., Fuggetta, A. "Assessing Process-Centered Software Engineering Environments," *ACM Transactions of Software Engineering and Methodology*, 6(3), pp. 283-328.

[Arango 1989] Arango, G. "Domain Analysis: From Art Form to Engineering Discipline," *Proceedings Fifth International Workshop on Software Specification and Design*, Pittsburgh, PA, ACM, New York, pp. 152-159.

[Basili et al. 1992] Basili, V. R., Caldiera, G., Cantone, G. "A Reference Architecture for the Component Factory," *ACM Transactions on Software Engineering and Methodology*, 1(1), pp. 53-80.

[Curtis et al. 1992] Curtis, B., Kellner, M. I., Over, J. "Process Modeling," *Communications of the ACM*, 35(9), pp. 75-90.

[Henninger 1996a] Henninger, S. "Accelerating the Successful Reuse of Problem Solving Knowledge Through the Domain Lifecycle," *Fourth International Conference on Software Reuse*, Orlando, FL, IEEE Computer Society Press, Los Alamitos, CA, pp. 124-133.

[Henninger 1996b] Henninger, S. "Building an Organization-Specific Infrastructure to Support CASE Tools," *Journal of Automated Software Engineering*, 3(3/4), pp. 239-259.

[Henninger 1996c] Henninger, S. "Supporting Software Development with Organizational Memory Tools," *International Journal of Applied Software Technology*, 2(1), pp. 61-84.

[Henninger 1997] Henninger, S. "Case-Based Knowledge Management Tools for Software Development," *Journal of Automated Software Engineering*, 4(1), pp. 319-340.

[Henninger 1999] Henninger, S. "Decision Support for Selecting Appropriate Software Process Models," *Software Engineering and Knowledge Engineering (SEKE '99)*, (submitted).

[Henninger et al. 1995] Henninger, S., Lappala, K., Raghavendran, A. “An Organizational Learning Approach to Domain Analysis,” *Seventeenth International Conference on Software Engineering*, Seattle, WA, ACM Press, New York, pp. 95-104.

[Kolodner 1993] Kolodner, J. L. *Case-Based Reasoning*, Morgan-Kaufman, San Mateo, CA.

[Osterweil 1987] Osterweil, L. “Software Processes are Software Too,” *Ninth International Conference on Software Engineering*, Monterey, CA, ACM, IEEE, Los Alamitos, CA, pp. 2-13.

[Senge 1990] Senge, P. *The Fifth Discipline: The Art and Practice of the Learning Organization*, Currency Doubleday, New York.

[Terveen, Wroblewski 1990] Terveen, L., Wroblewski, D. “A Collaborative Interface for Browsing and Editing Large Knowledge Bases,” *National Conference of the American Association for AI*, Boston, MA, AAAI, pp. 491-496.

[Terveen et al. 1995] Terveen, L. G., Selfridge, P. G., Long, M. D. “‘Living Design Memory’ - Framework, Implementation, Lessons Learned,” *Human-Computer Interaction*, 10(1), pp. 1-37.

[Walsh, Ungson 1991] Walsh, J. P., Ungson, G. R. “Organizational Memory,” *Academy of Management Review*, 16(1), pp. 57-91.

A Knowledge Management Lifecycle for Experience Packages on Software Engineering Technologies

Andreas Birk
Fraunhofer Institute for
Experimental Software Engineering
Sauerwiesen 6
D-67661 Kaiserslautern
Germany
email: Andreas.Birk@iese.fhg.de

Felix Kröschel[1]
VSS GmbH
Hahnstr. 70
D-60528 Frankfurt a. M.
Germany
email: fkroesch@vss.com

Abstract

Software engineering can benefit very much from customised knowledge management solutions. These should rely on reusable experience that is modelled explicitly and stored in central repositories. Few approaches exist yet that provide such knowledge management support to software engineering. Those that support it cover usually only part of the knowledge management lifecycle of the reusable artefacts.

This paper suggests a knowledge management lifecycle for experience about software engineering technologies and their application contexts. It primarily aims at supporting the planning of software projects and improvement programmes. The lifecycle model is substantiated by a tool implementation and evidence from an industrial trial application.

1. Introduction

Software engineering plays a key role in today's public life. Nearly every product of the manufacturing or service industries depends to a wide extent on software. The source of software engineering's impressive success is a tremendously fast progress in developing and deploying new technology. However, despite these achievements, stories about computing problems and failures are spread over the news quite frequently. These troubles seem not to fit into the picture of a successful new branch of engineering. A number of recent investigations have shown that the main cause of software engineering's failures is the same as for its successes: New technology–and, in the case of project failures, the inability to manage it successfully (cf. [KPM95] and [Gla98]).

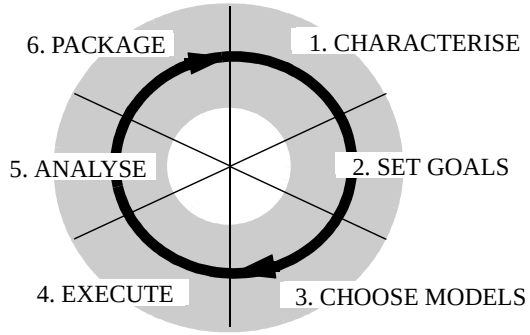

Figure 1: The Quality Improvement Paradigm.

The inherent risk of technology failure in software projects calls for a better management of our knowledge about technologies and their application contexts. Since the beginning of software engineering much effort has been put into the development of new technologies. Less attention has been paid to their application and empirical evaluation. We argue that this lack of information about when a certain technology can be applied most appropriately and when it should not be applied is the main reason for many technologies-caused project failures.

A knowledge management approach to the application of software engineering technologies has two basic requirements: (1) Precise and operational definition of software engineering technologies, and (2) the systematic investigation and documentation of the application contexts of technologies. While the first issue has already been subject to many research efforts, mainly in the area of process modelling (cf. [RV95], [CKO92]), the application contexts of technologies have hardly been addressed, yet. For this reason, we have been focusing our research around the following questions: (1) How can technology application contexts can be modelled? (2) how can concrete context models be gained, used for supporting project planning, and evolved based on experience from technology application? This paper presents results from this work. It introduces so-called *technology experience packages (TEPs)* for modelling technologies and their application context. Further, a lifecycle model for developing, using, and maintaining TEPs is presented. For each lifecycle task possible support is illustrated using a prototypical tool implementation.

