Customizing Software Engineering Experience Management Systems to Organizational Needs

Beim Fachbereich Informatik der Universität Kaiserslautern
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften (Dr. rer. nat.)

eingereichte Dissertation von

Dipl.-Inform. Carsten Tautz

Fraunhofer-Institut für Experimentelles Software Engineering
Kaiserslautern

Summary

The increasing internationalization of world trade intensifies the global competition forcing organizations to accelerate their technology changes and product innovations. Software development organizations are not exempted from this trend as software plays an increasingly important role for the value of products and services. The systematic acquisition, utilization, and maintenance of software engineering experience accelerates goal-oriented changes in software development practices. Thus, it has become a major competitive advantage.

Explicit software engineering experience is stored in a repository called the *experience base*. This dissertation presents a methodology based on the principle »learning from examples« for designing and implementing an experience base that contains all kinds of experience. The methodology consists of:

- A new **knowledge representation formalism** for experience base schemas. The formalism fulfills many practical needs (e.g., rationalized schemas and support for uncertain, imprecise, incomplete, and context-dependent information) for the first time. As such, it uses many features typically associated with case-based reasoning systems as well as ideas from other areas such as database systems.
- A **generic architecture** for the comprehensive tool support of an experience base. The architecture considers many practical needs such as the integration of already existing tools and learning in distributed environments.
- A **generic task framework**. The framework is the most comprehensive description for incremental, continuous learning (of all kinds of software engineering experience) based on a repository to date. Its description is based on the knowledge-level approach and, hence, implementation-independent.
- A **systematic, goal-oriented method for setting up an experience base**. The method describes how to engineer experience base schemas as well as how to tailor the generic architecture and instantiate the task framework to organization-specific needs. It is unique in the sense that its application does not only provide a schema, but also matching methods for populating (and updating) the experience base on the basis of the developed schema.

Systems designed using this methodology will provide faster and more effective access to software engineering experience available in an organization compared to existing solutions (usually a combination of »asking colleagues« and consulting lists and databases accessible over the intranet).

The methodology is complemented by a goal-oriented method for improving the technical infrastructure of a running experience base from a user´s point of view.

The components of the methodology have been validated in several industrial-strength projects including Fraunhofer IESE´s Corporate Information Network, which is presented in detail. A prototype for the tool architecture has been built using a commercially available case-based reasoning system as its modeling tool and retrieval engine.

Table of Contents

List of Requirements

List of Tasks

1 Introduction

In the business world, everyone is paid in two coins: Cash and experience. Take the experience first; the cash will come later.
Harold S. Geneen

In the industrial world, a radical change is emerging which is shaped by the ongoing globalization, internationalization, and liberalization of world trade [BWP98]. The change does not only result in a globally increased economic performance, but can also be characterized by an increased competition as well as accelerated technological change and product innovation. This confronts organizations with ever new challenges.

In this environment, »competitiveness derives from an ability to build, at lower cost and more speedily than competitors, the core competencies that spawn unanticipated products« [PH90, p. 81]. For an increasing number of organizations, software know-how has or will become one of these core competencies as it fulfills the three properties of a core competence [PH90]:

- **Potential access to a wide variety of markets.** Software has become an integral part of many products and services. Some services (e.g., telecommunications and banking) are no longer conceivable without software.
- **Significant contribution to the perceived customer benefits of the end product.** For many everyday products (e.g., telephones, hi-fi equipment, cars), software is a major part (e.g., it provides the functionality of telephone systems) or at least adds considerable value to the product (e.g., software lowers fuel consumption of cars). For many technical products, software development already makes up 75 to 80% of the total product development costs [Gla94].
- **Imitation by competitors is hard.** Software codifies know-how (e.g., the knowledge on how to lower fuel consumption). It takes years of experience to build up this know-how. As it is not visible, one can only observe the results (e.g., that a car has a low fuel consumption), but not the cause-effect relationships which explain the results. Thus, it is hard to imitate the functionality of the software.

  Similarly, a software development process model codifies know-how. For example, a process model that enables implementing many features within a short time span will outperform competitors regarding time-to-market and functionality. Again, as one (from the outside) can only observe that software products are created or updated fast (but not how this is accomplished), this kind of know-how is also hard to imitate by competitors.

Therefore, only those software development organizations with top performance regarding time-to-market, cost, and quality will survive the global competition in the long run [ABH+99].

In this context, the systematic acquisition, utilization, and maintenance of software engineering experience will become a decisive competitive advantage in the future. Basili and Rombach recognized this fact already in the late 1980´s when they advocated not to limit software reuse to code and software documentation, but to support reuse of all kinds of experience [BR88].

The systematic acquisition, utilization, and maintenance of knowledge in general has a long tradition in the »artificial intelligence« field of computer science [Ric89]. Because of the knowledge-intensive nature of software engineering experience management, it seems natural to support the ideas of Basili and Rombach by using techniques from the artificial intelligence field.

However, at the time this dissertation was being written, there existed no technical infrastructure capable of supporting the management of all kinds of software engineering experience and fulfilling the manifold requirements from practice (e.g., easy customizability) at the same time. Moreover, there was no goal-oriented, systematic method for designing a repository for software engineering experience.

This dissertation addresses these topics. It presents DISER, a methodology for designing and implementing software engineering repositories. DISER accelerates the design of an effective and efficient management system for software engineering experience. Systems designed using DISER will provide faster and more effective access to software engineering experience available in an organization compared to the state-of-the-practice (usually a combination of »asking colleagues« and consulting lists and databases accessible over the intranet).

Learning from examples

DISER is based on the principle »learning from examples«, a learning strategy from the field of »machine learning« [CMM84]: »Given a set of examples and counterexamples of a concept, the learner induces a general concept description that describes all of the positive examples and none of the counterexamples« [CMM84, p. 9]. In terms of software engineering this means that experiences gained during a software development project are stored as examples and counterexamples. To perform some task or to solve some occurred problem, other software development projects can retrieve these examples and counterexamples and induce consequences for their project. These consequences result in new guidelines for the project, for instance, in form of changed process models or templates. As these new guidelines are applied in the project, they are subject for gaining new (application) experiences (e.g., how successful the application of a guideline was). Together with these application experiences, the guidelines become new examples (and counterexamples). Thus, a loop for continuous, incremental learning is established.

Offer-driven versus demand-driven strategy

Since experience is recorded in the form it becomes available, the employment of the principle »learning from examples« can be viewed as an »offer-driven« strategy. Everything deemed to be potentially useful[1] to other projects is recorded. In contrast, a »demand-driven« strategy does not acquire experience on the basis of availability but on some identified improvement need of an organization, which is the result of some type of analysis (e.g., an assessment) [EB00a, EB00b]. Using this strategy, a problem might be that the needed experience is not readily (i.e., explicitly) available. Therefore, experience needs to be acquired using special knowledge acquisition techniques and documented using specialized model types [Bir00]. In this way, expert knowledge, which is typically experience aggregated from several projects, is made available. Such expert knowledge can be recorded in the form of software engineering technologies and offered for reuse using a repository [PRO00].

In practice, the application of both strategies is feasible depending on whether a particular (improvement) goal is to be attained or productivity is to be raised in general by sharing available experience. Both strategies have the setting up and running of a repository in common. They differ in the way the repository is built up. On one hand, the »offer-driven« strategy builds up the repository in a bottom-up fashion (aggregated knowledge is induced from examples). On the other hand, the »demand-driven strategy« builds up the repository in a top-down fashion (while technologies are applied, application experiences are recorded meaning that case-specific knowledge is obtained on the basis of aggregated knowledge). However, a common repository using the same technical infrastructure can be used for both strategies [Bro00]. Therefore, the strategies are combinable.

Although any mixture of both strategies is conceivable, the rest of this dissertation will primarily focus on the »offer-driven« strategy[2]. Consequently, the method »learning from examples« will play a central role.

The method »learning from examples« has been thoroughly investigated in artificial intelligence. One of the most prominent approaches in this area is that of *case-based reasoning* [AAB+95, Alt97, Wes95]. In case-based reasoning, examples are stored as *cases* in a *case base*. A case is a problem solution pair and documents how a particular problem has been solved (and with what result).

The problem-solving method using case-based reasoning is described by the case-based reasoning cycle [AP94]. A new problem is solved by adapting a solution of a similar problem that has occurred in the past [Ric98]. Depending on

1 In practice, the meaning of »potentially useful experience« needs to be defined more precisely. Typically, it is defined in terms of relevant topic areas and minimal quality requirements for the experience.

2 The »demand-driven« strategy is described in [Bir00].

what is stored in the cases, this method can be generalized as follows: A new problem is solved by adapting and combining the solutions of one or more similar problems that have occurred in the past.

1.1 Experience Factory Concept

The case-based reasoning cycle is very similar to the quality improvement paradigm (QIP) [BCR94a] in the sense that both describe how to achieve continuous, incremental improvement. The QIP is a six-step procedure for structuring software development and improvement activities. It involves three overall phases: planning, performing, and evaluating the software development project. The planning phase at the start of a new project is based on the explicit characterization (QIP1) of the initial situation, the identification of the project as well as learning and improvement goals to be achieved (QIP2), and the development of a suitable plan (QIP3) that shall achieve these goals. The plan then guides the performance of the development project (QIP4), which is a feedback loop of its own controlling goal achievement. The subsequent evaluation phase involves the analysis of the performed actions (QIP5) and the packaging of the experiences into reusable artifacts (QIP6). In contrast to the QIP, the case-based reasoning cycle is described on a more technical level. Both cycles can be matched resulting in an integrated model which can be used as the basis for an interaction model for software development projects and case-based reasoning systems (a more detailed discussion on this subject matter can be found in Section 4.2).

The QIP is one of the major principles underlying the experience factory concept [BCR94a]. The *experience factory* concept addresses the issue of managing all kinds of software engineering experience:

Definition 1: Experience factory

> *The* experience factory *is a logical and/or physical organization that supports project developments by analyzing and synthesizing all kinds of experience, acting as a repository for such experience, and supplying that experience to various projects on demand. It packages experience by building informal, formal or schematized, and productized models and measures of various software processes, products, and other forms of knowledge via people, documents, and automated support.*
>
> *[BCR94a, p. 472]*

To support the tasks of the experience factory, a repository which stores all kinds of software engineering experience is used. This repository is called the *experience base*. The experience base can be seen in analogy to the case base of a case-based reasoning system, whereas the experience factory corresponds to the organizational unit responsible for setting up and maintaining the case-based reasoning system [BBG+99].

However, the experience factory is responsible for much more than administrating the experience base:

> *The experience factory can be used to consolidate and integrate activities, such as packaging experience, consulting, quality assurance, education and training, process and tool support, and measurement and evaluation.*
>
> *[BCR94a, p. 475]*

As such, the experience factory is responsible for running software process improvement programs that include technology preparation and infusion. Numerous applications of the experience factory concept in practice have shown its effectiveness [BDY94, BZM+95, Das92, GC87, Hal96, HSW98, MB97, Rom96].

1.2 Focus of Dissertation

There are five kinds of aspects to be considered when setting up and running an experience factory:

1 **Strategic aspects** relate to the scope of the experience base, i.e., to the competencies to be built up by the experience factory. The scope of the experience base is subject to change when the environment changes (e.g., if the market requires to prepare and infuse a new technology).

2 **Organizational aspects** relate to issues such as organizational units, positions, and/or the existence of long-term projects for running an experience factory. These aspects have to be reconsidered if organizational changes take place within the software development organization.

3 **Managerial aspects** relate to issues such as success control, budget, available personnel, allocation of responsibilities to organizational units, allocation of people to tasks, etc.

4 **Methodological aspects** relate to the tasks that have to be performed as well as how they have to be performed.

5 **Implementation aspects** relate to issues about the technical infrastructure used for the experience base such as the architecture of the tools and the user interface employed.

This dissertation describes how to set up and run an experience base. Thus, it focuses on the strategic, methodological, and implementation aspects. A discussion of the organizational issues can be found in [BCC92], whereas managerial

aspects are covered by project management literature and papers on the experience factory concept (e.g., [BR88, BCR94a, Bas95, BC95]).

Hence, the dissertation takes on a technical view. This view can be characterized as »repository-based reuse of software engineering experience«.

1.3 Contributions of Dissertation

As pointed out above, this dissertation presents DISER, a methodology for designing and implementing a software engineering experience management system.

Definition 2: Software engineering experience management system

> *A* software engineering experience management system (SEEMS) *is a combination of an experience base and all software systems which provide functionality for storing, retrieving, evaluating, maintaining, disseminating, and utilizing software engineering experience of this experience base.*

DISER extends and refines the requirements and mechanisms for an experience base described in [BR91]. At the same time, it refines the case-based reasoning cycle of Aamodt and Plaza [AP94]. In addition, it considers constraints typically encountered in practice (see Figure 1) and consists of:

- A representation formalism for experience base schemas
- A generic architecture for the technical infrastructure of an experience base
- A generic framework for populating, maintaining, and utilizing an experience base in form of a task hierarchy
- A method for engineering an experience base schema, customizing the architecture, and instantiating the framework

Representation formalism

The manifold constraints regarding the software engineering experience required to develop a new representation formalism. Existing formalisms are not able to cope with the combination of the following requirements:

- Support for both formal and informal represented knowledge
- Support for certain and uncertain, consistent and (in some respects) inconsistent, precise and imprecise, complete and incomplete, stable and instable information
- Support for context-dependent information
- Inclusion of the most common classification techniques from information and library science
- Support for the management of dependencies among artifacts
- Support for the characterization of various types of artifacts in an artifact-specific way

In addition, existing representation formalisms do not support the documentation of rationales behind the schemas described using their constructs to the

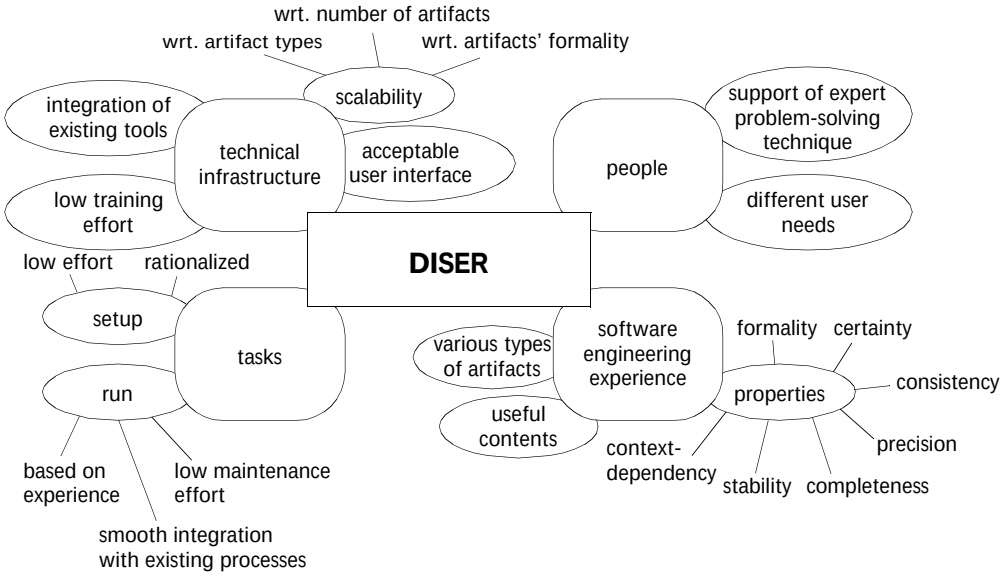


Figure 1: Selected practical constraints on DISER

degree necessary.[1] Hence, schemas are neither tailorable to changing organizational needs nor to the needs of other organizations (reuse of schemas with modification).

The representation formalism of DISER supports all this. It draws from ideas from several areas including database systems (e.g., semantic relationships among artifacts and implied consistency rules) [Che76, Dit97, GR95], case-based reasoning mechanisms (e.g., similarity-based retrieval with incomplete information) [Alt97], and knowledge-based mechanisms (e.g., value inferences and assertions) [Ric89]. Due to the inclusion of the most common classification techniques, the management of dependencies among artifacts, and documented rationales, it is possible to scale the experience base during its operation with respect to the number and types of artifacts to be stored as well as the formality of the stored experience. Schemas described using this representation formalism are modular and reusable. Thus, the development of an adequate schema (part of setting up an experience base) will require less effort than developing schemas from scratch.

In addition, the representation formalism allows to define value inferences (i.e., the automatic computation of characterization values) and to share parts of arti-

1 One reason for this »phenomenon« is the fact that most knowledge-based systems are designed for one purpose only. However, knowledge management systems are usually used for various purposes. Therefore, it becomes necessary to document which parts of the schema are needed for which purposes.

facts´ characterizations (e.g., context information such as the project in which the experience was gained). Thus, maintenance effort for recording and restructuring experience is kept at a minimum.

Architecture for a technical infrastructure

The schema defines the behavior of the *General Purpose Browser* which allows the recording, retrieval, and maintenance of the experience base´s contents. It is part of the generic architecture which also allows to integrate tools already existing in a given organization (e.g., CASE tools to edit requirements and design documents). Due to the familiarity of the users with the existing tools and the World-Wide Web (WWW) interface of the General Purpose Browser, the required training effort for using the technical infrastructure is low. Validation in several industrial-strength projects has shown that the user interface of the General Purpose Browser is acceptable.

A prototype of the tool infrastructure uses the commercially available case-based reasoning tool CBR-Works from tec:inno, Germany [tec00] as one of its components. CBR-Works serves as both the modeling tool for the experience base schema and the retrieval engine of the experience base. It was chosen, because it already fulfilled many of the practical constraints imposed on the representation formalism.

Framework for populating, maintaining, and utilizing an experience base

The generic framework for populating, maintaining, and utilizing the experience base describes how the architecture is used. It is based on the strategy »learning from examples«. This means that experience is recorded as it becomes available together with information of the context from which it originates. Later, such experience can be retrieved, assessed for its applicability, and tailored to the project situation at hand. This »imitates« experts who also recall similar problems from the past and adapt the corresponding solution(s) to the problem situation at hand. Thus, the framework supports the most natural technique of expert problem-solving.

The »learning from examples« strategy enables the recording of experience close to the point in time when it becomes available. Using this strategy, the experience base becomes a *communication medium* over which experience can be disseminated fast. This way, customer needs are documented early and, consequently, the development of innovative products (with many new features and short lead time) is supported.

Because recording of the experience is guided by a predefined schema, the effort for recording new experience is low and does not require an elaborate knowledge acquisition technique. During recording, experience is qualified. The qualification ensures a minimal quality of the experience so that only useful experience is stored in the experience base.

The task hierarchy making up the framework is the most comprehensive description for incremental, continuous learning based on a repository to date.

The task descriptions are based on Steel´s knowledge-use level approach (components of expertise) [Ste90], an extension of Newell's knowledge level approach [New82]. The knowledge-use level approach describes an intelligent system by arranging the tasks to be performed in a hierarchy and by specifying for each task:

- The goal to be attained by the task
- A method that either defines in what order to perform the subtasks of the task or how to perform the task if it is not further decomposed
- Contents (not structure!) of knowledge that is needed to perform the task
- Pragmatic constraints on the access and availability of the knowledge

In this dissertation, the task descriptions are extended by the requirements that should be fulfilled to support the task technically by the tool infrastructure. In this way, the (task) framework can be instantiated systematically for a given organization. It also provides the basis for tailoring the architecture to organization-specific needs via the listed (technical) requirements.

Method for engineering schemas, customizing the architecture, and instantiating the framework

The instantiation of the framework is based on the information needed and available at the particular organization. A goal-oriented, systematic method called GOODSEE (goal-oriented ontology development for software engineering experience) derives the information needs from stakeholders' interests via reuse scenarios. GOODSEE also surveys existing knowledge sources and derives knowledge acquisition scenarios which describe how available knowledge can be converted to needed information. Later, the needed information is conceptualized in form of an experience base schema (using the representation formalism). The conceptualization will guide the retrieval and utilization (part of the instantiated framework) of the stored information during the operation of the experience base. Corresponding recording methods (part of the instantiated framework) are defined based on the knowledge acquisition scenarios. Finally, the architecture is tailored to the needs of the organization.

By considering the needs of the users and the available knowledge sources within an organization, the tasks of the organization-specific instantiation of the framework smoothly integrate with the processes already existing in the organization. Different user needs are considered by allowing any number of reuse scenarios.

Since all (intermediate) results produced by GOODSEE are traceable to their »specification«, the setup of an experience base is systematic and goal-oriented. The produced results are rationalized. This allows to reuse (with modification) parts of the experience base documentation. Hence, scenarios and schemas do not have to be developed from scratch and methods for recording can be selected from a library. All this saves effort when setting up a new experience base.

GOODSEE is unique in the sense that its application does not only provide a schema for the experience base, but also matching methods for populating (and updating) the experience base on the basis of the developed schema. In contrast to existing schema development approaches (which are either too narrow or too broad), GOODSEE is both general enough to cover the whole domain of software engineering experience reuse and specific enough to be able to give meaningful, detailed descriptions which can be understood by a software engineer.

1.3.1 Related Areas

Reuse is not limited to the domain of software reuse. Several synonymous terms for »reuse« have been created:

- Case-based reasoning (reuse of problem solution pairs)
- Knowledge-based systems (reuse of problem solving knowledge for analysis or synthesis tasks)
- Knowledge management, knowledge sharing (reuse of both tacit and explicit knowledge)
- Information management [Ort91] (reuse of information in general)
- Decision support (reuse of existing information with the aim to make informed decisions)

These generalizations are associated with synonyms for the term »experience base«:

- Case base (experience base for cases of a case-based reasoning system)
- Knowledge base (experience base for explicitly represented knowledge of knowledge-based systems)
- Organizational memory (experience base for knowledge in general)
- Information system (experience base for information in general)
- Management information system (experience base for management information)
- Data warehouse (a technical infrastructure for an experience base that ensures the organization-wide availability of information needed for management support by importing and analyzing data from databases distributed throughout the organization [Glu97])

These synonyms do not only point out the importance of reuse, but also the complexity of the topic. This complexity cannot be ignored for software reuse, because the term »software engineering experience« cannot be clearly defined. In one organization the term may stand for software development documents only, while in another organization it may also include (software development) project and management experience. Often the latter type of experience is not specific to software development projects. Thus, related research areas such as case-based reasoning, knowledge-based systems, decision support, knowledge management, and data warehousing become relevant.

Case-based reasoning

As described above, case-based reasoning supports the strategy of »learning from examples«. As such, many methods and tools developed in this research area can be used for implementing an experience base. However, both methods and tools need to be extended and tailored to the specific needs of software engineering. In return, a new application domain for case-based reasoning is established. For example, the addition of »support for the management of dependencies among artifacts« via case references in CBR-Works led to the successful application of case-based reasoning for managing lessons learned at a large German insurance company [ABH+99] and for TQM in the health care sector [ABMN99]. Moreover, the improvement method described in Chapter 9 can be readily applied to case-based reasoning systems [ANT99b]. In addition, DISER can be generalized to cover a larger domain than software engineering experience reuse. Thus, the results of this dissertation can be used to complement existing methodologies such as the INRECA-II methodology [BA98, BBG+98, BBG+99] and more management-focused approaches such as the one of Stolpmann and Wess [SW99].

Knowledge-based decision support

»Experience, combined with the possibility of inexact matches, makes case-based reasoning an important technology in the field of decision support« [Ric98, p. 14]. In software engineering, decision support is useful for selecting the most adequate artifacts (among other purposes) [Bir00]. However, the actual utility of an artifact does not become known before the completion of its application. Hence, it has to be approximated by some kind of similarity function which allows inexact matches [AR99, Wes95]. DISER contributes to the area of decision support by providing a method for describing decision support scenarios and for engineering similarity functions.

Knowledge management

Another important related area to software reuse is knowledge management. As Yglesias points out, both knowledge management and software reuse manage artifacts that »range along a continuum from ideas to experiences to presentations and processes to models and code« [Ygl98]. However, knowledge management places most emphasis on the abstract end of this spectrum, whereas software reuse emphasizes the most constrained end. Although the steps used for knowledge management and software reuse are very similar on an abstract level, different methods are used to execute them. For knowledge management, human-based knowledge transfer (e.g., through group meetings and talks in the coffee kitchen) is often used, whereas software reuse typically employs some repository. Another difference concerns the representation: Knowledge management also deals with explicating tacit knowledge, whereas software reuse assumes that knowledge is represented explicitly and does not need to be explicated.

> *The emphasis on making the results of learning explicit and codified is central to systematic reuse. This goes beyond the general idea of organizational learning and involves a more radical shift to the ideas of a knowledge-creating organization [Non91]. A knowledge-creat-*

> *ing organization recognizes the creation of new knowledge as an integral part of every work process, and makes the capture of such knowledge central to its mission and business model. Where a learning organization might still treat learning activities as »process improvement« activities, the knowledge-creating organization considers systematic learning as strategic.*
>
> *[BSLC96, p. 3]*

Regarding knowledge management, DISER contributes to the idea of a knowledge-creating organization by means of a repository-based approach. Due to the inclusion of the most common classification techniques from information and library science in the representation formalism, DISER is able to cover the whole continuum of artifacts. Management of tacit knowledge is supported through knowledge maps (which can be modeled using the representation formalism). Knowledge maps are »yellow pages« that associate a particular topic with one or more experts [DP98]. These experts can be consulted for further information on the topic.

Information systems

Organizational memories [ADK98] or information systems in general [AN95] are used to store and retrieve explicit knowledge. In this sense, DISER can be generalized to a methodology for the design and implementation of information systems[1].

In the area of information systems, the focus has shifted from making information available to filtering out irrelevant information:

> *Over the last couple of years, the way people (not only scientists) access and search for information has been revolutionized. Information is at the fingertips of everyone, the people suffer now from information overload instead of having problems getting access to the information.*
>
> *[KLG98, p. 331]*

DISER contributes in this area by using a similarity-based and context-sensitive retrieval mechanism. The similarity-based retrieval does not only return a set of artifacts, but estimates their usefulness via a similarity function. Only the most similar artifacts need to be further investigated by the user. The similarity function is part of the experience base schema and thus subject to goal-oriented, systematic development. Moreover, a method is presented for improving the similarity function.

1 In fact, the methodology has been used successfully for supporting TQM in the health care sector [ABMN99].

The context-sensitive retrieval restricts applicable artifacts to those that have been validated in similar contexts. The explicit documentation of the context from which the artifacts originate also improves the understanding by the user. Hence, DISER copes with another problem of information systems, especially in the area of software engineering:

> *While many new methods, languages, tools, and formal methods have been created, little support exists for understanding when different techniques should be used and how they should be applied within the specific environment of software development organizations.*
>
> *[Hen97a, p. 24]*

Data warehousing

Information systems also play an important role in data warehouses. According to Gardner, »data warehousing is a process for assembling and managing data from various sources for the purpose of gaining a single, detailed view of part or all of a business« [Gar98, p. 54]. This situation is very similar to »converting« available information to needed information as it is described by the knowledge acquisition scenarios developed using GOODSEE. Thus, DISER contributes to data warehousing by providing a method for identifying the needs of a data warehouse´s stakeholders, eliciting requirements for the data warehousing process, and developing a schema for the needed views.

1.3.2 Research Hypotheses

This section summarizes the discussion about the contributions in form of hypotheses. They are formulated using the following constellations:

C1 **No knowledge sharing support.** The organization does not have an experience factory. Experience is transferred ad hoc from one colleague to another.

C2 **Knowledge sharing via knowledge brokers.** The organization employs dedicated knowledge brokers whose task is to facilitate experience sharing by suggesting expert colleagues.

C3 **Knowledge sharing via experience factory.** The organization employs an experience factory. In contrast to constellation C2, the objective is to store the experience explicitly in a central experience base. There are two ways of utilizing such an experience base: (a) a dedicated project supporter (a person working for the experience factory) queries the experience base to provide useful information to projects or (b) directly by the project team members.

This dissertation extends the »traditional« notion of the experience factory by a knowledge management component: Knowledge maps [DP98]. Consequently, constellation C3 is extended:

C4 **Knowledge sharing via extended experience factory.** This constellation is based on the assumption that it is not possible to explicate all knowledge. For instance, experts often act intuitively without being able to describe completely why they are acting the way they do. The constellation is defined by constellation C3 where the objective is modified as follows: The objective is to store the experience explicitly in a central experience base *if feasible*. Otherwise (i.e., if the experience cannot be readily documented or the acquisition effort would be prohibitively high) pointers to the experts are stored. In both cases, the experience needs to be annotated (e.g., with the topic of the experience and conditions when the experience can be utilized).

Using these constellations, the following hypotheses are postulated for DISER:

1 **Efficiency hypothesis.** Using DISER for setting up and running the experience bases for constellations C3 and C4, constellations C3 and C4 are superior to constellations C1 and C2 in the following sense: The information seeker[1] finds more useful experience items of a given kind per time period (in terms of both effort and duration) where »usefulness« is defined as the *perceived* usefulness by the information seeker regarding the experience item at the time of retrieval. It is measured on the subjective scale »useful«/»not useful«/»don't know«.

2 **Effectiveness hypothesis.** Using DISER for setting up and running the experience base for constellation C4, the combination of constellations C1 and C4 is superior to constellations C1 or C2 alone in the following sense: The information seeker finds more useful experience items for those kinds of experience items stored in the experience base where »usefulness« is defined as in the efficiency hypothesis.

3 **Applicability hypothesis.** DISER can be applied in industrial-strength projects with a justifiable amount of effort. DISER is considered to be applicable if people are willing to use the experience base on a regular basis. The amount of effort is considered to be justifiable if an organization is willing to pay for the application of DISER.

The hypotheses are discussed below.

1 An information seeker is a project team member that seeks experience for a problem or task at hand.

The first hypothesis expresses an expectation about the productivity gain (from the viewpoint of the information seeker). The hypothesis states that it will require less effort and time to get useful experience items (per experience item on average) from the experience factory than from colleagues. The underlying assumption is that it will require talking to several colleagues to get a satisfactory set of experience items, whereas only one person (at most) needs to be contacted if the experience items are fetched from the experience factory. If the experience base is queried directly, it depends on how easy it is to formulate a query. In case qualitative information is sought after, considerable effort may be necessary to formulate such a query (in comparison to deciding on the usefulness of the retrieved experience items) because often qualitative information can only be characterized meaningfully using keywords which are not confined to a predefined set. This introduces the problems of synonyms and homonyms forcing the information seeker to use a trial-and-error query strategy (further details regarding this issue can be found in Section 5.1). In contrast, quantitative information can be characterized more easily. Hence, it can be expected that the efficiency hypothesis is true for quantitative information if it can be shown for qualitative information.

To understand the effectiveness hypothesis, we have to distinguish between explicit and tacit knowledge. For knowledge that can be represented explicitly in the experience base, it is expected that the experience factory is more effective in terms of the number of useful experience items supplied. The underlying assumption is that the experience items are collected systematically from the projects after the projects have either been completed or reached a milestone. Hence, if an information seeker wanted to get a complete set of experience items, he would have to talk to a colleague from every current and past project. This does not only require a lot of effort, but may become impossible if all team members of a project have left the organization. In addition, aggregated experience (e.g., resulting from cross-project analyses) is only available in constellation C3 or C4 because it is typically not produced in constellation C1 or C2.

For tacit knowledge the situation is different. Tacit knowledge cannot be represented explicitly by its very definition. Therefore, a pointer to an expert is stored in the experience base (constellation C4) instead of the actual experience. Nevertheless, tacit knowledge must be characterized somehow, because it is obviously not enough to just list all employees for identifying the right expert in a given situation. However, it is difficult to characterize tacit knowledge as it is not completely known what it includes. Therefore, the characterization may be too narrow in comparison to the »real« knowledge of the expert. In such a case, constellation C1 could be more effective than C4. With qualitative information, another problem arises: Often it is not clear how detailed the information must be. For the information provider, a concise description suffices. However, if others want to decide on the applicability of the information, they might need additional information which can only be provided by the original author. The complexity of acquiring the complete context information is discussed in [Bir00]. In

addition, the experience fetched from experts is different from experience retrieved from the experience base. As discussed on page 3, expert knowledge is aggregated knowledge which is tailored to the needs of the information seeker. In contrast, the experience base can only provide exemplary experience from past projects if it is filled using the »offer-driven« strategy. Therefore, constellation C4 alone cannot guarantee a better effectiveness. Because of the different knowledge types, it is hypothesized that the combination of C1 and C4 is more effective than C1 or C2.

Finally, the first two hypotheses would not be worth anything if DISER would not be applicable in practice with a justifiable amount of effort. The arguments for this hypothesis have already been listed in the introduction of Section 1.3 (cf. Figure 1). They are based on the practical constraints that have been considered for the development of DISER.

The above hypotheses are validated for the extension of the experience factory concept proposed in this dissertation, that is, for qualitative experience:

- The efficiency hypothesis is validated via an experiment (Chapter 11).
- The effectiveness hypothesis is validated via an experiment (Chapter 11).
- The applicability hypothesis is validated via a case study (Chapter 10).

In the discussion above it has been argued that achieving a better efficiency and effectiveness is more difficult for qualitative experience. For these reasons, a validation of the hypotheses with qualitative experience can also be regarded as a validation of the hypotheses for quantitative experience.

1.4 Structure of Dissertation

DISER is systematically developed and validated (Figure 2 shows the logical flow between the chapters):

Chapter 2 — First, practical constraints regarding software engineering experience (cf. Figure 1 on page 7) are derived.

Chapter 3 — Requirements for a technical infrastructure of an experience base are derived based on these practical constraints, literature on reuse repositories, and the experience of the author.

Chapter 4 — Based on the experience of the author and the literature on managing repositories and information systems, a task framework for populating, utilizing, and improving an experience base is developed. The task descriptions refer to the requirements of the technical infrastructure to show how the tasks can be supported technically.

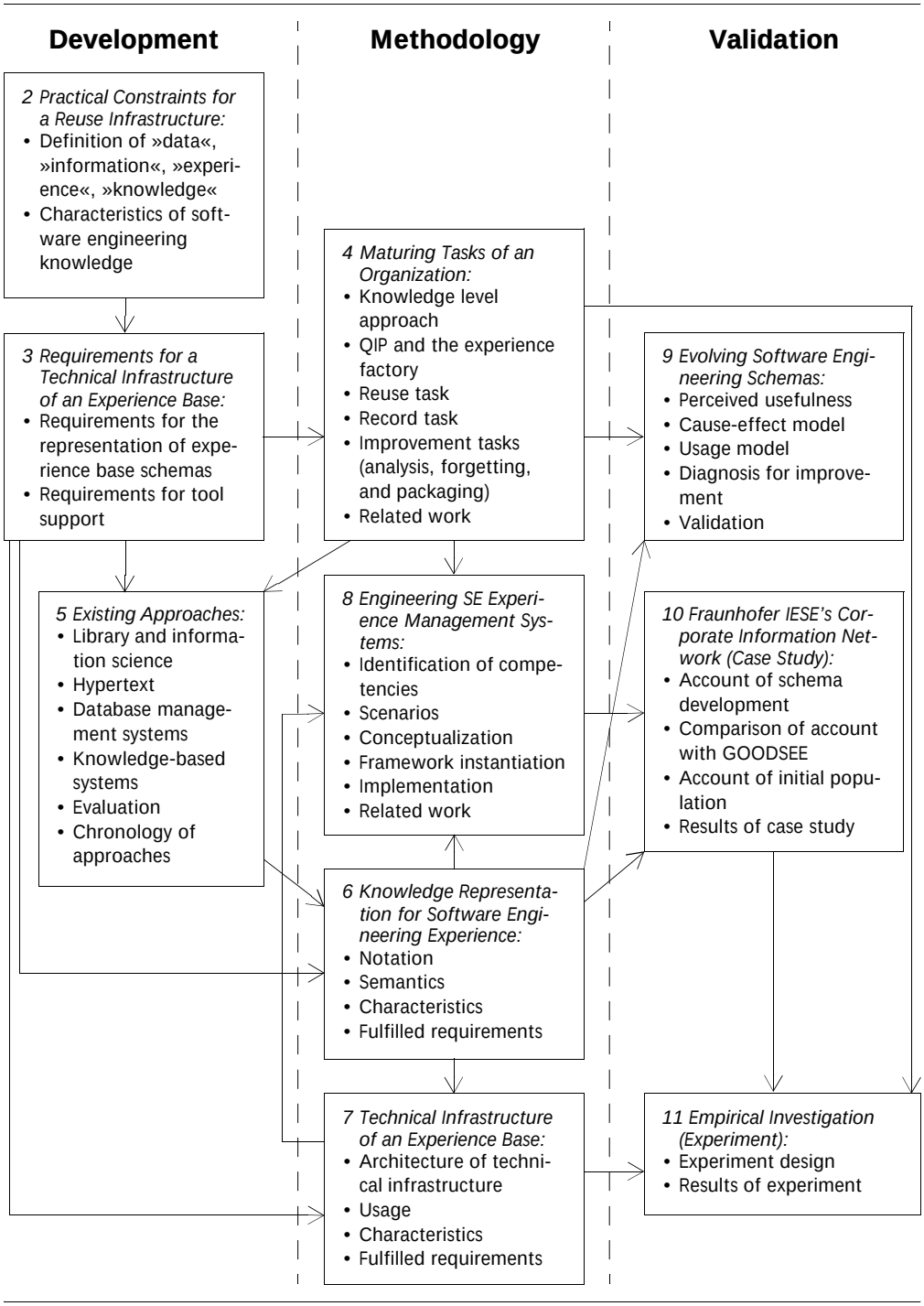


Figure 2: Chapter contents and the logical flow between them

The template used for describing the tasks is systematically derived from Newell's knowledge level approach [New82], which allows an implementation-independent description of the tasks.

The task hierarchy itself is a refinement of MIRACLE (model integrating reuse and case-based reasoning for lots of software engineering experiences) [ABvWT98, TA98], which integrates the QIP [BCR94a], the reuse-oriented software development model [BR91], and the case-based reasoning cycle [AP94].

Chapter 5 Next, existing approaches for software reuse are described and evaluated using the requirements and the task framework. This is done by describing classification and storage techniques (including their advantages and disadvantages) and illustrating these techniques using representative approaches from literature. The techniques are grouped by their origin (library and information science, hypertext, database management systems, and knowledge-based systems).

Chapter 6 As the result of the analysis of the knowledge representation approaches, a new knowledge representation formalism fulfilling the representation requirements of Chapter 3 is developed. The new representation formalism integrates and extends major existing approaches surveyed in Chapter 5.

Chapter 7 The knowledge representation formalism is used to define the schema of an experience base. The schema defines the behavior of the retrieval system of the experience base. The retrieval system is part of the technical infrastructure of an experience base. Therefore, the architecture of a technical infrastructure of an experience base, which fulfills the remaining requirements of Chapter 3, is developed next.

Chapter 8 Next, GOODSEE is developed, which is a method for engineering an experience base schema (based on the representation formalism), customizing the architecture, and instantiating the (task) framework.

Chapter 9 Based on the knowledge representation formalism and using the reuse tasks of the task framework, a method for improving an experience base is developed. The method uses parts of the task framework as a reference for its general usage model. Thus, the validation of this method also validates parts of the task framework. In addition, the method extends DISER by providing the means to improve the technical infrastructure of a running experience base using a cause-effect model for perceived usefulness and a diagnosis process.

Chapter 10 GOODSEE and part of the task framework are validated using a case study.

Chapter 11 Using the experience base resulting from the case study, the overall approach, the remaining part of the task framework, and the architecture is validated in an experiment.

Chapter 12 The dissertation is completed by listing the major results, analyzing how changes in an environment will affect the results, and discussing possible future work.

1.5 How to Read This Dissertation

The structure of the dissertation allows to read the dissertation in various ways for different purposes. This section points out the most common ways to read it.

First of all, each of the Chapters 2 through 11 have three parts: An introduction/ motivation part, a detailed part, and a summary part. It may be helpful for the reader to read the summary before reading the detailed part. This is especially true for chapters 3, 4, and 8 which are written in the style of a reference manual. The summary of these chapters contains a figure with an overview which can be used to guide the reader.

Usually, it is not necessary to read the complete dissertation. The parts to read depend on what the reader wants to accomplish.

To get an overview of the dissertation, it suffices to read the summary at the end of each chapter. Section 12.1 lists the major results of the dissertation.