Our approach builds on the Quality Improvement Paradigm (QIP) / Experience Factory (EF) approach [BCR94]. The QIP is a six-step cycle for continuous improvement in software engineering (see Figure 1). It serves also very well as a knowledge management

1. This work was performed while Felix Kröschel was with Fraunhofer IESE.

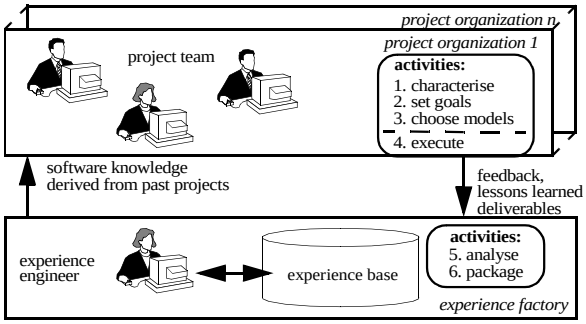

Figure 2: The Experience Factory.

paradigm for the software domain. The EF is the associated infrastructure for organisational learning (see Figure 2). It distinguishes the project organisation, whose main concern is software development, from the Experience Factory, whose main concern is to learn about software development and to support the project organisation with useful experience. Core component of the EF is the *Experience Base (EB)*, a repository of relevant software engineering experience. The EB contains a collection of *experience packages* that consist of a reusable artefact (e.g., a process model, an effort prediction model, or a code module) and information about when–i.e., in which situations–the reusable artefact can be applied.

Throughout this paper, emphasis is placed on software engineering technologies. Therefore, we refer to a specialised variant of experience bases: Technology experience bases (TEBs), that contain technology experience packages (TEPs). As software engineering technology we refer to every technique, method, or tool used for software engineering [BCR94]. Special attention is paid to process technologies, such as inspection methods and methods for performing measurement programmes.

Section 2 addresses the interface between knowledge management and software engineering. It provides a survey of selected knowledge management solutions from software engineering and other fields. The concept of *technology experience bases* is introduced in Section 3. Sections 4 to 7 present a lifecycle for managing knowledge about software engineering technologies and their application domains. A project on the development of a TEB and its experiences are reported in Section 8. Section 9 presents the main conclusions from our work.

2. Knowledge Management and Software Engineering

Knowledge management can provide beneficial support to software engineering (cf. [BSA99], [Eri92]). This section explores the interface between these two areas. First, requirements on knowledge management of software engineering technologies are identified (Section 2.1). These are then used in Section 2.2 to survey and characterise relevant knowledge management tools. Finally, conclusions are derived from this survey that pin-point further needs for methodology and tool support.

Knowledge management is a term that is still widely discussed and that has no comprehensive and generally accepted definition yet. A concise definition that is widely appropriate for software engineering follows a definition proposed by O'Leary [O'L98]:

> *Knowledge management is the formal management of knowledge for facilitating creation, access, reuse of knowledge, and learning from its application, typically using advanced technology.*

This definition reflects the need of a learning organization for identifying knowledge, for storing it appropriately, for making it accessible, and for easing its reuse. Experience gained from reusing the stored knowledge items should be deployed to further evolve and extend the stored body of knowledge.

2.1. Requirements on Knowledge Management of Software Engineering Technologies

The need for knowledge management support in software engineering–in particular for the management of knowledge about software engineering technologies and their application contexts–can be specified in the form of requirements. In the following, a set of such requirements is provided. Some of these have been derived from the literature ([Hen97b], [BR91]). Others stem from an analysis of knowledge management needs concerning software engineering technologies (cf. [BSA99]).

Support the entire lifecycle of reusable artefacts

A knowledge management system should support the entire lifecycle of the managed artefacts (or experiences or knowledge items), involving their creation, their retrieval and reuse, as well as the feedback of experience from using them.

Similarity-based retrieval

Retrieval of reusable artefacts should be based on appropriate similarity measures that do not require exact matches between a retrieval query (i.e., the specification of reuse requirements) and the retrieved reusable artefacts. This is necessary for knowledge reuse in software engineering, because the characteristics of software projects can differ quite much. So it is very likely that relevant reusable artefacts are characterised in a form that is not exactly matching a reuse situation.

Retrieval based on incomplete information

Retrieval of reusable artefacts should not require that for all attributes using which the artefacts are indexed information is provided in the retrieval query. This is necessary, because knowledge retrieval is usually performed at the beginning of a software project when some characteristics of the forthcoming project

Table 1: Comparison of selected knowledge management tools

	Support the entire lifecycle of reusable artefacts	Similarity-based retrieval	Retrieval based on incomplete information	Possibility to evolve the experience base	Support different types of knowledge	Precise representation of reusable artefacts	Intuitive characterisation of reusable artefacts	Links between reusable artefacts
Bore	●	●	●	◗	◗	●	◗	●
Msmt. Planning	◗	●	●	◗	◗	●	◗	●
KONTEXT	●	●	●	◗	◗	●	●	●
WebME	◗		◗	◗	◗	●	◗	●
IDS		◗	◗	◗	●	●	◗	
Ontobroker		◗	●	◗	●	●	◗	◗
Kactus			●	◗	◗	●	◗	◗
Dedal		◗	●	◗	◗	●	◗	●
QuestMap	❍			◗	●	●	◗	●
eule2	❍		◗	◗	●			●
LotusNotes/Domino	❍	❍	❍	◗	●	❍	❍	❍
LiveLink	❍	◗	◗	◗	●			●
AnswerGarden	❍			◗	◗			●

● = comprehensive support ◗ = support exists in part ❍ = support is not explicit, but the tool offers means for developing it

are not yet known. It should be possible to omit these characteristics when defining a retrieval query.

Possibility to evolve the experience base

The TEB should provide mechanisms for adding new knowledge artefacts, as well as for adding or removing attributes of existing objects. It should also be possible to generalise or specialise existing objects. This is necessary in order to support continuous learning and improvement in software engineering.

Support different types of knowledge

The experience base should support a variety of different knowledge types. They can have different representation structures, different access and usage processes, as well as different levels of maturity and reliability.

Precise representation of reusable artefacts

Reusable artefacts should be represented and stored–as far as possible–using a precise and sufficiently formal knowledge representation. This is needed for clarifying the semantics of the artefacts as well as for allowing for automated support of knowledge usage and maintenance.

Intuitive characterisation of reusable artefacts

Reusable artefacts–and in particular their application context–should be defined using terms that are intuitive to the knowledge users. This is important to assure that the experience base is accepted from its users and to allow for efficient usage processes.

Links between reusable artefacts

Explicit links should be established between related artefacts in the experience base. This is needed for being able to exploit relationships during maintenance and retrieval. For instance, technologies should be linked with the processes in which they can be applied.

There are also additional requirements that are important for the technical realisation of an experience base. Examples are access via the internet or intranet, view support, and information security. These are not addressed further, because the focus here is on the methodological concepts of knowledge management in software engineering.

2.2. A Survey of Knowledge Management Tools

Many knowledge management approaches rely mainly on the organisation of knowledge exchange between human agents (cf. [Wii95] [O'L98] [Sen90]). However, our objectives for knowledge management in software engineering are to make knowledge assets explicit (e.g., in the form of technology experience packages), to store them in repositories that are widely accessible (i.e., technology experience bases), and to provide computer support for the management of this knowledge.

There is quite a number of tools available–either as research prototypes or as commercial tools–that address knowledge management. Some of these are specialised to knowledge management in software engineering. Abecker et al. [ADK98] distinguish between process-oriented and product-oriented tools. Process-oriented tools aim at supporting or facilitating knowledge exchange between human agents during their work processes. Examples are groupware systems and intranet solutions. These tools are usually not particularly effective for supporting the management of knowledge about software engineering technologies and their application contexts. Product-oriented tools focus on the knowledge assets

to be reused. These tools offer appropriate concepts for the knowledge management of software engineering technologies and application contexts.

Table 1 shows a collection of selected knowledge management tools and describes them with regard to the requirements stated in the previous section. The following tools have been included into the survey: *Bore* ([Hen97a] [Hen97b]) manages knowledge that supports the execution of corporate software development processes. It contains guidelines, lessons learnt, and frequently asked questions, following a case-based approach to knowledge representation and management. Gresse von Wangenheim et al. [GvWB98] are developing a tool for capturing experiences about GQM measurement planning (in the following denoted as *Measurement Planning*). *KONTEXT* [Krö98] is a tool for managing the knowledge about software engineering technologies and their application contexts. It is described in more detail below, in Sections 4 to 7. *WebME*, a web-based tool for providing measurement data to project management has been developed at NASA's the Software Engineering Laboratory [TZ98]. These four tools are all knowledge management solutions that are designed for specific software engineering tasks. The following tools are designed primarily for other domains, or they provide generic knowledge management infrastructures.

The expert system *IDS* [IBM98] has been developed for the aircraft industry by making use of data-mining techniques for diagnosing purposes. *Ontobroker* [FDES98] is an intelligent search tool (e.g. for searching the internet) based on ontologies. *KACTUS* [SWI97] provides an interactive environment for browsing, editing and managing (libraries of) ontologies. A representation of a device model is used by the tool *Dedal* [CDR95] for indexing and retrieving multimedia information about a designed device. The hypermedia groupware system *QuestMap* (cf. [Shu97]) integrates different media and links the captured knowledge into a structure supporting the discussion process. *Eule2* [Rei97] is a knowledge-based system for supporting office work in the life insurance domain. It integrates knowledge bases of different knowledge-based systems. The collaboration process in an organisation is supported by the widely used groupware tool *Lotus Notes/Domino* [Lot]. For instance, it provides services for electronic mails, storing various types of documents and information, and it allows for annotating documents. The tool *LiveLink* [Ope] incorporates elements of collaborative work support and web agents on a document-centred knowledge base. By using *AnswerGarden* [MT90] two different types of knowledge are combined: recorded knowledge can be retrieved and individuals with knowledge of some kind are made known to the rest of the organization. Information access is organised through posing diagnosis questions.

The surveyed tools and their underlying methodologies represent quite different philosophies of knowledge management. It was a purpose of the survey to provide a broad overview and to place the four SE-specific tools into the context of other kinds of knowledge management tools. Please note that the classification schemes (i.e., the symbols in the table cells of Table 1) of different evaluation criteria (i.e., the table columns) can have slightly different semantics. The survey is not meant to rank the tools. Its objective is to illustrate that there exists a variety of technical solutions to common knowledge management requirements.

2.3. Observations from the Tools Survey

The knowledge management tools characterised in Table 1 can be evaluated using the requirements listed in Section 2.1. This evaluation provides an overview of the state of the art in knowledge management for software engineering applications. This section identifies the observations from the survey and briefly outlines needs for further tool support.

Support the entire lifecycle of reusable artefacts

The four tools that are specific to software engineering provide specialised support for more than one lifecycle phase. *Bore* and *KONTEXT* have a usage model that comprises all lifecycle phases from insertion of new knowledge via knowledge use to knowledge evolution. Other tools do either address only the core phase of knowledge use (then nothing is indicated in the table), or they support some kind of knowledge evolution without modifying specific artefacts. The latter is especially true for process-oriented knowledge management tools such as workgroup support systems.

Similarity-based retrieval

Similarity-based retrieval involving some kind of similarity function is realised in *Bore*, *Measurement Planning*, and *KONTEXT*. Other tools realise similarity-based retrieval using some other concepts such as heuristics. *Lotus Notes/Domino* offers the basic infrastructur for implementing similarity-based retrieval mechanisms.

Retrieval based on incomplete information

Again, *Bore*, *Measurement Planning*, and *KONTEXT* use explicit characterisation schemes for specifying queries and allow queries in which some characteristics are omitted (i.e., retrieval with incomplete information). This is also true for *Kactus* and *Dedal*. Other tools don't use characterisation schemes for knowledge representation and retrieval, but they allow for some kind of open retrieval.

Possibility to evolve the experience base

Dynamic extension and modification of the experience base is a standard feature that is provided by all surveyed tools in that new knowledge items can be added, removed, or modified. Specific support for

automated generalisation or specialisation is not provided at all.

Support different types of knowledge

Every tool represents multiple types or dimensions of knowledge. But only some tools do also allow for extending the set of pre-defined knowledge types or are fully open with regard to the managed knowledge types.

Precise representation of reusable artefacts

Most tools apply some formal or at least well-defined structured knowledge or data representation. But some, mainly the hypertext-based ones, do not have a particularly well-organised, finer-grained representation scheme.

Intuitive characterisation of reusable artefacts

KONTEXT is different from the other tools in that it offers a specific representation concept for characterising the reusable artefacts in an intuitive manner. This is done using a redundant characterisation structure that has one view which models the intuitive terms used by decision makers, while the other view provides a precise and unambiguous definition of context characteristics. Most other tools allow for using intuitive identifiers of knowledge items, but only as far as object identity can be assured.

Links between reusable artefacts

Most tools have some kind of links between reusable artefacts thus that relations between associated kinds of knowledge can be explored by the user. These clusters of linked artefacts thus establish some new kind of "higher-order" object and allow the user to explore it interactively. Ontology-based systems (i.e., *Ontobroker* and *Kactus*) do of course also implement links between objects. But these objects are mostly of similar kind, and the linked entities do not establish a really complex new knowledge structure (i.e., other than the ontology itself).

Another knowledge management approach that focuses particularly on software engineering is the Quality Pattern approach [LSH98]. It is a document-based approach that is not yet supported by specialised tools. Therefore most of the above evaluation criteria can not be applied to it appropriately. However, it demonstrates the importance of a semi-structured representation for different knowledge types that is combined with a system of typed links between the reusable artefacts.

As overall conclusion from the survey it can be noted that there exists a wide variety of knowledge management solutions. Some have been designed specifically for use in software engineering, and these offer quite a wide spectrum of features. For several of the requirements, *KONTEXT* offers the most advanced support among these tools. *Lotus Notes/Domino* is potentially satisfying all requirements. But it would require some considerable implementation effort to actually establish these functions.

Future developments would be needed most in the areas of comprehensive lifecycle support (i.e., offering specific support for inserting new knowledge and evolving the already represented knowledge), support for automated generalisation and specialisation of artefacts[1], improved integration of multiple different knowledge types, and the definition of intuitive views on the stored knowledge, which can be tailored to the needs of certain user groups.