For a deeper understanding of DISER it is suggested to start with Chapter 2 which defines and discusses fundamental terms which are used throughout the dissertation. The rest is determined by what the reader wants to do:

- **Maintain an existing experience base.** For maintaining an existing experience base, it is important to understand how new experience has to be recorded (Section 4.4.1) and how the technical infrastructure can be improved (Chapter 9). The latter requires basic knowledge of the reuse task (Section 4.3), the knowledge representation formalism (Chapter 6) and the improvement tasks of the task framework (Sections 4.4.2, 4.4.3, and 4.4.4 can be used as a reference).
- **Measure the success of an experience base.** The validation chapters 9, 10, and 11 contain examples for measuring the importance and usefulness of an experience base. For a deeper understanding, it is suggested to take a look at Sections 4.4.2 (analysis tasks) and 8.3 (tasks for defining relevant attributes).
- **Determine and install tools to be used.** The task framework (Chapter 4) describes all tasks that need to be performed. The framework also lists the requirements to be fulfilled for optimal tool support. Using Chapter 3 as a reference manual, the consequences of not fulfilling one of the requirements can be determined. The description of existing approaches (Chapter 5) can be used to determine advantages and disadvantages of the available retrieval systems. Based on this information, adequate tools can be selected. The envisioned technical infrastructure described in Chapter 7 can be used as a reference architecture. It stands for the optimal solution.

- **Set up a new experience base.** The case study (Chapter 10) describes an example for designing and implementing an experience base. For a detailed understanding of the case study, Chapter 8 can be used as a reference manual. Both Chapter 8 and Chapter 10 refer to the knowledge representation formalism (Chapter 6). To understand Chapter 8 in all its details, it will also be necessary to read Section 4.4.1 (record task) and Chapter 7 (technical infrastructure).

2 Practical Constraints for a Reuse Infrastructure

> *The development of a unified and coherent model that defines data, information, and knowledge is far from a straightforward task. Attempts to resolve this issue in the general case, e.g., to answer the question »What is knowledge?«, has been a major problem of philosophers and scientists since ancient times.*
>
> *[AN95, p. 193]*

The notion of *experience* is central to the experience base. It relates closely to those of data, information, and knowledge. In this chapter, these terms and the relationships between them are defined as they will be used throughout the dissertation.

Using these basic terms, the following notions for software-related knowledge (the contents of an experience base) are introduced: artifact, artifact characterization, and experience package. They are fundamental for the technical infrastructure of an experience base (Chapter 3 and Chapter 7) and the tasks associated with the experience base (Chapter 4).

As part of this dissertation, a technical infrastructure based on the characterizations of the artifacts stored in an experience base is presented. For this reason, the notion of artifact characterizations plays a vital role in the context of this dissertation. Therefore, this chapter not only defines the term, but also investigates the characteristics typically associated with characterizations. The results of this investigation are needed to define the requirements of a technical infrastructure for experience bases (subject of the next chapter).

2.1 Data, Information, Experience, and Knowledge

As Aamodt and Nygard point out, »the relation between information and knowledge is a source of much confusion and misunderstanding« [AN95, p. 192]. One might extend this statement to the point that the relations between information, knowledge, and *experience* are subject to confusion and misunderstanding.

In this dissertation, the terms shall be used in a consistent way which requires their precise definition. This is the aim of this section. Figure 3 shows an over-

view of the relations between the terms. Each of the terms is discussed in detail below.

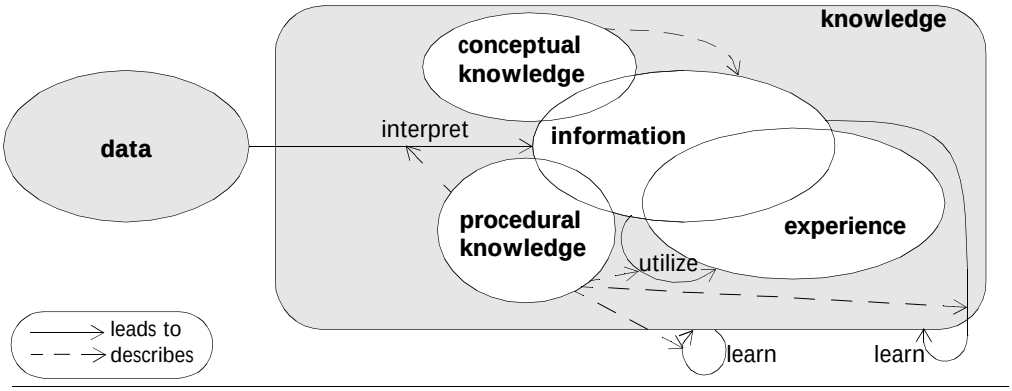


Figure 3: Relations between data, information, experience, and knowledge

2.1.1 Data

Webster's Dictionary defines *data* as »factual information (as measurement or statistics) used as a basis for reasoning, discussion, or calculation.« [Web95] This definition shows that it is not possible to distinguish data from information on a purely representational basis. That is, data and information »look the same« when stored in a computer or printed on paper. And this is true for explicit knowledge and experience, too.

For this dissertation, the perspective of Aamodt and Nygard is adopted, that is, the terms *data*, *information*, *knowledge*, and *experience* are defined by the roles they play in a decision-making process [AN95]. This perspective is the user's perspective when retrieving information from the experience base, because the user has to *decide* which information he wants to utilize in what way.

In the decision-making process, *data* is the starting point.

Definition 3: Data

> Data *are patterns with no meaning; they are input to an interpretation process, i.e., to the initial step of decision making. [AN95, p. 196]*

2.1.2 Information

If *data* is interpreted, it becomes *information*.

> Information *is interpreted data. Information is data with meaning; it is the output from data interpretation as well as the input to, and*

output from, the knowledge-based process of decision-making. [AN95, p. 196]

This puts a strong emphasis on the notion of interpretation, since the interpretation process transforms the syntactic structure of data into a semantic, meaningful entity. This distinction between data and information is also in accordance with ISO norm 2382 which defines information as »the meaning that a human assigns to data by means of the conventions used in their representation«. This definition clearly distinguishes between data (objective), information (subjective), and representation conventions (interpretation) [Çak90]. It stresses that data can be interpreted in different ways, that is, different interpretations of the same data leads to different information.[1]

For the purpose of this dissertation, the ISO definition is extended to also include the interpretation by an intelligent information system. Thus, in context of this dissertation,

Definition 4: Information

Information *is data with meaning that a human or an intelligent information system assigns to this data by means of the conventions used in their representation; it is the output from data interpretation as well as the input to, and output from, the knowledge-based process of decision-making.*

2.1.3 Experience

Experience is defined in Webster's dictionary as »knowledge or practical wisdom gained from what one has observed, encountered, or undergone.« [Ste98] As such it may be tacit or explicit (i.e., documented). This distinguishes experience from information, since the latter is always explicitly documented in form of data. All explicit experience is regarded as information. However, the reverse is not true. For instance, software documentation would be considered as information, but not as experience. Hence, information and experience overlap in Figure 3 on page 22. Another important distinction is that experience is always gained in a *context*, that is, it is context-dependent. This is reflected in the definition by Bullinger et al.: »Experience is knowledge embedded in a particular situation and gained through human sensory perceptions.« [BWP98] Both definitions also imply that experience can only be gained by humans.

1 This is an important aspect for the knowledge management of information when defining roles and tasks and assigning people to them. To mitigate the risk of unintended interpretations, a textual description of the data providers' interpretation (e.g., captured during a feedback session with the data providers) can be attached to the original data.

With respect to the decision-making process, experience is gained while applying the output of a decision-making process (e.g., a selected artifact) and is »concerned with what was true or false, correct or incorrect, good or bad, more or less useful, etc.« [Ric98, p. 6]

Definition 5: Experience

> Experience *is that gained through human senses from directly observing, encountering, or undergoing things during the participation in events or in a particular activity. Its validity is bound to the context of the events or activities.*

As stated above, experience becomes information to the recorder if it is documented explicitly and, for that matter, information to all people and systems that have access to this data and are capable of interpreting it.

When documenting an experience, it is important to also record the context in which it has been gained. Otherwise, it will be useless for others since its validity cannot be decided on, that is, incomplete data will lead to misinterpretation or ignorance. For instance, a particular code component may be reliable for a certain application domain. If the experience of reliability is recorded without the application domain, the code component may be used in application domains for which it has not been validated (optimistic approach) or ignored because its reliability is not ensured (pessimistic approach). In the latter case, the component may be ignored even if the application domain of the current project is the same as the one in which the experience has been originally gained. Consequently, the quality of the context description determines the fidelity of experience transfer from the person gaining the experience to all persons utilizing the experience.

2.1.4 Knowledge

According to the Webster's Dictionary, knowledge is »the range of one's information or understanding[1].« [Web95] This definition implies two things:

1 Given Definition 4, the knowledge is made up only of meaningful entities.

2 Knowledge is subjective. The range of information or understanding differs from individual to individual.

For the purpose of this dissertation, the definition is extended to include intelligent information systems (cf. Definition 4):

1 Here, *understanding* refers to »familiarity with a particular thing; skill in dealing with or handling something, e.g., ›an understanding of accounting practice‹« [Ste98].

Definition 6: Knowledge

Knowledge *is the range of learned information or understanding of a human or an intelligent information system.*

Procedural knowledge

Knowledge is not only made up of experience. It also includes *know-how* (labelled *procedural knowledge* in Figure 3 on page 22) that describes how to interpret data, how to utilize information to gain experience and/or more information, and how to learn in general, that is, how to acquire additional knowledge from existing information and other kinds of knowledge.[1]

Just as with experience, procedural knowledge, and for that matter, knowledge in general does not have to be represented explicitly (referred to as *understanding* in Definition 6). Usually, there is no explicit documentation of the complete knowledge. In this dissertation, all explicitly documented knowledge is referred to as *information*. Therefore, information is shown as a subset of knowledge in Figure 3 on page 22.

Note that the term *knowledge* is used here in a very general sense. It does not distinguish between »true« and »believed« knowledge. This is different from the influential branch of philosophy [AN95] and how the term is often used in the community of knowledge representation where the term *knowledge* is used exclusively for statements that are true in the world at large, and where the term *belief* is used if truth cannot be ascertained (e.g., [Rei91]). However, Definition 6 corresponds to the way the term is frequently used in *knowledge management* (e.g., [BWP98, Wii95]).

Conceptual knowledge

Knowledge can be categorized into many classes. We have already encountered procedural and experiential knowledge (experience). Another category of knowledge is *conceptual knowledge*, that is, if the structure of data or information is known, it is considered as knowledge by itself. Although conceptual knowledge about data helps to interpret these data, it is not a prerequisite to do so. For instance, one can understand the meaning of a requirements document without recognizing the underlying structure used to represent the requirements.

Conceptual information

Conceptual knowledge does not have to be represented explicitly. If it is, however, it becomes information (hence, conceptual knowledge and information overlap in Figure 3 on page 22) and is often referred to as *conceptual information*. It can be documented using concepts, their intensions, and rule-based dependencies. Each of these terms is defined below. In succeeding chapters, these terms will be refined as necessary.

1 In this dissertation, the term »procedural knowledge« is used in the narrow sense as defined here. In its general sense it would also include all kinds of procedures, especially software development processes.

Definition 7: Concept/instance

A concept *is a 3-tuple (concept name, extension, intension). The* extension *is the set of all* instances *that belong to the concept whereas the* intension *specifies the characteristics an instance must exhibit to belong to the concept.*

Definition 8: Intension

The intension *(characteristics of a concept) can consist of [Rei91]:*

- properties *that each instance of the concept has (e.g., all instances are red),*
- property classes *that are defined for all instances of the concept (e.g., all instances have a color),*
- semantic relationships to other concepts *that each instance of the concept has to have (e.g., for each tree there is a »has-parts« relationship to a stem), and*
- semantic relationships to instances *that each instance of the concept has (e.g., all instances of a concept »product« are produced by the same manufacturer M, an instance of the concept »manufacturer«)*

Definition 9: Rule-based dependencies

Knowledge of the kind »if statement A is true, then statement B is also true« is defined as knowledge about rule-based dependencies. [Rei91]

An example for a rule-based dependency is the following: »if the earth stands between the moon and the sun, then there is a moon eclipse«.

2.2 Characteristics of SE Knowledge

Up to this point, all terms have been defined in a general sense without taking into account the particularities of software engineering knowledge. This section explores these particularities by introducing the notions of artifact and experience package, relating these to the terms defined in the section above, listing property classes for software engineering knowledge, and specifying the classes' value ranges. In this way, the section characterizes software engineering knowledge that must be taken into account by any viable technical infrastructure for an experience base.

2.2.1 Artifacts and Experience Packages

The experience base is defined as the software-related knowledge (including meta knowledge) that may be of use in future projects [BR91, BCR94a]. But what does »software-related« really mean? If the experience factory is to support technical as well as *managerial* tasks of a software development project, we soon come to resource allocation. This raises the question for available personnel and their skills. To answer this question, project allocations, vacation and

travel plans, and the »CV« of people need to be stored in the experience base. Organization charts become references for the context information of experience. In short: information about the organization cannot be neglected even though such information is not software-specific. It can also be used for projects other than software development projects. For that matter, an experience base can be of value for *any* organization, not just for software development organizations.

Furthermore, some organizations may require to store software-related and other technical information in an integrated repository. This is especially true for embedded systems where hardware and software development go hand in hand. If hardware-related and software-related knowledge would be stored separately, it would be hard to document the relationships between them.

As can be seen, an experience base covers (at least potentially) a very large knowledge area. Consequently, two practical constraints must be met:

1 **Integration of existing information.** When a new experience base is set up, already existing information (e.g., in form of files or printed documents) must be integrated.

2 **Processing of knowledge in different ways.** The information stored in the experience base is utilized in different ways depending on the task to be achieved. This may require specialized tools that operate on a defined subset of the information. Often, some of these specialized tools already exist in the organization when an experience base is introduced and shall be continued to be used.

Therefore, the knowledge should be structured in a way that specialized tools can be utilized for defined subsets of the knowledge. A piece of information that can be processed by a specialized tool will be referred to as an *artifact*:

Definition 10: Artifact

> *An* artifact *is a piece of information that can be processed by a machine or a tool specialized for it. Artifacts are not restricted to documents.*

Examples for artifacts are software work products such as requirements documents, design documents, code components, test cases, but also executables, process models, quality models, project schedules, personnel allocation plans, organization charts, and documented experience (e.g., in form of lessons learned).

The term *artifact* is preferred here over the term *experience model* as defined in [BCR94a]. The latter term can lead to confusion, because the experience base does not only contain experience, but also other types of knowledge. For instance, a design document is not an experience according to Definition 5.

Artifact characterization

Artifacts are often of a proprietary format (i.e., the native format of the specialized tool it can be processed with). In particular, they are not extensible in the sense that additional information may be stored as part of the artifact. However, this represents a barrier for reuse, because usually additional information is necessary to make an artifact reusable (e.g., [BR91, EMT98]). Although logically part of the artifact, the additional information has to be stored separately. From now on, this additional information will be referred to as the *artifact's characterization*. Ideally, the characterization contains all information necessary to make the decision on whether to utilize the characterized artifact or not. However, in practice it may become necessary to view the artifact itself to make the final decision, because not all information that is needed for making the decision may be known at the time the artifact is stored.

The logical unit of artifact and characterization will be referred to as an *experience package* [BCR94a]:

Definition 11: Experience package

> *An* experience package *consists of an artifact and its characterization. The* characterization *contains necessary information to make the decision on whether to utilize the artifact.*

A few comments on artifacts and their characterizations follow to clarify the relationship between them (see Figure 4):

- **An artifact's characterization may consist of several instances of different concepts.**[1] Thus, there is a 1:n relationship between an artifact and its characterization's instances. For example, a process model can be characterized as a whole using a single concept (special case of a 1:1 relationship). But it may also be characterized in more detail, e.g., each of its parts (product models, activities, etc.) can be represented by its own concept. Then, a process model is characterized using one instance (of the concept »process model«) and as many instances (of the concepts »product model«, »activity«, etc.) for its constituents as needed. The instances will be connected using semantic relationships, and all instances together make up the characterization of the artifact.
- **The contents of an artifact and its characterization may overlap.** As stated above, artifact and characterization are separated for pragmatic reasons. The characterization shall contain all information necessary to make the decision on utilizing the artifact. This information may (at least partially) already be part of the artifact. For example, if a process model is characterized in detail, the structure of its characterization's instances may reflect the structure of the process model's constituents. Clearly, redundancy should be

1 From an object-oriented modeling point of view, a concept corresponds to a class definition.

avoided wherever possible or – where it cannot be avoided – an automatic transformation/synchronization should be put into place.
In the extreme case, artifact and characterization overlap completely. Then, the artifact can be reconstructed from its characterization. The artifact does not need to be stored separately, but can be constructed »on the fly«.

- **Artifact characterizations may overlap.** For instance, two artifacts each describing an experience, may share the same context. Therefore, the characterizations of the two artifacts overlap regarding the context information, that is, they share some instances.
- **»Artifact« and »experience« are not synonyms.** For example, a code component would typically not be considered as an experience. However, it may contain some experience. For instance, the programmer may include a comment like »the following code runs really slow on machine X«. This comment would typically be regarded as experience, because its author probably observed the documented fact (using his senses) and the comment has a restricted validity (i.e., it is valid only in the event that the code is run on machine X). Other artifacts (e.g., reports on case studies) document pure experience.
- **»Artifact characterization« and »experience« are not synonyms.** Characterizations typically consist of some experience and other information. For example, a code component may be characterized with the properties of »cyclomatic complexity« and »reliability« (among others). While the former would not be considered as an experience (the complexity will not change if the component is used in another domain), the latter would be (the reliability must have been proved through some »experiment« and is only valid for the environment in which the component has been tested).

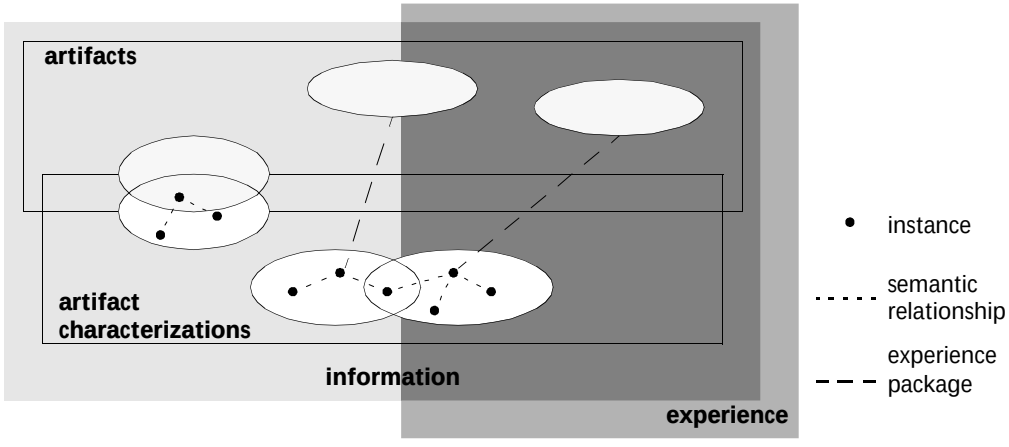


Figure 4: Relations between artifacts, artifact characterizations, and experience

2.2.2 Dimensions of Knowledge

Characterization information

In the previous subsection, the notion of artifact characterizations was introduced. The set of all artifact characterizations will be referred to as the *characterization information of the experience base* or simply as the *characterization information*. In the following, the nature of the characterization information is investigated further.

In general, information can be categorized in several dimensions. Reimer lists the following [Rei91]: Completeness, certainty, precision, and consistency. Other dimensions include formality, stability, level of abstraction, and context-dependency. Each of the dimensions will be described in the following.

Completeness

Completeness is the degree to which the characterization information actually contains all information needed to make the decision on utilizing an artifact.[1] If the conceptual knowledge, upon which the characterization information is based, is explicitly documented, one can distinguish between two cases:

1 The needed information is not supplied as part of the characterization although it is part of the conceptual knowledge (i.e., either there are semantic relationships to other concepts or there is a property class in the intension of the corresponding concept). In this case it is known that the artifact has this characteristic, but its concrete value is unknown. For instance, the cyclomatic complexity of a Pascal code component can always be computed. The absence of a concrete value as part of the component's characterization does not mean that the code does not have a complexity number.

2 The needed information is not documented as part of the conceptual knowledge. Here, it is not even known that the information is missing. In practice, this occurs frequently. Additional information needed may be identified through a decision-making process.

Certainty

Certainty is the probability that a given information is actually true.[2] For instance, before the start of a project, cost and duration of the project can only

1 Of course, information stored is never complete in the sense that only a portion of the knowledge existing in the world at large is explicitly represented. Therefore, the term »completeness« refers to what *shall* be covered by the experience base. This is determined by the purpose(s) for which the experience base is to be used.

2 Note that experience by its definition is uncertain for all contexts for which it has not been validated.

be estimated. If cost and duration are part of the project's characterization, they cannot be specified with a probability of 100%. However, as the project progresses, cost and duration become more and more certain.

Precision

Given information can be vague or imprecise. For instance, a project's duration may be specified to lie in the range of 1 to 24 months. Although this information is imprecise – the actual duration will lie within this interval –, it is 100% certain.

Information may be both uncertain and imprecise at the same time. For example, a project´s duration may be estimated to lie in the range of 10 to 12 months. Although this information is still imprecise (no exact value is given), it is also uncertain because the project´s duration may not lie within the given interval.

Often, there is a trade-off between precision and certainty. The more precise information is specified, the more uncertain it is. This means that uncertain information can be »made« certain by specifying it imprecisely. For example, assume that a project´s team size varied from 1 to 9 during the course of the project. If the team size is specified with the mean (i.e., 5), the information is uncertain, because the experience item that refers to the project may have been captured when the project´s team size was not 5. The value [2..8] is more certain, but also imprecise. Finally, the value [1..9] is 100% certain, but even more imprecise than [2..8].[1]

In contrast to incomplete information, imprecise information restricts the value of a characteristic whereas incomplete information does not give a value at all. Therefore, if the value range of a concept characteristic is known, the specification of all possible values for this particular characteristic has the same meaning as if no value would have been specified (= incomplete information). Hence, incomplete information can be interpreted as a special case of imprecise – but 100% certain – information (in case the value range is known).

1 The example assumes that only one project characterization is used for each project. Alternatively, a different project characterization could be used for every experience gained in the project. Although this strategy would allow the recording of more precise and certain characterization information, the strategy would also require additional characterization effort. Note, however, that a given experience may not refer to a specific point in time, but to a time span. For example, a communication problem may have been discovered during the enlargement of the team from 2 to 9 people during the coding phase. If this experience is to be recorded, it cannot be characterized with a value for the team size that is both certain and precise at the same time. Still, in this case the characterization information [2..9] is more precise than the value [1..9] of the example above.

Consistency

Consistency is the degree to which the characterization information is free of contradictions. In this dissertation, three types of inconsistencies will be distinguished:

1 **Inconsistent property values.** For instance, assume that two experiences were gained in the same project. If the context (in this case, the project) is characterized differently for each experience (e.g., once the duration is specified with 6 months and once with 7 months), then there is a contradiction, because the duration of the same project cannot be 6 and 7 months at the same time. This kind of inconsistency can be avoided by sharing characterization information (in the example above the project information) among several artifact characterizations (»overlapping artifact characterizations«). Another example concerns the dependencies among characteristics. For example, if a measurement goal is specified by the measurement object »development process« and the quality focus »cyclomatic complexity«, then there is a contradiction, because »cyclomatic complexity« is a product measure and not a process measure. This kind of inconsistency can be avoided by specifying such dependencies formally and checking characterization information automatically based on the specified dependencies.

2 **Inconsistency between an artifact and its characterization.** For instance, assume that a code module has 1.010 lines of code (LoC). If it is characterized with 1.000 LoC, then there is a contradiction, because the actual value does not correspond to the recorded value. This kind of inconsistency should be avoided. However, if precise characterization information is supplied by a human it may be difficult to ensure this kind of consistency. If the characterization information is generated by a tool, the correct implementation of the tool will yield consistent artifact characterizations.

3 **Inconsistencies among artifacts.** In a development project, artifacts typically depend on other artifacts. For example, a design document depends on a requirements document in the sense that the design document may have to be changed if the requirements document changes. In practice, it may be advantageous to document the semantic relationship between a requirements document and a design document even if they are inconsistent. To see this, assume that two consistent documents – a requirements document and a design document – are stored in the experience base and that the dependency between the documents is recorded as a semantic relationship. Assume further that the requirements document is changed and no longer consistent with the design document. If the design document is retrieved later, the changed requirements document may be more useful to the user than the old requirements document because ultimately, the system must fulfill the new requirements. Therefore, it is useful to record the logical dependency between the documents although they are physically inconsis-

tent.
This example shows that the software development practice requires the recording of inconsistent artifacts.

For the rest of this dissertation, the term *inconsistency* is used for the last type of inconsistency unless stated otherwise.

Formality

Information can be represented in the range of informal (e.g., as text) to formal (e.g., as mathematical formulas). Generally, the more formal the representation, the higher the observation/acquisition effort for the information, but at the same time analysis and decision-making support can be automated to a higher degree [Ste90]. Characterization knowledge can take on the whole range. For instance, names are typically represented as text whereas measurement data such as cyclomatic complexity can be represented in a formal way (in this case as integer).

Stability

Stability is the time until information becomes inaccurate, useless, or obsolete. Examples are:

- If a project characterization is entered upon project start by providing information and estimates available at that time, the original estimates for cost and duration should be replaced by the actual values in the experience base when the project has been completed. If the information is not updated it may become useless, because analyses based on this project characterization may come up with wrong conclusions.
- If a software development organization decides to no longer program in Fortran, all code components written in Fortran become useless.
- Experiences may be packaged in »higher-order artifacts«, making them obsolete [ABT98].

Level of Abstraction

Information may be more or less detailed (see discussion on number of instances to characterize a single artifact on page 28).

Context-Dependency

Some information (especially experience) cannot be utilized without knowledge of the context for which it has been validated.

2.2.3 Properties of Characterization Information

In this subsection, the characterization information of an experience base is classified using the knowledge dimensions presented in the previous subsection.

There are three situations in which characterization information is accessed: when new information is recorded, when already stored information is updated, and when existing information is retrieved. In the following, it is assumed that queries and artifact characterizations exhibit the same intensions. This means that queries are formulated using the same properties, property classes, and semantic relationships which the artifact characterizations have (as known from database systems [Dit97, GR95]).

The characterization information used for querying is different from the characterization information stored in the experience base, because query information has a short lifetime in comparison to characterization information stored in the experience base. Moreover, query information can be changed by the user during a retrieval process on an as-needed basis, whereas stored characterization information cannot. Therefore, both »kinds« of characterization information must be considered. The result is a classification of *query information* (characterization information available for retrieval) and *artifact information* (characterization information available for recording and updating).

Query information

Query information may be:

- **Incomplete** (either information is unknown or information is not supplied, because it is rated as unimportant or of minor importance regarding the retrieval goal)
- **Uncertain** (for instance, at the beginning of a project, the project characteristics can only be estimated)
- **Imprecise** (instead of uncertain information, imprecise information may be given)
- **Inconsistent** (for instance, two contradictory values are specified for property classes that are not independent of each other, e.g., if a product quality measure like »lines of code« is specified for a process model as part of a quality model query)[1]
- **Formal or informal** (depending on the degree of formality with which the information is stored in the experience base)
- **More or less abstract** (depending on the degree of detail that is used for storing information in the experience base)

1 Because query information is not associated with an artifact, inconsistency between an artifact and its characterization as well as inconsistencies among artifacts are not applicable.

Query information is always **instable**[1] (the query information needs to be retained only for the duration of retrieval), and **context-dependent** (every retrieval is performed for some purpose in a special environment).

Artifact information

Artifact information may be:

- **Incomplete** (either information is indefinitely or temporarily unknown; it is temporarily unknown if it is currently unknown, but efforts are under way to come up with the information)
- **Uncertain** (e.g., for long-running projects, an initial project characterization may be recorded at the beginning of the project; also experience is by its very nature uncertain)
- **Imprecise** (as a substitute for uncertain information)
- **inconsistent** (artifact characterizations are recorded over time; hence, it is likely that inconsistencies are introduced)
- **Formal or informal** (depending on the degree of automatic analysis required)
- **Stable or instable** (documentation of best practices has a longer lifetime than conclusions from empirical studies (in form of hypotheses) [ABT98], because relevance and accuracy of unvalidated conclusions are usually not known beforehand)
- **More or less abstract** (usually depending on the available tool support)

Artifact information should always be stored with the context of the artifact (i.e., the context in which the artifact has been created or utilized, and the context it is applicable for) as to allow for later analysis. In other words, artifact information is always (or at least should be) **context-dependent**.

The »profiles« of the characterization information presented above is inherent to software engineering knowledge. A decision-making process that does not take these characteristics into account, may provide suboptimal solutions.

2.3 Summary

The notion of *experience* is fundamental to the experience base. It is closely related to the notions of *data*, *information*, and *knowledge*. In this chapter, these terms have been defined. *Data* are patterns without meaning. In contrast, *information* is interpreted data, that is, data with meaning. The interpretation step requires *procedural knowledge*. If both recorder and user have the same

1 As will be shown in a later chapter, query information may also be retained for analyzing and improving the quality of the technical infrastructure. But even then the information is fairly instable, because usage habits change over time and once the technical infrastructure has been changed, query information becomes inaccurate.

procedural knowledge and share some data, they will have the same information. While information is always documented explicitly in the form of data, experience may not. *Experience* is gained by humans through observations performed in specific contexts. Experience is only valid for those contexts in which the observation has been made. Finally, *knowledge* is the range of learned information or understanding of a human or an intelligent information system. It may be tacit or explicit. A special kind of knowledge which describes the structure of information is referred to as *conceptual knowledge*.

Software-related knowledge (contents of an experience base) covers a broad spectrum. Therefore, specialized tools are needed to process portions of it. A piece of information that can be processed by a specialized tool is called an *artifact*. Information needed to make the decision on utilizing the artifact is stored separately as the *artifact's characterization*. Both the artifact and its characterization represent a logical unit called the *experience package*. The separation of an experience package into artifact and its characterizations is done for practical reasons only – it represents a practical constraint. Theoretically, an experience package can be stored physically as one item. But this is not feasible in practice, because (a) artifacts already exist (some only on paper) which are to be managed by the experience factory and (b) specialized tools usually require the artifacts to be of proprietary format. Thus, artifacts in general cannot be enriched by additional information.

To support the decision-making process on the utilization of an artifact, the nature of the artifact's characterizations has been further investigated in this chapter (see Figure 5). The results of this investigation will be input in form of practical constraints for the requirements of a technical infrastructure for an experience base. These requirements are subject of the next chapter.

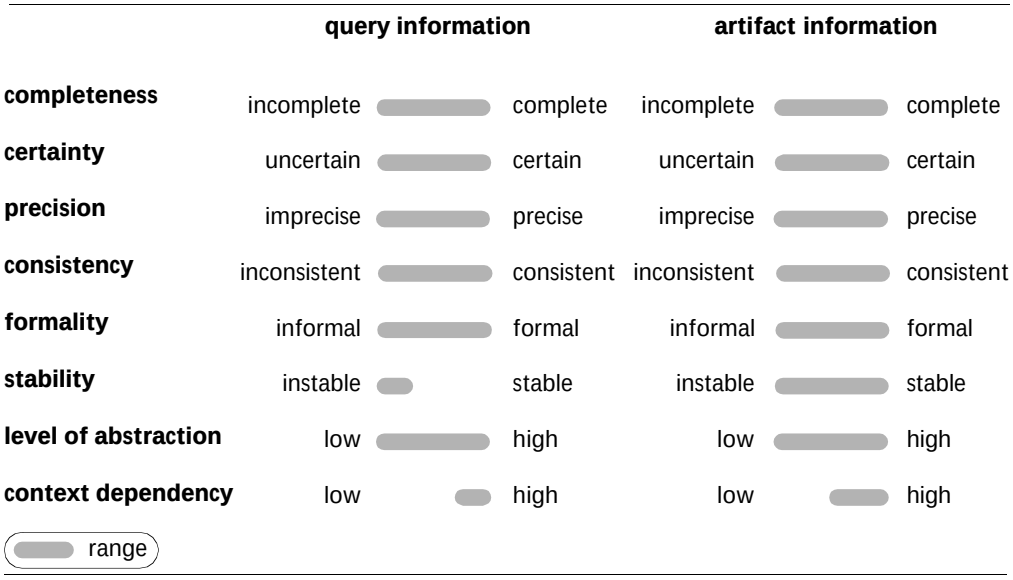


Figure 5: »Profiles« of query and artifact information

3 Requirements for a Technical Infrastructure of an Experience Base

Reuse is a simple concept: use the same thing more than once. But as far as software is concerned, it is sometimes difficult to define what is an object by itself, in isolation from its context.

[BCC92, p. 54]

In the previous chapter, the information to be stored in the experience base has been classified. However, the needed functionality of a technical infrastructure for an experience base has not been discussed yet. This is the aim of this chapter. Before the requirements are listed, the term *technical infrastructure* shall be defined.

Definition 12: Technical infrastructure of an experience base

> *The* technical infrastructure *of an experience base consists of all software systems needed to store, retrieve, evaluate, maintain, disseminate, and utilize the information of this experience base.*

The requirements have been elicited through years of experience gained in experience factory related industrial-strength and research projects. In these projects, experience bases were created for health care centers, software experience centers, software inspections, software tools, and lessons learned [ABH+99, BKP94, ABT98]. Also, the author has gained some experience in setting up organizational information systems [Ste95, TBF95].

Requirements description

Although a technical infrastructure cannot be defined in detail independently of the organizational processes that utilize the infrastructure, some generic requirements can be stated by surveying published requirements for software engineering repositories. These requirements are listed below. Each requirement is described using the following information:

- **Requirements number.** Each requirement has a unique number for reference purposes.
- **Requirements identification.** Each requirement has a unique title that reminds the reader of its contents.
- **Requirements statement.** The statement specifies the requirement. If the requirement (or a similar one) has been identified explicitly in the literature, the source is also acknowledged in the requirements statement.
- **Prerequisites.** If a requirement can only be fulfilled if some other requirements are fulfilled, the other requirements are listed here.
- **Explanation.** The explanation comments on the requirements statement. It usually motivates the requirements statement.

- **Example.** The (optional) example shows what the requirement is »good« for.
- **Consequences if not fulfilled.** Not all of the requirements listed below must be implemented for the technical infrastructure of an experience base. The set of requirements to be fulfilled largely depends on the organizational context in which the technical infrastructure will be used. However, if some requirements are not implemented, it will have consequences regarding basic functionality of the technical infrastructure. For instance, the performance of some organizational task may become more difficult or require more effort. Through the explicit documentation of the consequences, the limitations of existing approaches (see Chapter 5) can be determined.
- **Refinements.** Some requirements are defined on an abstract level at first. These requirements are refined later. They are fulfilled if their refinements are fulfilled. In the end, all requirements that are not further refined must be implemented.

3.1 General Requirements

In this section, all requirements are listed that pertain to all instantiation, maintenance, and operation functionality of the technical infrastructure.

Req. R1: ease of use

The technical infrastructure must be easy to use by both the administrators of the experience base and the end users. [PD91]

Prerequisites: none

Explanation: With all other requirements fulfilled, the structure of the characterization information of the experience base will be very complex. This requires an intelligent assistant that aids the usage [GABT98].

Recently, web technology has been identified as an easy-to-use technology for experience bases [Con98].

Consequences if not fulfilled: People will not accept the infrastructure. Obviously, this is a mandatory requirement.

Refinement: none

Req. R2: support for incremental, continuous learning

Incremental, continuous learning must be supported.

Prerequisites: none

Explanation: Experience can only be gained through utilizing information (see Section 2.1, Definition 5). If experience is to be reused effectively, it has to be maintained continuously and disseminated. Also – since experience can have a short lifetime –, functionality must be provided for »forgetting«.

Consequences if not fulfilled: Artifacts cannot be stored in the experience base one at a time, but only in clusters or – even worse – the experience base may need to be redesigned. This means that the experience base will be updated seldom; the organization as a whole will learn at a slow pace.

Refinement: This requirement is fulfilled if the requirements R14, R40, and one or more of R32, R33, R34, and R39 are fulfilled. If the requirement is extended to include the continuous improvement of the technical infrastructure as well, then the requirements R11 and R15 must also be fulfilled.

Req. R3: tool integration

The technical infrastructure must integrate existing tools. [ABT98, ABH+99, MR87]

Prerequisites: none

Explanation: The integrated tools are the specialized tools needed to create and change the artifacts stored in the experience base. The integration of these tools (to the degree that the repository can invoke the tool with the chosen artifact as argument) ensures the transparency for the user (so he does not need to know which kind of artifact can/must be edited with which tool).

Seen from the other side, specialized tools should be able to query the experience base for relevant information, but also to store detailed characterization information that can be exported from the artifact. The automatic creation of characterization information was discussed in Section 2.2.

Example:

- A requirements document is edited or created from a template using a text editor.
- A project schedule is edited using a project planning program.

- To realize reuse for process modeling on a low level of abstraction, a process modeling tool needs to query the experience base for process model constituents (e.g., for activities, product models, and resource models).

Consequences if not fulfilled:

1 Tool invocations will not be transparent to the user. He has to know when to use which tool for what purpose.
2 Characterizations on a low level of abstraction (i.e., below »artifact level«) will be practically impossible, because the manual characterization will be effort-intensive and error-prone.

Refinement: none

Req. R4: administrative information

The conceptual information must include administrative information such as owner, author(s)[1], and creation date of the artifact. [BR91, EMT98]

Prerequisites: none

Explanation: Sometimes questions regarding an artifact arise during its utilization. In such a case, it should be possible to contact the owner or one of the authors of the artifact.

Example: During the tailoring of a retrieved code component, the programmer does not understand why a certain algorithm has been chosen. The consultation of the experience base shows that this design decision is not recorded. However, through the administrative information the original programmer can be identified and asked.

Consequences if not fulfilled: Finding the author will take more effort; goal-oriented active dissemination of new experience cannot be supported technically, that is, as new experience becomes available, it cannot be disseminated to people interested in it (cf. R30 (change notification) on page 59).

Refinement: none

1 The *owner* has the responsibility for the artifact. He decides who has access to the artifact, when it is to be changed, and when it is to be removed from the experience base. An *author* is someone who created or modified the artifact in the past.

Req. R5: access rights

It must be possible to restrict access rights for certain users or user groups.

Prerequisites: R4

Explanation: Not all information in an organization is public. Therefore, it must be possible to restrict the access to both, the characterization information and artifacts.

Example: If effort data is stored in the experience base, the individual figures should be visible only for the person entering the data and the data analysis team. All others should only be able to see the information in an aggregated form (anonymous data handling is necessary for the success of measurement programs).

Consequences if not fulfilled: Everybody will have access to everything. In case of measurement programs this means that people will be reluctant to provide data, which will cause the measurement programs to fail.

Refinement: none

Req. R6: network access

It must be possible to access the experience base from a network. [EMT98]

Prerequisites: none

Explanation: In many organizations the experience base (at least parts of it) is accessed by project teams. This means that many people have access to the experience base. Also, more and more projects (and experience factories for that matter) operate in distributed environments. These two reasons require remote access to the experience base.

Consequences if not fulfilled: Only a small group of people working together locally can have access to the experience base.

Refinement: none

3.2 Instantiation and Maintenance of Technical Infrastructure

In this section, all requirements are listed that pertain to the tailoring of the conceptual knowledge underlying the characterization information.

Req. R7: storage of various kinds of artifacts

The technical infrastructure must be able to store all kinds of experience packages. [ABT98, GABT98]

Prerequisites: none

Explanation: To provide effective support for software development and improvement, various kinds of artifacts need to be stored and characterized. Each kind of artifact must be characterized using different characteristics.

As implied by Definition 10 on page 27 an artifact may be of a proprietary format. It may be physically stored by the specialized tool in various ways, e.g., as a file or database. This has to be considered when storing the artifact in the experience base.