3. Technology Experience Bases

The technology experience base (TEB) is the basic repository of knowledge about software engineering technologies and their application contexts onto which our approach is based. It has been introduced briefly in Section 1. In the following, the structure of TEBs is explained in some more detail.

A TEB has basically two kinds of contents: (1) A collection of technology experience packages (TEPs) and (2) a collection of background definitions and taxonomies (see Figure 3). A TEP contains the definition of the respective technology (usually in the form of a process model) and specifies for which process (e.g., software design or measurement) it can be applied, which product quality (e.g., reliability or maintainability) of which product type (e.g., embedded control systems or medium-sized information systems) can be yielded using the technology, and in which context situation this application and quality impact of the technology can be expected (cf. [Bir97]). The context situation is defined through a set of *context characteristics*. A context characteristic is an attribute/value pair that defines a characteristic of a software project such as team size or degree of management commitment. The attribute and its definition is referred to as *context factor*. A collection of associated context factors forms a *context model*.

Background definitions and taxonomies are needed for achieving uniform definitions of the contents of TEPs, for avoiding redundant storage of these TEP contents, and to provide an index structure for experience retrieval. Hence, each concept contained in a TEP has an associated background definition and is part of a taxonomy. A TEB contains–among others–taxonomies of technologies, processes, products, and product qualities.

4. A Knowledge Management Lifecycle for Technology Experience Packages

The tool survey in Section 2 has shown that most knowledge management approaches are lacking yet a comprehensive lifecycle support for the managed knowledge artefacts. In this section and the following ones we use the case of technology experience pack-

1. Basic technologies for this are provided for instance from the field of machine learning.

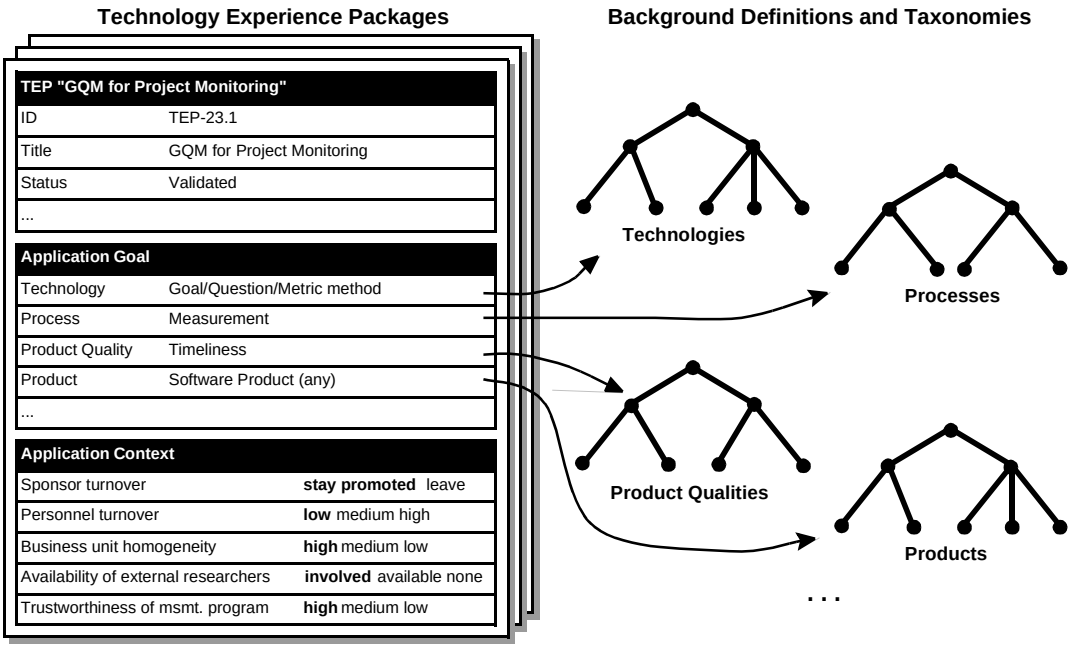

Figure 3: Technology experience packages and their relations to background definitions and taxonomies.

ages to illustrate how such a comprehensive lifecycle support can look like. The approach is substantiated by examples from a specialised knowledge management tool.

The knowledge management lifecycle for TEPs involves three main phases (Figure 4):

- Gaining TEPs using *knowledge acquisition* and other techniques.
- Using TEPs to support software engineering, e.g., for *technology selection* during the planning of software projects or improvement programmes.
- Updating and evolving TEPs based on the *empirical evaluation* of software projects that have applied them.

In addition, we briefly address the topic of building and installing a TEB (Section 5).

For gaining TEPs, a particular emphasis is placed on knowledge acquisition techniques, because we claim that much knowledge about the application context of software engineering technologies exists already implicit in the minds of experienced software professionals, or it is distributed over a large number of different media throughout a software organisation (cf. [BSA99]). Knowledge acquisition is considered to be the most beneficial source of knowledge when setting up a new TEB. The usage of TEPs and their empirical investigation are closely related to software projects or improvement programmes for which appropriate technologies are needed, and in which they are used.

A prototypical tool has been implemented for supporting the knowledge management lifecycle for TEPs. The tool is called *KONTEXT* (*KnOwledge maNagement based on the application conTEXt of software engineering Technologies*). It offers functions for the knowledge modelling tasks involved in knowledge acquisition and empirical evaluation of TEPs, as well as for the entire technology selection process. In the following sections, these functions of *KONTEXT* are explained in more detail.

5. Knowledge Acquisition of Technology Experience Packages

Most software organisations and experienced software professionals know much about the application contexts of software engineering technologies. However, this knowledge is often implicit or distributed over a large number of sources and hardly accessible. For this reason, knowledge acquisition is the preferred approach to quickly gain a large number of TEPs and to populate an initial TEB.

Alternative or additional information sources for developing TEPs are the software engineering literature, software measurement programmes, data mining and information research (using past project documentation, existing measurement data, or other organisational files), as well as empirical investigations such as surveys or case studies. However, all these techniques can also benefit from prior knowledge acquisition efforts that provide grounded hypotheses about the technologies and their application context.

The initial construction of TEBs, which should afterwards be extended and updated continuously, can in principle be done according to the following basic steps:

1. Identify and define the set of relevant technologies.
2. Determine the required target set of TEPs by specifying their processes, product types, and product qualities.
3. Conduct a pre-study of past projects in which the

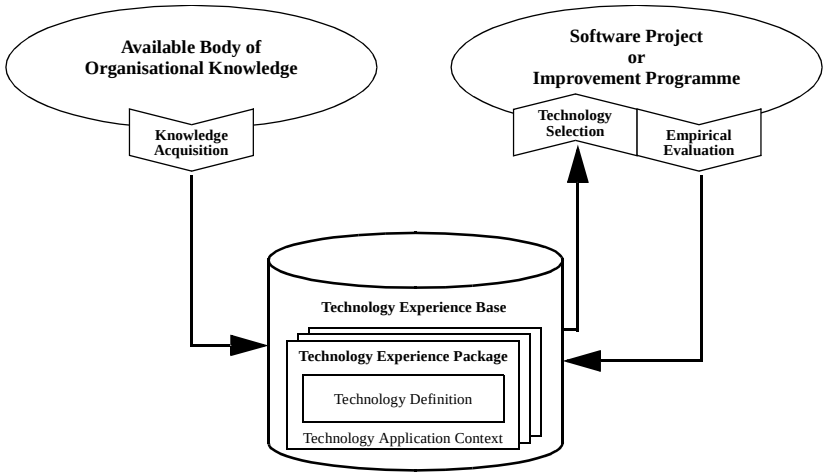

Figure 4: The lifecycle model of technology experience packages.

technologies have been used and collect relevant information from literature.

4. Develop background definitions and taxonomies of all relevant concepts that are to be represented in the TEB (e.g., precise operational technology definitions, process taxonomies, product quality definitions and taxonomies).
5. Conduct knowledge acquisition in order to gain context characteristics of the TEPs.
6. Model the TEPs based on the knowledge gained from knowledge acquisition.
7. Verify the modelled TEPs.
8. Validate the TEPs.

Figure 5 depicts the user interface of the TEP modelling component of *KONTEXT*. The leftmost part of the window guides the user through the process of modelling a collection of TEPs. It involves the insertion of a new technology, initialisation of a new TEP, modification of a selected or new TEP, as well as verification and validation of TEPs. The rightmost part of the window offers the functionality to perform the selected process step, in this case the modification of a selected TEP. The entire TEP is displayed and each component of it can be modified or extended.

Verification and validation of TEBs are very important. *KONTEXT* supports the automatic verification of TEPs such as completeness checks and some basic consistency checks. The extensive user guidance of *KONTEXT* avoids widely that wrong kinds of information could be inserted. Validation is currently supported through offering a well-structured report view on TEPs for inspection. Plans for future validation features involve several kinds of simulation-based and multi-expert checking.

6. Technology Selection Support Using Technology Experience Packages

A TEB can be beneficial for a multitude of tasks in software engineering, such as technology transfer and software risk management. As initial task to be supported, we focus on the selection of technologies during the planning of software projects and improvement programmes. We claim that this task is the one where decision support based on TEBs occurs most frequently and where it has the most direct impact on software development.

The selection of software engineering technologies during the planning of software projects and improvement programmes should follow the paradigm of informed decision making. There is a large variety of factors affecting the “right” decision, and the result of the decision can be highly critical to the success of a software organisation. Hence, tool support is required for supporting human decision making rather than for prescribing an “optimal solution”.

We base the decision support process implemented in *KONTEXT* on the principles of comprehensive reuse as introduced by Basili and Rombach [BR91]. This results in the following multi-staged decision support and selection process:

1. Determine the candidate processes for which a new technology can be applied, the product type to be developed, and the product quality goal to be achieved. – This information allows to pre-select a set of candidate technologies and their TEPs. From the context characteristics of these TEPs, a customised characterisation questionnaire can be constructed. It will be used for characterising the forthcoming project or improvement programme in a form that is suitable to conduct similarity-based retrieval of those TEPs that are most appropriate for the given reuse situation.
2. Characterise the forthcoming software project or improvement programme using the customised characterisation questionnaire. – Using this characterisation, a similarity-based ranking of the candidate TEPs can be conducted. This ranking is expected to support the final selection decision to be drawn by the human decision maker (i.e., the planner of the project or improvement paradigm).
3. Select the most appropriate technology (or technol-

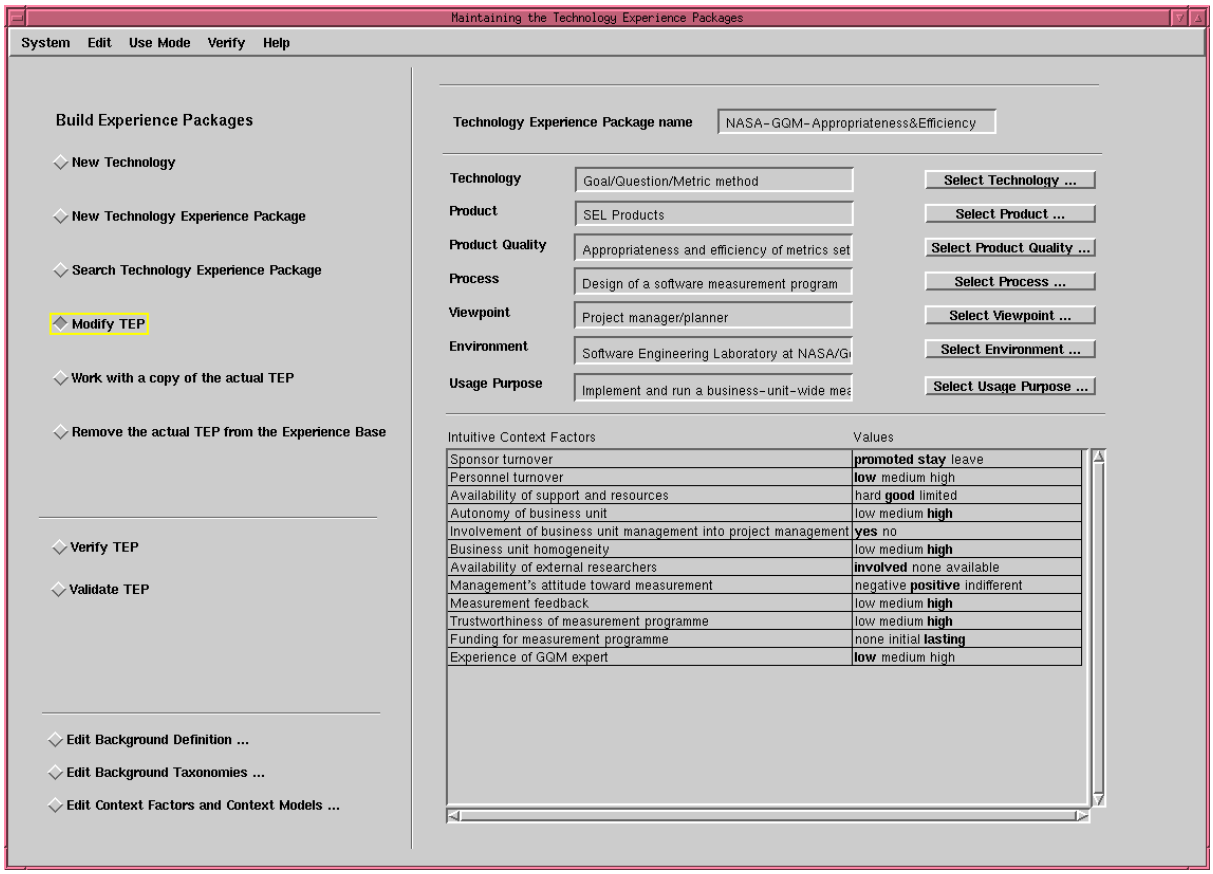

Figure 5: Modifying a technology experience package.

ogies) for the forthcoming project or improvement programme. – This selection should be justified explicitly in order to facilitate the achievement of commitment and as a basis for later evaluation of the decision.