Example: Examples for artifacts include:

- Process models (e.g., for Cleanroom and Ada waterfall model) [ABT98, BR91, Bas95]
- Product models such as forms and templates [ABT98, BR91]
- Method and technique models and evaluations (e.g., best method for finding interface faults) [Bas95]
- Software best practices [ABT98, Hen97a]
- Design method tools [OB92]
- Technologies [Bir00, FT98]
- Measurement plans [GvWB98]
- Data [Vis94]
- Quality models such as reliability models, defect slippage models, ease of change models [ABT98, BR91, Bas95]
- Software work products such as requirements documents, software designs, code components (e.g., Ada generics for simulation of satellite orbits), test cases and procedures, change requests, etc. [ABT98, Bas95]
- Change and defect models and baselines such as defect prediction models [Bas95]
- Resource models and baselines such as local cost models and resource allocation models [ABT98, Bas95]
- Project information such as project plans and schedules
- Organizational information such as business goals and organization charts [GvWB98]
- Lessons learned about the above artifacts such as problem-solution statements, guidelines, and improvement suggestions (e.g., risks associated with an Ada development) [ABT98, Bas95, BT98a, Sar95]
- Observations (results and success reports) [ABT98]

Consequences if not fulfilled: The technical infrastructure will be capable of storing only a predefined set of artifact kinds. This means that expanding the infrastructure to accommodate unanticipated kinds of artifacts and transfer-

ring the infrastructure to another organization (or organizational unit for that matter) will be difficult.

Refinement: none

Req. R8: explicit representation of the conceptual knowledge

The conceptual knowledge underlying the artifact characterizations must be represented explicitly. [GABT98]

Prerequisites: none

Explanation: All information systems have conceptual knowledge. This knowledge may be documented implicitly (e.g., through the forms presented to the user) or explicitly. In the latter case it can be documented informally (e.g., as text) or formally (in the sense that a program can interpret it). »Explicit representation« means formally documented. Only if the conceptual knowledge is formally documented, it can be interpreted by an information system, and thus, has a value for the system. It becomes *conceptual information* for the system (cf. discussion on page 25).

Meta knowledge

The fulfillment of this requirement leads to a three-layered knowledge representation. At the most abstract level – the *meta knowledge level* –, the representation formalism used to represent the conceptual knowledge is described. The conceptual knowledge in turn describes how the characterization information is represented [ABT98, ABH+99]. As Table 1 shows, this three-parted representation is common practice.

Table 1: Three-layered representation

Level	Experience base	UML [Rat97]	Database management systems	Process modeling	Domain analysis
3	Characterization information	Instances	Data	Processes/process traces	Systems
2	Conceptual knowledge	Class diagrams	Schema, data model	Process model	Domain model
1	Meta knowledge	Meta model	Data description language	Process modeling language	Domain modeling language

Note the last two columns: process traces, process models, descriptions of the modeling languages, systems, and domain models may all be artifacts of an experience base (and thus be characterized as part of the characterization information). The conceptual knowledge must allow to characterize each of these artifact kinds (requirement R7) as well as the relationships between the artifacts (requirements R18, R38, and R41). For example, to reuse a process

model, the question »which process modeling language has been used to create the particular process model?« must be answered.

Example: Explicitly represented conceptual knowledge can be used (among others) for user guidance (both for entering and retrieving characterization information), as means of communication among people (to discuss the adequacy of structure underlying the characterization information), and as a communication vehicle for tools (generic tools can retrieve the conceptual information, and thus do not need to be adapted if the structure underlying the characterization information is changed).

Consequences if not fulfilled: There are two fallback positions to this requirement:

1 The conceptual knowledge is documented explicitly, but informally. In this case, the document can still be used as a means of communication among people. However, inconsistencies can be introduced between the documented conceptual knowledge and the implicit conceptual knowledge used by the information system. Also user guidance and tool communication will not be possible.
2 The conceptual knowledge is not documented at all. In this case, the conceptual knowledge cannot even be used as a means of communication.

Refinement: none

Req. R9: separation of characterization and conceptual information

Conceptual information must be treated differently than characterization information. [ABH+99, TG98b]

Prerequisites: R8

Explanation: The recording of new artifacts or the update of an artifact's characterization affect the behavior of the retrieval, but they do not require the end users to think in new terms when using the system. On the contrary, changes to the conceptual information are much more critical. Therefore, the circle of people who is allowed to change the conceptual information should be restricted.

This requirement is further substantiated by R13.

Example: If a new property class is added to a concept, the user must understand its meaning. If a concept is split up into two (because the information captured in the second part is shared by the characterizations of other artifact kinds), the user may be confused the next time he uses the system.

Consequences if not fulfilled: If everybody who is allowed to record an artifact, is also allowed to change the conceptual information, then there will not be a common rationale underlying the experience base.

Refinement: none

Req. R10: rationalized conceptual information

The conceptual information must be rationalized. [BR91]

Prerequisites: R9

Explanation: To transfer a technical infrastructure of an experience base to another organization, we need to tailor the conceptual information to the specific needs of the organization in a systematic way. This not only requires the ability to change the conceptual information, but also some kind of rationale that ties the given conceptual information to its underlying assumptions. »Such a rationale enables us to identify the impact of different environments and modify the conceptual information in a systematic way.« [BR91, p. 306]

Example: Experience bases for various companies vary with respect to the:
- Kinds of artifacts stored
- Information used to characterize the artifacts
- Processes used for learning, forgetting, and utilizing artifacts

Consequences if not fulfilled: The conceptual information will have to be reengineered at a high cost due to missing rationales before transferring or improving the technical infrastructure.

Refinement: none

Req. R11: modularity of conceptual information

The conceptual information must be modular.

Prerequisites: R10

Explanation: Tailoring effort for the conceptual information can be saved if the conceptual information can be constructed from a library of modules or with the help of patterns. Experience has shown that it is much easier to prune conceptual information than to expand it.

Example: Examples for patterns and modules include:
- How to model versions of artifacts

- How to model the development history of artifacts
- Conceptual information for artifacts that are processed with tools frequently installed at companies
- Conceptual information for artifact kinds that are stored by many organizations (e.g., lessons learned)

Consequences if not fulfilled: The tailoring of the conceptual information will take more effort than necessary. It may be more effective to develop the conceptual information from scratch.

Refinement: none

Req. R12: accommodation of new artifact kinds

The infrastructure must be capable of storing artifacts that were not anticipated at the time the infrastructure was installed.

Prerequisites: R7 and R11

Explanation: Over time, additional uses of the experience base may be identified requiring the storage of artifact kinds that were not anticipated.

Note: This requirement is of central importance to the infrastructure of the experience base. Since all kinds of software-related information shall be stored, it usually takes a long time to build up an experience base. This means that the infrastructure has a long lifetime and must be able to grow with the demands of the organization.

Example: After the experience base has been introduced to the organization, it is decided to include the artifacts of the code repository as well. Before, code components were stored separately from the rest of the software development information (in the code repository which already existed at the time the experience base was introduced).

Consequences if not fulfilled: The effort for expanding the experience base to new artifact kinds will be prohibitively high.

Refinement: none

Req. R13: maintenance of conceptual information

Changes of the conceptual information must be reflected in the artifacts' characterizations or the conceptual information must be versioned.

Prerequisites: R9

Explanation: The artifacts' characterizations may become inconsistent with the conceptual information if the conceptual information is changed. Therefore, either the conceptual information must be versioned (so it is clear to which concept version an instance refers) or the artifacts' characterizations have to be adapted. Which adaptation operation should be invoked depends on the change operation performed on the conceptual information and the amount of effort to be spent for »re-characterizing« the artifacts.

Example: If the conceptual information is extended by administrative information (see requirement R4), the administrative information of already stored artifacts should be set to a predefined value, e.g., the current date for the creation date of the artifact.

Consequences if not fulfilled: Characterization information can become invalid, meaning that the old artifacts' characterization can no longer be retrieved.

Refinement: none

3.3 Maintenance of Experience Packages

The last section dealt with the maintenance of conceptual knowledge whereas this section deals with the maintenance of characterization information.

Req. R14: maintenance of experience packages

The technical infrastructure must support maintenance of experience packages, i.e., it must be capable of handling stable and instable experience packages by providing functionality for learning, forgetting, and versioning. [ABT98, GABT98, PD91, RM88]

Prerequisites: none

Explanation: As an organization learns more and more about its software development practices, new artifacts must be recorded in the experience base (*learning*). This may involve the restructuring of the characterization information of the experience base. Also, obsolete or useless information

must be either removed (*forgetting*) or replaced by new information (*versioning*). (In general, information should not be removed or replaced physically, but only excluded from future retrieval results. This allows to still inquire information about artifacts that have been marked as forgotten or replaced.)

Example: Several improvement suggestions that have been accumulated in the experience base for a particular document are worked in. These improvement suggestions now become obsolete and are marked accordingly (i.e., they are only needed for recalling the history of the changed document). Only the latest version of the document is now visible in the experience base.

Consequences if not fulfilled: There is the danger of creating a »knowledge cemetery«. Queries will return more and more obsolete or useless artifacts. Consequently, the ratio of useful to useless answers decreases and with it the value of the experience base as a whole.

Refinement: This requirement is fulfilled if R17, R27, and R28 are fulfilled.

Req. R15: data collection for evaluation

The infrastructure must allow the collection of measurement data. [ABT98, EMT98, Nic98]

Prerequisites: none

Explanation: To improve the technical infrastructure in a goal-oriented manner, data has to be collected. The collected data can be attributed to the artifacts (in this case they should be stored as part of the artifact's characterization) or to the retrieval system as a whole.

Example: Exemplary measures include:

- The number of times an artifact has been utilized
- The number of times an artifact appeared as the result of a query
- The number of successful and unsuccessful searches
- The total number of artifacts in the experience base, grouped according to their kind

Consequences if not fulfilled: Either the data must be entered manually (e.g., by changing the artifacts' characterization) or no data is collected. The former alternative is error-prone whereas the latter does not allow a systematic and continuous improvement of the technical infrastructure. This violates that everything related to software development (and the experience base is part of it) should be improved on a continuous basis.

Refinement: none

Req. R16: data analysis for improvement

The infrastructure must support the analysis of collected data.

Prerequisites: R15

Explanation: To improve the technical infrastructure, it is not enough to just collect data. Rather, the data has to be analyzed and appropriate actions for improving the infrastructure have to be defined.

Example: The infrastructure lists all characterizations whose artifacts have not been checked out for the last two years.

Consequences if not fulfilled: The data has to be analyzed manually. This is an error-prone and effort-intensive procedure.

Refinement: none

3.3.1 Learning

Req. R17: artifact recording

The user (typically the administrator) must be able to record a new artifact into the experience base.

Prerequisites: none

Explanation: The recording of an artifact involves the storage of the artifact itself as well as the elaboration of the artifact's characterization. A variation of the recording function is the recording of a new version of an artifact.

Consequences if not fulfilled: Incremental, continuous learning can not be supported.

Refinement: none

Req. R18: various characterizations of one artifact

It must be possible to characterize an artifact in various ways. [RM88]

Prerequisites: none

Explanation: The same artifact may serve different purposes. Therefore, it may have more than one characterization. The relationship between the artifact's characterizations must be recorded.

Example: A software development organization may develop its own development tools, e.g., a compiler. The executable of such a compiler would be considered as both a software system (i.e., a software product) and a development tool.

Consequences if not fulfilled: The artifact might have to be stored multiply; this can result in inconsistencies between artifact copies or their characterizations.

Refinement: none

Req. R19: tolerance of different levels of abstraction

It must be possible to characterize artifacts of the same kind on different levels of abstraction. [GABT98]

Prerequisites: none

Explanation: Over time, the development environment of an organization changes. Tools may be purchased which have their proprietary format for storing artifacts. For the one who wants to reuse such an artifact, it does not matter which tool was used for its creation. However, if the new tool is able to export detailed characterization information, the user might also want to take advantage of this fact by performing reuse on a lower level of abstraction.

Example: In the past an organization documented their development processes using a word processing system. Process models were characterized manually using a single concept. Now it has procured a process modeling tool. The tool is able to export detailed characterization information, that is, in addition to the top-level characterization of the process models, constituents of the process model (e.g., roles, product models, process descriptions) are also characterized as reusable parts in the experience base. This allows, for example, to query for all process models where a particular role is involved (although such a query would return only process models created with the new tool). Consequently, old and new process models are characterized on different levels of abstraction.

The policy of the organization does not require that all new process models are created using the new tool. Therefore, old and new process models can

be reused. In conjunction with requirement R3, the right tool should be invoked depending on the representation of the artifact.

Consequences if not fulfilled: The tool integration may not be transparent (R3 not fulfilled) or only one tool per artifact kind can be used.

Refinement: none

Req. R20: tolerance of incomplete information

It must be possible to store incomplete characterization information.

Prerequisites: none

Explanation: see Section 2.2.3.

Note: This requirement implies that retrieval also tolerates incomplete characterization information.

Consequences if not fulfilled: The experience base will be of limited use, because the characterization of artifacts must be restricted to information that is always available.

Refinement: none

Req. R21: tolerance of uncertain information

It must be possible to store uncertain characterization information.

Prerequisites: none

Explanation: see Section 2.2.3.

Note: This requirement implies that retrieval also tolerates uncertain characterization information. Two retrieval techniques that tolerate uncertain characterization information are similarity-based retrieval[1] and browsing. Retrieval techniques that rely on exact matches are not suitable.

Consequences if not fulfilled: The experience base will be of limited use, because the characterization of artifacts must be restricted to information that is 100% certain. Alternatively, the information may not be stored at all (if R20 is fulfilled) or it may be stored imprecisely (if R22 is fulfilled).

Refinement: none

Req. R22: tolerance of imprecise information

It must be possible to store imprecise characterization information.

Prerequisites: none

Explanation: see Section 2.2.3.

Note: This requirement implies that retrieval also tolerates imprecise characterization information.

Consequences if not fulfilled: The experience base will be of limited use, because the characterization of artifacts must be restricted to precise information. Alternatively, the information may not be stored at all (if R20 is fulfilled) or it may be stored as uncertain information (if R21 is fulfilled). In both cases, information is lost.

Refinement: none

Req. R23: transparency of duplicated information

If characterization information is stored redundantly, it should be transparent to the user.

Prerequisites: none

Explanation: As stated in Section 2.2.1, artifact characterizations may overlap. The information in these overlapping regions may be stored redundantly. However, the duplicated information must be consistent.

Example: Two lessons learned share the same context.

Consequences if not fulfilled: Inconsistencies can be introduced by the system even though these inconsistencies do not exist in the real world.

Refinement: none

1 The underlying assumption is that the characterization information is not arbitrary, but that the specified value for a property class is the *most likely* value. If the specified value is not the actual value, the actual value will be close to the specified value. In other words, the further away a value is from the specified value, the more unlikely it is that it is the actual value. For example, if the lines of code of a code component is estimated to be 1,000, the probability that it is actually 10,000 (difference 9,000) is lower than the probability that it is 2,000 (difference 1,000).

Req. R24: tolerance of inconsistent information

It must be possible to store and document inconsistent information. [RM88]

Prerequisites: none

Explanation: »Because of the long-term nature of software development processes, it might be necessary to store intermediate information that does not yet conform with the desired consistency criteria.« [RM88, p. 7] There are three types of consistency (see Section 2.2.2): consistency between the parts of an artifact's characterization, consistency between an artifact and its characterization, and consistency across artifacts.

While the first two types of consistency should only be tolerated temporarily, i.e., until the characterization has been validated, the latter type of consistency may persist for quite some time as the following example shows.

Example: Suppose that traceability information among requirements documents, design documents, and code components are recorded as part of the characterization information. Let us further assume that the requirements document *R*, the design document *D* and the code component *C* are consistent and stored as part of the experience base. If *R* and *C* are changed to *R'* and *C'* respectively, but *D* remains unchanged due to time pressure, then *R'*, *D*, and *C'* are inconsistent. Possibly *D* is not changed until *C'* is requested by another project.

Consequences if not fulfilled: Either intermediary information can only be stored if it is consistent[1] or the information cannot be marked as »inconsistent«. The former case may cause a delay in making information available for other projects which will in turn slow down the pace of organizational learning. The latter withholds important information from the user (e.g., the information that two artifacts are inconsistent can be regarded as a project risk).

Refinement: none

1 Some inconsistencies can be resolved by using uncertain or imprecise information.

Req. R25: integrity constraints

It must be possible to check the integrity of the experience base to detect inconsistencies. [MR87]

Prerequisites: R24

Explanation: Inconsistent information (especially inconsistent artifacts) reduce the reusability of the artifacts. Therefore, to raise the reusability of artifacts, inconsistencies should be identified and removed.

Example: In the example of R24, the design document *D* is not consistent with the requirements document *R'* and the code document *C'*. This reduces the reusability of *C'*, because its design must be updated before it can be reused. This would require a reengineering effort on part of the project that wants to reuse *C'*. However, the experience factory can perform *preventive maintenance* by making *D* consistent with *R'* and *C'*.

Consequences if not fulfilled: The reusability of artifacts will be lowered; retrieval results may be misleading.

Refinement: none

Req. R26: accommodation of growing collection

The infrastructure must accommodate a continually expanding collection of artifacts. [PD91]

Prerequisites: R10, R17

Explanation: As more and more artifacts are recorded, it may become difficult to distinguish between them. This means:

1 The characterization information should be extended so that a distinction of the artifacts becomes possible
2 The representation of the characterization knowledge must allow the larger collection, that is, retrieval must remain efficient

Example: Artifacts of very small collections can be displayed as a single list. Larger collections can be organized using enumerated classification, that is, artifacts are retrieved using a browser. Still larger collections can either be indexed through keywords or classified according to several dimensions.

Consequences if not fulfilled: Retrieval will become ineffective as the collection grows.

Refinement: none

Req. R27: artifacts' status

Artifacts' characterizations must include a status. The status defines for what purposes an artifact is retrieved.

Prerequisites: R17

Explanation: The status of an artifact may change as it matures or degrades in importance. Depending on the status of an artifact, certain tasks using this artifact may be performed while others may not. Thus, the explicit documentation of the status allows the enforcement of policies.

This requirement's implementation can also be used to implement the »forgetting« of artifacts.

Example: The notions of »obsolete« and »useless« are relative. A status can be used to define policies on how to handle artifacts. For instance, if errors in a code component are corrected, only the new version should be utilized by succeeding projects (see also requirement R28). Note that the experience packages should not be deleted physically, because running projects (or future maintenance projects) may inquire information about the old (and now obsolete) artifacts. In this case, it is »OK« to retrieve an obsolete artifact.

Consequences if not fulfilled: Policies cannot be enforced; »forgetting« is difficult to realize.

Refinement: none

3.3.2 Versioning and Configuration Management

Req. R28: versioning

The infrastructure must provide version management for the artifacts stored in the experience base. [EMT98]

Prerequisites: R27

Explanation: An artifact may exist in several versions and variants. Since various versions and variants of an artifact may be in use by various projects at the same time, all versions and variants of an artifact (and the experience related to them) must be stored in the experience base. It is important that their mutual relationships are documented. Conceptually, they are part of the characterization information.

Consequences if not fulfilled: Each version and variant of an artifact will be stored as a unique, that is, independent artifact. The fact that these »independent« artifacts are related will not be documented. Thus, the version history of an artifact cannot be recorded.

Refinement: none

Req. R29: configurations

The infrastructure must provide configuration management for the artifacts stored in the experience base.

Prerequisites: R24, R28

Explanation: Most artifacts depend on other artifacts. The process of creating an artifact usually has one or more artifacts as input. If one of the input artifacts is changed later on, it has to be checked whether any artifacts that depend on the changed artifact have to be changed too. This requires the explicit documentation of artifact dependencies. Conceptually, they are part of the artifacts' characterizations.

Example: Suppose that a requirements document *R* and a design document *D* are compatible, that is, all requirements of *R* are fulfilled by *D*. If *R* is changed, *D* has to be analyzed if it needs to be changed too.

Consequences if not fulfilled: The development history of an artifact cannot be recorded.

Refinement: none

3.4 Dissemination

The storage of artifacts alone does not allow for improvements. The artifacts must also be delivered to the users utilizing the artifacts. This section lists the respective requirements.

3.4.1 Active Dissemination

Req. R30: change notification

The infrastructure must be able to notify end users of changes in the experience base. [EMT98]

Prerequisites: R29 and R4

Explanation: If some artifact has been changed:

- The users of an older version of the artifact (i.e., people who copied and possibly modified it and then integrated it in some surrounding system that is still used and maintained) should be informed.
- The owners of the artifacts which depend on the changed artifact (i.e., people who used the changed artifact for reference purposes only) should be informed.

This is a first step towards ensuring the consistency of the information stored in an experience base.

Also, if a new artifact is recorded, some people in the organization may want to be notified, that is, end users should be able to subscribe to selected kinds of artifacts.

Example: Scenarios for this requirement are manifold:

- At the beginning of an implementation, code components are retrieved in order to be integrated into the new software system. This fact is recorded in the experience base (as part of the versioning information: »who checked out the artifact?«). One of the components is also used in another software development. There, errors in the component are identified and corrected. The updated component is recorded in the experience base as a new version. The technical infrastructure notifies the users of the old version that a new version has been recorded. The notified project benefits, because it can now use the higher quality artifact for their software system; thus raising the quality of the entire system.
- If a new contract has been finalized (and thus recorded in the experience base), all project managers of the organization are informed.

Consequences if not fulfilled: Either the affected people must be notified manually by the experience factory team or newer versions are discovered by chance.

Refinement: none

3.4.2 Passive Dissemination/Retrieval

Req. R31: artifact preference information

Artifacts' characterization must include preference information. [KS96]

Prerequisites: none

Explanation: A query of the experience base will not always be successful. If no artifact in the experience base fits the needs of the user, he still does not necessarily have to start from zero. In such a case, it can be helpful to retrieve a template for the artifact to be created. Another possibility is to provide the user with exemplary artifacts. However, it is usually difficult for the user to decide which of the stored artifacts are exemplary, because what is »exemplary« depends to a large degree on the policies of an organization.

Example: A requirements engineer is looking for a requirements document, but does not find one that could be tailored with a reasonable amount of effort. Therefore, he retrieves a template for requirements documents. But how is this template to be filled in? The requirements engineer decides to look at examples. However, there are several requirements documents stored in the experience base that use the chosen template as their basis. Which one is the best? It is a requirements document that is complete (i.e., all placeholders of the template are filled out) and reflects the intention of the one who defined the template. Whether an artifact fulfills the latter condition cannot be decided by the requirements engineer. Therefore, he needs additional information on whether the artifact conforms to the intention.

Consequences if not fulfilled: Application policies cannot be enforced; decision-making process for exemplary artifacts will be prolonged.

Refinement: none

Req. R32: browsing

It must be possible to browse the characterization information. [EMT98]

Prerequisites: none

Explanation: For small artifact collections (in the order of tens), browsing suffices as a retrieval technique [EMT98]. Browsing can also be used for follow-

ing semantic relationships among artifacts. In this way, the understanding of retrieved artifacts is supported.

Consequences if not fulfilled: Other means of retrieval must be provided; it is not possible to follow semantic relationships among artifacts to better understand them.

Refinement: none

Req. R33: textual search

It must be possible to search for artifacts using textual input.

Prerequisites: none

Explanation: The artifacts' characterizations may contain informal descriptions. These can be utilized by a textual search. The requirement can be extended to the artifacts themselves (requires that artifacts are text documents).

Note 1: An advantage of textual search is that it requires little effort when storing a new artifact, because the text can be indexed automatically (see Section 5.1.5). Yet, it allows to query the experience base. By comparison, purely schema-based approaches require a higher effort to characterize artifacts.

Note 2: Textual search is of limited use to search for quantitative experience. Without background knowledge on similar words, textual search cannot accommodate uncertain information.

Consequences if not fulfilled: It is not possible to search for terms that have not been considered as part of the conceptual information.

Refinement: none

Req. R34: similarity-based retrieval

The infrastructure must be able to retrieve similar artifacts. [ABT98, ABH+99, BR91, GABT98, Hen96, OHPDB92, PD91]

Prerequisites: none

Explanation: In practice, the artifact matching exactly the user's need may not exist. In such a case, similar artifacts should be returned as the result of a query. Here, »similar« means that the artifact can still be utilized (although

the user's needs are not completely met) or that it can be tailored the user's needs with a comparably low amount of effort.

Consequences if not fulfilled: Support for decision making by the information system is not possible, that is, the user has to find the most similar artifact by himself. This means: Either browsing must be employed to find a suitable artifact or reuse with modification will not be possible.

Refinement: none

Req. R35: tolerance of incomplete query information

It must be possible to query the system with incomplete information. [ABT98, GABT98]

Prerequisites: none

Explanation: see Section 2.2.3.

Consequences if not fulfilled: Either browsing must be employed to find a suitable artifact or the technical infrastructure will not be usable.

Refinement: none

Req. R36: tolerance of uncertain query information

It must be possible to query the system with uncertain information. [GABT98]

Prerequisites: none

Explanation: see Section 2.2.3.

Consequences if not fulfilled: Either browsing, incomplete, or imprecise query information must be used.

Refinement: none

Req. R37: tolerance of imprecise query information

It must be possible to query the system with imprecise information.

Prerequisites: none

Explanation: see Section 2.2.3.

Consequences if not fulfilled: Either browsing, incomplete or uncertain query information must be used.

Refinement: none

Req. R38: context information

The characterization information must include context information. [ABT98, ABH+99, BR91, GABT98, RM88]

Prerequisites: R29

Explanation: To record experience, a context must be specified (see Definition 5 on page 24). But the context may also be recorded for artifacts that document information other than experience.

Context information includes the:
- Application domain the artifact has been created for
- Application domains the artifact has been utilized in so far
- Artifacts from which this artifact has been created (configuration management)
- Solution domain [BR91], that is, the process/technique used to create the artifact
- Object that was observed (in case the artifact documents an experience)

Context information can be utilized for:
- Decision-making on whether to utilize an artifact (in this case, applicability experience must be available; see also requirement R42)
- Understanding an artifact (documentation of its »creation history«; see requirements R29 and R4)
- More effective retrieval (see requirement R39)

Note: This requirement is of central importance to the infrastructure of an experience base, because experience must always be recorded together with the context it was gained in.

Consequences if not fulfilled: Experience cannot be stored in the experience base; it will be harder to understand an artifact.

Refinement: none.

Req. R39: context-sensitive retrieval

It must be possible to restrict the retrieval of artifacts to those developed for or applied in contexts similar to the current context/situation. [ABT98, ABH+99, BR91, GABT98, RM88]

Prerequisites: R34 and R38

Explanation: Not every artifact is applicable in every situation. Therefore, it should be possible to specify the current context for a query, so only applicable artifacts are returned. Also, an artifact may exhibit different qualities in different contexts. Therefore, if certain qualities are required, the context in which the artifact is to be utilized must also be specified.

Another reason for this requirement is to support the first step of the quality improvement paradigm (QIP) [BR88, BCR94a]. Here, the experience base is queried for all artifacts that could potentially be used for the current project. For this purpose, the characteristics of the project (and the organizational unit in which the project is performed) are listed and the experience base returns all artifacts that are applicable in such a context. This gives an overview of available knowledge for the current project. Based on this information, missing information can be identified. One of the project's goals may then be to fill these »information gaps«.

Since each project is unique, project characterizations (= contexts for experience gained and applied in the projects) vary as well. Hence, context-sensitive retrieval requires the ability to find *similar* contexts.

Note: This requirement is also of central importance to the infrastructure of an experience base. Without it, information stored in the experience base cannot be filtered to the project at hand. Since many information items will accumulate in the experience base (the more types of artifacts are stored in the experience base the more likely is an information flood) over time, the ability to filter out the relevant from irrelevant information becomes critical to the success of the infrastructure.

Example: Experience is only applicable in the context it has been gained in. For instance, a code component may be reliable for a toaster, but not for the ground control of a satellite.

Consequences if not fulfilled: Retrieval will not be effective for large artifact collections, because many irrelevant artifacts will be returned; it will be effort-intensive to perform the first step of the QIP.

Refinement: none

3.5 Utilization

After an artifact has been selected, it is utilized. Requirements pertaining to the utilization of artifacts from the experience base are listed in this section.

Req. R40: check-out of artifacts

It must be possible to check-out artifacts from the experience base. [EMT98]

Prerequisites: none

Explanation: After an artifact has been selected from the experience base, a local copy must be made which can be tailored to the project-specific needs.

If the infrastructure supports active dissemination, the check-out must be recorded in the experience base (cf. requirement R30).

The recording of who checked-out the artifact also allows to ask users for their application experience (cf. requirement R42).

Consequences if not fulfilled: The experience base will be a write-only archive.

Refinement: none

Req. R41: interface information

The characterization information must include interface information. [BR91, Mat84a]

Prerequisites: none

Explanation: To answer the question whether an artifact can be integrated in the current environment or system, not only information about the artifact itself and its context is required, but also about the artifact's interface, that is, information about preconditions that must be fulfilled to utilize the artifact.

If the interface requirements of the artifact do not match the characteristics of the environment into which the artifact is to be integrated, either the artifact can be tailored to the environment or the environment characteristics can be changed to meet requirements of the artifact's interface. In this way, the interface information can also be used to estimate the effort for utilizing the artifact.

Example: To decide whether a code component can be integrated in the system, the following interface information should be available [Mat84a]:
- Input/output parameters
- Global variables accessed by the component
- Other code components required to utilize this one
- Required input/output devices
- Memory size required to use the software
- Interrupts that affect the software

Consequences if not fulfilled: The user will not be able to (a) decide whether an artifact is applicable (based on the artifact's characterization) and (b) estimate how much effort it will take to utilize the artifact (based on its characterization).

Refinement: none

Req. R42: application history

The context information of an artifact's characterization must include information about its application history. [EMT98]

Prerequisites: R38

Explanation: For utilizing an artifact it is of interest in which situations it is meant to be applied and in which situations it has been applied before. The latter should include application experience [EMT98, GB97]:
- Whether the application of the artifact was successful or failed (including the reason for failure)
- Problems met in understanding, tailoring and using the artifact (and how they were solved)
- Guidelines for those tasks (based on the occurred problems)
- Effort required for those tasks

Information about the application history helps in deciding whether to utilize an artifact and in estimating the effort for utilizing the artifact. It can also be used to improve the artifact during maintenance. For example, problems met in understanding can be alleviated by adding comments.

Consequences if not fulfilled: The quality of the utilization decision will be worse; effort estimation for utilizing an artifact will be more difficult.

Refinement: none

3.6 Summary

In this chapter, basic functional requirements for a technical infrastructure of an experience base have been elicited along with their dependencies (see Figure 6) and consequences. As a side effect, important information that should be part of the characterization information has been identified (see Figure 7).

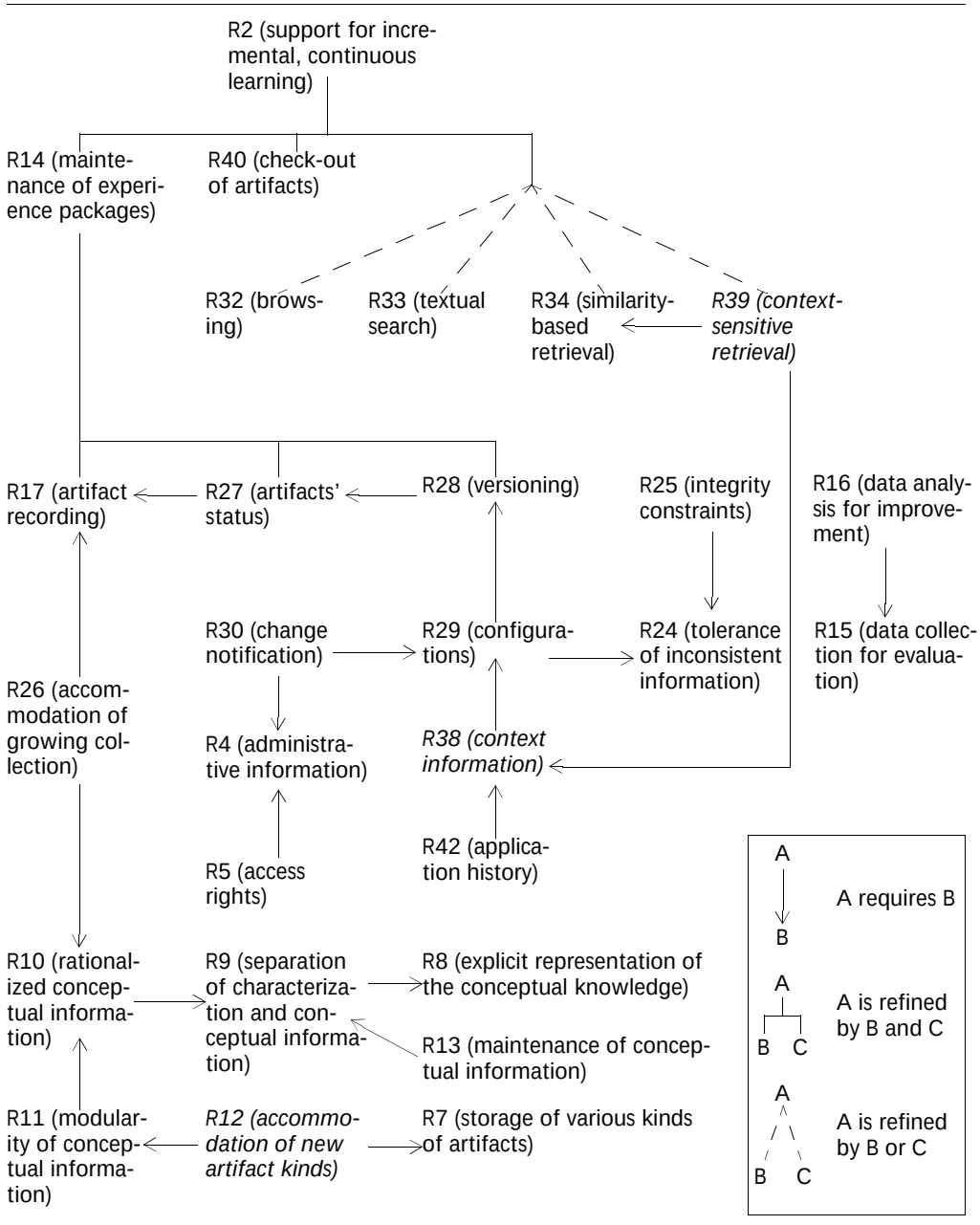


Figure 6: Dependencies between requirements (requirements independent of all others not listed)

Figure 7: Minimal contents of an artifact characterization

Information about the artifact itself:
- Administrative information (owner, author(s), creation date, …)
- Access rights
- Access information (how often the artifact has been checked out, how often the artifact has appeared as the result of a query, …)
- Preference information
- Status
- Functionality/purpose

Information about the interface of the artifact:
- Inputs/outputs
- Prerequisites for utilization
- …

Information about the context of the artifact:
- Application domain artifact has been created for
- Application domains artifact has been applied in (including application experience)
- Solution domain (process/technique used to create this artifact)
- Artifacts from which this artifact has been created
- Version information (older/newer version of this artifact, variants of this artifact, people who have checked out this artifact, …)
- …

The requirements are based on insights from the people who coined the term »experience base«, especially those requirements documented in [BCC92, BCR94a, BR91, MR87, RM88]. These were complemented by requirements that follow from the identified practical constraints (Chapter 2) and requirements elicited through recent industrial projects both in the area of incremental, continuous learning [ABT98, KS96] and software reuse [EMT98]. Finally, the requirements were grouped and analyzed for dependencies among them.

The requirements listed in this chapter can be used to choose a concrete infrastructure for an experience base according to the needs of an organization. Doing so requires deciding whether the consequences are »OK«. If so, the corresponding requirement can be dropped. Existing and emerging tools can also be classified according to the requirements they fulfill (see Chapter 5). Then, the needed requirements can be matched against those fulfilled by existing systems.

Of central importance to the technical infrastructure of an experience base are the requirements R12, R38, and R39 (printed in italics in Figure 6) together with the requirements they depend on. Without R12, new artifact types cannot be added to the experience base. This is a problem, because the buildup of an experience base usually takes a long time. Thus, the infrastructure should be able to grow with the needs of an organization. Without R38, no experience at all can be stored. Finally, without R39, information irrelevant for a project cannot be filtered out. Although recognized as early as 1988 (e.g., [BR88]), these requirements have not been supported by a technical infrastructure yet (see

Chapter 5). The implementation of these requirements is thus a major contribution of this dissertation to the state-of-the-art.

The basic functionality provided by these requirements allows the infrastructure to be used in many ways. It does not prescribe a sequence in which the functions are to be performed. Therefore, a systematic engineering of learning, dissemination, and utilization tasks is needed. For the operationalization of these tasks, the basic functionality of the infrastructure may be used. This will be the subject of the next chapter.

4 Maturing Tasks of an Organization

The system's job is to increase the overall quality and effectiveness of the user's work. Therefore, it is the total problem solving ability of the human user and the computer – in cooperation – which is the factor to optimize, not that of the computer system itself.

[AN95, p. 21]

Whereas the previous chapter investigated the main requirements for a technical infrastructure of an experience base, this chapter will take a closer look at the tasks to be performed for populating, utilizing, and maintaining the experience base. This corresponds to a top-down approach, whereas the approach taken by the previous chapter can be viewed as a bottom-up investigation.

As can be seen from the previous chapters, the contents of an experience base cover (at least potentially) a very wide range of knowledge. No single tool will be able to handle all of this knowledge in every conceivable way. Therefore, it is essential to have a generic framework that can be instantiated according to the needs of an organization. This framework must be supported by a flexible technical infrastructure that provides generic functionality as to minimize implementation efforts for setting up an experience base in a new environment.

Environments vary regarding both the kinds of knowledge to be captured and the processes to be performed for populating, maintaining, and utilizing the experience base. The former are influenced by strategic aspects, whereas the latter are constrained by organizational and managerial aspects. In total, there are five kinds of aspects to be considered when setting up an experience factory (cf. Section 1.2):

1 **Strategic aspects** relate to the scope of the experience base, i.e., to the competencies to be built up by the experience factory. The scope of the experience base is subject to change when the environment changes (e.g., if the market requires to prepare and infuse a new technology). Ultimately, the strategic aspects decide on the kinds of artifacts to be captured. This will be subject of Chapter 8.

2 **Organizational aspects** relate to issues such as organizational units, positions, and/or the existence of long-term projects for running an experience factory. These aspects have to be reconsidered if organizational changes take

place within the software development organization. A discussion of these issues can be found in [BCC92].

3 **Managerial aspects** relate to issues such as success control, budget, available personnel, allocation of responsibilities to organizational units, allocation of people to tasks, etc. They are covered by project management literature and papers on the experience factory concept (e.g., [BR88, BCR94a, Bas95, BC95]).

4 **Methodological aspects** relate to the tasks that have to be performed as well as how they have to be performed. They are subject of this chapter. Organizational and managerial aspects constrain the methodological solutions by limiting tasks to be performed (through defined responsibilities) and by restricting accessibility of knowledge (e.g., through budget and personnel limitations).

5 **Implementation aspects** relate to issues about the technical infrastructure used for the experience base such as the architecture of the tools and the user interface employed. The needed technical infrastructure is partly determined by the methodological solution chosen for an organization, because certain functionality is needed to support the chosen methods efficiently and effectively. Implementation issues are subject of Chapter 6 and Chapter 7.

The remainder of this chapter describes a generic framework for the methodological aspects. It uses a task-method decomposition that arranges the tasks to be performed in a hierarchy. Appropriate tasks and methods (describing how the tasks are to be performed) for a given organization can be derived using this framework while considering strategic, organizational, and managerial aspects.

Each task is described by its goal, its inputs and outputs, its subtasks, a method for performing it, and the requirements needed for performing the method. The latter allows the organization-specific configuration of a technical infrastructure once the methodological aspects have been decided upon. The task description itself is implementation-free in the sense that it does not specify how the needed knowledge is to be represented. It only defines what kind of knowledge is needed (input) and produced (output). The decision of the actual representation is deferred to the implementation of the technical infrastructure.

In addition, a type is given for each task defining a tailoring action for instantiating the particular task for a new environment. The framework is instantiated by:
- Removing tasks not to be performed within a particular organization
- Assigning roles to each of the tasks to be performed
- Selecting methods from libraries to achieve the goals of the tasks

The chapter is structured as follows. In Section 4.1, a suitable description formalism for the framework is systematically derived. Using this formalism, Sec-

tions 4.2 through 4.4 decompose the maturing task of an organization. For this purpose, Section 4.2 integrates existing models of continuous learning (here, the quality improvement paradigm [BCR94a] and the case-based reasoning cycle [AP94]) and reuse-oriented software development [BR91]. The top-level task is then decomposed into the subtasks »reuse« and »learn«. Both subtasks are further decomposed in Sections 4.3 and 4.4. Section 4.5 shows how various policies can be realized using the framework, while Section 4.6 summarizes the chapter and compares the framework to related literature.

4.1 Knowledge Level Approach

To describe the populating, maintaining, and utilizing of an experience base in an implementation-free way, techniques from the field of artificial intelligence can be applied if we think of the experience factory organization as an *intelligent system*[1]. In this section, a suitable description formalism is derived based on Newell's knowledge level approach [New82], which is widely accepted by the knowledge engineering community [Stu99].

In his presidential address of the American Association for Artificial Intelligence in 1982, Newell proposed the notion of the *knowledge level* by formulating the *knowledge level hypothesis*:

Knowledge level hypothesis

> *There exists a distinct computer systems level, lying immediately above the symbol level, which is characterized by knowledge as the medium and the principle of rationality as the law of behavior.*
>
> *[New82, p. 99]*

By distinguishing sharply between the knowledge level and the symbol level, it is possible to describe the behavior of a system[2] independent of the way knowledge is represented. According to Newell, an intelligent system (called an *agent*) is described at the knowledge level by *goals* it wants to achieve, *actions* it can perform, and a *body of knowledge*. Only the content, but not the structure of the knowledge is described. The behavior of an agent is determined by what the agent knows (*body of knowledge*), what it wants (*goals*) and the means it has to interact with its environment (*actions*) through the principle of rationality:

1 The term »system« is used here in a very broad sense and refers to the sociotechnological system of the experience factory organization (i.e., the union of the project organization that develops software systems and the experience factory).

2 Although Newell's knowledge level hypothesis refers only to computer systems, the notion can be easily generalized to systems where some of the actions are performed by humans.

Principle of rationality

If an agent has knowledge that one of its actions will lead to one of its goals, then the agent will select that action.

[New82, p. 102]

It turned out that it is difficult to use Newell's highly intentional and purpose-oriented way of describing a system for practical systems descriptions and modeling, because the principle of rationality does not consider any practical constraints [AN95]. This led to extensions of the original notion (e.g., [Ste90, Sti89]).

Knowledge-use level

Steels, for example, introduced a *knowledge-use level* that considers pragmatic constraints such as time and space limitations [Ste90]. The knowledge-use level focuses on:

- The decomposition of a *task* into manageable subtasks
- The ordering imposed on the tasks
- The kind of access to the knowledge that will be needed
- How pragmatic constraints can be overcome

The latter two bullets restrict the representation that can be chosen. This implies a mapping from the knowledge level to the knowledge-use level:

- A top-level goal of an agent is associated with a top-level task.
- A problem-solving method (part of the agent´s knowledge) determines what subtasks (*actions*) have to be performed in what order to achieve the goal of the top-level task. Each of these subtasks has its own goal to be achieved and, thus, may be decomposed further.
- At the lowest level, that is, a task that is not decomposed further, a method performs the task directly.
- The agent´s knowledge is thus divided in *procedural knowledge* (called *methods*) and knowledge needed as input for some task (called *domain knowledge*).

Components of expertise

To summarize, a description on the knowledge-use level is made up of the following *components of expertise* [Ste90]:[1]

- Tasks arranged in a hierarchy
- Problem-solving methods that either define in what order to perform the subtasks of a task or how to perform a task that is not further decomposed

1 This description formalism is similar to process modeling languages, in particular MVP-L [BLRV92]. Here, process models correspond to tasks and methods, whereas product models correspond to the domain knowledge. MVP-L also allows the formal decomposition of process models until they are finally described informally at the lowest level. The needed knowledge is described using consume relationships and entry criteria, whereas the goal is described by exit criteria. In contrast to the knowledge-use level approach, MVP-L does not allow to describe the goal informally (important for soft goals [BMSB95]) or pragmatic constraints on the product models, and does not distinguish between tasks and methods.

- Access to knowledge that is needed to perform some task including both (a) contents (domain knowledge) and (b) pragmatic constraints imposed on (a)

Distinguishing the notions of tasks and problem-solving methods has the advantage that knowledge acquisition and representation is no longer independent of how the knowledge will be used in concrete problem-solving (as opposed to the development of earlier expert systems where the method selection depended only on the goal to be achieved) [Ste90]. For instance, the propose-and-revise method implies two kinds of knowledge: one for proposing solutions and another one for revising the solutions. Based on the pragmatic constraints of both kinds of knowledge, an appropriate problem-solving method can be chosen. Thus, the knowledge-use level approach can be viewed as a goal-oriented method for selecting an appropriate problem-solving method and, consequently, also as a method for the acquisition of domain knowledge.

An appropriate problem-solving method depends on:
- Conceptual aspects
- Pragmatic aspects
- Available domain knowledge

Especially pragmatic aspects play an important role in distinguishing between different methods: For example, if solutions cannot be proposed based on known regularities in the form of rules, a rule-based determination of a solution is not possible. Instead, case-based reasoning could be employed which bases the proposing of solutions on problem-solution situations of the past. Such problem-solution pairs can be acquired continuously as the system is used. However, if the cost of observation plays an important role, case-based reasoning is less appropriate, because the continuous acquisition of cases requires a constant effort.

The concept of problem-solving methods also allows reuse of conceptual knowledge. If tasks of the same generic nature are studied, the same type of conceptual knowledge and the same sort of inferences operating over such knowledge is needed. For instance, classification typically makes use of knowledge in the form of hierarchically arranged items. So far, quite a few generic tasks such as classification, diagnosis, information retrieval, decision support, and construction (including configuration, planning, and design) have been identified [Ste90, Ric98]. Generic tasks are defined independent of their application and can be instantiated by defining the knowledge they are to process.

The above discussion suggests a methodology for acquiring necessary knowledge to select and instantiate appropriate problem-solving methods for a given goal [Ste90]:

1 Characterize task identifying the major features – the description, the input and output, the generic task, the pragmatic constraints, and the knowledge available

2 Investigate the set of possible methods and associated knowledge needed

3 If a method suggests a decomposition, instantiate the tasks implied by the method. Reiterate for each of the subtasks from 1 until tasks have been reached that are directly solved by the application of knowledge

In the following the components of expertise will be applied to the system »around« the experience base, that is, the objective is to develop a knowledge-use level description for the tasks associated with an experience base. The pragmatic constraints on the software engineering knowledge in general have been discussed in Chapter 2. What remains is to identify the tasks needed for managing the knowledge of an organization, to arrange these into a hierarchy, and to identify the task-specific knowledge needed.

In practice, not all of the methods associated with these tasks can be automated as they depend on organization-specific policies or explicate tacit knowledge (which, of course, cannot be accomplished without an expert having the tacit knowledge). Still other methods cannot be defined generically, that is, independent of the kind of artifacts they operate on. This leads to the need to identify *artifact-independent* and *artifact-dependent* methods to be performed. For the artifact-independent methods, the *automatable* parts need to be identified. These parts must match with the functionality elicited in the previous chapter.

Because the components of expertise focus on task-decomposition and methods, the description on the knowledge-use level (for the tasks associated with an experience base) can be instantiated to fulfill organization-specific needs: Although not all methods can be defined in an artifact-independent manner, the goals associated with the tasks can often be stated in a generic way. This means that instantiating the knowledge-use level description involves the following tailoring procedures:

- **Removing superfluous tasks.** If a task (e.g., modification of retrieved components) is not to be performed, the task (including all of its subtasks) may be removed. Consequently, the knowledge needed for the task (and its subtasks) does not need to be acquired (if it is not needed by some other task). Vice versa, this means that certain tasks cannot be performed if the needed knowledge cannot be represented by the chosen technical infrastructure (see Chapter 5).
- **Defining artifact-specific methods.** Methods that cannot be described independent of the kind of artifact (e.g., adaptation of an artifact) must be defined separately for all kinds of artifacts stored in the experience base. It is expected that, with time, a library of methods for all kinds of artifacts can be

built up. This will then allow to configure knowledge-use level descriptions for experience bases from existing »components«.

- **Detailing tasks in artifact-specific ways.** Some tasks are so complex that they would fill entire guidebooks if treated with all possibilities. Such tasks (e.g., the task »collect« which has the objective to acquire new experience; its subtasks can potentially cover all of the methods known for knowledge acquisition) must be detailed on an as-needed basis. Again, it is expected that, with time, a library can be built up for the most common applications.

The following knowledge-use level description provides a framework that can be tailored to organization-specific needs by making local changes at predefined locations within the task decomposition using this limited set of tailoring procedures. One should start with the top-level task and proceed down the hierarchy as suggested by the structured methodology for knowledge acquisition on page 75. Reuse on the task/method level is supported through usage of »task« and »method« libraries, thus reducing tailoring effort to a minimum. It is this tailoring procedure that makes the components of expertise attractive for experience bases.

4.2 Cycle of Continuous Improvement

To derive an appropriate task-decomposition, we must start with the two major principles underlying the notion of the experience factory:

Quality improvement paradigm

1 The quality improvement paradigm (QIP) is the fundamental paradigm underlying all actions of an experience factory organization [BCR94a]. It is a six step cycle as shown in Figure 8. Over the years, several variants of this cycle have

[BCR94a]	[BDY94]	[BCC92]	[BZM+95]
characterize	plan	plan	understand
set goals			
choose artifacts			assess
execute	execute	execute	
analyze	evaluate	analyze	
package		synthesize	package

Figure 8: Quality improvement paradigm (QIP)

been developed. These variants are shown in the table next to the original cycle. This indicates that it is not important how many steps the QIP has. What is important is that it constitutes a closed cycle:

- The cycle consists of an »application« part (QIP 1 through QIP 4) and a »learn« part (QIP 5 and QIP 6).
- Between QIP 6 (of an old cycle) and QIP 1 (of a new cycle) capitalization of knowledge takes place, that is, knowledge learned in the previous cycle can be applied in the next cycle.

Experience factory organization

2 The experience factory is a physical and/or logical organizational unit of its own; it is not part of the project organization. However, it interacts with the project organization by collecting, analyzing, and synthesizing all kinds of experience that is relevant to software development and by supplying experience back to projects on demand [BCR94a]. Figure 9 depicts this principle graphically. Although both the project and the experience factory are involved in all steps of the QIP, the lead for performing the six QIP steps can be assigned to the organizational units of an experience factory organization [BC95]. Figure 9 shows that the project is responsible for the »application« part whereas the experience factory is responsible for the »learn« part of the QIP.

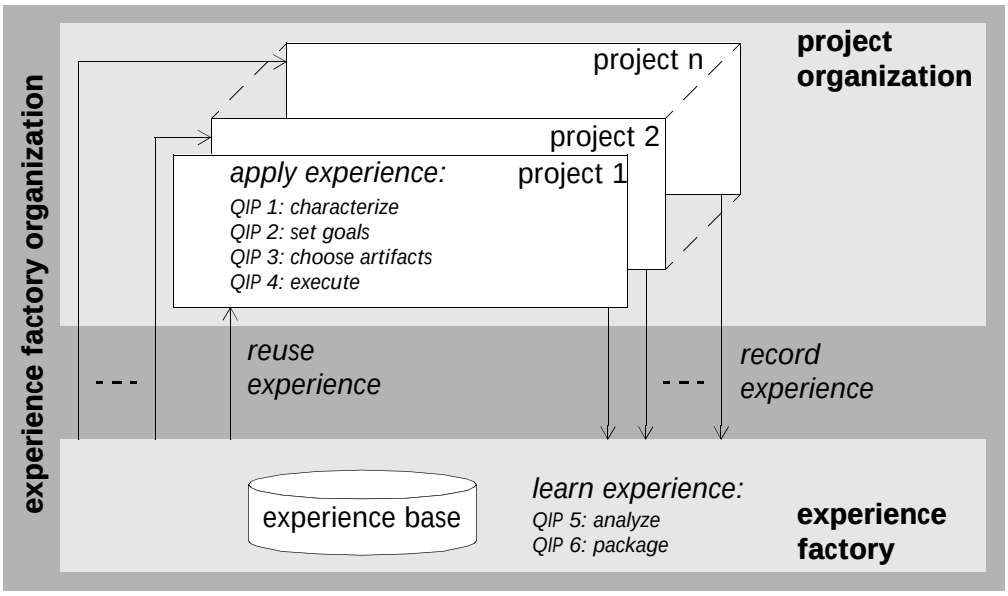


Figure 9: Experience factory organization

For the technical infrastructure of an experience base, the processes »reuse experience« and »record experience« are of major interest. To »serve« the experience factory organization right, the relationships of these processes to the six QIP steps must be investigated. This is the objective of the remainder of this section.

Case-based reasoning

As pointed out above, the fact that the steps of the QIP constitute a cycle is a prerequisite to realize continuous learning. Independently from the development of the QIP, such cycles have been proposed by other communities concerned with continuous learning (for a comparison see [Bas95]). A technology that is based on the principle of continuous learning is *case-based reasoning (CBR)* [Alt97]. A widely accepted model for case-based reasoning is the CBR cycle described by Aamodt and Plaza [AP94]. It is further refined by a task-method decomposition. Both the four step cycle as well as the task-decomposition of the four steps is shown in Figure 10. The cycle shows in the center the knowledge the tasks operate on. In addition, the »problem« at the top indicates that the cycle is performed for one reason only: to solve problems.

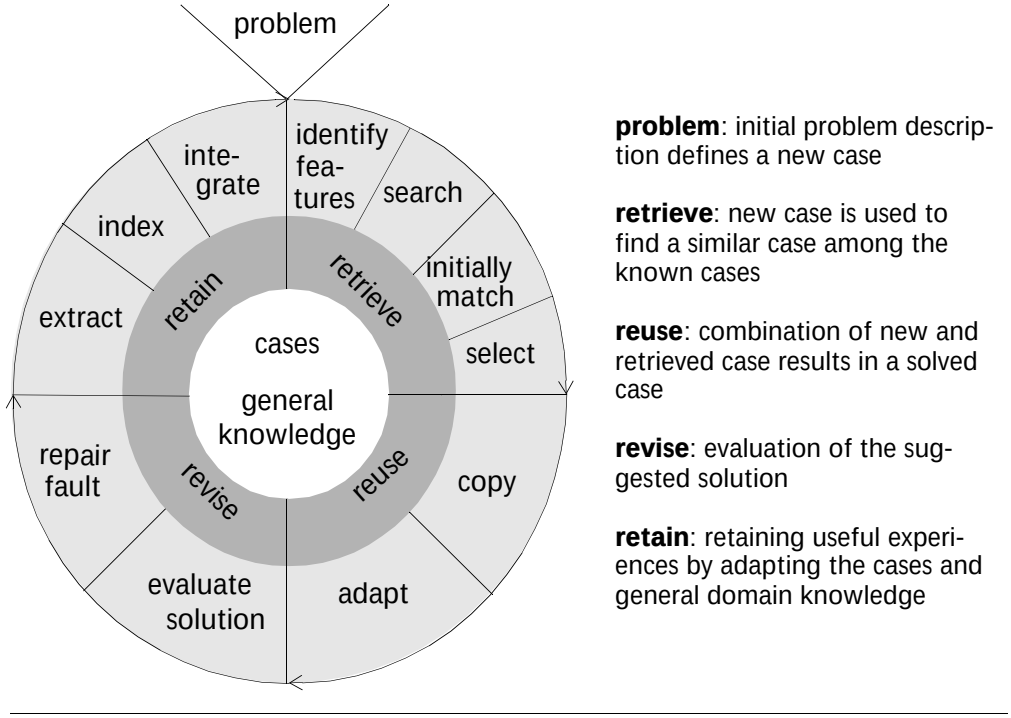


Figure 10: The CBR cycle and its task-decomposition on the first level

The CBR cycle views the steps as tasks that must be performed for each problem to be solved. If more than one problem is solved at the same time, the cycle is instantiated for each problem. This corresponds to an activity-oriented model. For example, the cycle may be performed several times during project execution (once for every artifact reused). In contrast, the QIP is often viewed as a phase-oriented model. For example, it can be used to describe the phases for a project: a project is planned, executed, and evaluated. During the execution, several artifacts are developed. The QIP is instantiated only once for the project. In addition to this project level view, the QIP can be used to describe the improvement activities on an organizational level (see Table 2 on page 80) – again, during a step several tasks may be performed (e.g., during the execution step, several projects

may be performed according to the project level QIP) corresponding to a phase-oriented view.

Table 2: Improvement steps on project level and organization level

Project level	QIP step	Organization level
characterize project and identify relevant models to be reused	QIP 1: characterize	characterize organization and identify future trends
define project goals in measurable terms and derive related measures	QIP 2: set goals	set up improvement goals and hypotheses in measurable terms
choose appropriate processes and develop project plan	QIP 3: choose artifacts	identify projects or pilots for investigating the hypotheses
perform project according to plan, collect data, and provide on-line feedback for project guidance	QIP 4: execute	perform projects or pilots and collect data
analyze project and collected data, and suggest improvements	QIP 5: analyze	analyze projects and pilots, and evaluate hypotheses
package analysis results into improved reusable artifacts	QIP 6: package	package experiences for use in future projects

However, the phase-oriented view is only an approximation of reality. For example, the development of a measurement plan is part of QIP 3 [GB97]. For the construction of the measurement plan, an old measurement plan stored in the experience base may be reused. Although final experience in the application of the measurement plan can only be recorded after the project, some of the experience (e.g., the cost of tailoring the old measurement plan to the new needs) can be recorded as early as in step 3 of the (project level) QIP. Thus, it can be useful to view the QIP as an activity-oriented model. In this case, the QIP is instantiated for every information item to be utilized or recorded in the experience base. Table 3 shows the interpretation of the QIP for the above example.

Table 3: Activity-oriented view of the QIP

Measurement plan	QIP step	Tailoring experience for measurement plans
Retrieve existing relevant measurement plans	QIP 1: characterize	Retrieve existing tailoring experience for the measurement plan at hand
Estimate effort to create a measurement plan from scratch; the goal is to need less effort; …	QIP 2: set goals	Collect the tailoring effort and the changes done to the measurement plan
Choose the most appropriate measurement plan among those retrieved in QIP 1; estimate the tailoring effort (see right column); tailor the chosen measurement plan to the project´s needs (see right column) or create a new one from scratch – whichever requires less effort	QIP 3: choose artifacts	Consider all tailoring experience for changes similar to the ones needed for this project; estimate the required tailoring effort based on this experience
Apply the measurement plan in the project	QIP 4: execute	Tailor the measurement plan
Analyze the application of the measurement plan; make suggestions for improvement	QIP 5: analyze	Check the tailoring effort; is the effort more or less than expected? Why?
Either record the new measurement plan or attach the application experience to the reused measurement plan	QIP 6: package	Record the tailoring effort, the changes done to the measurement plan, and the analysis results

MIRACLE

Based on the activity-oriented view of the QIP, the QIP can be compared to the CBR cycle and its task-decomposition. A common model can be constructed encompassing both the decomposed tasks of the CBR cycle and the QIP [TA97]. Figure 11 shows this model called MIRACLE (model integrating reuse and case-based reasoning for lots of software engineering experiences) [ABvWT98, TA98]. Matching the CBR cycle with the QIP allows to operationalize the QIP using the tasks and methods of case-based reasoning.

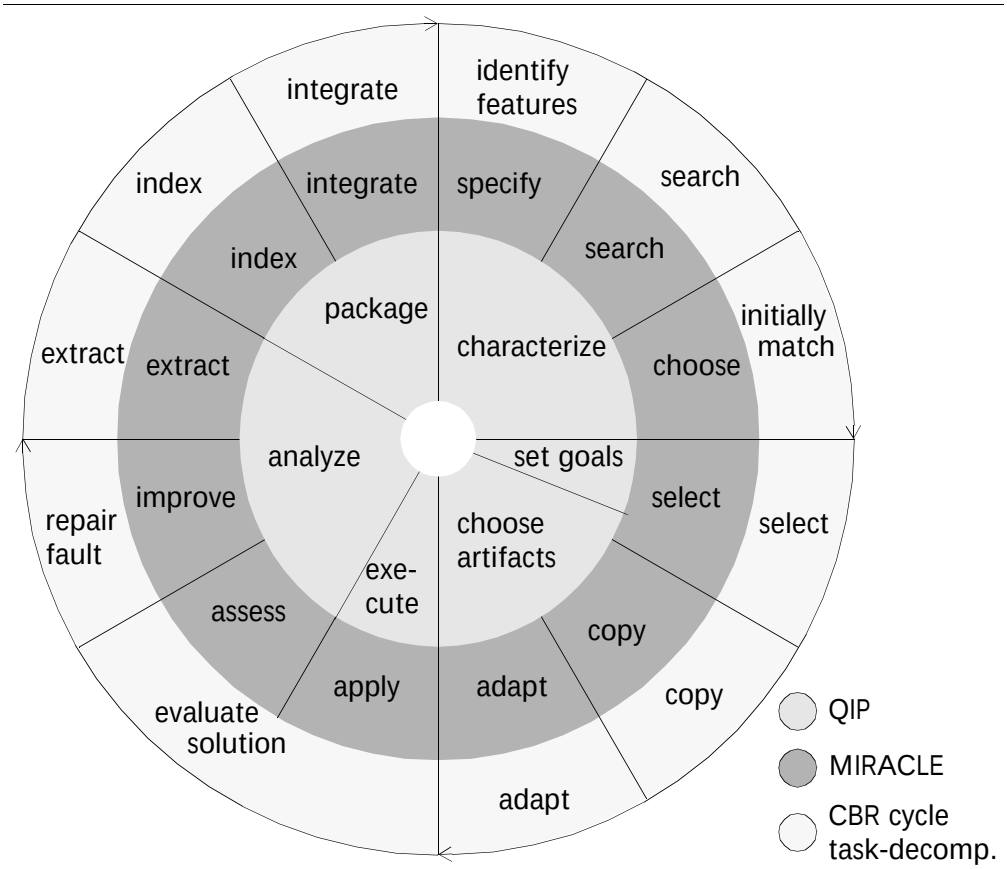


Figure 11: Matching MIRACLE with the QIP and the CBR cycle task-decomposition

Although quite detailed, the task-decomposition of Aamodt and Plaza does not cover all tasks to be performed »around« the experience base. This is because neither the QIP nor the CBR task-decomposition detail the application of knowledge (step »apply« of MIRACLE). They also fail to detail the task for recording knowledge to the degree necessary and miss out on such important tasks as developing and maintaining the conceptual knowledge of the experience base.

So far it has been shown that the activity-oriented view of the QIP can be operationalized using the tasks and methods of CBR and that these tasks map onto MIRACLE. The relationship between the QIP and the »reuse« and »record« pro-

cesses of Figure 9 on page 78 has not been shown yet. To do so requires a deeper understanding of »reuse« and »record«. This can be found in [BR91].

Reuse-oriented software development model

Basili and Rombach detail the »reuse« process by requiring mechanisms for identifying, evaluating, selecting, and modifying artifacts from the experience base [BR91]. These can be complemented by mechanisms for specifying the needed experience and incorporating the retrieved experience into some system *S* (thus modifying it) [TA97]. Basili and Rombach also give some vague hints on what is needed for recording experience. Finally they define »(re-)packaging« as a task operating on the contents of the experience base and not as a task (solely) operating on experience coming from projects. Consequently, (re-)packaging may be performed without recording new experience from projects.

Although not explicitly mentioned, *forgetting* is also an important task that operates on the contents of the experience base (cf. [SK95, LW98] on maintaining case bases). Due to changing environments or the availability of new high quality experience, it may become necessary to remove or replace old experience. Both, (re-)packaging and forgetting require triggers that determine when and what to (re-)package or forget. Usually the tasks are triggered as a consequence of analyzing the contents of the experience base. Figure 12 summarizes the discussion. The figure also shows that creating the needed experience from scratch is an alternative to reusing some already existing experience (e.g., if modifying the existing experience requires more effort than creating it from scratch).

As with the case-based reasoning tasks, the mechanisms for recording and reusing experience may be matched with MIRACLE [TA98] (see Figure 13). Processes internal to the experience factory are not part of the cycle.

Matching the mechanisms for recording and reusing experience to MIRACLE also matches the mechanisms with the steps of the QIP (also shown in Figure 13). Through this mapping the relationship between the QIP and the processes »reuse« and »record« from Figure 9 has been established. Note that the »reuse« process relates to the »application« part of QIP, whereas the »record« process and the processes internal to the experience factory relate to the »learn« part. Therefore it seems straight forward to refine the maturing task of an organization into a »learn« and a »reuse« task.

Both Figure 9 and Figure 12 suggest that recording and reusing experience do not need to constitute a closed loop. For example, a project may develop a measurement plan from scratch. This deliverable is recorded, but since no predecessor of this measurement plan exists in the experience base, reuse never took place. In another example, the project may reuse experience from the experience base (e.g., a measurement plan with its tailoring effort), but not record *updated* experience. For instance, the tailoring of the existing measurement plan may not have been performed because it required more effort than creat-

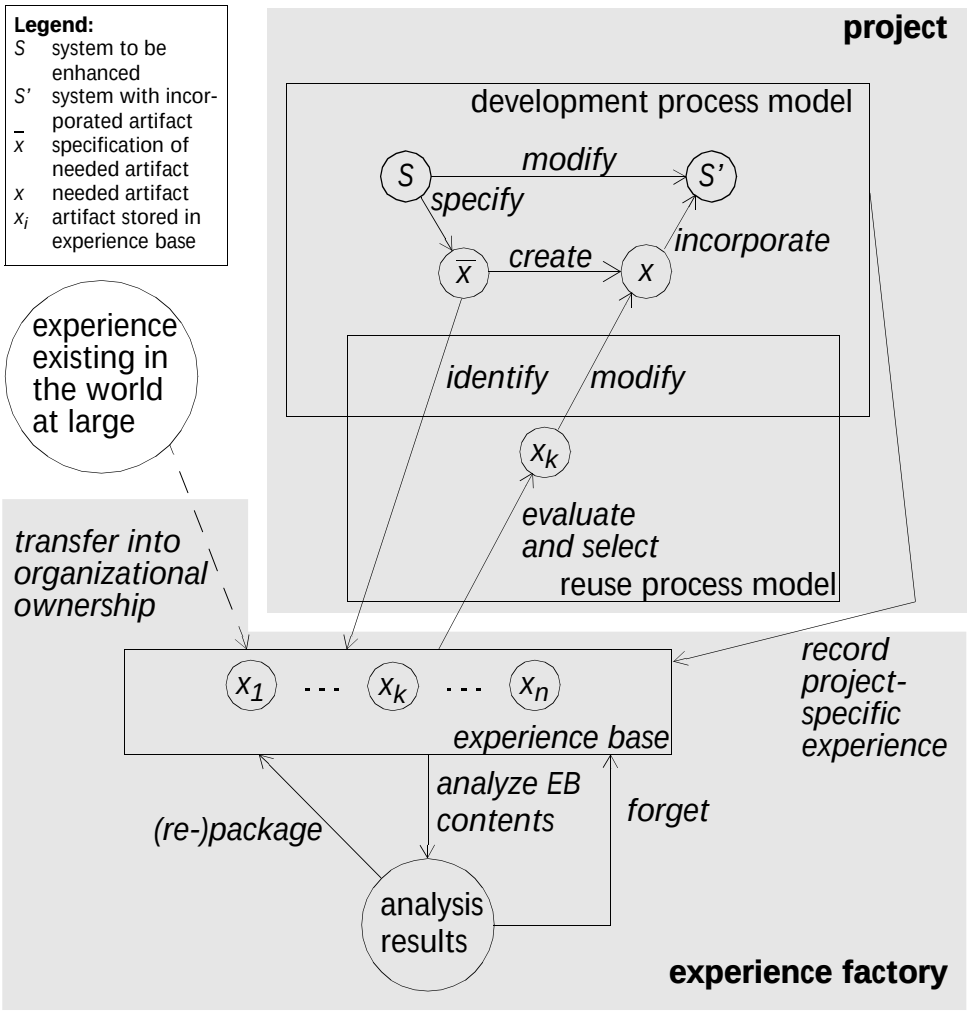


Figure 12: Reuse-oriented software development model based on [BR91]

ing a new one from scratch. Instead, the new measurement plan is recorded. Although the decision to create a new measurement plan was made based on the retrieved measurement plan and the reused estimated »tailoring effort«, neither the measurement plan nor the tailoring effort was updated.

The rest of the chapter will decompose the »reuse« and »learn« tasks further. The »reuse« task is decomposed first, because the information needs of a project define what must be learned. These information needs must therefore be considered when decomposing the »learn« task.

Task description

Each task is described using the following information:

- **Task number.** Each task has a unique number for reference purposes.

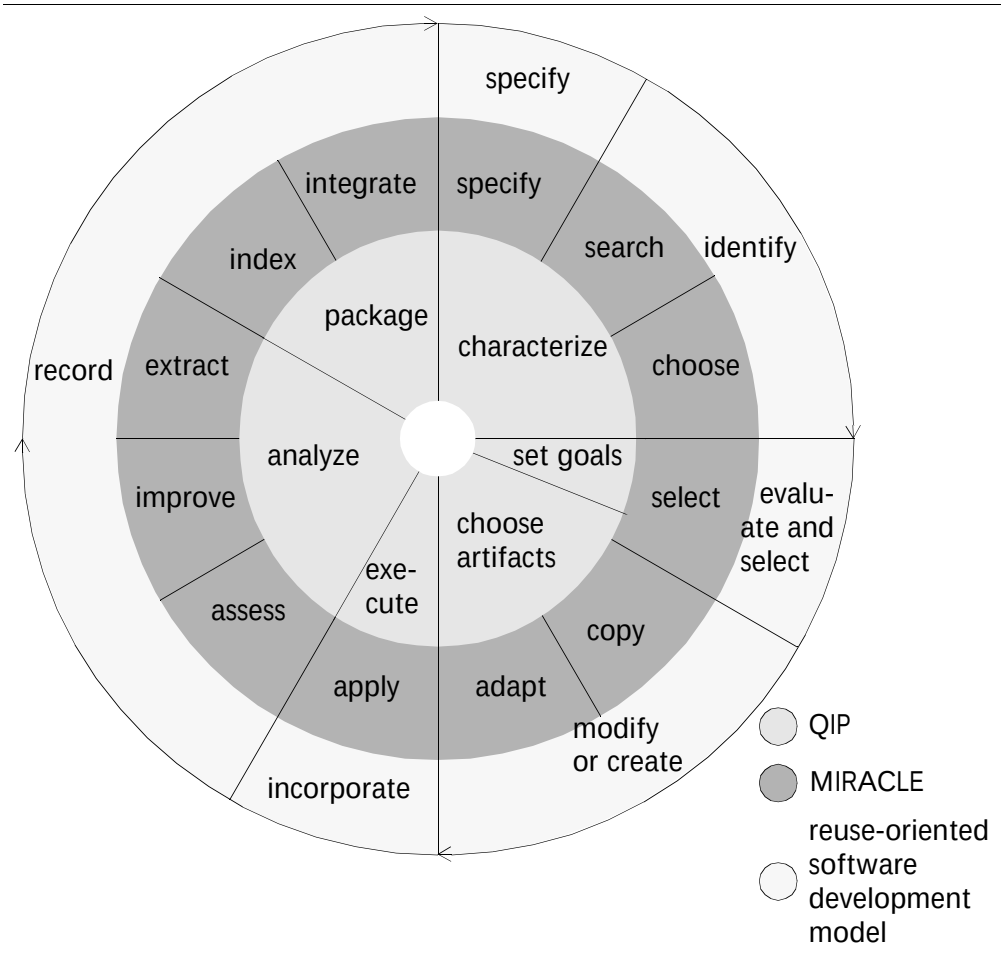


Figure 13: Matching MIRACLE with the QIP and the reuse-oriented software development model

- **Task identification.** Each task has a unique title that reminds the reader of its contents.
- **Objective.** The goal of the task.
- **Inputs and outputs.** The interface of the task. The phrase »state of the practice« refers to the organization´s state of the practice in form of an experience base.
- **Type.** The type specifies how the task can be operationalized by a method. Possible types are:
  - **Generic.** The task can be accomplished using a generic method that is independent of the kind of artifact for which the task is performed. Typically, this means that the method (see below) operates only on the artifact´s characterization.
  - **Externally triggered.** Although subtasks can be identified, the exact order in which the subtasks are to be accomplished cannot be foreseen, because the order depends on triggers external to the task. Hence, the order of execution is not under control of the method.

- **Artifact-specific.** The task cannot be completed without knowledge about the artifact itself. The method for completing such tasks typically makes assumptions about the structure of an artifact. Therefore, a decomposition into subtasks cannot be specified in general. A typical example for such a task is the editing of an artifact. For instance, while measurement plans can be edited using a specialized measurement plan editor enforcing a certain process for developing the plan, a code component may be edited using a simple text editor which does not provide any further support.
- **Context-specific.** The way the task is accomplished depends on the contents of the artifact. Although subtasks to be performed can be identified, their exact order cannot be specified due to an incomplete understanding of the task. Such tasks require the interaction of a human. In contrast to externally triggered tasks, the order in which the subtasks are to be performed does not depend on an external event. A typical example is the tailoring of an artifact to given project-specific requirements.

Buildup of method library

The type is of central importance for building up libraries of methods. Generic and externally triggered tasks need to be tailored only once to organization-specific needs. Artifact-specific tasks must be instantiated for each kind of artifact stored in the experience base. However, a library can be built up for artifact-specific tasks. The selection of an artifact-specific method depends on (a) the goal of the task and (b) the available conceptual knowledge about the artifact [Stu99]. Finally, context-specific tasks cannot be tailored (with the current state of the art). However, as our understanding of these tasks increases, they may eventually be classified as »generic« or »artifact-specific«.

- **Decomposition.** Subtasks of the task (if existent). Optional subtasks, that is, subtasks not performed for every instance of the task, are printed in *italics*.
- **Method.** Either a description of how to perform the task directly (in case there is no decomposition) or the order in which the subtasks are performed.
- **Requirements.** The set of requirements from Chapter 3 that must be fulfilled to support the method effectively. Note that the *actual* set of requirements also includes refinements of the listed requirements as well as those requirements upon which the listed requirements depend (see Figure 6 on page 67).

  Moreover, the listed set of requirements constitutes a necessary set of requirements for supporting the task as only those requirements are listed that are necessary for performing the method. A sufficient set of requirements includes the requirements of all the subtasks.

Task T1: mature

Objective: build up competencies through learning from experience

Input: state of the practice (of the organization in form of an initial experience base); artifacts from project organization other than modified artifacts; tacit knowledge from project organization; experience existing in the world at large

Output: improved state of the practice in form of an updated experience base; information about new artifacts stored in the experience base; training material; change plan

Type: externally triggered

Decomposition: *learn (T32, page 106)*, *reuse (T2, page 87)*

Method: As pointed out earlier, the subtasks are triggered by events outside the scope of the task. For example, »reuse« is triggered by a project request whereas »learn« is either triggered by the availability of new artifacts from a project or internally by the experience factory through some periodic event. However, there is the obvious restriction that an artifact must be »learned« before it can be reused.

This task is special in the sense that it does not operate on a single artifact but on the whole experience base. It is also never completed, because maturing is a continuous activity. Its inputs are supplied continuously to the task and its outputs are produced continuously as well. Of course, the task may be aborted for political reasons.

Requirements: R2 (support for incremental, continuous learning)

4.3 Reuse Task

Prieto-Díaz and Freeman state that code reuse involves the three steps of (a) accessing, (b) understanding, and (c) adapting the code [PDF87]. Barnes and Bollinger point out that adapting a retrieved component is not enough [BB91]: The adapted components have to be combined (*compositional reuse*). Compositional reuse [BP84] develops a system by assembling existing parts together, possibly creating »glue« between the parts if they do not fit. Thus compositional reuse becomes a configuration problem: Out of a library a set of artifacts must be chosen that – if assembled together – require minimal glue between them. How this can be done is shown, for example, by Penix and Alexander [PA97]. Consequently, Biggerstaff lists four steps needed for effective reuse: (a) find, (b) understand, (c) modify, and (d) compose artifacts [BR87].

Another approach is *generative reuse* [BP84]. Here, patterns are reused instead of finished building blocks. For example, Neighbor´s Draco system generates code from requirements formulated in an application-specific language [Nei84]. In this case, patterns describe the transformation from requirements constructs to the constructs of the target implementation language.

Hybrid reuse [GW95] combines compositional and generative reuse. The approach uses *hybrid domain-specific kits* that consist of

> *reusable components, a carefully designed framework which captures the architecture or shape of a family of related products, and some form of »glue code« which could be either a typical language such as C or C++ or some problem-oriented language specific to the application domain.* *[GW94]*

Considering these remarks results in a refinement of the reuse-oriented software development model of Basili and Rombach [BR91] and the decomposition presented in [ABH+99].

Task T2: reuse

Objective: save effort while delivering better quality in less time

Input: specification of needed experience; system the experience is to be applied in; feedback indicators; state of the practice

Output: modified artifact; application experience for utilized artifacts; access statistics/user feedback (for improving the software engineering experience management system)

Type: generic

Decomposition: retrieve (T3, page 89), utilize (T22, page 100)

Method: The performance of this task depends on the artifacts available in the experience base and the experience needed. Needed experience may be [ABH+99]:

1. A new artifact
2. (Tacit) knowledge from an expert
3. Information for supporting management, e.g., for identifying problems and risks

Code reuse is an example of the first case and has been described above. Existing building blocks (for compositional reuse) and patterns (for generative reuse) are retrieved to construct the needed artifact. In case a pattern is returned, a second retrieval task becomes necessary for the identification of the appropriate generator. Several iterations of retrieving and utilizing an

artifact may be necessary. Iterations are necessary for composing an artifact from several building blocks. That is, if no artifact is available that fulfills the requirements completely (even if it is modified), a partial solution may be applied. Then the retrieve-utilize cycle is repeated for the »remaining gap«. Iterations may also become necessary if the task »utilize« fails (e.g., if the subtask »understand« reveals that the retrieved artifact is not appropriate).

In the second case, the knowledge required is not stored in the experience base. However, the experience base may contain a pointer to somebody who has the knowledge. The retrieval result is utilized by talking to a person! The answer of the person is the »modified artifact« output by the task.

In the third case, the returned experience is not modified explicitly. The information is retrieved and analyzed. The analysis corresponds to »utilize« and the analysis results represent the »modified artifact« output by this task.

Requirements: R18 (various characterizations of one artifact)

4.3.1 Retrieval Task

Retrieval plays a key role in achieving effective reuse. If existing information is not found, organizations run the risk of reinventing wheels. A simple solution would be to list all artifacts stored in the experience base. A user could look through the list and pick out appropriate artifacts. However, such a list could overwhelm the user with information, and

> *from the perspective of a software production staff, profitability depends largely on component availability and on easily locating components, assessing their applicability, and incorporating them into the software system being developed.*
>
> *[WES87, p. 53]*

The »easy locating« brings us to the following task:

Information filtering

> *Another important task is information focusing and filtering, i.e., to find the type of information that is relevant in a particular context. Given the huge amount of information available for most professional tasks, and the accelerating increase of information, this is currently a problem that concerns many researchers in the information system and AI fields. [AN95, p. 21]*

Therefore, the aim for tool support of the retrieve task is to return all relevant information items, but no more. From these items, the user may select the most appropriate item(s). The notion of *relevance*, however, cannot be defined objectively [Har92]. What is relevant to the user may change even while performing

the retrieval. Thus, relevance is a highly dynamic concept. We will come back to this issue in more detail in Chapter 9. All this amounts to aiding the decision-making process on which artifacts to utilize. This is reflected by Richter's definition of the task »decision support«:

Decision support

> *The role of decision support is to help decision makers, rather than to replace them. In many situations, there is no precisely stated problem but rather a complex problem structure. The output of a support system is usually not a solution but rather an advice or a useful piece of information.*
>
> *[Ric98, p.14]*

Auxiliary information

The quality of decision support depends on the information available about the artifacts. For example, auxiliary information such as the artifact's quality, popularity, importance, reception by critics, status of its authors, and the accuracy of the artifact´s characterization may greatly influence the decision-making process [Coo97]. Moreover, policies may be enforced using auxiliary information. For instance, »standard« solutions should be used unless there are good reasons to use »special« solutions. Such a policy requires the documentation of »standard« and »special« solutions. »Special« solutions must state under what circumstances they should be preferred over the »standard« solution [Bur98].

The technical infrastructure of an experience base should be flexible enough to accommodate any auxiliary information deemed necessary for the effective decision support. In the following, we will abstract from concrete auxiliary information since the needed information can be encoded in the conceptual knowledge underlying the experience base.