In *KONTEXT*, we have implemented several features that we consider beneficial for informed decision making in software engineering. Our pilot implementation demonstrates that these are practical solutions for advanced tool support for knowledge management in software engineering:

- Investigation of background information within the decision support process (e.g., technology definitions can be viewed before the final selection decision is drawn).
- Exploration of concurrent scenarios during the decision process. For instance, different characterisations of the forthcoming project can be used concurrently for similarity-based retrieval, and the results can be compared.
- Offering multiple different decision support mechanisms (e.g., multiple similarity functions or different multi-attribute decision making algorithms [MPE96]; the decision maker can select the mechanism he/she regards most appropriate for the current type of selection).
- Backtracking and iteration of sub-steps of the decision process before the final decision is drawn. An example is the application of different decision support mechanisms using the same baseline information. Such a redundant decision process can increase the confidence in the results gained.
- Explicit justification of the final decision, in order to motivate the decision maker that he/she fully rationalises the decision.

The user interface of *KONTEXT* offers these features in its technology selection dialogue. It is shown in Figure 6. The rightmost column of the window guides the user through the multiple actions involved in technology selection. Among others, these are the preparation of the decision making by providing needed baseline information (e.g., characterisation of the forthcoming software project), the selection of a decision support method, and the drawing of the final decision and justifying it. The rightmost part of the window shows the dialogue for selecting baseline data for running a decision support algorithm that has been selected in a previous step. The baseline data involves one of possibly multiple alternative characterisations of the forthcoming project and a set of candidate TEPs. Since *KONTEXT* logs all activities of the decision process and their intermediate results, a hierarchy of gradually constrained sets of candidate technologies results from the iterative application of decision support algorithms. The second list in the window presents this hierarchical list of sets of candidate technologies and allows for selecting one of these sets.

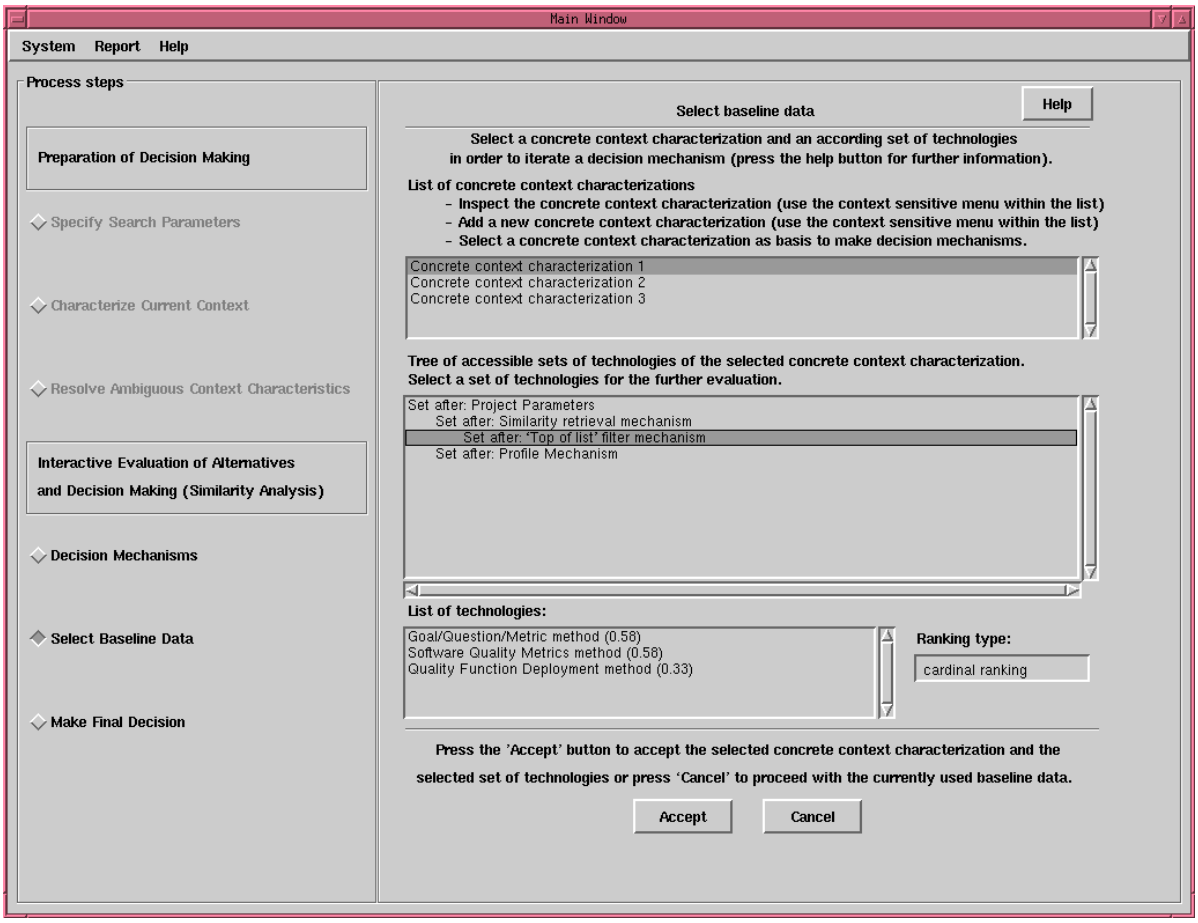

Figure 6: Selecting a technology

The technologies contained in the selected candidate set are depicted in the list at the bottom of the window.

For later evaluation of the application of the selected technology, the context characterisation and the entire trace of the decision support process are stored in the TEB. The following section explains how this information is used for refining and updating the TEB.

7. Empirical Evaluation of Technology Experience Packages

Objective of the empirical evaluation of technology experience packages is to possibly refine and update the contents of the TEB. When a technology is applied in a software project, this can be used as a case to investigate whether the information contained in a TEP is appropriate. The core question of empirical analysis is: Does the technology, when applied for the specified process and within the defined application context, really have an observable impact on the respective product quality? This question can be split into two separate investigations: (1) Has the product quality been achieved?, and (2) is the actual context situation of the project or improvement programme the same as defined in the TEP's context model and as it was expected when the technology was selected?

Depending on how the answers to these questions look like, there can be different consequences of empirical evaluation of TEPs:

- If the evaluation shows that the technology had significant impact on the product quality and that the actual application context was the same as in the TEP, then the contents of the TEB might be confirmed and provided with additional evidence. This would create deeper trust in the effectiveness of the technology when used later in similar application contexts.
- If the technology's impact on the product quality or the actual context characteristics were not as expected, then some kind of correction of the TEB is needed. Depending on the exact evaluation results and the contents of the TEB, the modifications can be quite different, ranging from small modifications of context models to the introduction of new variants of technologies and the partial re-design of multiple TEPs. For instance, it could turn out that a TEP on software inspections should better be split into two separate TEPs for variants of software inspections with and without inspection meeting. This would also require new, modified context models that clearly distinguish between the application contexts of the two inspections variants.

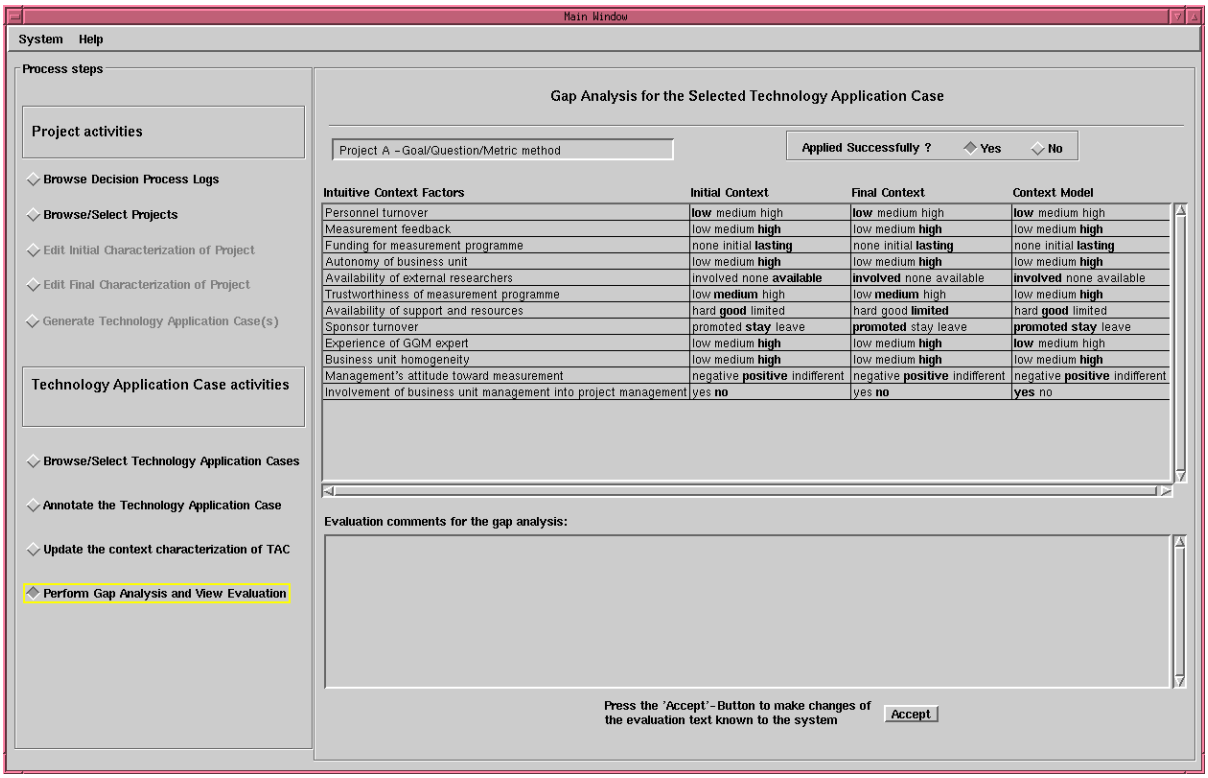

Figure 7: Empirical post-project analysis of a technology application case.

A detailed explanation of the analysis strategy for the empirical evaluation of TEPs is described in [BJvS99]. The main objective is to identify a causal link between application of the technology and the achieved product quality, under special consideration of the impact of the project context.

KONTEXT supports empirical evaluation by the following features:

- The trace of the technology selection process (cf. Section 6) including the initial characterisation of the project context (i.e., the expected characteristics of the forthcoming project) are stored in the TEB as a basis for future evaluation.
- The actual context situation at the end of the project can be supplied. It is then stored in the TEB.
- *KONTEXT* offers a user dialogue that guides through the process of analysing the deviations between initially expected context characteristics, actual project characteristics, and the TEP's context model (see Figure 7).

The object model of *KONTEXT*'s TEBs (see Figure 8) contains objects for representing information about projects and about the application of selected technologies in these projects (*technology application case*). This way, *KONTEXT* integrates abstract information about reusable artefacts (i.e., the TEPs) with concrete information about the actual reuse of these artefacts. This is not only beneficial for empirical analysis of technology application. It also provides information relevant for informed decision making during technology selection.

8. An Example Technology Experience Base

A technology experience base of the presented kind is currently being developed in ESPRIT project PROFES[1]. The objective of PROFES is to support industries that have strong product-related quality requirements, such as the embedded systems industry, with an improvement methodology that focuses improvement actions on those elements of the software development process that contribute most to the critical product quality factors. It places emphasis on the continuous learning about the impacts that software engineering technologies and the processes in which they are applied have on product quality. This information is modelled in the form of so-called product/process dependencies (PPDs) and stored for reuse in a central repository.

The PPD repository is widely analogous to the technology experience base introduced above. PPDs deploy the representation scheme of TEPs. The usage processes on the PROFES PPD repository are designed in accordance with the presented TEP lifecycle model.

The PROFES PPD repository contains information about software engineering technologies that has been collected from multiple literature sources and from three industrial software organisations that have participated in the PROFES project. It will be offered for public use via the internet. So other software organisa-

1. ESPRIT project 23239, PROFES (PROduct Focused improvement of Embedded software Processes); funded by the European Commission. http:/www.iese.fhg.de/Profes

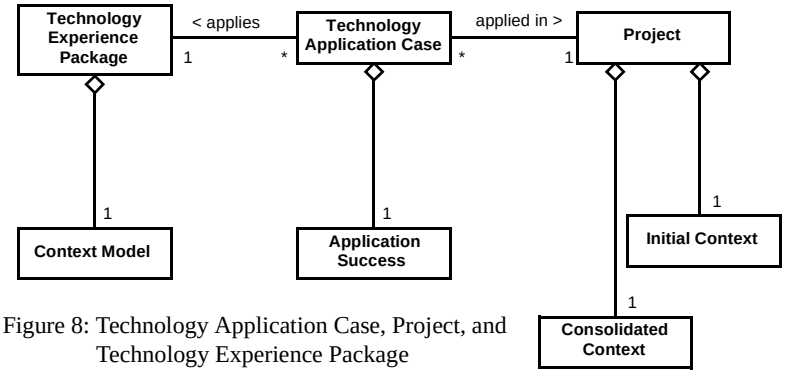

Figure 8: Technology Application Case, Project, and Technology Experience Package

tions can use the PROFES PPD repository to support the identification of improvement actions (i.e., the selection of technologies that are to be applied in forthcoming software projects) in their improvement programmes. They also will be able to feed back their experience and thus help to evolve and extend the repository.