Task T3: retrieve

Objective: find the most appropriate artifacts for the problem at hand

Input: specification of needed experience; state of the practice

Output: characterizations of artifact(s) to be utilized[1]; access statistics/user feedback

Type: generic

1 It is assumed that all relevant experience is captured in artifacts. For example, »pointers« to human experts are described by the topics on which an expert can be consulted. Here, the listing of the topics constitutes the artifact.

Decomposition: *specify (T4, page 90)*, identify (T8, page 92), evaluate (T11, page 94), select (T19, page 98) (cf. [AP94, ABH+99, BR91, IK96, Sen97, TA98, ANT99b])

Method: Generally, the subtasks are performed sequentially from left to right. Two conditions may result in an iteration after evaluate, that is, in a re-specification:

- Too many artifacts have been returned by the system; better filtering is needed
- Inappropriate artifacts have been returned; the specification must be restated more precisely

The second condition may also occur as a result of select.

The task *specify* is performed for systems using queries. For systems supporting only browsing, the task is not performed.

Note: The sequence »identify«, »evaluate«, and »select« represents a three-stage process for finding adequate artifacts. It extends the two-stage process of the MAC/FAC model (many are called but few are chosen) [GF91] by a third (human-based) process step:

1 A cheap computational process is used to restrict the large number of artifacts stored in the experience base.
2 A more expensive computational process judges the usefulness of the artifacts part of the restricted set, thereby restricting the set even further and also recommending the best candidate artifact.
3 A human makes the final decision by investigating the artifacts´ characterizations and – if necessary – by viewing the artifacts themselves.

Requirements: none

Task T4: specify

Objective: formulate query

Input: specification of needed experience; optionally the final query of an earlier »specify« task

Output: final query

Type: generic

Decomposition: collect descriptors (T5, page 91), interpret problem (T6, page 91), infer descriptors (T7, page 92) [AP94]

Method: Generally, the subtasks are performed sequentially from left to right. If the subtask »interpret problem« rejects the query, the task is restarted. In

this case, the subtask »collect descriptors« is started with either the »initial query« produced during the first iteration, the »checked query« as produced by »interpret problem«, or by the »final query« of an earlier »specify« task.

Requirements: none

Task T5: collect descriptors

Objective: formulate initial query

Input: specification of needed experience; optionally: the »final query« of an earlier »specify« task, the »initial query« of an earlier »collect descriptors« task, or the »checked query« of an unsuccessful »interpret problem« task.

Output: initial query

Type: generic

Decomposition: –

Method: The specification of the needed experience is expressed in the language of the retrieval system. This means that the needed experience is described using descriptors: Attribute values, keywords, formal formulas, etc.

The collection of descriptors can start with:
- An empty query form
- A predefined query
- A query issued earlier during the reuse task

Requirements: R9 (separation of characterization and conceptual information), R35 (tolerance of incomplete query information), R36 (tolerance of uncertain query information), R37 (tolerance of imprecise query information)

Task T6: interpret problem

Objective: correct descriptors if necessary

Input: initial query

Output: checked query

Type: generic

Decomposition: –

Method: It is checked whether the specified descriptors of the initial query make sense within the context. For example, specifying a quality model for the number of defects of a development process does not make sense,

because the number of defects is a measure for work products. Two strategies are possible:
1 Reject initial query
2 Disregard descriptors

For descriptors unknown to the system there are also two strategies:
1 Ask the user for an explanation (e.g., how similar the descriptors are to descriptors known to the system)
2 Disregard the descriptors

Requirements: R25 (integrity constraints)

Task T7: infer descriptors

Objective: complete checked query

Input: checked query

Output: final query

Type: generic

Decomposition: –

Method: Using general knowledge (e.g., in the form of completion rules) or case-specific knowledge (e.g., through recalling characterizations using exact matches and disregarding those descriptors which are irrelevant or inappropriate for the current problem), the checked query is completed by the system.

Requirements: R25 (integrity constraints)

Task T8: identify

Objective: find a set of artifacts with the potential to satisfy project-specific reuse requirements

Input: state of the practice; specification of needed experience; optionally: final query

Output: set of characterizations of potential artifacts

Type: generic

Decomposition: *browse (T9, page 93)*, *search (T10, page 93)*

Method: For small collections (up to about 50 artifacts), browsing will suffice to find potential artifacts [GFW94]. For larger collections, more sophisticated

search techniques are necessary. Therefore, the subtasks represent alternatives. If none of the subtasks are performed, the set of artifacts stored in the experience base cannot be restricted. In this case all stored artifacts are viewed as potential artifacts.

Requirements: none

Task T9: browse

Objective: find a set of potential artifacts using navigation

Input: state of the practice; specification of the needed experience

Output: set of characterizations of potential artifacts

Type: generic

Decomposition: –

Method: A list of potential artifacts is found by navigating along a predefined hierarchy.

Requirements: R6 (network access), R32 (browsing)

Task T10: search

Objective: find a set of potential artifacts using »knock out« criteria

Input: state of the practice; final query

Output: set of characterizations of potential artifacts

Type: generic

Decomposition: –

Method: The set of all stored artifacts in the experience base is filtered using »knock out« criteria. An important »knock out« criterion is the kind of artifact sought after. For example, if a user requests a requirements document, he is certainly not interested in project schedules. Another »knock out« criterion is the status of an artifact. Typically, obsolete artifacts should not be returned as the result of a search.

Other »knock out« criteria may be defined, but are usually artifact-specific. For example, if a quality model for processes is requested, quality models for products may be discarded.

Requirements: R6 (network access), R7 (storage of various kinds of artifacts), R27 (artifacts' status)

Task T11: evaluate

Objective: (a) characterize the degree of discrepancies between potential artifacts and needed experience; (b) predict relative net benefit of utilizing potential artifact

Input: final query; set of characterizations of potential artifacts

Output: prioritized list of characterizations of potential artifacts; access statistics/user feedback

Type: generic

Decomposition: recommend (T12, page 95), *examine (T16, page 97)* [ANT99b]

Method: First »recommend« is performed, then »examine«.

Note 1: The decomposition of »evaluate« into two subtasks has been made, because the task can be partially automated. The task »recommend« represents the automatable part whereas »examine« represents the part that must be performed by the user. The exact border depends on the application knowledge available. If precise expert knowledge about the applicability of an artifact exists, then »examine« does not need to be performed [Bir97].

The method described here assumes that the system cannot decide whether a given artifact can be utilized (with modification) in a given context. Therefore, it returns similar artifacts that have been utilized in similar contexts. The applicability decision is left to the user.

Note 2: The net benefits of the artifacts do not only depend on the experience requested, but also on the quality of the artifacts themselves. This means that artifacts must exhibit a minimal quality (e.g., artifacts that have not been validated will not be reused)[1]. Hence, the artifact's quality should be part of its characterization. Because the task of evaluating is based on the artifacts' characterizations, the quality of the characterization (e.g., completeness and accuracy) is also of great importance for successful reuse. Hence, the learning tasks must ensure a high quality of the characterizations.

1 This is a requirement for the recording of an artifact. Artifacts not fulfilling the minimal quality requirements should not be recorded.

During the evaluation, a trade-off between the artifact's quality and the cost of modifying it to the project-specific needs must be made.

Requirements: none

Task T12: recommend

Objective: suggest an ordering of the potential artifacts based on their characterizations

Input: final query; set of characterizations of potential artifacts

Output: ordered list of characterizations of potential artifacts augmented with the similarity between the final query and each potential artifact

Type: generic

Decomposition: calculate similarity (T13, page 95), explain similarity (T14, page 96), sort (T15, page 96)

Method: For each characterization of a potential artifact, the similarity between the query and the characterization of the potential artifact is calculated and explained. Thus, a similarity value is associated with each characterization. The characterizations are then ordered by decreasing similarity value associated with them.

Requirements: none

Task T13: calculate similarity

Objective: compute similarity between the final query and a potential artifact

Input: final query; characterization of potential artifact

Output: similarity value; similarity computation trace

Type: generic

Decomposition: –

Method: The actual computation depends on the representation chosen for the characterizations. However, a common strategy is to compute local similarities for each descriptor of the query and to combine these local similarities to a global similarity for the whole characterization.

The similarity computation function is part of the conceptual knowledge of the experience base.

Requirements: R9 (separation of characterization and conceptual information), R20 (tolerance of incomplete information), R21 (tolerance of uncertain information), R22 (tolerance of imprecise information), R23 (transparency of duplicated information), R24 (tolerance of inconsistent information), R33 (textual search), R35 (tolerance of incomplete query information), R36 (tolerance of uncertain query information), R37 (tolerance of imprecise query information), R39 (context-sensitive retrieval)

Task T14: explain similarity

Objective: explain the ordering of the artifacts by explaining the similarities for each potential artifact

Input: similarity computation trace

Output: similarity explanation

Type: generic

Decomposition: –

Method: Using the trace information generated by the similarity calculation, an explanation is produced for the user how the similarity value was derived.

Requirements: R34 (similarity-based retrieval)

Task T15: sort

Objective: order the potential artifacts in decreasing order of their (predicted) net benefit

Input: set of characterizations of potential artifacts; similarity value for each characterization; similarity explanation for each characterization

Output: ordered list of characterizations of potential artifacts augmented with the similarity between the final query and each potential artifact

Type: generic

Decomposition: –

Method: Sort the characterizations of potential artifacts in the order of decreasing similarity.

Requirements: R34 (similarity-based retrieval)

Task T16: examine

Objective: prioritize set of potential artifacts and give feedback for improving the retrieval system

Input: ordered list of characterizations of potential artifacts augmented with the similarity between the final query and each potential artifact

Output: prioritized list of characterizations of potential artifacts; access statistics/user feedback

Type: generic

Decomposition: prioritize (T17, page 97), explain prioritizing (T18, page 98) [ANT99b]

Method: First »prioritize«, then »explain prioritizing« is performed.

Requirements: none

Task T17: prioritize

Objective: restrict the set of potential artifacts and prioritize the recommended set of potential artifacts using their characterizations

Input: ordered list of characterizations of potential artifacts augmented with the similarity between the final query and each potential artifact

Output: prioritized list of characterizations of potential artifacts

Type: generic

Decomposition: –

Method: The characterizations of the potential artifacts are examined in the order suggested by the input. Using the characterizations and additional background knowledge of the user, the user decides whether the application of the artifact might have a net benefit. All artifacts whose net benefit is estimated to be negative are disregarded. The rest of the characterizations are ordered by decreasing (estimated) benefit of the artifacts.

If not already specified in the query, interface and context information as well as other auxiliary information is used during this task. Also, browsing may be used to examine related information (e.g., application history of or lessons learned about an artifact). Therefore, documenting explicitly the relationships between artifacts does not only allow to search for artifacts that have been used in similar contexts, but also to better understand the artifacts.

Requirements: R4 (administrative information), R5 (access rights), R29 (configurations), R31 (artifact preference information), R32 (browsing), R41 (interface information), R42 (application history)

Task T18: explain prioritizing

Objective: give feedback to improve retrieval system

Input: ordered list of characterizations of potential artifacts augmented with the similarity between the final query and each potential artifact; prioritized list of characterizations of potential artifacts

Output: access statistics/user feedback

Type: generic

Decomposition: –

Method: Ideally, the suggested ordering by the system matches the prioritization of the user. If not, the ordering may be improved. For this purpose, feedback from the user is needed. How such an explanation can be transformed into improvements of the software engineering experience management system (SEEMS) is subject of Chapter 9.

Requirements: R15 (data collection for evaluation)

Task T19: select

Objective: select best suited artifact(s)

Input: prioritized list of characterizations of potential artifacts

Output: characterizations of artifact(s) to be utilized; access statistics/user feedback

Type: generic

Decomposition: view (T20, page 99), *explain selection (T21, page 99)*

Method: First, the actual artifacts are viewed. Based on the outcome of this task the final selection is made. If this selection contradicts the input (e.g., the artifact with the highest priority is not selected), an explanation is requested from the user allowing the improvement of the SEEMS.

Requirements: none

Task T20: view

Objective: make final selection

Input: prioritized list of characterizations of potential artifacts

Output: characterizations of artifact(s) to be utilized

Type: artifact-specific

Decomposition: –

Method: The actual artifacts (not their characterizations) are viewed using some editor. Here, additional knowledge about the artifacts is collected. Through viewing, questions like »Is the artifact well-structured?« and »Can the artifact be easily understood?« can be answered.

The additional information may change the estimations for the net benefit of the artifacts. Those artifacts with the highest benefit are selected as the artifacts to be utilized.

Usually, only the best suited artifact is returned as the result of this task. However, sometimes several very similar artifacts may be merged effectively to fulfill the specification of the needed experience. In this case, a set of artifacts is the result of this task.

Requirements: R3 (tool integration), R5 (access rights)

Task T21: explain selection

Objective: give feedback to improve the SEEMS

Input: prioritized list of characterizations of potential artifacts; characterizations of artifact(s) to be utilized

Output: access statistics/user feedback

Type: generic

Decomposition: –

Method: Ideally, the best suited artifact is the one with highest priority. If not, the characterization information may be improved. For this purpose, feedback from the user is needed. How such an explanation can be transformed into improvements of the SEEMS is subject of Chapter 9.

Requirements: R15 (data collection for evaluation)

4.3.2 Utilize Task

Task T22: utilize

Objective: take advantage of existing experience

Input: specification of needed experience; characterizations of artifact(s) to be utilized; system the experience is to be utilized in; feedback indicators; state of the practice

Output: modified artifact; application experience for utilized artifacts

Type: context-specific (to the artifact(s) to be utilized)

Decomposition: *check-out (T23, page 101)*, understand (T24, page 101), *merge artifacts (T25, page 102)*, *modify artifact (T26, page 102)*, *change constraints (T29, page 103)*, incorporate (T30, page 104), give feedback (T31, page 104) [DvK95]

Method: After selecting the most appropriate artifact(s), the artifact(s) are utilized as follows:

1 A permanent local copy of the selected artifact(s) is made so it can be modified (»check-out«); this task is skipped in case the experience need is fulfilled by viewing the artifact (Task T20).
2 The selected artifact(s) are investigated until they are understood as far as necessary for their application (»understand«).
3 If more than one artifact has been selected, the artifacts are merged into a single artifact (»merge artifacts«).
4 The artifact (either the selected or the merged) is modified to suit the project-specific needs (»modify artifact«).
5 The system where the artifact is to be incorporated is modified (»change constraints«). This task is performed if it requires less effort to modify the system than to extend the artifact.
6 The artifact is incorporated into the system (»incorporate«).
7 Finally, feedback for future applications of the artifacts is given (»give feedback«). For this purpose, data (e.g., effort) may be collected during the first six steps.

Often, the subtasks 2 through 4 are performed iteratively.

Note: The term »system« is used here in a broad sense. A system can be anything from a software system to an environment in which a software system is developed. For example, if the artifact to be incorporated is a guideline, a modification of the system amounts to changing the constraints on the development project as a whole (thus the task name »change constraints«).

Requirements: none

Task T23: check-out

Objective: copy the selected artifact(s) into the project base

Input: characterizations of artifact(s) to be utilized

Output: artifact(s) to be utilized; data

Type: artifact-specific

Decomposition: –

Method: The actual artifacts are copied to the local environment where their copy may be modified. At the same time, the check-out is registered in the experience base. This is important so the user who checked-out the artifact can be informed in case some defects become known to the experience factory. The registration information can also be used to ask for application feedback after some predefined period.

Requirements: R40 (check-out of artifacts)

Task T24: understand

Objective: find out how to utilize the selected artifact(s)

Input: artifact(s) to be utilized

Output: understanding of the artifact(s) to be utilized; data

Type: artifact-specific (to the kind of artifacts to be utilized)

Decomposition: –

Method: The »golden rules of reusability« state: »Before you can reuse something, you need to (a) find it, (b) know what it does, and (c) know how to reuse it.« [Tra95, p. 93] The latter two are different aspects of understanding an artifact. While (b) has already been done during retrieval (Task T16 and Task T19), (c) typically needs some more effort.

However, the same methods can be used for this task as for examining (Task T16) and selecting (Task T19). Therefore, next to invoking a special editor for viewing the artifact(s), browsing can be employed [BR87].

For »pointers« to human experts, this task also involves talking to the experts. During the talks, tacit knowledge becomes explicit knowledge. If several experts are interviewed, the result of task »understand« are »extended« artifact(s) that need to be merged.

Note: The understanding of an artifact may be tacit (if the same person merges, modifies, and integrates the artifact) and/or explicit (in form of notes and documentation).

Requirements: R3 (tool integration), R5 (access rights), R6 (network access), R29 (configurations), R32 (browsing), R41 (interface information), R42 (application history)

Task T25: merge artifacts

Objective: combine several artifacts to a single artifact

Input: artifact(s) to be utilized; understanding of the artifact(s) to be utilized

Output: final artifact to be utilized; data

Type: artifact-specific (to the kind artifacts to be utilized)

Decomposition: –

Method: A new artifact is created by taking parts from each of the artifacts to be utilized. Perhaps, some »glue« must be inserted to connect the parts.

Requirements: R3 (tool integration)

Task T26: modify artifact

Objective: bridge the gap between the artifact to be utilized and the needed experience

Input: final artifact to be utilized or artifact to be utilized with an understanding of it (if merging of artifacts does not take place); specification of experience needed; state of the practice

Output: modified artifact; data

Type: context-specific

Decomposition: modify manually (T27, page 103), generate (T28, page 103)

Method: At least one of the subtasks must be performed. If patterns are utilized, the following sequence is typical. First the patterns are modified manually (»modify manually«), then an artifact is generated (»generate«), and finally the generated artifact may be modified manually (»modify manually«).

Note: The idea of integrating compositional and generative reuse can be traced back to Griss [Gri93] who introduced domain-specific kits [GW95].

Requirements: R3 (tool integration)

Task T27: modify manually

Objective: adapt an artifact to project-specific needs

Input: (final) artifact to be utilized or generated artifact; understanding of the artifact to be utilized

Output: modified artifact; data

Type: artifact-specific (to the kind of the modified artifact)

Decomposition: –

Method: The artifact is edited.

Requirements: R3 (tool integration)

Task T28: generate

Objective: generate artifact to be included in the final system

Input: (final) artifact to be utilized or modified artifact (in form of a pattern); state of the practice

Output: generated artifact

Type: generic

Decomposition: –

Method: First, an appropriate generator must be retrieved. Usually, the choice of generators is restricted to a small set, because it is related to the artifact to be utilized. Thus it can be retrieved using the browsing mechanism. Then the generator is invoked to generate the wanted artifact.

Requirements: R3 (tool integration), R32 (browsing)

Task T29: change constraints

Objective: adapt the system so the modified artifact can be incorporated readily

Input: system the experience is to be applied in; modified artifact or final artifact to be utilized (if modification does not take place) or artifact to be utilized with an understanding of it (if merging and modification does not take place)

Output: modified system; data

Type: artifact-specific (to the kind of artifact the system represents)

Decomposition: –

Method: The system is changed by considering the modified artifact to be incorporated.

Requirements: R3 (tool integration)

Task T30: incorporate

Objective: incorporate the modified artifact into the system

Input: modified artifact or final artifact to be utilized (if modification does not take place) or artifact to be utilized with an understanding of it (if merging and modification does not take place); modified system or system the experience is to be applied in (if modification of the system does not take place)

Output: extended system; data

Type: artifact-specific (to the artifact included and the meaning of the term »system« – see note under »utilize (T22, page 100)«)

Decomposition: –

Method: The artifact is incorporated into the system.

Requirements: R3 (tool integration)

Task T31: give feedback

Objective: give feedback to improve decision-making for future applications

Input: data collected during the previous tasks (e.g., effort); feedback indicators

Output: application experience for utilized artifacts

Type: generic

Decomposition: –

Method: The experience gained during the previous subtasks of the task »utilize« is made explicit. This not only includes the interpretation of data, but also the documentation of important observations. The feedback needed is guided by the characterization schema for application experience (feedback indicators).

Requirements: R42 (application history)

4.4 Learn Task

According to Arango,

> *learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same tasks more efficiently and more effectively the next time.*
>
> *[Ara89, p. 85]*

Three types of changes in the system can be distinguished:

1 Changes of the contents of the experience base (deletion, insertion, update of artifacts and their characterizations)

2 Changes of the conceptual knowledge underlying the contents (e.g., the addition of a new property to a characterization schema)

3 Changes of the functionality of software tools which are part of the technical infrastructure of the experience base (e.g., the addition of an editor for a particular kind of artifact)

In the rest of this section, changes of the first type will be discussed. Changes of the second type are subject of Chapter 8 and Chapter 9. Changes of the third type are subject to strategic and tactical decisions on part of the experience factory. These are not within the scope of this dissertation.

Among the existing learning strategies, three are of particular importance from the viewpoint of the experience factory [CMM84, Wii95]:

1 **Learning by being told.** Organized knowledge (e.g., in the form of deliverables) is provided by the project organization to the experience factory. The experience factory selects the most relevant facts or transforms the knowledge into more useful forms. They serve as examples for information requests by future projects. The application of this learning strategy is called »recording«.

2 **Learning by observation or discovery.** Based on the analysis of observed or presented entities in provided material, some of the entities are classified into a preexisting or new representational structure, which can characterize or even explain the material. The result of this learning strategy is *experience* (see also Definition 5 on page 24). The application of this learning strategy is called »characterizing«.

3 **Learning from examples.** New knowledge is created from existing examples and possibly counterexamples. The application of this learning strategy is called »packaging«. Usually, an analysis is required to find relevant examples and counterexamples. As the result of creating new knowledge, old knowledge (examples and counterexamples) may become obsolete. Thus, it should be removed from the experience base.

The discussion leads to refinements of the notions of »recording« and »packaging« as introduced by Basili and Rombach [BR91] and extends them by introducing the notions of »analyzing« and »forgetting«.

Task T32: learn

Objective: make decision support more efficient and effective

Input: state of the practice; artifacts from project organization other than modified artifacts; tacit knowledge from project organization; experience existing in the world at large; modified artifacts; application experience for utilized artifacts; access statistics/user feedback

Output: improved state of the practice; information about new artifacts stored in the experience base; training material; change plan; feedback indicators

Type: externally triggered

Decomposition: *record (T33, page 107)*, *analyze SEEMS (T50, page 116)*, *forget (T54, page 119)*, *package (T55, page 120)*

Method: The task is triggered either by events external to the experience factory, by periodic or internal events. External events may be:

- The availability of new artifacts or the request of a project to make tacit knowledge explicit. In this case, the »record« subtask is performed. Afterwards, the task »analyze SEEMS« may be performed to check whether the tasks »forget« and/or »package« must be performed. If new contents are produced as the result of »package« the cycle is iterated. In case, a new version of an already existing artifact is inserted into the experience base, the old version needs to be removed (task »forget«).
- Changes of the environment, for example, the request to build up knowledge in a new competence area. In this case, the task »package« is invoked. If the »package« produces new contents, these contents have to be recorded (task »record«).

Periodically or whenever is time, the task »analyze SEEMS« is performed. As a result, »forget« and/or »package« and – if »package« produces new contents – »record« may have to be performed. The cycle (»analyze SEEMS« – »package« – »record«) may be iterated until »analyze SEEMS« produces no more improvement suggestions.

Requirements: R2 (support for incremental, continuous learning), R14 (maintenance of experience packages)

4.4.1 Record Task

Recording of new experience packages is the only way of expanding the collection of the experience base. Thus, the task »record« is vital for increasing the competence of the experience base and to improve the state of the practice.

Task T33: record

Objective: add competence to the experience base

Input: state of the practice; modified artifacts; artifacts from project organization other than modified artifacts; tacit knowledge from project organization; experience existing in the world at large; application experience for utilized artifacts; changed experience packages; aggregated artifacts

Output: improved state of the practice; information about new artifacts stored in the experience base

Type: generic

Decomposition: collect (T34, page 108), *store (T35, page 108)*, *qualify (T39, page 110)*, *publish (T48, page 115)*, *inform (T49, page 115)* [vvK96]

Method: In general, the subtasks are performed sequentially from left to right. New artifacts may be rejected during »collect« or »qualify«. If the new artifact is rejected during »collect«, none of the other subtasks are performed. If the new artifact is rejected during »qualify«, »publish« and »inform« are not performed.

Each of the tasks operates with a different granularity. The output of »collect« is a set of artifacts as delivered by the project organization or available in the world at large. For each of these artifacts, »store« is performed. The subtask »store« splits these artifacts into reusable parts (for example, a code component may be split into its functions to not only allow reuse on the component level, but also on the function level). Each of these parts is qualified. If qualified successfully, the part is published and – if the policy demands – the publication information is disseminated to the users of the experience base (»inform«).

Requirements: R7 (storage of various kinds of artifacts), R26 (accommodation of growing collection)

Task T34: collect

Objective: capture new or improved artifacts

Input: modified artifacts; artifacts from project organization other than modified artifacts; tacit knowledge from project organization; experience existing in the world at large; application experience for utilized artifacts; changed experience packages; aggregated artifacts

Output: location of artifacts to be stored

Type: artifact-specific

Decomposition: –

Method: There are many methods to collect artifacts. They include the whole range of knowledge acquisition methods (e.g., [Lio98]). In principle, from the viewpoint of the experience factory the collection can be performed actively or passively [vvK96, ABH+99]. Active collection means that the experience factory scans the organization for existing artifacts, or mediates in explicating tacit knowledge. Passive collection means that the project organization or the experience factory (as the result of a »package« task) suggest new artifacts to be stored.

Examples for active collections are the creation of learning histories [KR97], domain analysis methods (e.g., [Ara94, FPDF98, KKL+98, Nei84, Sim91]), ethnographic analysis (e.g., [Wii95], and postmortem meetings (e.g., [CDF96]).

The task can also be used to initiate further actions. For instance, if a new project is started, its characterization should be recorded in the experience base [ABH+99]. At the same time, goals for filling »knowledge gaps« can be negotiated with the project team.

Requirements: none

Task T35: store

Objective: make a new experience package available in the experience base so it can be qualified

Input: location of an artifact to be stored; tacit knowledge from project organization

Output: set of initial characterizations of reusable parts

Type: generic

Decomposition: copy (T36, page 109), split (T37, page 109), characterize initially (T38, page 110)

Method: Sequential performance of »copy«, »split«, and »characterize initially«.

Requirements: none

Task T36: copy

Objective: make a new artifact available for local modification

Input: location of an artifact to be stored

Output: new artifact

Type: artifact-specific

Decomposition: –

Method: The artifact is copied to the local experience base environment where it can be modified.

Requirements: R3 (tool integration), R5 (access rights)

Task T37: split

Objective: identify the parts of the artifact that must be characterized

Input: new artifact

Output: set of reusable parts

Type: artifact-specific

Decomposition: –

Method: The reusable parts of the artifact are identified. In the simplest case, only the artifact as a whole is reusable. In this case, there is only one reusable part – namely the artifact itself.

Note: Recall that the granularity of the artifact is defined by the available tool set (see Definition 10 on page 27). To enable reuse on a finer level of granularity, reusable parts of an artifact are identified as part of the task »record«. Consequently, it is not enough to characterize only the artifact. Rather, there must be a characterization for each reusable part. The artifact is always considered as a reusable part itself.

Requirements: R3 (tool integration)

Task T38: characterize initially

Objective: store available information about the parts

Input: set of reusable parts; tacit knowledge from project organization

Output: set of initial characterizations of reusable parts

Type: generic

Decomposition: –

Method: The characterization can be entered manually or it can be exported from a specialized tool or both. The characterization includes context information (i.e., relationships to other reusable parts in the experience base). Although a characterization is stored, the parts must not appear as the result of a query (yet), because the parts have not been qualified yet. This is achieved by attributing the status »submitted« to the parts.

Note: It is also possible that a set of (consistent) artifacts is to be reused as a single entity. For instance, a code component may be reused including its specification and test cases. Although such an aggregated artifact does not exist in reality, a characterization for this »virtual« artifact may be created. Consequently, the characterization of the aggregated artifact must be updated if an artifact is to be stored that »belongs« to the aggregated artifact. Such relationships are part of the context information.

Requirements: R3 (tool integration), R5 (access rights), R7 (storage of various kinds of artifacts), R18 (various characterizations of one artifact), R19 (tolerance of different levels of abstraction), R20 (tolerance of incomplete information), R21 (tolerance of uncertain information), R22 (tolerance of imprecise information), R24 (tolerance of inconsistent information), R27 (artifacts' status), R38 (context information)

Task T39: qualify

Objective: accept (with modification) or reject a new part

Input: state of the practice; initial characterization of a reusable part

Output: complete characterization of the reusable part (if accepted)

Type: generic

Decomposition: analyze part (T40, page 111), review (T43, page 112), evolve (T46, page 114), complete characterization (T47, page 114)

Method: First the part is analyzed. If the analysis reveals major shortcomings, the artifact is rejected or evolved and analyzed again. Otherwise it is

reviewed. While the analysis is solely done by the experience factory (preparation of the review), users (i.e., people from the project organization) participate in the review. After the review, the part is evolved according to the review results. Finally, the characterization of the (accepted) part is updated and/or completed.

Note: Experience has shown that there needs to be a defined process that controls the contents, structure, and validity of experience packages [Hen97a, Hen97c] and that encourages reuse by making the artifacts »easy to find, adapt, and integrate into new systems« [BB91, p. 18]. How much can be invested in such efforts depends among others on the expected levels of reuse.

Requirements: none

Task T40: analyze part

Objective: prepare review

Input: initial characterization of reusable part; state of the practice

Output: set of characterizations of similar parts already stored in the experience base; test results

Type: generic

Decomposition: analyze quality (T41, page 111), check for existing parts (T42, page 112) [Ara94]

Method: The subtasks can be performed in any order.

Requirements: none

Task T41: analyze quality

Objective: check whether minimal quality requirements are fulfilled

Input: initial characterization of reusable part

Output: test results (degree to which minimal quality requirements are fulfilled)

Type: artifact-specific

Decomposition: –

Method: Data is collected for quality factors defined objectively (e.g., size of description).

Requirements: R3 (tool integration)

Task T42: check for existing parts

Objective: check whether similar artifacts already exist in the experience base

Input: initial characterization of reusable part; state of the practice

Output: set of characterizations of similar parts

Type: generic

Decomposition: –

Method: The initial characterization is used to retrieve similar parts. The aim is to find parts that are viable alternatives for the new part (so replacement or merging can be suggested by the reviewers) or that contradict the new experience (in which case it must be documented when to utilize which part). Properties of the retrieved parts that differ from the properties of the new part are marked. The context of the parts found is also documented.

Requirements: R20 (tolerance of incomplete information), R21 (tolerance of uncertain information), R22 (tolerance of imprecise information), R24 (tolerance of inconsistent information), R34 (similarity-based retrieval), R35 (tolerance of incomplete query information), R36 (tolerance of uncertain query information), R37 (tolerance of imprecise query information), R38 (context information)

Task T43: review

Objective: reject or accept and classify part

Input: initial characterization of reusable part; test results; set of characterizations of similar parts

Output: classified part; suggested improvements (if accepted)

Type: generic

Decomposition: analyze reuse potential (T44, page 113), suggest improvements (T45, page 113)

Method: First, it is determined whether the part has reuse potential. If so, the part is accepted and improvement suggestions are made. If not, the part is rejected.

Requirements: none

Task T44: analyze reuse potential

Objective: classify part

Input: initial characterization of reusable part; test results; set of characterizations of similar parts

Output: classified part (classification includes initial characterization of the part and the test results)

Type: artifact-specific

Decomposition: –

Method: The reusable part is reviewed to make the decision on whether to reject or publish the part or whether to merge it with other (existing) parts. Both the part itself and its characterization are reviewed. The part is reviewed regarding its contents and its layout/structure.

In the following, exemplary factors are listed that can be used for making the decision on publishing:

- Difficulty/complexity of problem solved by the part [BB91]
- Usefulness for third parties [LSH98]
- Understandability [Mat84b]
- Definiteness (degree of clarity to which the part´s purpose, capability, constraints, and interfaces are defined) [Mat84b]
- Adaptability [Mat84b]
- Relevance [Sar95]
- Newness/innovativeness of part [vvK96]
- Consistency with already stored parts [vvK96]
- Generality [vvK96]

The review will lead to good, typical, important, misleading, or unnecessary parts. Such preference information should also be part of the review result.

Requirements: R3 (tool integration), R31 (artifact preference information)

Task T45: suggest improvements

Objective: formulate change requests to improve reuse potential

Input: classified part; set of characterizations of similar parts

Output: suggested improvements

Type: artifact-specific

Decomposition: –

Method: Based on the analysis results, improvements are suggested. Typical reuse scenarios should be identified as part of this task. Suggested improvement do not need to be limited to the new classified part. Examples are [Ara94]:

- Merge similar parts/artifacts into a single part/artifact
- Merge variants using parameterization
- Replace an existing part/artifact (amounts to creating a new version)

Requirements: R29 (configurations)

Task T46: evolve

Objective: improve reuse potential

Input: classified part; set of characterizations of similar parts; suggested improvements

Output: improved part

Type: artifact-specific

Decomposition: –

Method: The suggested improvements are implemented. For example, reuse scenarios are supported through a guideline on how to modify the part if it fulfills the needs of the reuse scenario. Also, defects that might have been detected during »analyze part« or »review« are corrected.

Requirements: R3 (tool integration)

Task T47: complete characterization

Objective: prepare publication

Input: improved part; classification of classified part; suggested improvements

Output: complete characterization of reusable part

Type: generic

Decomposition: –

Method: The characterization of the improved part is complemented by the analysis and review results. In case the new part replaces one or more old parts, the configuration and version information has to be updated too. This includes marking the old parts as obsolete (cf. task »forget«).

Requirements: R3 (tool integration), R5 (access rights), R20 (tolerance of incomplete information), R21 (tolerance of uncertain information), R22 (tolerance of imprecise information), R24 (tolerance of inconsistent information), R29 (configurations)

Task T48: publish

Objective: make the new artifact available for retrieval

Input: set of complete characterizations of reusable parts (of an artifact)

Output: improved state of the practice

Type: generic

Decomposition: –

Method: The artifact's status is changed to »published«.

Note: For some artifacts it may be necessary to publish them *preliminarily*. For example the characterization of a project that runs for several years should already be stored during the project so knowledge of that project can be stored with its context. At the end of the project, the characterization is updated and published as a regular artifact [ABH+99].

Requirements: R27 (artifacts' status)

Task T49: inform

Objective: notify potential users of the new artifact(s)

Input: set of complete characterizations of reusable parts

Output: information about new artifacts stored in the experience base

Type: artifact-specific

Decomposition: –

Method: There are many possibilities. Catalogs with artifacts published during the last month can be disseminated to the users of an experience base. Alternatively, users can be notified by e-mail every time an artifact is published. A subscription service (users subscribe to kinds of artifacts they are interested in) can restrict notification messages to relevant ones.

Other notification criteria include status changes (e.g., a person who checked out an artifact is notified if the artifact is replaced by a new version). These can be refined:

- **Corrective maintenance.** In case corrections are made, all persons utilizing the artifact should be notified.
- **Perfective/preventive maintenance.** In case the interface is extended (e.g., additional functionality for a code component) or a new variant of the artifact is created, the persons utilizing the artifact should be notified on a wanted basis. If the interface is changed, however, all persons utilizing the artifact should be notified.

Requirements: R30 (change notification)

4.4.2 Analysis Tasks

Analysis can be performed from different points of view [Nic98]:
- Organization as a whole
- Project
- Experience factory

Here, we are only concerned with the latter viewpoint. Analysis methods can aim at analyzing the utility of each artifact or the whole SEEMS.

Task T50: analyze SEEMS

Objective: find potential for improving the SEEMS

Input: state of the practice; access statistics/user feedback (for improving the SEEMS); set of defined objectives

Output: improvement suggestions

Type: generic

Decomposition: *analyze efficiency (T51, page 117)*, *analyze effectiveness (T52, page 118)*, *analyze artifact (T53, page 118)*

Method: At least one of the subtasks must be performed. Otherwise, there will be no improvement suggestion. The subtasks may be performed in any order, however, it is likely that improvements in efficiency and effectiveness can be realized through a detailed analysis on the artifact level (task »analyze artifact«). Therefore, a possible strategy is to take a look at the global indicators for efficiency and effectiveness first and then to analyze specific artifacts (e.g., based on the retrieval results of unsuccessful retrieval tasks).

In general, analysis can be restricted to an excerpt of stored artifacts. Examples of such groups are:
- Result of a retrieval task
- All artifacts of a particular kind

- All artifacts developed by projects with certain properties (e.g., small projects)
- All artifacts of a particular application domain
- All artifacts developed with a particular technology or process model
- All artifacts related to a particular artifact

As can be seen, analysis makes heavy use of context-sensitive retrieval. Through context-senstitive retrieval it is also possible to analyze the impacts of changes in the environment (e.g., legislative changes for a particular application domain or software development practice in general).

Requirements: R39 (context-sensitive retrieval)

Task T51: analyze efficiency

Objective: find suggestions for improving the efficiency of a SEEMS

Input: state of the practice; access statistics/user feedback; set of defined objectives

Output: improvement suggestions

Type: generic

Decomposition: –

Method: Indicators are examined to identify improvement potential and to make improvement suggestions regarding the defined objectives. Exemplary indicators are:

- Average effort needed to retrieve an artifact
- Average effort needed to get some required experience
- Average effort needed to record an artifact
- Average time elapsed from submission to publication of an artifact

The actual indicators used will vary from organization to organization and must be defined as part of the setup of a SEEMS (e.g., using the goal/question/metric technique [NT99]).

Exemplary improvement suggestions are:

- Change number of artifacts in the experience base
- Redefine tasks and methods for maturing
- Add relationships between artifacts in the artifacts´ characterization
- Package artifacts stored in the experience base (e.g., by combining several related artifacts into a single artifact)

Requirements: R16 (data analysis for improvement)

Task T52: analyze effectiveness

Objective: find suggestions for improving the effectiveness of a SEEMS

Input: state of the practice; access statistics/user feedback; set of defined objectives

Output: improvement suggestions

Type: generic

Decomposition: –

Method: Indicators are examined to identify improvement potential regarding the defined objectives and to make improvement suggestions. Exemplary indicators are:

- Number of successful/unsuccessful retrieval tasks
- Number of artifacts stored in the experience base
- Number of retrieval tasks per time period
- Usefulness perceived by the users (cf. Chapter 9, [NT99])

The actual indicators used will vary from organization to organization and must be defined as part of the setup of a SEEMS (e.g., using the goal/question/metric technique [NT99]).

Exemplary improvement suggestions are:

- Change number of artifacts in the experience base
- Change similarity definition used for similarity-based retrieval
- Add auxiliary information to characterizations of artifacts
- Raise minimal quality requirements for stored artifacts
- Package artifacts stored in the experience base (e.g., by combining several related artifacts into a single artifact)

Requirements: R16 (data analysis for improvement)

Task T53: analyze artifact

Objective: find improvement suggestions for artifacts stored in the experience base

Input: state of the practice; access statistics/user feedback

Output: improvement suggestions

Type: artifact-specific

Decomposition: –

Method: Indicators are examined to identify improvement potential and to make improvement suggestions. Exemplary indicators are:

- Number of times the artifact has been checked-out over the number of times the artifact has been part of a retrieval result
- Number of times the artifact has been part of a retrieval result over the total number of retrieval tasks during the lifetime of the artifact
- Number of times the artifact has been reused successfully
- Number of applications in a given time period

The actual indicators used will vary from organization to organization and must be defined as part of the setup of a SEEMS (e.g., using the goal/question/metric technique [NT99]).

Exemplary improvement suggestions are:

- Forget artifact
- Improve quality of artifact
- Add reuse documentation to the artifact

In general, this will require to examine the artifact.

Requirements: R3 (tool integration), R16 (data analysis for improvement)

4.4.3 Forget Task

Task T54: forget

Objective: remove obsolete artifacts from the experience base

Input: state of the practice; improvement suggestions (set of artifacts to be removed)

Output: improved state of the practice

Type: generic

Decomposition: –

Method: The status of the artifact is set to »obsolete«. Related artifacts are examined. If they have become obsolete too, the task is performed recursively.

Alternatively, the experience package could be removed physically. However, one should be careful about doing so, because:

1 The artifact may be in use by some running project. In this case, there is the chance that information related to the artifact will be requested in the future.

2 The artifact may be included in some other artifact. In this case, information about the forgotten artifact may be requested if the other artifact is modified later on.
3 The artifact may be of use for future analyses about the development history of the SEEMS.

Requirements: R27 (artifacts' status)

4.4.4 Package Task

Task T55: package

Objective: increase efficiency and effectiveness of the SEEMS

Input: state of the practice; improvement suggestions

Output: aggregated artifacts; changed experience packages; improved state of the practice; set of defined objectives; feedback indicators; training material; change plan

Type: generic

Decomposition: *structure (T56, page 121)*, *adapt (T57, page 121)*, *aggregate (T60, page 122)*

Method: Three types of improvement suggestions can be distinguished:
1 One or more existing experience packages shall be changed (e.g., update of the artifacts themselves, update of the artifacts´ characterizations, development of additional documentation). In this case, »adapt« is performed for each experience package.
2 Out of a set of existing or adapted experience packages, a new experience package shall be created. In this case, »aggregate« is performed for each new experience package.
3 The conceptual knowledge underlying the experience base shall be changed. In this case, »structure« is performed. As a result, all of the affected experience packages must be adapted (task »adapt«) unless the technical infrastructure provides versioning for the schema. If the conceptual knowledge shall be extended, the artifacts of the new kind may be developed using existing artifacts (task »aggregate«).

Requirements: none

Task T56: structure

Objective: change the conceptual knowledge of the experience base and adapt the technical infrastructure accordingly

Input: state of the practice; improvement suggestions

Output: changed schema of the experience base (part of the improved state of the practice); feedback indicators; training material; change plan; set of defined objectives

Type: generic

Decomposition: see Chapter 8

Method: see Chapter 8

Requirements: R12 (accommodation of new artifact kinds), R13 (maintenance of conceptual information)

Task T57: adapt

Objective: improve an experience package

Input: state of the practice; improvement suggestions

Output: changed experience package

Type: generic

Decomposition: *adapt artifact (T58, page 121)*, adapt characterization (T59, page 122)

Method: If »adapt artifact« is performed, it is performed first. Then, »adapt characterization« is performed. The task »adapt characterization« is performed in any case, because a change of the artifact should be reflected by its characterization.

Requirements: none

Task T58: adapt artifact

Objective: improve an artifact

Input: state of the practice (including the artifact to be adapted); improvement suggestions

Output: changed experience package (with the artifact changed)

Type: artifact-specific

Decomposition: –

Method: The artifact is changed according to the improvement suggestions.

Note: Major changes require a re-recording of the artifact.

Requirements: R3 (tool integration), R28 (versioning)

Task T59: adapt characterization

Objective: improve characterization of an artifact

Input: state of the practice (including the characterization to be adapted); improvement suggestions; optionally: changed experience package (with the artifact changed)

Output: changed experience package (with the characterization changed)

Type: artifact-specific

Decomposition: –

Method: The characterization of an artifact is changed according to the improvement suggestions. Major changes (e.g., creation of »virtual artifacts« like lessons learned and reuse modification documentation) require a re-recording of the artifact.

Requirements: R3 (tool integration)

Task T60: aggregate

Objective: combine existing or adapted artifacts to a new one

Input: state of the practice; changed experience packages; improvement suggestions

Output: aggregated artifacts

Type: artifact-specific

Decomposition: –

Method: A new artifact is developed based on existing or adapted artifacts. Examples include [BCR94a]:

- Write the functional specification of a code component
- Turn a lessons learned document into a management decision support system
- Build a cost model from empirical data

- Automate methods into tools

Note: The aggregated artifacts are subject to recording.

Requirements: none

4.5 Exemplary Instantiations of the Task Decomposition

The presented framework can be instantiated to reflect various strategies and policies of an organization regarding the populating, maintaining, and utilizing of an experience base. This section outlines two examples.

4.5.1 Problem-Driven Versus Experience-Driven Learning

As stated in the beginning of Section 4.4, the subtasks of »learn« are triggered by events. Which subtasks are triggered by which event is subject of the experience factory policies.

The task decomposition presented can be used for the various policies. For example, learning policies of the experience factory can vary between two extremes [BCC92]:

1 **Experience-driven.** Whenever experience becomes available through projects, it is recorded. This means that problems are always recorded together with their solutions (and an evaluation how good the solution was).

2 **Problem-driven.** Whenever a challenging situation is identified during the course of a development project, the problem is passed to the experience factory. The experience factory actively solves the problem by developing solutions. The solution is given to the project. Finally, problem, solution, and application experience are recorded.

In both cases, internally triggered analyses and packaging will take place (»preventive maintenance«).

Where a given experience factory is positioned between these poles depends on the responsibilities given to the experience factory by its organization. Thus, the task decomposition must be flexible enough to support various organizational setups.

With the presented task decomposition, both extremes can be realized:

1 **Experience-driven.** New experience is acquired with the task »record«.

2 **Problem-driven.** Projects articulate their problems through retrieval. If a solution already exists, it is returned by the retrieval and is applied. If no feasi-

ble solution exists, feedback given during the retrieval will trigger an analysis (task »analyze SEEMS«). At this point, a solution may be developed using the already available experience (task »aggregate« of »package«). The new solution is recorded. At the end of the task »record«, the project is informed (task »inform«) about the new solution. Finally, the project retrieves and applies the solution.

Since both of the extremes are supported by the task decomposition, any mixture of experience-driven and problem-driven policies is supported too.

4.5.2 Domain Analysis

Domain analysis can be understood as a method for learning. More precisely,

Definition 13: Domain analysis

> domain analysis *means organizing descriptions of components and architectures common to the applications analyzed.*
>
> *[Ara94, p. 19]*

This definition implies that the analyzed applications already exist. In terms of the presented framework, this means that the applications have already been recorded in the experience base as artifacts. For the domain analysis, components and architectures of the applications, which are already stored in the experience base, have to be identified. Usually, this requires a refinement of the applications' characterizations (components and architectures become reusable parts). In the presented framework, the »package« task is started. The subtask »structure« develops an adequate characterization schema for components and architectures. Then, the characterizations of the applications are adapted (»adapt characterization«). Possible methods for finding the new characterizations include reverse engineering techniques.

The (extended) characterizations are then re-recorded (task »record«). As part of this task, their similarities are analyzed (task »check for existing parts«). During the task »review«, suggestions for combining similar components and architectures are made. The task »evolve« actually combines similar components and architectures, whereas the task »complete characterization« documents the applications the components and architectures belong to. The descriptions of components and architectures common to the applications are organized.

In domain analysis, the final results are evaluated. This is represented through the »analyze SEEMS« in the presented framework.

The above analogy is based on the common domain analysis process (CDAP) described in [Ara94]. CDAP is based on eight commonly used domain analysis methods. Because CDAP is more detailed than outlined above, it can be used as a basis for refining the framework for two kinds of artifacts: components and

architectures. Nevertheless, Arango also points out some shortcomings of existing domain analysis methods. For instance, »authors appeal to some form of validation steps in their process,« but there are no procedures for model validation [Ara94, p. 37]. Here, the presented framework extends existing domain analysis methods by systematically collecting user feedback, by providing access statistics, and by suggesting improvements based on this input.

4.6 Summary

This chapter describes the tasks of populating, maintaining, and utilizing the experience base using Steel´s knowledge-use level approach (components of expertise) [Ste90], an extension of Newell's knowledge level approach [New82]. The knowledge-use level approach describes an intelligent system by arranging the tasks to be performed in a hierarchy, and by specifying for each task:

- The goal to be achieved by the task
- A method that either defines in what order to perform the subtasks of the task or how to perform the task if it is not decomposed further
- Contents (not structure!) of knowledge that is needed to perform some task
- Pragmatic constraints on the access and availability of the knowledge

Figure 14 shows the hierarchical decomposition of the maturing task introduced in this chapter. The upper levels of the breakdown are based on [BR91], whereas the lower levels are inspired by representative literature from the areas of case-based reasoning, organizational memories, knowledge management, domain analysis, and software reuse (see Table 4), thus integrating and complementing the processes of these areas. The resulting hierarchy has been synthesized and refined through years of experience in the area of experience factory [ABT98, ABH+99, ANT99b, BR88, Bec96, BT98a, Bro98, Gäß95, GB97, Nic98, Stu95, Tau93, TA97, TG98b].

In this dissertation, the tasks have been described systematically and comprehensively using the knowledge-use level approach for the first time. It is the most comprehensive description for incremental, continuous learning based on a repository to date. The description is systematic in the sense that not only the goal and the way to perform a given task are specified, but also the technical requirements needed to support the task effectively. Inputs and outputs define the product flow between tasks. The description is comprehensive in:

- **Breadth.** The description covers all tasks needed to set up, populate, utilize, and improve an experience base.
- **Depth.** The tasks (including the task »structure«, see Chapter 8) are broken down to a level of detail where further refinements are either senseless (because their instances depend on the technical infrastructure employed; e.g., task »calculate similarity«) or artifact-specific (i.e., a refinement must consider particularities of the kind of artifact to be processed; e.g., task »collect«). A strategy has been proposed in this chapter to set up libraries for

Figure 14: Maturing tasks

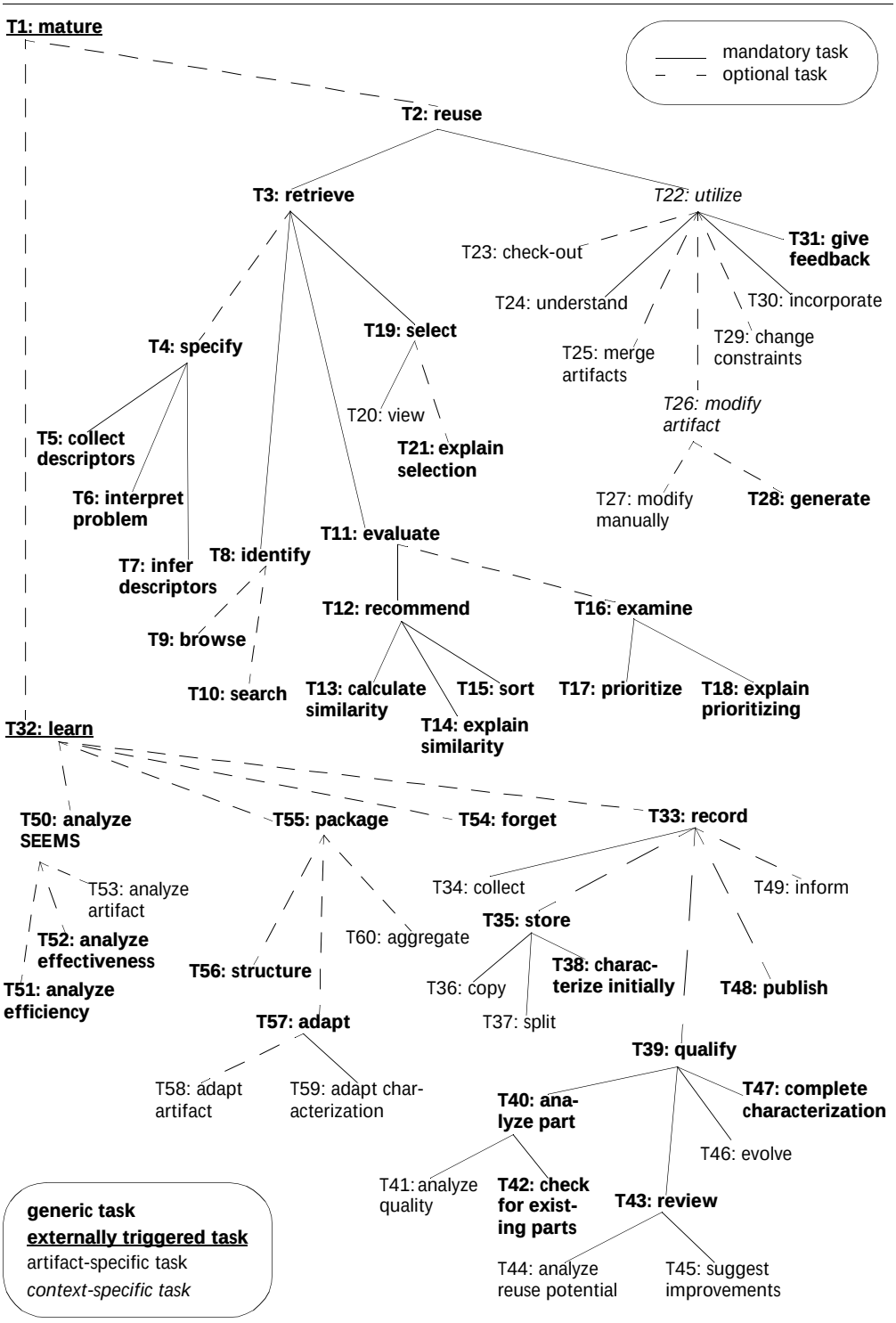


Table 4: Related literature to the presented task decomposition

Task	[AP94]	[Ara94]	[BB91]	[BR91]	[BCC92]	[BR87]	[BWP98]	[CB91]	[CDF96]	[DvK95]	[IK96]	[KS96]	[KBM98]	[LW98]	[PDF87]	[RY97]	[Sar95]	[Sen97]	[SCK+96]	[Tra94]	[vvK96]	[Wil95]
T32: learn				X	X									X								
T50: analyze SEEMS		X												X								
T51: analyze efficiency																						
T52: analyze effectiveness																						
T53: analyze artifact														X								
T55: package				X	X									X								
T56: structure		X		X	X		X					X		X					X			
T57: adapt				X										X								
T58: adapt artifact																						
T59: adapt characterization																						
T60: aggregate		X																				
T54: forget														X								
T33: record	X			X	X		X	X					X	X							X	
T34: collect		X		X			X	X	X			X						X	X		X	X
T35: store	X			X			X	X											X		X	
T36: copy																						
T37: split																						
T38: characterize initially	X	X	X	X				X										X				
T39: qualify		X		X			X											X		X		
T40: analyze part	X																					
T41: analyze quality	X							X														
T42: check for existing parts	X	X			X											X						
T43: review	X	X						X				X					X	X				
T44: analyze reuse potential		X	X					X														
T45: suggest improvements		X			X																	
T46: evolve	X		X					X														
T47: complete characterization	X	X	X	X				X										X				
T48: publish							X										X					
T49: inform							X														X	
T2: reuse	X	X	X	X						X				X								
T3: retrieve	X		X	X	X		X			X				X							X	
T4: specify	X																					
T5: collect descriptors	X			X						X	X		X					X				
T6: interpret problem	X																					
T7: infer descriptors	X																					
T8: identify	X									X	X		X					X				
T9: browse																						
T10: search	X			X							X											
T11: evaluate	X			X							X		X					X				
T12: recommend	X										X											
T13: calculate similarity	X			X							X											
T14: explain similarity	X										X											
T15: sort											X											
T16: examine											X											
T17: prioritize																						
T18: explain prioritizing																						
T19: select	X			X						X	X		X					X				
T20: view																						
T21: explain selection																						
T22: utilize	X				X	X	X			X	X											
T23: check-out	X																					
T24: understand						X				X				X				X				
T25: merge artifacts										X												
T26: modify artifact			X	X		X				X			X		X							
T27: modify manually			X		X	X				X								X				
T28: generate			X			X				X												
T29: change constraints			X			X				X												
T30: incorporate			X			X				X			X									
T31: give feedback										X												

these artifact-specific refinements (see page 85). Using this strategy, commonalities and differences can be identified resulting in a type change from »artifact-specific« to »generic«. This way, the presented framework is extended in depth as the state-of-the-art advances.

The knowledge-use level description represents a framework that

- is flexible enough to be tailored to company-specific needs and policies, i.e., increases potential for reuse across organizations [PD91]
- guides the tailoring to company-specific needs, i.e., it facilitates management control
- enables comparisons among processes employed in various organizations (by mapping the processes onto the tasks and comparing them on the task level)
- enables analyses of employed processes to detect strengths and weaknesses (by mapping the processes onto the tasks and checking (a) which requirements for the corresponding tasks are fulfilled and (b) which tasks are performed only implicitly or not at all)
- closes the gap between the tasks to be performed for running an experience base and the functionality provided by a technical infrastructure of an experience base, i.e., it helps in identifying support tools
- enables the demand-driven buildup of libraries for reusable problem-solving methods in the context of running an experience base (a problem-solving method achieves the goal of some task while considering pragmatic constraints on the availability and accessibility of needed knowledge)

As such the framework provides a goal-oriented means for setting up an experience base by defining the methodological and implementation aspects of an experience factory. Thus, it complements the definition of the organizational and managerial aspects as described by Basili, Caldiera, and Cantone [BCC92].

The framework can also be used to evaluate a given reuse approach. A reuse approach consists of a methodological part and a tool part implementing (at least partially) the methodological aspects. By analyzing the coverage of the methodological part with respect to the presented framework, the useful requirements for the technological part can be identified using the requirements enumerated by the framework for the tasks covered. Then, the completeness of the reuse approach can be assessed. This will be done in the next chapter for reuse approaches described in literature.

5 Existing Approaches

People can't share knowledge if they don't speak a common language.
T. Davenport

In the previous chapters, requirements for an experience base have been elicited. They include the pragmatic constraints on the knowledge to be stored in the experience base (Chapter 2), the technical requirements to be fulfilled (Chapter 3), and the tasks of populating, maintaining, and utilizing the experience base which have to be supported (Chapter 4). These requirements represent a framework which can be used for evaluating existing approaches regarding their suitability for implementing the technical infrastructure of an experience base.

This chapter surveys relevant areas for software component repositories and describes exemplary existing approaches from each area to determine in how far they fulfill the elicited requirements. According to Frakes and Gandel, the relevant areas are library and information science, hypertext systems, and knowledge-based systems [FG90]. In this dissertation, the area of database management systems is reviewed as well, because recent advances in the direction of object-oriented and object-relational database management systems also contribute to the organization of software engineering repositories. For each area, the major techniques are shortly introduced before examples from software reuse are discussed. Many of the selected examples date back a few years. The reason is that most of the more recent approaches (including software development repositories [Ort99]) employ a mixture of the techniques described in this chapter and thus cannot be associated with a single technique or area.

Following their descriptions, Section 5.5 evaluates the approaches regarding their contributions with respect to the framework. It will be shown that none of the approaches meet all of the requirements. In particular, none of the approaches are capable of handling imprecise and inconsistent characterization information. Also, none of the systems support the maintenance of the experience base in a satisfactory manner.

In the summary section, the chronological development of the approaches is presented together with the major insights and assumptions that led to the development of ever new approaches. The chapter summary closes with a (knowingly simplified) characterization of the state-of-the-art.

5.1 Library and Information Science

Collecting and providing on demand books and other kinds of documents to interested communities dates back a very long time (e.g., the well-known library of Alexandria). Since then, many techniques for searching collections have been devised. With the rise of multimedia technology, libraries are no longer limited to (textual) documents, but may also contain images, audio samples, and even digital videos. At the same time, traditional libraries increase their budget for providing electronic services in form of CD-ROMs, on-line access catalogs, and on-line databases [FAFL95]. In fact, more advanced techniques for searching collections have become feasible only through the use of computing technologies [Gau95].

In the future, we will see the merging of electronic and paper-based artifacts, that is, electronic catalogs will provide access to electronic artifacts and documents stored in traditional libraries, making up digital libraries [LM95]. The collections will also include instable artifacts with a short lifetime, for instance, shared annotations of library users. This trend makes the research community of digital libraries interesting for experience bases. However, in contrast to digital libraries, collections for software developments tend to be rather focused. Various kinds of artifacts and the relationships between them can be clearly identified. This leads to small numbers of artifacts for each artifact kind (quality is more important than quantity [Spa99]) in comparison to the large heterogeneous collections of digital libraries that cover a wide spectrum of knowledge available in the world at large (e.g., over the World-Wide Web [SC99]).

As there are libraries of all sizes for all kinds of purposes [LM95], it is, nevertheless, worthwhile to look at the ways of organizing access information for library collections. Due to the heterogeneity of the artifacts and users of libraries, the organization is based on term vocabularies. Numerical information is typically not used for searching. The techniques for organizing access information cover a wide spectrum ranging from sorted arrangements to sophisticated information retrieval techniques. The techniques can be structured as shown in Figure 15 [FP94, Gau95].

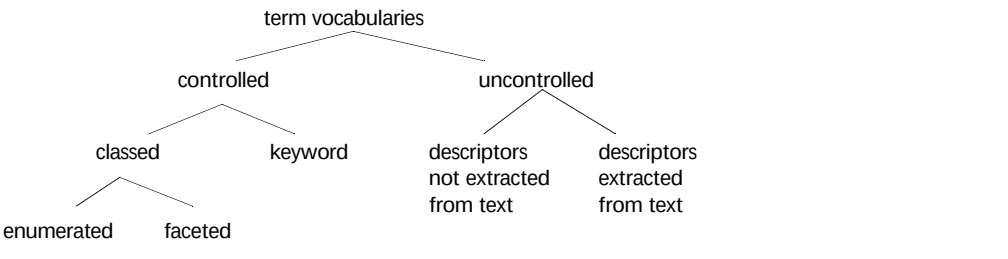


Figure 15: Techniques from library and information science for organizing access information

When using terms from the everyday language, one has to deal with the ambiguities of natural language. Here, homonyms (the same term is used for differ-

ent things) and synonyms (various terms are used for the same thing) cause problems if the person characterizing and the person searching do not use the terms consistently [Gau95].

Homonyms cause rarely any problem in informal text descriptions, because their meaning can be derived from the context. However, in information science homonyms are often used as descriptors without any textual context. Hence, misunderstandings are very likely. For instance, the term »software product« can be used for executables of a software system, but also for the whole software system documentation or parts thereof. If someone looks for software products with the meaning of »executables« in mind, the retrieval system might also select all requirements, design, and code documents. These selections are clearly a burden on the person searching. Strategies for overcoming the problems of homonyms are [Gau95]:

- **Limit scope of library.** Through the limitation of the scope, many meanings of a term lie outside the subject area of a library. For instance, a code library will not contain requirements and design documents.
- **Use explanations.** Terms are made unambiguous through supplementing them with an explanation, for example, »software product (executable)« and »software product (code component)«.
- **Use thesauri.** Homonyms can also be made unique by arranging them in hierarchies. This requires the documentation of broader, narrower, and related terms for each term. Such arrangements are called *thesauri*.

Synonyms cause substantial problems in information science. For instance, if the term »code module« is used for searching for code artifacts, then all those artifacts characterized by »code component« will not be selected. Hence, homonyms and synonyms can be viewed as antagonists. While homonyms cause too many artifacts to be selected, synonyms cause too little artifacts to be selected. Strategies for overcoming the problems of synonyms are [Gau95]:

- **Search with all synonyms.** During characterization, no particular preventive actions are taken. Consequently, the person searching must look under all synonyms. This strategy has the disadvantage that the number of descriptors increase, because all synonymous terms are legal descriptors. The main disadvantage is, however, that it will be difficult for the person searching to think of all relevant synonyms. Often, some of the synonyms are forgotten leading to incomplete retrieval results.
- **Use general naming conventions.** Through conventions consistent use of the term space can be achieved. For instance, terms can be taken from some standard (e.g., IEEE standards). Other choices concern whether nouns are always written in singular or plural, whether abbreviations and acronyms are preferred over written out words, and language conventions (e.g., US English or British English).
- **Use a controlled vocabulary.** A predefined vocabulary defines which term is to be preferred for a set of synonyms.

- **Use equivalence classes.** Synonyms are not confined to terms having exactly the same meaning. Rather, synonyms may also have nearly the same meaning, that is, such terms differ in their connotations or they partly overlap (e.g., »software development« versus »programming«). To avoid a large number of descriptors, synonyms (including those terms with nearly the same meaning) are viewed as an *equivalence class*. The class is typically referred to by a preferred term for all terms in the class. For each single class it must be decided on the degree of variation allowed for the meanings of the terms in that class. For the core subject areas covered by a library, equivalence classes should be defined narrowly while for peripheral subject areas, equivalence classes should be defined more widely. The goal is to have roughly the same number of artifacts in each equivalence class.
- **Use thesauri.** The systematic arrangement of terms does not only help solving problems with homonyms, but also helps to deal with terms having nearly the same meaning. Terms having nearly the same meaning are related terms. Thus, cross references point to terms having nearly the same meaning. In addition, thesauri can define preferred terms. Hence, thesauri are a means to handle problems of both homonyms and synonyms.

All measures taken toward a clear definition of terms and the distinction between their meanings are referred to as *terminological control*. Effective and efficient retrieval requires good terminological control.

With this background, the following sections introduce the various organization principles of Figure 15.

5.1.1 Enumerated Classification

Among all organization principles, enumerated classification is the simplest one [Gau95]. Here, the subject area is divided into separate areas called *classes*. Metaphorically, each area corresponds to a bookshelf. All classes must be disjunct. Each class is represented through a descriptor. At the same time, each class of a enumerated classification represents an equivalence class, that is, the descriptor is the *preferred term* for a set of synonymous terms. Ideally, each artifact is assigned to exactly one class.

To ease the searching for artifacts, the classes are systematically arranged. Classes can be sorted according to some criterion (e.g., alphabetically) and displayed as linear lists. But other arrangements, for instance, as hierarchies, are also conceivable. For example, Figure 16 shows an excerpt from a hierarchically arranged enumerated classification of UNIX commands (classes are printed in **boldface**).

It is often difficult to find a set of disjunct classes for an enumerated classification schema. For instance, in the example given in Figure 16, the UNIX com-

Figure 16: Example for an enumerated classification of UNIX commands

```
directory operations
    create
        ln, mkdir
    destroy
        rmdir

file operations
    create/modify
        edit
            ed, vi, view, emacs, ex, vedit
    destroy
        rm
```

mand »rm« can also be used to destroy directories with the option »-r«. This can be solved by allowing multiple descriptors for a single artifact. In this case, an artifact is no longer assigned to exactly one class, but to a small number of classes.

Advantages of enumerated classification:
- The organization principle is natural and easily understandable.
- If not arranged linearly, the arrangement helps the users to understand the relationships among the descriptors.
- The arrangement provides a natural search technique for navigating in the classification schema.

Disadvantages of enumerated classification:
- The classes must (should) be disjunct.
- For each area and each artifact, there must (should) be exactly one fitting class available.
- A detailed classification schema (for high indexing precision, that is, a small number of artifacts per class) requires a large number of classes.
- If not arranged linearly, the rigid arrangement makes the classification schema difficult to change.
- Arranging the descriptors systematically is often difficult, partly because for technical reasons, but also because adequate classifications are subject to discussions among people who have differing viewpoints.

Enumerated classification is the right choice for small collections. Large collections with high indexing precision require many classes, making the classification schema clumsy and, therefore, useless [Gau95].

Exemplary software repositories employing enumerated classification are common practice. They are typically implemented using file systems with an appropriate directory structure where the directory names correspond to the descriptors.

5.1.2 Faceted Classification

A principle problem of enumerated classification is the difficulty to search with incomplete information. For instance, if we are interested in all »destroy« commands, we would have to look into two places using the enumerated classification schema of Figure 16: directory and file operations. It is not possible to reclassify the artifacts dynamically (e.g., by distinguishing between »create/modify« and »destroy« on the top level). This becomes possible through faceted classification. In the example, two *facets* can be identified: (a) operations (»create/modify« and »destroy«) and (b) the objects the commands operate on (»directory« and »file«).

Faceted classification is capable of integrating several aspects in one characterization [Gau95]. Each aspect is represented by a facet. Each facet is specified using an enumerated classification schema. Hence, faceted classification is a generalization of the enumerated classification technique. Ideally, each artifact is characterized by exactly one class from each of the facets. However, as with enumerated classification, it may prove difficult to define schemas with completely disjunct classes. Therefore, multiple entries for a facet may be used, but it should be the exception.

Facets must be specified in a way that all artifacts can be viewed meaningfully under all facets. Yet, in practice, facets may not be applicable for some unanticipated artifacts or may simply not provide a needed class. This problem can be solved by adding a special value »not applicable« to the classification schema of the facet. It may even be useful to add several special values, for example, »unknown«, »not applicable«, and »known and applicable, but none of the other classes match«.

Advantages of faceted classification:

- In comparison to enumerated classification, faceted classification allows a much more detailed characterization of the artifacts. At the same time, the number of classes across all facets is typically less than the number of classes for an equivalent enumerated classification schema.
- The name of a facet annotates the descriptors of the facet. For instance, the facet »team size« sets the descriptors »small«, »medium«, and »large« into context.
- The faceted classification schema guides the characterization of artifacts by telling the librarian the aspects which are important to the library user.
- It is easier to reach a consensus among people on a set of small enumerated classification schemas than on one comprehensive enumerated classification schema.
- Faceted classification is able to deal with incomplete queries. Facets not specified for a query match all characterizations.

Disadvantages of faceted classification:

- In comparison to the enumerated classification, characterization requires more effort because a descriptor must be chosen from each facet.
- Since each facet is specified using an enumerated classification schema, the disadvantages of (non-overlapping) enumerated classification are inherited. These are: the subject area (an aspect) must be covered completely, the classes must be disjunct, and each artifact must (should) be assignable to exactly one class.

Faceted classification is suitable for domains with rather well defined topic areas as opposed to broad, heterogeneous collections [PD91]. This is the reason why faceted classification is popular with engineering and scientific disciplines such as software engineering. Prieto-Díaz and Freeman used faceted classification for a code component library [PDF87]. They used six facets for characterizing code components:

1 Function (add, append, close, …)
2 Objects (arguments, arrays, backspaces, …)
3 Medium (array, buffer, cards, …)
4 System type (assembler, code generation, code optimization, …)
5 Functional area (accounts payable, accounts receivable, analysis structural, …)
6 Setting (advertising, appliance repair, appliance store, …)

The facets consider context information, for instance, the facet »setting« corresponds to the application domain for which a code component has been developed.

Terminological control is achieved by using a thesaurus for full synonyms (terms with the exactly same meaning) and weighted conceptual graphs for terms with similar meanings. Figure 17 shows an example where the terms »measure« and »add« are more similar (distance 6) than »measure« and »move« (distance 16).

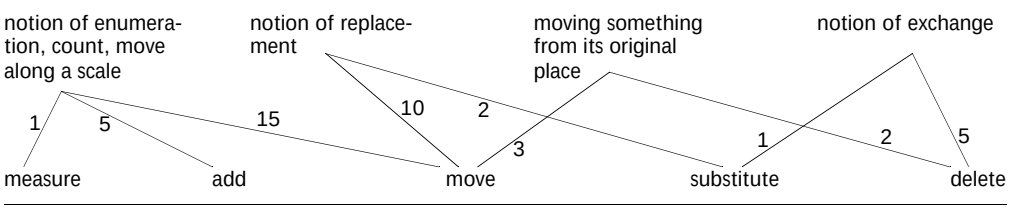


Figure 17: Example of a partial weighted conceptual graph for the function facet (taken from [PDF87])

Retrieval is done by matching the query exactly. If the query does not return enough artifacts, the query can be expanded by including additional query terms. These terms are suggested to the user sorted by their conceptual closeness. For instance, a query with »move« as its function facet can be expanded by also including code components that are characterized by »delete« (5), »substitute« (12), »measure« (16), and »add« (20). The user can choose to which

degree the query shall be expanded. This technique allows to find similar code components.

In addition, matching code components are evaluated with respect to the application effort required. This effort is defined to be a function of component size, complexity of component structure, quality of documentation, and programming language. User experience is a modifier for the first three measures. The evaluation result is computed using fuzzy set theory, that is, the system assigns a reusability value to each component retrieved. Then, the components are sorted according to their reusability.

Another example of faceted classification is the feature-oriented reuse method (FORM) for domain-specific reference architectures [KKL+98]. The terms used in the faceted classification schema of Kang et al. are not disjunct. The consistency of characterizations are enforced using *composition rules*. For example, the term »DOS« of the facet »Operating system« must be selected together with the term »multi-tasker« of the facet »S/W platform«. Composition rules can be interpreted as rule-based dependencies (see Definition 9 on page 26). They guide the user in specifying the query by interpreting the problem on-line (T6, page 91) and allowing only the specification of compatible descriptors.

5.1.3 Controlled Keyword Systems

Classification systems require disjunct classes. The development of classification schemas with disjunct classes is effort-intensive, and sometimes it is difficult to satisfy the viewpoints of all users. An alternative is to use keyword systems. For keyword systems, descriptors need not be disjunct. In fact, an overlap of the descriptors is desired.

Keyword systems are based on the principle of term combination, that is, an artifact is characterized using a variable number of adequate keywords [Gau95]. Likewise, a query consists of a set of keywords. An artifact characterization matches a query if all keywords of the query are contained in the characterization of the artifact. Advanced retrieval systems will even allow a disjunctive combination (»or« semantic) in addition to the conjunctive (»and« semantic) or the specification of terms that shall *not* be part of the artifact's characterization.

Each keyword stands for an aspect of the artifact and thus can be interpreted as an equivalence class. A set of keywords describes the intersection of the equivalence classes associated with the keywords. For instance, coding guidelines for software of a chemical plant can be characterized by the keywords »coding«, »guideline«, »programmer«, »manufacturing control«, and »chemical plant«.

Controlled keyword systems limit the descriptors to a predefined set of keywords. Usually, keywords are arranged in alphabetical order.

Advantages of controlled keyword systems:

- It is less effort-intensive to develop a set of allowed keywords than to develop classification schemas. The development of the vocabulary can be performed semiautomatically for textual artifacts using a set of reference artifacts.
- The meaning of descriptors may overlap.
- Already a small set of descriptors allows a detailed characterization.

Disadvantages of controlled keyword systems:

- More effort is needed for characterizing artifacts in comparison to classification techniques, because – at least theoretically – the whole list of keywords must be considered to mark all relevant keywords. In contrast, the characterization using classification schemas is finished once one term has been selected from each facet.
- Terminological control is more difficult to achieve than in classification systems. For instance, disjunct keywords (e.g., »small team« and »large team«) should not be part of the same characterization. Thesauri may help, but the buildup of thesauri takes away the advantage of developing the vocabulary with a low amount of effort. In effect, a thesaurus-supported keyword system is a faceted classification system where each artifact may be characterized with a variable number of (applicable) facets.

In software engineering, a controlled keyword system has been used for retrieving object-oriented code (classes and libraries) [DFB99, DFF97]. In the descriptor-based approach to object-oriented code reuse, a thesaurus is constructed semiautomatically by an application engineer by extracting keywords from interface descriptions of classes and libraries and performing a term-frequency analysis. Each descriptor consists of a verb object pair (e.g., »connect server«) and is annotated with its importance for each application domain covered by the descriptor base. The underlying assumption is that the relevance of a descriptor depends on the application domain for which a code artifact is sought after. To retrieve an artifact, the user first scans the thesaurus. From this input, the inference of suitable descriptors is guided using the thesaurus. For example, once the user has spotted a verb that could be the first term of a descriptor, the system automatically presents all objects listed in its controlled vocabulary, which are coupled with that verb in at least one characterization. Query results are based on the relevance assigned to the descriptors and the similarity between the descriptors of the queries and the descriptors used for characterizing the code artifacts (defined by the thesaurus).

5.1.4 Uncontrolled Keyword Systems

One disadvantage of controlled keyword systems is the effort needed to characterize artifacts. This disadvantage can be overcome by using uncontrolled vocabularies. Here, artifacts are characterized using arbitrary keywords – a predefined list of keywords does not have to be considered.

Advantages of uncontrolled keyword systems:
- Low effort for characterizing artifacts.
- Unanticipated artifacts can be characterized adequately, because new terms can be introduced at any time.

Disadvantages of uncontrolled keyword systems:
- Low terminological control, because thesauri are not used (otherwise it would be a controlled keyword system by definition).
- Saved effort for characterizing is shifted to the user searching for artifacts.

No example could be found in the literature for a software engineering library using uncontrolled keyword systems where keywords were not extracted from the artifacts.

5.1.5 Full-Text Search

One of the most expensive and error-prone activities in information processing is the manual characterization of artifacts. By automatically extracting terms from artifacts using information retrieval techniques, this activity requires a minimum of effort [SM83].

Advantages of full-text search:
- Effort for characterizing artifacts is minimal.
- Unanticipated artifacts can be characterized with no problems, because new terms can be introduced at any time.
- Large, heterogeneous collections can be characterized.
- Costs for bad investments where a library is built up with a detailed characterization for all artifacts, but which is never used extensively, is considerably lowered. Full-text search is much cheaper for libraries which are used seldom. In contrast, the saved amount of effort for manually characterizing artifacts is negligible for frequently used libraries.
- Natural language interfaces for queries can be constructed easily. Using the same algorithm for characterizing queries and artifacts, it is possible to extract keywords from queries. Users do not need to learn any special query language.

Disadvantages of full-text search:
- Only textual artifacts in digital form can be characterized. Other forms of informations (e.g., diagrams) are not considered for characterization.
- Low terminological control, although some heuristics (e.g., usage of stop word lists to prevent irrelevant keywords such as articles to be part of the characterization; term combinations appearing in several documents suggest related terms) can be employed.

- Many software engineering artifacts cannot be characterized automatically, because the information needed for utilizing them is simply not documented explicitly in the artifact (cf. Chapter 2).

In software engineering, an approach for automatically constructing software libraries is the GURU system [MB94]. GURU characterizes UNIX man pages using the heuristics of:

- **Open-class words.** In general, open-class words include nouns, verbs, adjectives, and adverbs, whereas closed-class words are pronouns, prepositions, conjunctions, and interjections. The heuristic considers only open-class words as descriptors.
- **Lexical affinity.** In GURU, two terms are lexically related if they are open-class words and are not further apart than five words. Hence, descriptors are not single words, but pairs of words.
- **Term-frequency analysis.** The more often a lexical affinity occurs in the artifact, the more relevant it is for characterization. However, this strategy alone may lead to also accepting noise (e.g., the lexical affinity »be move«) as descriptors. To avoid this, only lexical affinities are used for characterizations that (a) occur frequently within one artifact and (b) whose terms do not occur frequently *across* the artifacts of the library collection.

Finally, the artifacts of the library are arranged in a hierarchy using a clustering analysis technique. Thus, an enumerated classification schema is assembled automatically. The classes of the resulting classification schema contain exactly one artifact. Therefore, the hierarchy connects related artifacts.

Retrieval is performed in two phases:

1 A natural language query is characterized using the same technique as for characterizing the artifacts. The query characterization is then matched against the artifacts' characterizations. Not only exact matches are returned, but also similar characterizations.

2 Starting with one of the returned artifacts, the user can browse the automatically assembled hierarchy. However, the authors point out that the automatically constructed hierarchies are not always of a good quality and should only be used to aid when nothing relevant has been retrieved using the full-text retrieval.

5.2 Hypertext

Organization principles from library and information science assume all characterizations to be independent of each other. Therefore, dependencies among artifacts are not considered for retrieval. However, in software engineering artifacts do not exist in isolation. For instance, a design document can be viewed as

an implementation of its specification. When a design document is retrieved, its specification is also of interest for understanding the design. Techniques from library and information science would typically not return a specification if the user asks for a design. The same problem arises when retrieving reusable parts, because usually the reusable parts (cf. task »record (T33, page 107)«) are highly interrelated (otherwise they would not be part of the same artifact).

This problem can be overcome by providing the means to navigate to related artifacts or reusable parts starting from an initial artifact (as has been done by [MB94]). Hypertext systems provide the capability to author and traverse such networks. While its roots go back to Vanevar Bush [Bus45], the term »hypertext« was coined much later by Ted Nelson during the 1960's [Nel67].

The principle underlying hypertext is simple [AMY88]: *Information units* (also called *nodes*, *frames*, *notecards*, etc.) are interconnected by links. Users *navigate* or *browse* a hypertext by selecting links to travel from one unit to another. Systems that allow information units to contain information forms other than text (e.g., graphics, sounds, and videos) are called hyper*media* systems. The most well-known hypermedia system is probably the World-Wide Web (WWW).

Advantages of hypertext and hypermedia systems:
- The contents of information units can be of any form and can vary from unit to unit.
- Links allow arbitrary structures.
- They are easy to use.

Disadvantages of hypertext and hypermedia systems:
- Users may get lost in the »jungle« of links (»lost in hyperspace« syndrome). This problem can be alleviated by keeping a navigation history.
- It may be difficult to find the right information unit to start working with.
- The maintenance of hypertexts is error-prone and effort-intensive. Without tools it is difficult to maintain hypertext systems with more than 100 information units [BPS91]. Partial solutions to the maintenance problem are to treat links separately from information units (i.e., information units should not contain the physical address of a destination of a link) [CHDH94] and to use consistency checkers [TBF95].

Over the years, various extensions have been defined ranging from general conventions for hypertext systems [AMY88, Ste95] over the definition of types for the information units and links [Bog92] to the possibilities of defining hierarchical information units (e.g., [Har88]).