First experience from the industrial applications of PROFES substantiate core components of the presented TEP lifecycle model: The TEP representation schema has proven appropriate for representing the information needed within technology experience bases, knowledge acquisition has shown effective for gaining TEPs, and the importance of product quality goals, processes, and context factors has been demonstrated (cf. [BJvS99]).

9. Conclusions

The systematic management of knowledge about software engineering technologies and their application contexts is expected to reduce the risk of technologies-caused failure of software projects. Hence, knowledge management can be regarded an important means for further improvement of software engineering practices.

The Quality Improvement Paradigm (QIP) / Experience Factory (EF) approach ([BCR94]) provides appropriate solution concepts for realising customised knowledge management of experience on software engineering technologies that is packaged for reuse during the planning of software projects and improvement programmes. This experience is modelled explicitly in the form of so-called *technology experience packages* (TEPs) that are stored in repositories called *technology experience bases* (TEBs).

We have suggested a structure for organising TEBs and a comprehensive lifecycle model (1) for gaining TEPs through knowledge acquisition techniques, (2) for using them to support technology selection during the planning of software projects and improvement programmes, as well as (3) for updating and evolving TEPs based on the empirical evaluation of software projects.

Our approach differs from existing ones in that it addresses all three phases of the lifecycle model at a considerably detailed technical level. It provides guidance for organising and running technology experience bases. A prototypical tool called *KONTEXT* has been implemented in order to illustrate and substantiate the methodological concepts. An industrial trial application has provided first empirical evidence for the validity of the approach.

Acknowledgements

We want to thank our collegues at the SLI department of Fraunhofer IESE for the many valuable discussions and the feedback they have provided to our work. Markus Nick has provided comments on an earlier version of this paper.

References

[ADK98] Andreas Abecker, Stefan Decker, and Otto Kühn. Organizational memory. In *"Das aktuelle Schlagwort" im Informatik Spektrum*, volume 21 of *4*, pages 213–214. Springer Verlag, August 1998.

[BCR94] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. Experience Factory. In John J. Marciniak, editor, *Encyclopedia of Software Engineering*, volume 1, pages 469–476. John Wiley & Sons, 1994.

[Bir97] Andreas Birk. Modelling the application domains of software engineering technologies. Technical Report 014.97/E, Fraunhofer IESE, August 1997.

[BJvS99] Andreas Birk, Janne Järvinen, and Rini van Solingen. A validation approach for product-focused process improvement. Technical Report IESE-Report No. 005.99/E, Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany, 1999.

[BR91] Victor R. Basili and H. Dieter Rombach. Support for comprehensive reuse. *IEEE Software Engineering Journal*, 6(5):303–316, September 1991.

[BSA99] Andreas Birk, Dagmar Surmann, and Klaus-Dieter Althoff. Applications of knowledge acquisition in experimental software engineering. In *Proceedings of the 11th European Workshop on Knowledge Acquisition, Modeling, and Management*

(EKAW'99), Berlin, 1999. Springer.

[CDR95] CDR, Center for Design Research. http://gummo.stanford.edu/html/gcdk/dedal/index.html, 1995.

[CKO92] Bill Curtis, Marc I. Kellner, and Jim Over. Process modeling. *Communications of the ACM*, 35(9):75–90, September 1992.

[Eri92] Henrik Eriksson. A survey of knowledge acquisition techniques and tools and their relationship to software engineering. *Journal of Systems and Software*, (19):97–107, 1992.

[FDES98] Dieter Fensel, Stefan Decker, Michael Erdmann, and Rudi Studer. Ontobroker: The very high idea. In *Proceedings of the 11th International Flairs Conference (FLAIRS-98)*, Sanibel Island, Florida, May 1998.

[Gla98] Robert Glass. *Software Runaways.* Prentice Hall, 1998.

[GvWB98] C. Gresse von Wangenheim, A. von Wangenheim, and R. Barcia. Case-based reuse of software engineering measurement plans. In *Proc. of the 9th Int. Conference on Software Engineering and Knowledge Engineering (SEKE)*, 1998.

[Hen97a] Scott Henninger. Capturing and formalizing best practices in a software development organization. In *Proc. of the 9th Int. Conference on Software Engineering and Knowledge Engineering (SEKE)*, 1997.

[Hen97b] Scott Henninger. Case-based knowledge management tools for software development. In *Automated Software Engineering: An International Journal*, volume 4. Kluwer Academic Publishers, 1997.

[IBM98] IBM. http://www.software.ibm.com./data/ids/, 1998.

[KPM95] KPMG Ltd. Runaway projects–cause and effects. *Software World*, 26(3), 1995.

[Krö98] Felix Kröschel. A system for knowledge management of best software engineering practice. Master's thesis, University of Kaiserslautern, Kaiserslautern, Germany, November 1998.

[Lot] Lotus. http://www.lotus.com.

[LSH98] Dieter Landes, Kurt Schneider, and Frank Houdek. Organizational learning and experience documentation in industrial software projects. In *Proceedings of the Workshop on Organizational Memories at the European Conference on Artificial Intelligence '98*, pages 47–63, Brighton, England, August 1998.

[MPE96] Mansooreh Mollaghasemi and Julia Pet-Edwards. *Making Multiple-Objective Decisions*. IEEE Comuter Society Press, 1996.

[MT90] Ackerman M.S. and Malone T.W. Answergarden: A tool for growing organizational memory. In *Proc. of the ACM Conference on Office Information Systems*, pages 31–39, 1990.

[O'L98] Daniel O'Leary. Using AI in knowledge management: Knowledge bases and ontologies. *IEEE Intelligent Systems*, pages 34–39, May/June 1998.

[Ope] Open Text. http://www.opentext.com.

[Rei97] Ulrich Reimer. Knowledge acquisition for content selection. In *21st Annual German Conference on AI '97*, Freiburg, September 1997. http://www.dfki.uni-kl.de/km/ws-ki-97.html.

[RV95] H. Dieter Rombach and Martin Verlage. Directions in software process research. In Marvin V. Zelkowitz, editor, *Advances in Computers, vol. 41*, pages 1–63. Academic Press, 1995.

[Sen90] Peter M. Senge. *The Fifth Discipline. The Art and Practice of The Learning Organization*. Bantam Doubleday Dell Publishing Group, Inc., New York, 1990.

[Shu97] S. Buckingham Shum. Negotiating the construction and reconstruction of organisational memories. *Journal of Universal Computer Science*, 3(8):899–928, 1997.

[SWI97] Department of Social Science Informatics Sociaal Wetenschappelijke Informatica. http://www.swi.psy.uva.nl/projects/newkactus/reports.html, 1997.

[TZ98] Roseanne Tesoriero and Marvin Zelkowitz. A web-based tool for data analysis and presentation. *IEEE Internet Computing*, 2(5):63–69, 1998.

[Wii95] Karl Wiig. Knowledge management methods. Practical approaches to managing knowledge. In *Knowledge Management: The Central Management Focus for Intelligent-Acting Organizations*, volume 3. Schema Press, Ltd, 1995.