In software engineering, hypertext systems have been used for software work products [GS90].The Document Integration Facility (DIF) allows templates (both on the organization and the project level) to be defined for information units which represent in this case the work products. Forms define relationships

between templates (by predefined links). Projects instantiate these forms and templates.

Another example of using hypertext systems for documenting software development information are quality pattern [HK97]. A quality pattern is a generic template for information that can be represented as a problem-solution pair (Figure 18). The template consists of sections, each representing an information unit. Guidelines for filling out a quality pattern suggest where to place links (i.e., between which sections or where to place links to related quality pattern and other artifacts) [Fel96].

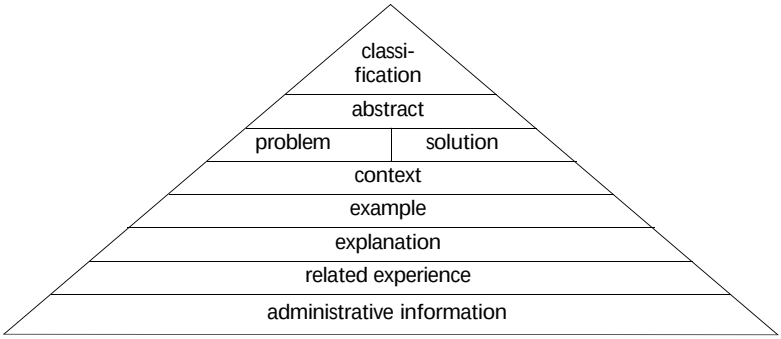


Figure 18: Structure used for quality patterns

Hypertext systems can be enhanced by techniques from the library and information science. For example, the quality pattern approach uses several techniques for finding relevant quality pattern:

- Enumerated classification schema (a top-level WWW page puts all existing quality pattern into context)
- Faceted classification (in the »classification« section, each quality pattern is characterized using one term for each of the facets »type«, »object«, »purpose«, »viewpoint«, »environment«, »analysis technique«, and »restrictions«
- Full-text search

Isakowitz and Kauffman also employ a hybrid approach [IK96]. Using faceted classification, a manageable set of reusable software objects is identified. This set is then arranged as a hypertext system: Each reusable software object represents an information unit; software objects which share a common value in one of their facets are arranged in a *guided tour* (i.e., a closed loop). The automatically constructed hypertext system allows the user to browse the initial set returned by the retrieval system. Thus, hypertext systems are another way to deal with similarity.

5.3 Database Management Systems

Database management systems (DBMS) can be seen from an implementation-oriented view and from a modeling-oriented view. From the implementation-oriented view, DBMS are a means to realize the techniques of library and information science. In addition, they can aid in the maintenance of hypertext systems. In fact, most of the examples given so far use a DBMS as part of their tool implementation. From a modeling-oriented view, DBMS provide general means to structure data.

This section examines database management systems from a modeling-oriented view. In contrast to the techniques of library and information science, DBMS also allow quantitative data to be used for characterizing artifacts as well as the documentation of relationships among artifacts. In contrast to hypertext systems, information units are highly structured, that is, all information units (characterizations) of the same type exhibit exactly the same structure. Thus, characterizations stored in a DBMS tend to be more detailed and formal in comparison to hypertext systems. They can be searched using queries.

5.3.1 Relational DBMS

The schema of a relational DBMS (RDBMS) is defined by tables (or relations), attributes, and types [Cod70]. Tables consist of a set of attributes. A type is associated with each attribute. The set of available types is predefined. Exemplary types are »integer«, »real«, and »text«, but may also include specialized types for sounds, videos, etc. in multimedia RDBMS. Because ranges for numerical attributes can be used for searching, RDBMS support imprecise query information. An attribute may also be a *foreign key* to the same or some other table. Foreign keys are specifications of links.

The attributes prescribe how an entry (characterization) is to be entered into a table, that is, all entries in a table are specified using the same attributes. Integrity rules (e.g., which attributes must be specified, ranges for attributes) restrict possible attribute values. Thus, attributes correspond to the facets of faceted classification. In contrast to faceted classification, types other than terms (= type »text«) may be used. However, classification schemas for terms cannot be specified as part of the database schema. Instead, they must be specified using entries of a separate table for each classification schema of a facet.

Advantages of RDBMS:

- Specification of schemas is mathematically founded and operations on characterizations are defined mathematically.
- RDBMS technology is common, that is, tools are readily available for building information systems based on relational schemas.

Disadvantages of RDBMS:

- The mapping from objects of the real world to the relational schema is not straight forward. For example, as outlined above, the realization of techniques from the library and information science may prove to be difficult.
- The set of operations on the data entries is limited. More sophisticated functionality (e.g., the automatic characterization of texts using keywords) requires a front end.

Although RDBMS can be used to maintain a collection of software-related artifacts, »standard« RDBMS do not seem to be sufficient for efficient and effective retrieval as most authors who employ RDBMS for libraries of software artifacts extend the basic functionality provided by RDBMS (e.g., [DFF97]). In the following, a few more examples are discussed in detail.

For finding software components, Browne et al. extended the notion of the relational schema by systematically arranging the value ranges of the attributes, thus implementing a »full-fledged« faceted classification [BLW90]. Figure 19 shows such a structured relational schema. The relational part is represented by the table, whereas the structured part is represented by the arrangement of the value ranges. All kinds of structures are supported including ordered lists, hierarchies, lattices, and networks.

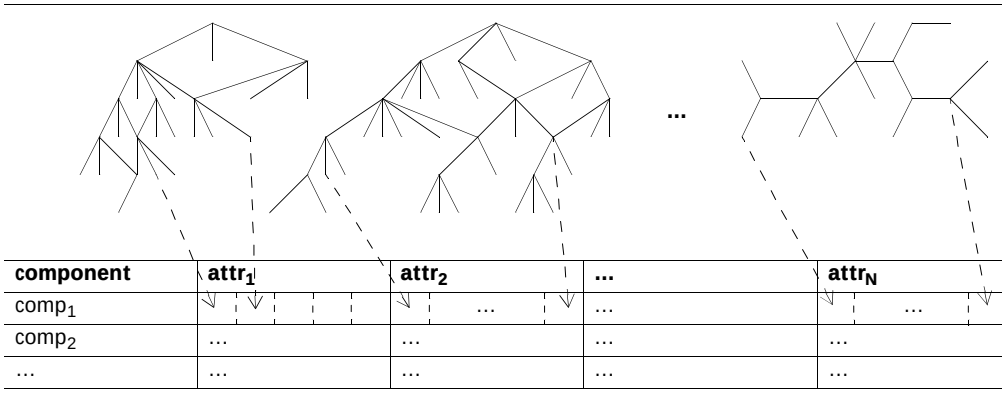


Figure 19: Structured relational schema [BLW90]

The Reusable Software Library (RSL) stores several types of software components including functions, procedures, packages, and programs [BAB+87]. Database entries are extracted automatically from specially marked comments in the source code. RSL uses a relational schema as a basis, but its query interface is a natural language interface using a keyword classification. The initial retrieval also returns components that match only partially the user's query. Because the retrieval may return too many components, an alternative retrieval technique using a score component is suggested by the authors. First, the score component asks the user some preliminary questions to limit the set of potential components. The user is then asked to rate the importance of relevant attributes.

Based on this input, the score component recommends a set of components. The score component works similar to the fuzzy logic approach by Prieto-Díaz and Freeman [PDF87], however, RSL is not based on the user's experience, but asks the user to weight software attributes. The score component can be used for well-understood kinds of artifacts (e.g., data structures). For other kinds of artifacts, determining preliminary questions may be difficult.

In the ALMA environment, RDBMS are used on two levels: the instantiating and the instantiated level [vDD+88]. At the instantiating level, concepts and their intensions are specified. These meta-models are an extension of standard entity-relationship models (e.g., [Che76]) in that n-ary relations can have attributes, they can be defined on unions of entity types, type specialization with multiple inheritance is supported, and a mechanism is provided for defining views for different users and/or tools. Meta-models are stored in the model database (MDB) from where they are instantiated to project databases (PDB). In this respect, the MDB serves as a template for PDBs (cf. [GS90]). At the instantiated level, PDBs are used to record the projects' artifacts. An update component allows to extend the structure of PDBs by extending their meta-model without the loss of already existing project artifacts. The ALMA environment allows reuse only within projects, but not across.

5.3.2 Object-Oriented DBMS

As the examples in the previous section have shown, the expressiveness of relational schemas are insufficient for complex systems. Relations are limited to 1:1 and 1:n relationships; n:m relationships cannot be specified directly, but must be »simulated« using extra tables leading to unnatural data models [Dit89]. Often, tables share some attributes which, again, cannot be modeled directly. Finally, operations are limited to inserting, deleting, changing, and searching table entries (using exact matches). There is a wide semantic gap between possibilities offered by modern programming languages and operations provided by RDBMS. All of this results in low access efficiency.

Object-oriented database management systems (OODBMS) unite two trends:

1 The extension of RDBMS to include concepts of object-oriented development and programming including object-oriented data modeling.

2 The enhancement of object-oriented programming languages (e.g., C++, Java) by database properties (persistence data storage, multi-user access, etc.).

An object-oriented data model is made up of a set of classes [KM93]. Each class consists of a set of typed attributes and operations. Types may be simple types (integer, real, text, reference, etc.) or compound types (list, set, record, etc.).

Through bidirectional lists of references it is possible to model n:m relationships. In addition, object-oriented mechanisms like inheritance are supported. Existing OODBMS provide some rudimentary query mechanisms, but vary from provider to provider. However, user-defined operations may be added to provide needed functionality.

Advantages of OODBMS:
- Real world objects can be modeled more naturally in comparison to RDBMS.
- Possibility for user-defined operations offers high flexibility and increases access efficiency.
- The same notation for data modeling (e.g., UML [Rat97]) can be used for both an object-oriented database schema and the data to be processed by a software system.

Disadvantages of OODBMS:
- Most of the existing OODBMS suffer from the lack of query facilities.
- Most of the existing OODBMS do not support schema changes beyond the addition of classes. In contrast, it is no problem to add or delete attributes from tables or drop entire tables of RDBMSs.
- There is no full-fledged standardized query language for OODBMS such as SQL [ISO99].

In software engineering, Chee, Jarzabek, and Ramamoorthy used an object-oriented schema for representing project experience [CJR96]. The purpose of the system is to provide experience to project managers. It is capable of answering questions like »Which are high risk modules?«, »What is the actual cost of completed phases to date as compared to the expected cost?«, and »Which software products are adequately tested?«. The special query language PMQL (project management query language), which is based on SQL, allows to query the objects stored.

The Ithaca Tools Environment [FNP92] supports reuse for object-oriented development. The software information base (SIB) stores descriptions of applications, designs, implementations, and correspondences. The latter describe how the three former description kinds are interrelated (e.g., which requirements relate to which designs and implemented objects). Because descriptions may reference each other, the SIB shares properties of both OODBMSs and hypertext systems. The selection tool supports the search for development information by browsing and querying functionality. The query facility provides a formal query language and an application domain-based thesaurus [FFD96]. A query is specified using weighted keywords that describe the functionality of the searched component. A set of candidate components is returned. For each of the candidate components, a confidence value is given which expresses how well the component matches the user's query.

Gäßner and Stummhöfer implemented a system for comprehensive reuse (CORE) of all kinds of software experience [Gäß95, Stu95]. CORE marks an extension of a system developed by the author of this dissertation for the reuse of process models [Tau93]. It consists of two major components: a transformer that converts a CORE schema into an object-oriented schema for OBST (an OODBMS) [CRS+92] and the user interface COUSIN (CORE user interface). A CORE schema is described using WieSpra (Wiederverwendungssprache – German for »reuse language«) and consists of a set of classes with attributes and relations between the classes. Integrity rules (i.e., 1:1, 1:n, n:m relationships) for relations can also be specified, thus taking away one of the disadvantages of OODBMS (since integrity rules are not supported by many OODBMSs). In addition, WieSpra provides constructs for distance definitions. If defined adequately, they sort candidate artifacts according to their expected modification effort. Distances are computed between each potential artifact and the query. Distance definitions are based on the attribute values of the potential artifact and the attribute values of the query as well as on parameters that can be specified by the user. Searching for artifacts is performed in two phases:

1 A query-based retrieval is performed to find a good starting point for the second phase. The retrieval is performed similarity-based using the distance defined in the CORE schema. Compound queries that correspond to join operations of RDBMS (queries encompassing more than one concept, e.g., a query for a design document may also include a characterization of the requirements document, the design was derived from) are supported.

2 Starting from any of the returned artifact characterizations, the collection may be browsed using the relations among the characterizations.

5.3.3 Object-Relational DBMS

The disadvantages of OODBMS over RDBMS and vice versa have led to a crossing of both approaches called object-relational DBMS (ORDBMS). ORDBMS unite the advantages of RDBMS (extensive query facility, schema evolution, integrity control, etc.) with those of OODBMS (concepts of inheritance, user-defined operations and types, etc.) [Kim96]. Currently, there are no widely accepted standards for ORDBMS although extensions of SQL with object-relational elements are in the process of being standardized [Luf99].

Advantages of ORDBMS:
- Combines advantages of RDBMS and OODBMS, while avoiding some of their disadvantages

Disadvantage or ORDBMS:
- No widely accepted standards yet

All kinds of software engineering experience can be stored in the reuse repository described in [Fel99]. The reuse repository is divided into two sections called the experiment-specific and organization-wide section, respectively. The experiment-specific section contains project-specific experience such as project plans and deliverables. In contrast, the organization-wide section contains experience that can be reused across projects. It is subdivided into the areas of process modeling, measurement, qualitative experience, technologies, and component repositories. The artifacts' characterizations are stored in the ORDBMS, whereas the artifacts are stored as files. However, the ORDBMS intercepts the file access operations, thereby ensuring the consistency between the artifacts´ characterizations and their corresponding artifacts. Relationships among artifacts can be documented. The following types of relationships are distinguished: measures/ measured_by, is_about/has_part, uses/used_in, gains/gained_in, uses/ used_with, and explains/explained_in.

5.4 Knowledge-Based Systems

All approaches discussed so far focus on the storage and retrieval of data. Some of the systems feature a component to rank the retrieved artifacts according to their applicability (e.g., [PDF87]). Through knowledge about similarity, these systems are able to learn in the following sense. Assume that five artifact characterizations (A_1, A_2, A_3, A_4, A_5) match a query q and that A_1 is estimated by the system to be the most applicable artifact. Assume further that three new artifact characterizations (A_6, A_7, A_8) that match q are stored. If the same query q is issued again, the system may return a new most applicable artifact, e.g., A_7. Thus, the system has *learned* that there is an artifact which is better applicable than A_1 – namely A_7. In terms of Section 2.1, parts of the characterization (data) are interpreted as information, and knowledge (in form of a similarity definition) causes the system to behave differently, i.e., to make another recommendation. In contrast, a standard DBMS[1] is not able to learn in this sense. It would return all eight characterizations as a match for q without giving a hint about their applicability.

Knowledge-based system

Systems that base their behavior (partly) on knowledge are called *knowledge-based systems*. A fundamental assumption of the knowledge-based system paradigm is captured by the »knowledge representation hypothesis«:

1 In this dissertation, deductive DBMS are not considered to be standard DBMS. Deductive DBMS are rule-based [BS96]. Rules allow, for instance, to infer new data sets or information. In this sense, deductive DBMS can be seen as knowledge-based systems.

Knowledge representation hypothesis

Within an intelligent system there exist »structural ingredients« which we as observers take to represent the system's knowledge. These structural ingredients not only exist, but also play a causal role in producing the system's intelligent behavior.

[AN95, p. 202]

Knowledge representation (KR), KR construct, KR formalism, representation structure

The way knowledge is represented is specified using a *knowledge representation formalism* [Rei91]. A formalism consists of a set of *knowledge representation constructs* (short: constructs) and a set of operations that retrieve or change instances of the constructs. Construct instances are called *representation structures*. All representation structures together make up the *knowledge representation (KR)* of the system.

This section surveys the most important classes of representation formalisms. One of the difficulties in describing KR formalisms is the fact that no standards exist. Many varieties and extensions of the basic constructs exist in literature, each solving some special problem. For instance, certain extensions deal with the various dimensions of knowledge introduced in Section 2.2. Due to the limited space, these extensions cannot be discussed here. The interested reader is referred to some introduction on knowledge representation (e.g., [Ric89, Rei91]).

5.4.1 Logic-Based

Logic-based representation formalisms are based on mathematical formulas, most often on first order predicate logic [Ric89]. Using well-established rules (e.g., the modus-ponens rule) and self-defined axioms, new knowledge can be inferred.

Advantages of logic-based representations:
- Semantic is precisely defined (mathematically).
- If artifacts are specified formally, this representation formalism is a natural means to characterize the artifacts.

Disadvantages of logic-based representations:
- Most artifacts cannot be specified formally. They also require informal text description for their characterization.
- If many axioms are specified, logic-based representations are difficult to comprehend.

Penix and Alexander use formal specifications for retrieving and assembling code components [PA97]. Code components are specified using two predicates, a precondition and a postcondition. A component solves completely a problem if it results in one of the problem's valid outputs for all legal inputs of the problem. The retrieval mechanism of Penix and Alexander also allows a partial match

that matches only with respect to the postconditions. A partial match must be completed by another component matching completely a new problem with the same precondition as the original problem, but a postcondition that satisfies the precondition of the previously selected component. Figure 20 shows possible matches (I_P, O_P, I_C, and O_C denote precondition and postcondition of problem and component specification respectively).

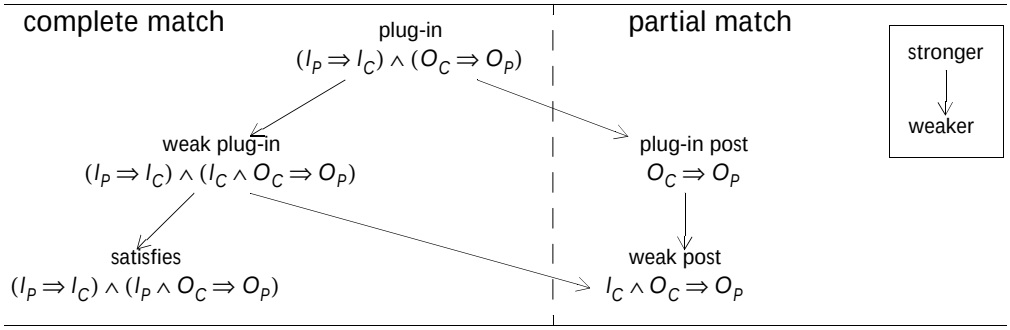


Figure 20: Lattice of specification matches [PA97]

Chen and Cheng use both specifications and signatures of functions to match code components [CC97]. They also consider generalization for matching specifications, that is, if *H* is a function implementing the function specification *h*, then *H* also implements the function specification *g* that is more general than *h*. This generalization corresponds to the complete match of Penix and Alexander.

Mili et al. use formal specifications to match components to a query as well [MMM94]. However, in case an exact query fails, they retrieve and order similar components based on the predicted modification effort [JDFM97].

5.4.2 Rule-Based

A rule-based system consists of a set of rules called *rule base*, a *fact base*, and an interpretation process that checks rules for their applicability and executes applicable actions [Rei91, Ric89]. A rule is a pair consisting of a precondition and an action. The action can be performed if the precondition is true. The precondition refers to the fact base (which can be represented using any representation formalism). The action typically changes the fact base, but can also ask the user for some input or output some computation results.

Advantages of rule-based systems:

- As a construct, rules are adequate for representing rule-based dependencies.
- Rules support human expert thinking (especially for generalized knowledge) in terms of rules and heuristics [And88].
- Rules are modular, that is, single rules can be added or removed at any time independent of other existing rules. This is possible, because rules do not

refer to each other directly, but only indirectly through changes of the fact base.

Disadvantages of rule-based systems:
- Information must be stable and well- or semi-structured.
- It is difficult to overview (intellectually) the control flow even for a small number of rules. This means that rule-based systems are difficult to comprehend, maintain, and verify.
- Since the control flow is not specified explicitly, performance problems can occur.

In software engineering, rule-based systems have been used at the Software Engineering Laboratory (SEL) of NASA's Goddard Space Flight Center (GSFC) for trend analysis and guidance of software development projects [HKV92]. Trend analysis analyzes the trend of the measures of interest. If the current value is outside the expected range, probable causes of the deviation are presented. Guidance provides recommendations on how to solve problems that have been detected through trend analysis and overall assessment functions.

The Software Management Environment (SME) uses databases containing objective and subjective data about the project as its fact base. For historical reasons, two rule-based systems have been implemented:

1 The first system is used for trend analysis and guidance. Figure 21 shows an excerpt of the rule base. The example shows that one possible reason for an unexpected high number of software changes (problem or deviation) is an »inexperienced development team« (see rules of type I on the left of the figure). To evaluate the truth of the reason, rules of type II are used (shown on the right of the figure). In the example, team experience is determined by four factors (experience with application, language, environment, and tools). The factors are rated either by other rules of type II or by looking up the ratings in the databases with subjective and objective data. The weights are used to determine the accuracy of the conclusion. In the example, all factors are determined by subjective data [HKV94]. Experience is rated either as »low«, »normal«, »high«, or »unknown«. Unknown values are ignored for the determination of the team experience. All other values are mapped as follows: »low« to -0.25, »normal« to 0, and »high« to 0.25. The numbers are added and divided by the sum of the weights of the known values. If the result is greater than or equal to 0.5, team experience is considered to be high. If the result is lower than or equal to -0.5, the team experience is considered to be low (i.e., the team is inexperienced). Otherwise, the team experience is considered to be normal.

2 The second system is used only for trend analysis. Its rule table is made up of only one table (see Figure 22). It is used whenever the current ratio of measures (there are nine predefined measure ratios, e.g., number of software

changes per line of code) is deviating from normal. Deviations are determined through analyzing the collected objective data. In addition, the project phase is determined. The example shows three possible reasons if the condition stated on the left evaluates to true. Each reason is associated with a probability value. If more than one rule is applicable and the same reason is diagnosed, the probabilistic values are combined. Finally, the reasons are sorted in descending order of their probability and presented to the user.

Figure 21: Excerpt of a rule base

problem or deviation	**reason or explanation**	**factors that affect team experience**	**weight**
above normal software changes	the development team is inexperienced OR …	experience with application	0.25
		experience with language	0.25
		experience with environment	0.25
		experience with tools	0.25

Figure 22: Excerpt of a rule base

condition	**reason or explanation**	**weight**
if late in coding phase and software changes per line of code are above normal	good quality test plan	0.25
	OR	
	unstable specifications	0.25
	OR	
	error-prone code	0.50

5.4.3 Semantic Nets

Semantic nets go back to models of human memory by cognitive sciences [Rei91]. The models are based on experimental investigations. They suggest that semantically related concepts are represented by structures which are connected in an adequate manner. If a particular concept structure is activated, the activation is spread across all connections of the activated structure. Thus the activation levels of all directly connected structures are raised. The connections can be attributed with a value that determines by how much the activation level should be raised. If the activation levels are raised above a certain threshold, their corresponding concepts are remembered.

Such networks can be represented by nodes (representing concept structures) and edges (representing connections). For semantic nets, the value of the connection is replaced by a label that marks the type of connection. Retrieval is performed by matching subgraphs. Figure 23 shows an example of a semantic net for code components, whereas Figure 24 shows exemplary queries and their result. Note how properties and semantic relationships are inherited using the »is-a« relationship.

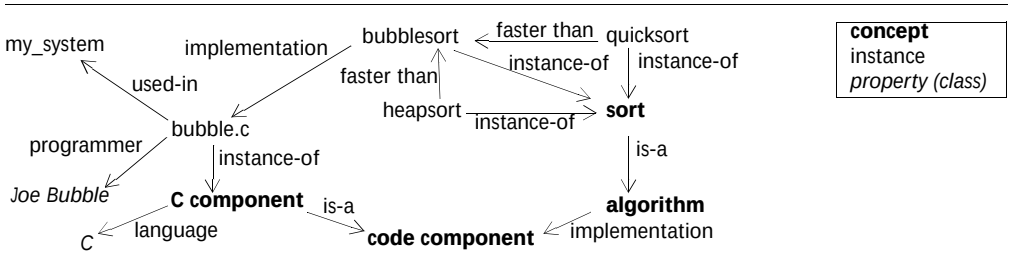


Figure 23: Example of a semantic net (based on [FG90])

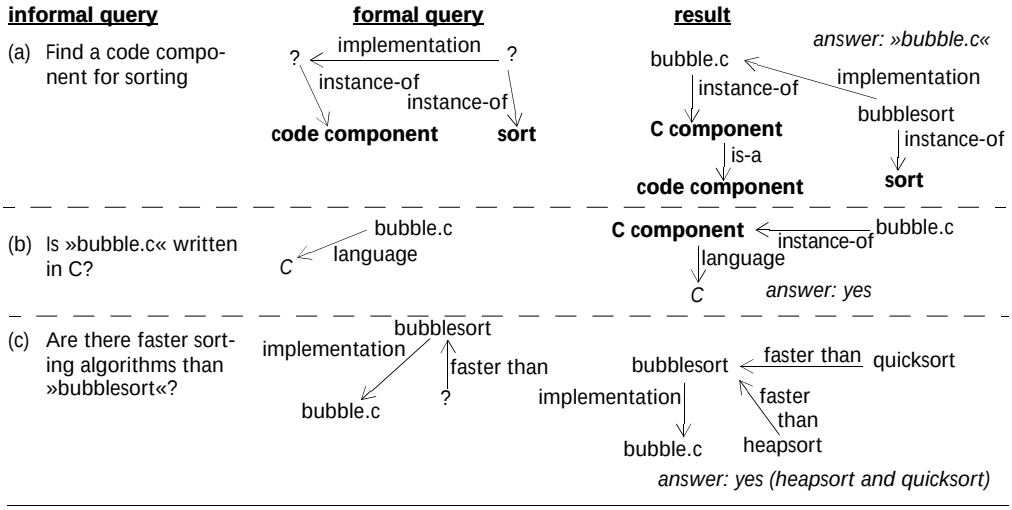


Figure 24: Exemplary queries of the semantic net shown in Figure 23

Advantages of semantic nets:

- Network structures can be comprehended more easily than rules and formulas. For example, the matching can be overviewed (intellectually) more easily than the control flow of rule-based systems.
- On average, the system resources needed to conduct a search is less than for those approaches that use a representation formalism that do not allow any means of structuring (e.g., logic-based and rule-based KRs). This is because the search is usually confined to a small section of the net (cf. the exemplary queries in Figure 24).

Disadvantage of semantic nets:

- Rule-based dependencies cannot be represented as simple and elegant as with other representation formalisms, in particular, logic-based and rule-based systems. The same is true for incomplete knowledge.

In the Reusable Software Library (RSL), the contents are organized using a semantic net [SWT89]. The semantic net does not only contain the software components as concepts, but also related concepts. For instance, benchmark features are stored for characterizing benchmark programs. The semantic net can be used for browsing. It is complemented by a rule-based system that

guides the user in storing and finding software components. For instance, the rule-based system can be invoked at any node of the semantic net to »jump« to nodes other than the directly related nodes based on questions posed to the user.

5.4.4 Frame-Based

In semantic nets, information units are very small. For instance, if a characteristic of an instance is to be recalled, an access operation is necessary that navigates along the corresponding edge from the concept to its characteristic (e.g., in Figure 23, the edge »programmer« has to be traversed to find out the author of »bubble.c«). Frame-based representation formalisms store all information that is closely related to a particular concept or instance in a single structure called *frame*.

Frame-based representation formalisms have their origin in the notion of the schema as viewed by cognitive sciences [And88]. Thus, they go back to a model of human memory too. However, in contrast to semantic nets, the model is not associative, but schema-based. A *schema* is a model for a memory structure that not only considers associations between concepts, but also pays attention to the phenomenon of stereotypical patterns of memory. As an example, let us consider the concept of a code component. The activation of the corresponding schema could be triggered by the need to create a code component. The schema activation results in the mental presence of a (stereo)typical code component with type definitions, functions, variable declarations, and code instructions, maybe in a particular language using specific programming guidelines. A schema activation causes specific expectations that yield a context-dependent interpretation of objects and events. For example, the size of a code component is often seen as a synonym for »lines of code«. If some other schema would be active, e.g., one for a text document, »size« would be interpreted as »number of words« or »number of pages«, but typically not as »number of lines«.

In computer science, schemas are represented by frames. A frame consists of a number of attributes (also called *slots*), each standing for a particular characteristic of the concept or instance the frame represents. There are two types of attributes: terminal and nonterminal attributes. The value of terminal attributes are properties, whereas the value of nonterminal attributes are other frames. Thus, nonterminal attribute values represent semantic relationships. In this way, all concepts and instances that belong to the semantic context of a given concept or instance are grouped together in a single representation structure. Attribute values can be restricted, for example, by specifying value ranges and cardinality for the attributes. Since all relevant attributes of each concept are predefined, incomplete information (in the form of unknown attribute values) can be represented by storing a special value (e.g., »unknown«) for the corresponding attribute.

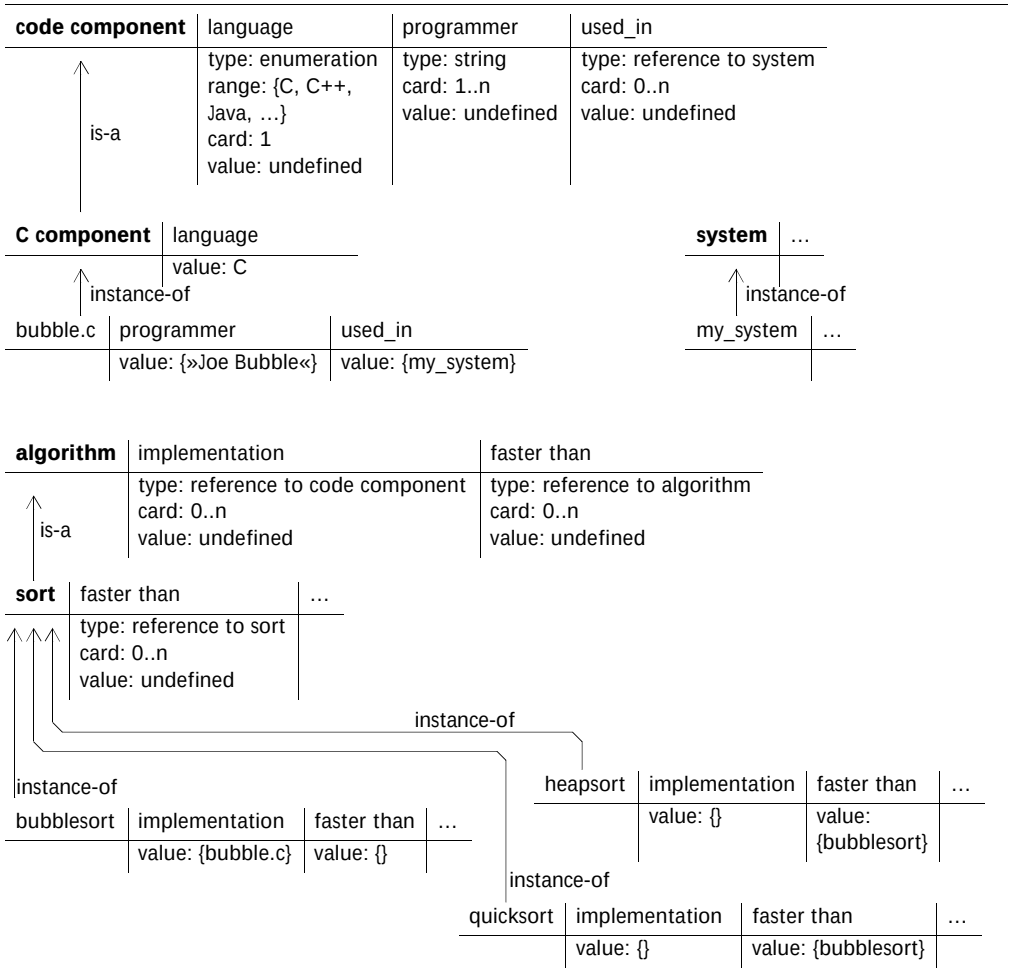


Figure 25: Example of a frame-based representation

Figure 25 shows the example of Figure 23 using a frame-based representation. Note how attributes (and their values) are inherited: the instance »bubble.c« has three attributes: language (value: *C*; inherited from »C component«), programmer (value: *Joe Bubble*; type and cardinality inherited from »code component« through »C component«), and used_in (value: *my_system*; type and cardinality inherited from »code component«). Exemplary retrievals are shown in Figure 26 (only instance frames are shown as a result).

The examples show that frame-based representations are very similar to object-oriented database management systems. What distinguishes frame-based representations from OODBMS is that frame-based representations can infer new knowledge. For example, binsort [AHU83] is faster than quicksort and heapsort. If the frame-based system knows that »faster than« is a transitive relationship, it can infer that binsort is also faster than bubblesort even though this is not documented explicitly. Such knowledge can be specified using, for example, rules. In addition, operations can be assigned to attributes. If attribute values are

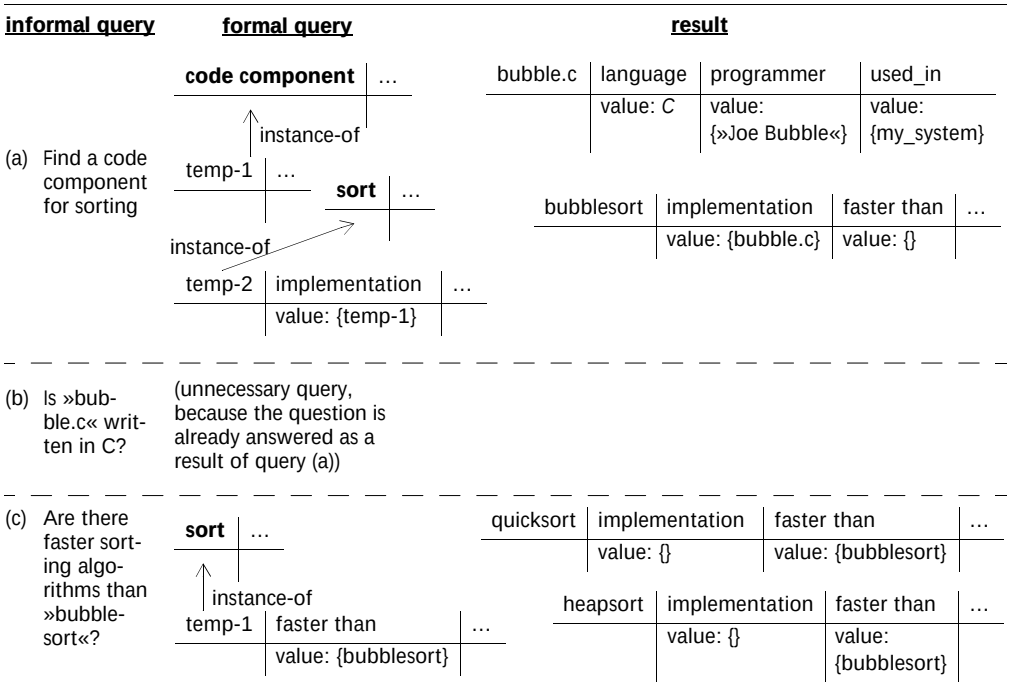


Figure 26: Exemplary queries of the knowledge representation shown in Figure 25

changed, these operations are triggered and can infer attribute values of other concepts and instances.

Advantages of frame-based representation formalisms:

- Frames allow good guidance for structuring knowledge by predefining the needed knowledge for each concept.
- Because all properties and semantic relations that belong to a concept or instance are captured in a single frame, they can be accessed directly after the frame has been retrieved. No further access operations are necessary. In contrast, in semantic nets it is not clear a priori what belongs to a concept and what not.
- Inference rules can be attached to attributes. Applicable rules are thereby restricted to those of activated frames. This typically results in a better efficiency of the inference processes.

Disadvantage of frame-based representation formalisms:

- Rule-based dependencies cannot be represented as simple and elegant as with other representation formalisms, in particular, logic-based and rule-based systems. Hence, in practice frame-based representation formalisms are often complemented with rule-based constructs.

In software engineering, Devanbu et al. use a frame-based representation formalism to store and retrieve information about Definity 75/85, a large switching

system consisting of about one million lines of C code [DBSB91]. Their information system, called LaSSIE, provides information about the:

- Domain model (relations between code and conceptual objects and actions of the software switching domain)
- Architecture
- Features (associations between basic system functions and customer features such as »call forwarding«)
- Code (relations among code-level components)

LaSSIE's retrieval component has a formal query language (ARGON) and a natural language front end which translates a query in natural language into an ARGON query. Only instances are returned as the result of a query. Basic concepts of the KR are actions, objects, doers, and states. These are further specialized. In queries, either instance names or concept names may be used. In case, too few or too many instances are returned, the query can be broadened or narrowed by navigating in the concept hierarchy for any of the attributes values. LaSSIE's classifier associates new instances automatically to the proper concept. This eases the maintenance of the knowledge representation.

LaSSIE's knowledge representation was developed manually. This, of course, is an effort-intensive task. Girardi and Ibrahim developed a system that automatically generates a frame-based representation from textual software components [GI94, GI95]. Software descriptions are analyzed morpholexically (i.e., standard forms and grammatical categories of words as well as their semantic relationships to other words in a thesaurus are identified), syntactically, and semantically. Queries (in natural language) are processed similarly. The frame-based representation of the query is then matched with the processed software descriptions using a similarity analysis.

With the exception of SME, all knowledge-based systems discussed so far were developed for the reuse of code components. Oivo's ES-TAME goes beyond code reuse by supporting the complete quality improvement paradigm cycle [Oiv94]. Oivo distinguishes between passive software engineering models (life cycle models, project models, resource models, design techniques, quality models, etc.) that contain mostly descriptive knowledge and active GQM[1] models that contain mostly procedural knowledge [OB92]. GQM models make the descriptive knowledge of the software engineering models (SEMs) operational by manipulating and using the knowledge of SEMs in setting goals, answering questions, and collecting data. Oivo uses various kinds of relationships among the models:

1 GQM stands for goal/question/metric paradigm and is an innovative technique for goal-oriented software engineering measurement [BDR96].

- **Is-a/children.** This kind denotes single inheritance. Although multiple inheritance is not supported, is-a relationships can be changed during runtime resulting in a view mechanism. Inherited attribute values are not forgotten. Therefore, they become accessible again if the is-a relationship to the original model is reestablished.
- **Instance-of/instances.** This kind has the usual semantic.
- **Part-of/has-parts.** This kind describes composite objects.
- **Defined-by/defines.** This relationship is used if an instance can be defined by one or more other instances. For example, a GQM goal can be defined by subgoals. If the GQM goal and all its subgoals are each represented by an instance, the GQM goal would have a defined-by attribute referencing its subgoals.
- **Compatible-objects.** Two instances are compatible if they can be used together. For example, the function point technique might be compatible with MIS projects, but not with real-time projects. The relationship »compatible-objects« is utilized for reuse. Starting out with some artifact (e.g., a design document), the compatibility relationships reference other instances that can be used in conjunction with the current instance.
- **Counterpart.** Counterpart relationships are normally used to define relationships between objects which are used in the same context to build a larger scheme. For example, a counterpart relationship can be established between data flow diagram models and design level coupling models, because both are used in the same context to assess the quality of the system design.
- **Dynamic-attribute.** This relationship allows to associate an instance's attribute with the attribute of another instance. For example, the number of source lines of code (SLOC) may be stored in the product model. To avoid redundant storage, the attribute may be associated with the SLOC attribute of the resource estimation and defect slippage models. Thus the SLOC value is maintained only in one place and automatically propagated if it is changed.

To summarize, the system supports the user in creating GQM and measurement plans. The GQM models actively collect data from the SEMs and interpret them appropriately (e.g., by adding the effort figures for all subsystems to the total effort figure for the system). Retrieval for reuse is realized via the »compatible-objects« relationship.

Ostertag's Extensible Description Formalism (EDF) is also capable of organizing an experience base with various kinds of artifacts [Ost92]. EDF does not only provide means to define concepts and attributes, but also rule-based dependencies among attributes (using assertions) and how to compute similarities for retrieval. Attribute types are not limited to enumeration types, but also include numeric and text types. EDF is based on an earlier prototype called AIRS [OHPDB92] which was used to retrieve similar packages based on package characteristics and a characterization of the needed operations.

5.4.5 Case-Based

Approaches that abstract from existing knowledge (e.g., [HKV92, SWT89]) usually require a large amount of effort to develop and maintain the knowledge representation. »This is serious because a system then easily gets a static character, that is, changing knowledge content is something that should be avoided.« [AN95] Therefore, many approaches refrain from generalizing existing reusable parts. They simply *characterize* the reusable parts. By doing so, a 1:1 relationship is established between a reusable part and its characterization. The reusable parts become solutions to decision-making problems. However, often there is no perfect solution stored (this would result in reuse without modification), but solutions may be stored that can be easily tailored to solve the problem at hand. Two strategies can be employed to find partially matching parts:

1 **Widening the scope of the query** (e.g., by using wildcards, not specifying certain attribute values, or using a broader term for some attribute values). This strategy is used, for example, by [BLW90, DBSB91, PDF87].

2 **Computing the similarity between a query and a reusable part.** Here, the underlying assumption is that similar problems (represented by characterizations) can be solved with similar solutions. For example, [DFF97, Gäß95, JDFM97, Ost92, Tau93] make use of this assumption. The strategy has the additional advantage of being able to order the partially matching parts according to their estimated applicability.

The second strategy is the underlying principle of case-based reasoning [AP94, Alt97, BSW96, Ric98]. According to Richter, the knowledge stored by case-based reasoning systems can be divided into four *knowledge containers* [Ric98]:

1 **Case base.** The case base consists of a set of *cases*. Each case represents a problem-solution pair. In software reuse libraries, a case is comprised of a characterization and its corresponding reusable part.

2 **Vocabulary.** The vocabulary is used to describe the cases. It is typically represented using a frame-based representation formalism. While the vocabulary corresponds to concept frames, cases can be interpreted as their instances.

3 **Similarity.** Knowledge in this container defines what »similar« means. It is most often represented by mathematical functions or – if the value range of an attribute is finite – by a table, but more complex representations are also in use in research prototypes.

4 **Solution transformation.** Retrieved solutions can be tailored to suit the problem at hand. The knowledge stored in this container tells how this can be done. For example, if the solution is described through a set of attributes,

rules may be used to infer certain attribute values based on the query specification.

The notion of knowledge containers has two important aspects:

1 The knowledge stored in one container can be changed (almost) independently of knowledge stored in other containers. This eases maintenance. For example, new cases can be inserted or old cases removed at any time without having to change the vocabulary, the similarity, or solution transformation.

2 Knowledge can be transferred between containers, in particular, there is a trade-off between case-specific knowledge (stored in the case base) and general knowledge (stored in the other containers). For example, a large case base with little solution transformation knowledge can be used instead of a small case base with lots of solution transformation knowledge.

Advantages of case-based reasoning systems:
- Similarity estimates the applicability on an ordinal scale. In contrast, DBMS only match or do not match a reusable part to a query without giving a hint how close the reusable part is to the needed part.
- In contrast to the techniques of library and information science, many case-based reasoning systems allow – in addition to symbol types which can be used for enumerated classification schemas – numeric and text types as well as compound types such as sets. With sets, overlapping classification schemas and keywords can be defined.
- In contrast to faceted classification systems, variants of a faceted schema can be used. Faceted schemas can inherit attributes from other faceted schemas. For example, the faceted schema for »sort algorithms« inherits all attributes from the faceted schema for »algorithms« and can define additional attributes. This ensures that all reusable parts can be characterized appropriately.

Disadvantages of case-based reasoning systems:
- If no cases are available, it is hard to define an appropriate vocabulary (this, however, is also true for the development of DBMS and classification schemas).
- Retrieval is not as efficient as for DBMS, because of the similarity computation although some optimization techniques exist (e.g., INRECA trees that are based on k-d trees [ATW96]). Hence, case-based reasoning systems should not be used for huge amounts of data sets.

Numerous case-based reasoning systems have been built for reusing software. For instance, CABAROS retrieves object-oriented code components [BE95, Rei94]. Code components can be retrieved either by a functional specification of a class, a method, or a combination thereof. Each method of a class is associ-

ated with a code fragment which is the solution to the specified code component (= problem). A class or method matches if their similarity to the query is above a certain threshold which can be specified by the user. Compound queries (e.g., a class specification with a set of method specifications) match if (a) the class specification matches with a class *C* and (b) »most« of the methods match and are part of *C*. Since it is possible that no class exists that matches all specified methods, CABAROS also attempts to find classes that provide most of the specified methods. Explanations for differences in attribute values explain the consequences, that is, how the class or method can be tailored to the user's needs. Later, these ideas were reimplemented in the Smalltalk system VisualWorks [CHEK96]. Another reimplementation [BE95] used the INRECA prototype [AAB+95].

Gonzáles and Fernández also developed a CBR system for object-oriented code reuse [GF97]. In their system, a case contains a description, an associated solution, and a justification of the solution. The description corresponds to the functional description of a component. The case description has two parts: (a) a set of keywords extracted from the component's documentation using an automated full-text characterization technique and (b) a set of attribute-value pairs. The user is offered two kinds of retrieval techniques. One uses a natural language interface and matches cases using (a). The other uses a graphical interface for specifying appropriate attribute values and searches for cases using (b). The solution (that is, the code itself) is represented as a string. Finally, the justification corresponds to the interface information of an artifact. It documents design decisions and commitments, feasible customizations, and dependencies among implementations. Thus, it serves as the basis for modifying components.

Gómez arranges code components in a hierarchy of abstraction levels [Gom97]. The software system is located at the top of the hierarchy, whereas the final code is located on the bottom of the hierarchy. The hierarchy can be built up bottom-up. The model underlying the case-based reasoning system considers user feedback regarding the degree of success (to predict the applicability of a case), frequency of use (to mark appropriate cases as obsolete), adaptation of cases, and various views on the cases.

Figure 10 on page 79 shows that case-based reasoning is not restricted to retrieval, but supports the complete cycle of continuous, incremental learning. Katalagarianos and Vassiliou present a system that implements the whole cycle for abstract data types [KV95]. Code components are characterized by their specification consisting of a set of pairs of the form <name, type>. Two pairs are similar if they have the same name and (a) a user has told the system that the types are similar during a previous reuse process or (b) both types have a common supertype and the user accepts the types to be similar. Adaptation rules are associated with each pair of similar types. For example, a »stack of integer« can be converted to a »stack of char« by substituting all occurrences of »integer« by »char«. If none of the specifications match, a new code component may be cre-

ated from scratch. If components are retrieved that are not relevant, their specifications may be extended by the user thus avoiding that the irrelevant components are suggested again for the same query. Hence, the system learns interactively.

5.5 Evaluation

Although the approaches presented cover a wide spectrum, they all contribute to the state-of-the-art of a technical infrastructure for an experience base. Approaches can contribute to knowledge representation on the two levels of conceptual and meta knowledge (see discussion on meta knowledge on page 45). On the conceptual knowledge level, contributions focus on the definition of concepts, their intensions, rule-based dependencies among the concepts, and similarity. On the meta knowledge level, contributions focus on how to define conceptual knowledge.

Meta knowledge can be represented explicitly in the form of a representation formalism using constructs and operations. Here, it is of interest in how far the constructs support the conceptual knowledge needed by the methods for populating, maintaining, and utilizing the experience base. The more natural the representation formalism, the easier will be the maintenance of the conceptual information (prerequisite for the task »structure«).

The constructs restrict the set of possible operations on both the conceptual and the characterization information. These operations may be further restricted by a given implementation. A given implementation may automate a task, guide the user in accomplishing a task (e.g., by providing meaningful characteristics for a query), or simply support a task (without providing guidance to the user, for instance, if the user must enter a query with a defined syntax as plain text).

In the following, contributions on the meta knowledge level are evaluated according to the constructs provided (see Table 5), the tolerated dimensions of artifact characterizations (see Table 6), and the tasks the approach supports, guides, or automates (see Table 7).

Table 5 lists for each approach the scope it was developed for and the kinds of conceptual knowledge that can be defined using the approach. The kinds of conceptual knowledge are based on the definition of »intension« (Definition 8 on page 26). Here, property classes are further classified as typed (»with value range«) or untyped (»without value range«). Semantic relationships are also classified as typed (i.e., references can point only to (instances of) predefined concepts) or untyped (i.e., references can point to any concept). Note that pure keyword systems do not allow the definition of concepts in the sense as defined in Definition 7 on page 26. Therefore, the four columns are not marked for these approaches. If the approach allows to define some kind of rule-based

dependencies (see Definition 9 on page 26), the corresponding column is marked. Since similarity plays an important role for the experience base (cf. R34 (similarity-based retrieval) on page 61), a column has been added to indicate whether an approach allows similarity definitions.

The table shows:

- Although only representative approaches have been chosen for the survey, the survey shows that most approaches focus on code reuse. Some approaches widen the scope to work products. Only a few approaches have different foci (e.g., [CJR96, HKV92, OB92, Tau93]).
- Most approaches support only a fixed number of concepts. Only a few approaches attempt to cover all kinds of software engineering experience (e.g., [Gäß95/Stu95, OB92, Ost92]).
- Most approaches that allow the definition of concepts use typed property classes and – if semantic relationships are supported – also typed relationships.
- Only a few approaches use rule-based dependencies to ensure the consistency of the stored knowledge. This aids in the maintenance of the characterizations.
- Many approaches employ some kind of similarity to find similar artifacts. However, the flexibility of the similarity definitions varies from none (e.g., [JDFM97/MMM94]) to full control (e.g., [Gäß95/Stu95, Ost92])
- Only [Gäß95/Stu95, OB92, Ost92] allow the definition of rule-based dependencies, similarities, and typed property classes and semantic relationships. However, all three approaches provide different rule-based dependencies:
  - Integrity rules as known from DBMS [Gäß95/Stu95]
  - Value propagation via »dynamic-attribute« relationships [OB92]
  - Assertions that must always be true [Ost92]

Table 6 shows the various dimensions supported by each approach for the artifact characterizations (cf. Section 2.2.2). The criteria for considering a dimension as supported are as follows:

- **Incomplete.** The knowledge representation constructs underlying the approach must allow (a) storage of a partial characterization and (b) search over partial characterizations. For example, the faceted classification as defined by Prieto-Díaz and Freeman [PDF87] does not allow the storage of incomplete characterizations. However, if a special term »undefined« is added to each of the facets, incomplete characterizations can be stored. Through an appropriate definition of the conceptual distance in the weighted conceptual graph a meaningful retrieval is still possible. Therefore, the corresponding column is marked in this case.
- **Uncertain.** Either the approach must provide a similarity-based retrieval (e.g., [Ost92]) or the certainty of the information must be explicitly documented (e.g., [HKV92]).

Table 5: Contributions on the meta knowledge level: provided constructs

approach	scope	property classes		semantic relationships		rule-based dependencies	similarity
		with value range	without value range	to predefined concepts	to any concepts		
[BLW90]	software components	X					
[BAB+87]	software components	X					X
[CJR96]	project information	X		X			
[CC97]	code components						
[DFB99, DFF97]	object-oriented code artifacts						X
[DBSB91]	software functions			X			
[Fel99]	all software engineering artifacts	X		X			
[FNP92]	object-oriented code artifacts	X		X			X
[Gäß95, Stu95]	all software engineering artifacts	X		X	X	X	X
[GS90]	work products		X		X		
[Gl94]	software descriptions						X
[Gom97]	software components	X					
[GF97]	object-oriented code components	X	X	X			X
[HKV92]	project management information					X	
[HK97]	quality patterns		X	X			
[JDFM97, MMM94]	software objects						
[IK96]	code components	X					
[KKL+98]	reference architectures (subsystem model, process model, module model)	X		X		X	
[KV95]	abstract data types	X		X			X
[MB94]	UNIX man pages						X
[OB92]	all software engineering artifacts	X		X		X	
[Ost92]	all software engineering artifacts	X		X		X	X
[PA97]	software components						
[PDF87]	code components	X					X
[BE95, Rei94]	object-oriented code components	X		X			X
[SWT89]	code components	X		X			
[Tau93]	software process models	X					X
[vDD+88]	project artifacts	X		X			
methodology of this dissertation (DISER)	all software engineering artifacts	X		X	X	X	X

- **Imprecise.** The representation constructs of the approach must allow the specification of ranges or sets with an »or« semantic for both artifact characterizations and queries.
- **Inconsistent.** The approach must allow the documentation of inconsistent relationships. For example, if a design document is changed without updating the corresponding requirements document, the relationship between both documents is considered to be inconsistent.[1] Yet, the relationship should be documented because the old requirements document is (still) the best design specification available.
- **Informal (textual).** The approach must allow natural text as part of the artifact characterizations.
- **Formal (non-textual).** The approach must allow other means than natural text to characterize artifacts. Other means not only include numerical and logic-based specifications, but also keywords extracted from a text or terms taken from a classification schema.
- **Different levels of abstraction.** The approach must allow the characterization of artifacts on different levels of abstraction. This means that an artifact´s characterization can be accessed at different levels of abstraction.
- **Context-dependent.** There must be knowledge representation constructs for the intended application domain, the application history, the solution domain, and the development history of the artifact (cf. Figure 7 on page 68).

The table shows:
- None of the approaches support imprecise or inconsistent characterizations.
- Most approaches do not support uncertain characterization information.
- Only a few approaches allow both informal and formal information to be part of the characterization (here, [Gäß95/Stu95, GF97, Ost92]).
- Although the four types of context-dependencies listed above is not explicitly mentioned in the description of the approaches, some approaches allow the documentation of contexts using the available knowledge representation constructs.

Finally, Table 7 shows the supported tasks. The table distinguishes between methodological (i.e., operations that can be implemented) and implementation (i.e., operations that actually were implemented) aspects. Thus, the evaluation is technology-oriented. For this purpose, only the generic tasks that are at the bottom of the task-method decomposition (see Figure 14 on page 126) are of interest. Strictly seen, the tasks are only supported if all of the requirements listed under »requirements« of the corresponding task description are fulfilled. How-

1 Note that this type of inconsistency is different from those inconsistencies detected by rule-based dependencies. Rule-based dependencies define restrictions on the characterization, whereas the type of inconsistency referred to here concerns the artifacts themselves.

Table 6: Contributions on the meta knowledge level: dimensions of artifact characterizations

approach	incomplete	uncertain	imprecise	inconsistent	informal (textual)	formal (non-textual)	different levels of abstraction	context-dependent: application domain	context-dependent: application history	context-dependent: solution domain	context-dependent: development history
[BLW90]	X					X					
[BAB+87]	X					X					
[CJR96]	X					X					
[CC97]						X					
[DFB99, DFF97]	X	X			X		X				
[DBSB91]	X					X	X				
[Fel99]	X				X	X			X	X	X
[FNP92]	X	X				X					
[Gäß95, Stu95]	X	X			X	X	X	X	X	X	X
[GS90]	X				X		X	X	X	X	X
[Gl94]	X				X						
[Gom97]	X	X				X	X				
[GF97]	X	X			X	X	X				
[HKV92]						X	X	X		X	
[HK97]	X				X		X	X		X	X
[IK96]	X					X					
[JDFM97, MMM94]						X					
[KKL+98]						X	X				
[KV95]	X					X					
[MB94]	X	X			X		X				
[OB92]						X	X	X	X	X	X
[Ost92]	X	X			X	X	X	X	X	X	X
[PA97]						X					
[PDF87]	X					X		X		X	
[BE95, Rei94]	X	X				X					
[SWT89]						X	X				
[Tau93]	X					X		X		X	
[vDD+88]	X					X		X	X	X	X
methodology of this dissertation (DISER)	X	X	X	X	X	X	X	X	X	X	X

ever, this would lead to spare coverage. Therefore, the requirements are relaxed (with the consequences listed under the requirements that do not have to be fulfilled). More specifically, the requirements that must be fulfilled for supporting the tasks are as follows:

- **analyze efficiency (T51, page 117) and analyze effectiveness (T52, page 118).** R15 (data collection for evaluation), R16 (data analysis for improvement); *for guidance:* relevant indicators must be predefined and an automatic trend analysis must alert the administrator of the experience base if deviations are detected.

- **structure (T56, page 121).** R9 (separation of characterization and conceptual information), R13 (maintenance of conceptual information); *for guidance:* provision of templates for knowledge representation constructs (e.g., through usage of a graphical user interface).
- **forget (T54, page 119).** R27 (artifacts' status); *for guidance:* predefined criteria when task is to be performed and automated analysis using the criteria.
- **characterize initially (T38, page 110).** R20 (tolerance of incomplete information), R21 (tolerance of uncertain information); *for guidance:* listing of relevant descriptors
- **check for existing parts (T42, page 112).** R34 (similarity-based retrieval); *for guidance:* listing of relevant descriptors
- **complete characterization (T47, page 114).** R17 (artifact recording); *for guidance:* listing of relevant descriptors
- **publish (T48, page 115).** R27 (artifacts' status); guidance not meaningful
- **collect descriptors (T5, page 91).** Query-based retrieval; *for guidance:* listing of relevant descriptors
- **interpret problem (T6, page 91).** Query-based retrieval, rule-based dependencies must be applied to query information; *for guidance:* the rule-based dependencies are used during the query formulation, that is, the system restricts the value range of attributes »on-the-fly« based on attribute values the user has already entered
- **infer descriptors (T7, page 92).** Query-based retrieval, completion knowledge must be applied to query information; guidance not meaningful
- **browse (T9, page 93).** Information units must be traversable via links; *for guidance:* links must be highlighted
- **search (T10, page 93).** Query-based retrieval; guidance not meaningful
- **calculate similarity (T13, page 95).** Defined similarity; *for guidance:* user is asked for values not part of the artifacts´ characterization
- **explain similarity (T14, page 96), sort (T15, page 96).** Defined similarity; guidance not meaningful
- **prioritize (T17, page 97).** Always supported since this task is actually to be performed manually by the user (not indicated in the table); *for guidance:* user is asked for values not part of the artifacts´ characterization
- **explain prioritizing (T18, page 98).** Automated if system explains the ordering based on user input for task »prioritize«; *for guidance:* system prompts the user for an explanation only if prioritization does not correspond to the system's recommendation
- **explain selection (T21, page 99).** R15 (data collection for evaluation); *for guidance:* system prompts the user for an explanation only if selection does not correspond to the prioritization
- **generate (T28, page 103).** R32 (browsing) for finding a suitable generator; automated if suitable generator is invoked automatically based on the artifact to be utilized; guidance not meaningful
- **give feedback (T31, page 104).** R15 (data collection for evaluation); *for guidance:* relevant indicators must be predefined

Table 7: Contributions on the meta knowledge level: supported, guided, and automated tasks

- [ ] task cannot be supported
- s task can be supported by approach
- g user can be guided in accomplishing the task by approach
- a task can be automated by approach
- **s** task is supported by implementation
- **g** user is guided by implementation
- **a** task is automated by implementation
- n/a not applicable

	T32: learn								**T2: reuse**													
	T50: analyze SEEMS		**T55: package**		**T33: record**				**T3: retrieve**												**T22: utilize**	
approach	**T51: analyze efficiency**	**T52: analyze effectiveness**	**T56: structure**	**T54: forget**	**T38: characterize initially**	**T42: check for existing parts**	**T47: complete characterization**	**T48: publish**	**T5: collect descriptors**	**T6: interpret problem**	**T7: infer descriptors**	**T9: browse**	**T10: search**	**T13: calculate similarity**	**T14: explain similarity**	**T15: sort**	**T17: prioritize**	**T18: explain prioritizing**	**T21: explain selection**	**T28: generate**	**T31: give feedback**
[BLW90]			**s**	s			**g**	s	**g**				**a**								
[BAB+87]			s				s		**g**		**s**		**a**	**g**	a	**a**	**g**	a			
[CJR96]			s	s			**g**	s	**g**				**a**								
[CC97]							**s**		**s**				**a**								
[DFB99, DFF97]			n/a		**a**	**s**	**s**		**s**		**s**		**a**	**a**	a	**a**					
[DBSB91]			s	s			**s**	s	**s**			**g**	**a**								
[Fel99]			s		**g**	**s**	**g**	s	**g**			**s**	**a**								
[FNP92]			s	s	**s**	**s**	**s**	s	**s**			**g**	**a**	**a**	a	**a**					
[Gäß95, Stu95]			s	s	**g**	**g**	**g**	s	**g**			**g**	**a**	**a**	a	**a**					
[GS90]			**s**				**s**					**g**									
[Gl94]						n/a	**a**		**a**				**a**	**a**							
[Gom97]			s	g	g	g	g	s	g				a	a	a	a					g
[GF97]			**as**	s	**ag**	**s**	**g**	s	**ag**			**g**	**a**	**a**	a	**a**					
[HKV92]													**a**								
[HK97]				s			**g**	s	**g**			**g**	**a**								
[IK96]			s	s			**g**	s	**g**			**g**	**a**				**g**				
[KKL+98]			s				**s**		**g**	**g**		**g**	s								
[KV95]				n/a			**g**	n/a	**s**				**a**	**a**	**a**				**g**	**a**	
[JDFM97, MMM94]							**s**		**s**				**a**	**a**	a	**a**					
[MB94]			**a**		**a**	n/a	n/a		**s**			**g**	**a**	**a**	a	**a**					
[OB92]			s	s			**g**	s				**g**									
[Ost92]					**s**	**s**	**s**		**s**	**s**		**g**	**a**	**a**	a	**a**					
[PA97]							**s**		**s**				**a**								
[PDF87]			s	s			**g**	s	**g**				**a**	**g**	a		**g**				
[BE95, Rei94]					**s**	**g**	**s**		**g**				**a**	**a**							
[SWT89]			s				**s**		**s**			**g**	**a**								
[Tau93]			s	s	**g**	**s**	**g**	s	**g**				**a**	**a**	a	**a**					
[vDD+88]			**s**	s			**g**	s	**g**				**a**								
methodology of this dissertation (DISER)	s	**s**	**s**	**g**	**g**	**g**	**g**	**s**	**g**	**s**	**s**	**g**	**a**	**a**	a	**a**		g	g	a	s

The table shows:

- None of the implementations support the analysis of the technical infrastructure or the application feedback. [Gom97] state the intention by listing some exemplary indicators.
- None of the systems support the task »forget«. Thus, they are not able to deal satisfactorily with outdated artifacts.
- None of the implementations support the task »publish«. Therefore, information is publicly available as soon as it is stored in the experience base – regardless whether it has already been qualified or not.
- All systems that offer browsing functionality, also guide the user.
- Only a few systems support the task »structure«.
- Only a few systems support the task »characterize initially«, mainly because they are not able to deal with incomplete and uncertain characterization information. If the requirements R22 (tolerance of imprecise information) and R24 (tolerance of inconsistent information) would have been among the support criteria for this task, none of the systems would have supported the task.
- Only one system supports the tasks »explain selection« and »generate«.

5.6 Summary

This chapter has surveyed existing approaches for representing the contents of repositories. Existing approaches use techniques from the research areas of library and information science, hypertext, database management systems, and knowledge-based systems. Figure 27 gives an overview.

5.6.1 Chronology of Approaches

If a software engineering repository is viewed as a digital library, techniques from library and information science can be exploited. While enumerated classification, which typically arranges the artifacts into a hierarchy and can be implemented using a simple directory structure, is an easily comprehensible organization principle, its classification schemas tend to grow very large – even for moderately sized repositories. Besides the maintenance problem of such a monolithic schema, it is also often difficult to find a consensus for the classification schema among all users in the first place. Consequently, it becomes difficult for the user to retrieve adequate artifacts. A landmark for alleviating these problems has been set by Prieto-Díaz and Freeman [PDF87, PD91]. They employ the organization principle of faceted classification, which allows to characterize artifacts using a predefined set of facets. The value range of each facet is defined by an enumerated classification schema. Facets can be used to represent different views of the characterization. At the same time, the total number of descriptors needed for artifact characterizations is reduced. However, the effort for characterizing an artifact increases, because an artifact is no longer character-

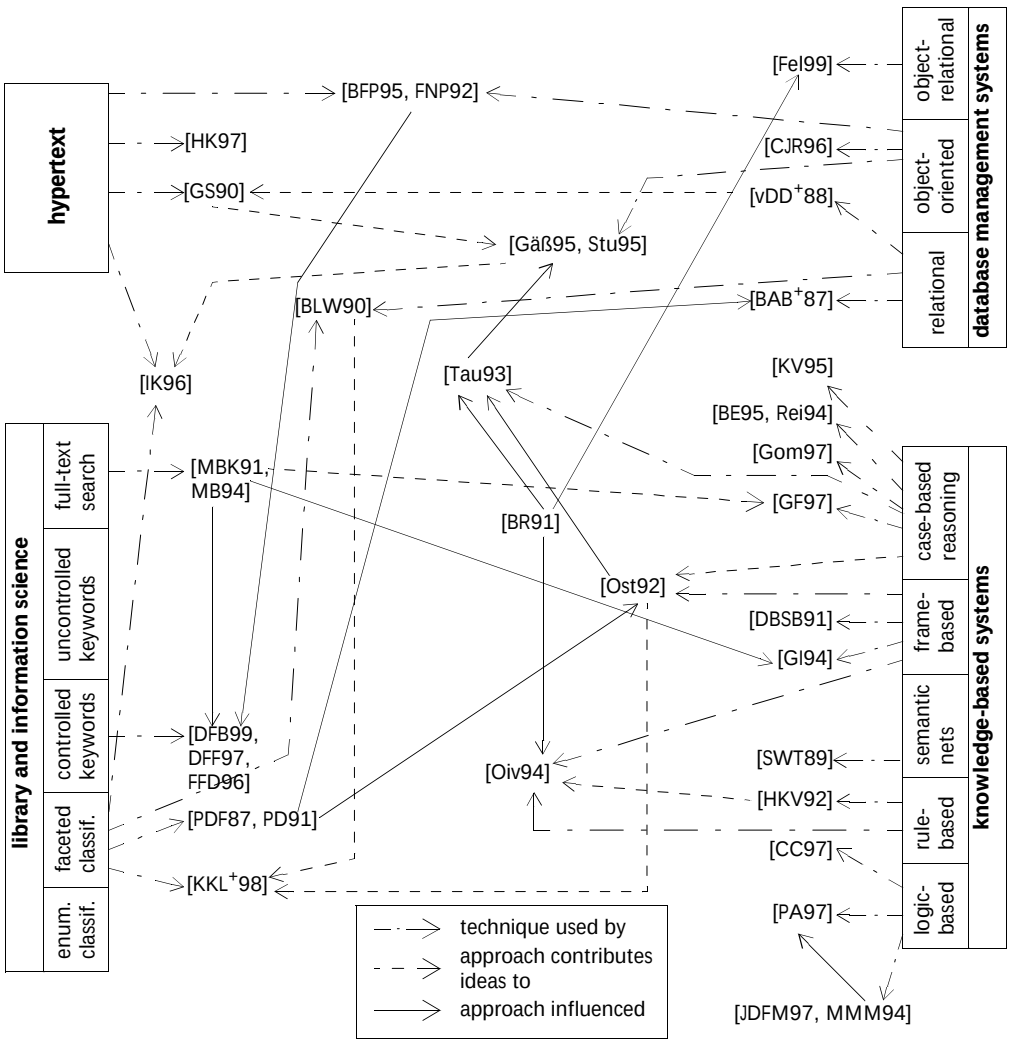


Figure 27: Existing approaches surveyed in this chapter

ized by placing it in a single class, but by placing it in one class for each facet. Maarek et al. [MBK91, MB94] automated the characterization process by using information retrieval techniques. However, later it was recognized that automatically characterizing artifacts results in low quality characterizations, partially because not all information needed for reusing an artifact is explicitly documented in the artifact. Therefore, semiautomatic approaches have been proposed (e.g., [DFB99, DFF97, FFD96]).

Techniques from library and information science consider each artifact to be independent of the other stored artifacts. Thus, the techniques do not allow the documentation/exploitation of relationships between artifacts or between reusable parts of the artifacts. The maintenance and exploitation of relationships is

one of the strengths of database management systems (DBMS). For software reuse, relational DBMS have been used to extend the capabilities of »traditional« library search systems. For instance, Browne et al. [BLW90] implement the faceted classification technique using a relational DBMS. In contrast to [PDF87], their system is not restricted to hierarchical classification schemas for the facets. More general arrangements, such as graphs, are also supported. Although not explicitly mentioned, it is possible to support several kinds of artifacts (each kind having its own faceted classification schema) as well as relationships between the artifacts due to the use of relational schemas. Burton et al. [BAB+87] implemented a score component which guides the user by recommending artifacts based on an importance rating of the user.

A somewhat different approach is reuse by standardization. Templates are defined which are used as guidelines by the projects. Typically, there are templates for all kinds of work products. The templates specify not only what information must be supplied, but also how the artifacts are related. This approach was first described by van Lamsweerde et al. [vDD+88]. Later, Garg and Scacci [GS90] presented an approach that is similar in spirit. However, they used a hypertext system instead of a relational DBMS to implement the idea.

In contrast to DBMS, hypertext systems allow the documentation of any kind of information by using text and graphics. Relations between information units are documented through links which allow the user to navigate from one information unit to another. Thus, links can be exploited to find information (e.g., as done by the quality pattern approach [HK97]). However, the informal representation of the contents makes it difficult to maintain hypertext systems. Hence, a more rigorous definition of hypertext systems is needed. For object-oriented development, Fugini et al. [BFP95, FNP92] provide a solution by using an object-oriented organization of the artifacts. A hypertext user interface allows the browsing of the collection, whereas a keyword technique from library and information science [DFF97, FFD96] allows to search for artifacts *across* all information units – something that is not possible in pure hypertext systems. Later, the limitations of relational DBMS for software repositories (e.g., only 1:1 and 1:n relationships) were also realized by Chee et al. [CJR96] who built a repository for project information. However, the weak query ability of object-oriented DBMS led recently to the concept of object-relational DBMS. [Fel99] used this technology for organizing an experience base.

Although DBMS are good for ensuring the consistency among artifacts, »traditional« DBMS do not behave in an intelligent way. This means that extensions for finding similar artifacts and inferring characterization information from existing information must be programmed manually. In addition, major extensions of DBMS may result in inefficient storage and retrieval systems.

Knowledge-based systems are specialized to produce such intelligent behavior. At first, knowledge-based systems for software reuse were developed indepen-

dently of other existing approaches, that is, they picked up the ideas from existing and well-established knowledge representation formalism in the field of artificial intelligence. For instance, Solderitsch et al. [SWT89] used semantic nets to organize software components (and related information) and Devanbu et al. [DBSB91] used a frame-based representation for organizing information about a large switching system. Hendrick et al. [HKV92] employed a rule-based representation for guiding project managers. The implementation described in [HKV92] utilizes information stored in existing DBMS to derive conclusions using the rules – thus building a bridge between information handled by »traditional« DBMS and by the knowledge-based system.

However, the effort for acquiring the stored knowledge in all three cases was high. Maintenance proved to be quite effort-intensive as well. Hence, later approaches tend to store only information that can be directly associated with a particular artifact as exercised by all information science, hypertext, and DBMS approaches. Information about a collection of artifacts has to be avoided (cf. also [Men99] on this issue). For example, rules, that are derived from data across many projects (such as those of [HKV92]), must be checked and updated – at least theoretically – after the completion of each project. Oivo [Oiv94] built a system using a knowledge representation based on frame-based and rule-based representations. In contrast to [DBSB91], frames stored only information that can be traced back to a single artifact. Rules were valid independent of the stored artifacts, because they did not document facts about the collection (as in [HKV92]), but defined dependencies among characteristics in general (e.g., a rule may document that the value for »lines of code« used in the efficiency model for coding is the same as the value for »lines of code« recorded as part of the software system‘s characterization).

More recently, logic-based representations and case-based reasoning technology have been exploited for software reuse. Logic-based representations (e.g., [CC97, JDFM97, MMM94, PA97]) retrieve artifacts using formal specifications of artifacts. They are suited for artifacts that are (or must be) specified formally. Because most of the artifacts in an experience base are not of this kind, these approaches cannot be easily generalized to all kinds of artifacts. A technology that supports the characterization of artifacts in a manner similar to DBMS is case-based reasoning. Case-based reasoning [AP94, Alt97, BSW96, Ric98] is a generalization of faceted classification because it not only supports facets with enumerated classification schemas, but also quantitative (e.g., integer, real, and dates) and (informal) textual facets. It extends DBMS by supporting directly various arrangements of terms (e.g., sorted lists, term hierarchies and lattices as known from library and information science) for one facet. It complements both faceted classification and DBMS by supporting similarity-based retrieval. Finally, its philosophy is to store knowledge in the form it becomes available in practice (for the experience base, this means as artifact characterizations). Thus, it reduces considerably the effort for knowledge acquisition and maintenance. Exemplary approaches using case-based reasoning include [BE95, Rei94] (object-

oriented code) and [KV95] (abstract data types). Gómez [Gom97] discovered that mechanisms for forgetting obsolete experience based on frequency of use can be easily integrated. Gonzáles and Fernández [GF97] complement the case-based reasoning approach by a full-text search capability. However, the user has to decide whether he wants to use the faceted approach (= case-based reasoning) or natural language (= full-text search) to retrieve some component.

In parallel to these recent advances, hybrid approaches using techniques from knowledge-based systems began to appear. For instance, Girardi and Ibrahim [GI94] used a frame-based representation to extract descriptors from artifacts automatically combining information retrieval techniques with knowledge-based techniques. Although not explicitly mentioned, Kang et al. [KKL+98] organized their information for reusing domain-specific reference architectures by employing facets and rule-based dependencies in form of assertions. Already a few years earlier, a more general approach combining faceted classification (characterization through attribute value pairs), frame-based representation (characterizations with the same attributes are in the same class), and assertions was described by Ostertag [Ost92]. Ostertag's Extensible Description Formalism also allows the definition of distances (antonym to similarity) between artifact characterizations and queries. A shortcoming of the system is the fact that the user is not guided when querying (or entering) a new characterization. This bears the danger that the user specifies values for attributes that are completely irrelevant for the query because none of the existing characterizations might contain a value for the specified attributes. Hence, the author of this dissertation used the idea of distance computation for storing and retrieving software process models, but integrated it with the user guidance of faceted classification [Tau93]. Later, Gäßner and Stummhöfer [Gäß95, Stu95] generalized this approach for all kinds of artifacts. They also introduced the notion of consistency rules for characterizations referencing each other as known from relational DBMS. The system combines the described query functionality with the navigation functionality of hypertext systems to further explore and understand the context of the artifacts recommended by the system. A system similar in spirit was developed simultaneously by Isakowitz and Kauffman [IK96].

5.6.2 Conclusion

Many ways of organizing reuse information have been tried. One of the hard lessons learned is the fact that one should be very careful in storing abstract knowledge as it has been done traditionally in knowledge-based systems [Men99]. These systems are not maintainable with a feasible amount of effort if experience is to be added on a continuous, incremental basis.

The comparably large number of hybrid systems show that no well-established technique for storing and retrieving information can be readily used. Frakes and Pole [FP94] show that there is no silver bullet. Appropriate approaches depend

on what artifacts are to be characterized, which tasks and operations are to be supported, and on the expected usage frequency, but also on user preferences. Frakes and Pole suggest to offer alternative ways of retrieving.

Although many approaches exist, the evaluation section demonstrates that none of the techniques fulfill all of the requirements listed in Chapter 3. In particular, no system is able to cope with imprecise and inconsistent information. None of the approaches document the rationale for explicitly stored conceptual knowledge in a satisfactory manner, that is, in a way that the conceptual model can be used as a communication vehicle among people working for the experience factory. Also, none of the systems support application feedback and analysis of the technical infrastructure as required in Chapter 4. These shortcomings also affect the maintainability of the experience base. For example, without user feedback, the technical infrastructure cannot be adjusted to the users' needs.

Overall, the state-of-the-art regarding the technical infrastructure of an experience base can be characterized by classifying existing approaches into those that support:

- Only one or a small number of predefined kinds of artifacts (mostly code artifacts)
- All kinds of artifacts, but which restrict the knowledge dimensions of characterizations (to precise, consistent information) unnecessarily and have deficiencies in the maintenance of the experience base

The next chapter presents a representation formalism that solves the identified problems.

6 Knowledge Representation for Software Engineering Experience

Make it as simple as possible, but not simpler.

Albert Einstein

The explicit, rationalized, and modular representation of the conceptual information of an experience base (= schema) is essential for transferring and tailoring experience bases to organization-specific needs (see requirements R8 (explicit representation of the conceptual knowledge) on page 45, R10 (rationalized conceptual information), and R11 (modularity of conceptual information)). Representation formalisms describe how to specify conceptual information. Such representation formalisms need to fulfill the technical requirements stated in Chapter 3. In addition, the schemas described using the constructs of the representation formalisms must also be easily comprehensible so the schemas can be used as a communication vehicle among the people responsible for structuring an experience base.

The survey of existing approaches in the previous chapter has shown that many approaches for representing software engineering experience exist, but none of them alone is sufficient for implementing a software engineering experience base. Therefore, the approaches need to be integrated and complemented to meet the requirements elicited in Chapter 3.

This chapter briefly describes REFSENO (representation formalism for software engineering ontologies[1]), a representation formalism that was developed to meet the objectives outlined above. The detailed definition of REFSENO can be found in [TG98a] and consists of a notation (Section 6.1 gives an overview) and semantics (Section 6.2 gives an overview) part. Benefits and limitations of REFSENO are described in Section 6.3, whereas Section 6.4 lists the requirements from Chapter 3 that REFSENO fulfills. Finally, Section 6.5 summarizes the main points.

1 An *ontology* is an explicit representation of a conceptualization [Gru95].

6.1 Notation

This section presents an overview of REFSENO´s constructs [TG99]. The constructs are used to specify the schema of an experience base. The complete definition can be found in [TG98a].

In the following, the usage of the constructs is illustrated with excerpts of a schema for lessons learned (see also Chapter 10). Lessons learned are informal, qualitative experience items that allow to record any problem solution statement [GvWB98], observation, guideline, or improvement suggestion [BT98a]. Their goal is to promote the repetition of good practices and to avoid making the same mistake twice. Lessons learned can be captured with a comparably low amount of effort, because their contents do not need to be formalized. As such, lessons learned complement more formal data collections, for example, those employing the goal/question/metric (GQM) technique [vSB99]. If the GQM technique is used, interpretations of data (as supplied during feedback sessions) can be recorded together with a graphical representation of the data as lessons learned.

6.1.1 Concepts

A concept is a construct modeling software engineering artifacts such as documents, guidelines, observations, or improvement suggestions (cf. Definition 7 on page 26). In addition, abstract concepts can be introduced representing information shared by various concepts to avoid redundancies in the schema. For example, an observation and a guideline may share a common context for which they are valid.

A concept is specified by a 9-tuple (*name*, *extension*, *intension*, sim_{artif}, $sim_{I/F}$, sim_{ctxt}, *assertion*, *description*, *scenarios*) as described in Table 8. Concepts are represented using tables. Table 9 shows an excerpt of a concept glossary. The intension, similarities, and assertion of a concept are represented using a concept attribute table (see Section 6.1.2 and Section 6.1.4) while the objects of the concept's extension are represented using instance tables (see Section 6.1.6).

Table 8: Components of a concept specification

Component	Description
name	unique name used for reference purposes
extension	set of all instances belonging to the concept

Component	Description
intension	the set of all attributes an instance must exhibit which belongs to the concept (i.e., all instances of the *extension* are characterized using the same attributes). There are two kinds of concept attributes: Terminal (Section 6.1.2) and nonterminal concept attributes (Section 6.1.4). Furthermore, each attribute belongs to one of three layers: Artifact (characterizes artifact itself), interface (characterizes how a particular instance can be incorporated into the surrounding system), and context (characterizes the environment in which the instance has been applied or can be applied). The context layer also contains attributes describing the quality of the instance in the specified environment [BR91] (cf. Figure 7 on page 68).
sim_{artif}, $sim_{I/F}$, sim_{ctxt}	similarity functions [Alt97] associated with the concept. They compute the similarity between two instances of the *extension* based on the *intension* of the concept returning a real number between 0 (denoting total dissimilarity) and 1 (denoting total similarity, i.e., equivalence). Their values are combined to a single similarity value using a weighted sum formula.
assertion	condition that all instances of the extension must fulfill at all times (should include an annotation, i.e., a natural language explanation)
description	narrative text defining the software engineering artifact
scenarios	set of 3-tuples where each tuple describes a retrieval goal and consists of a name, a purpose, and a set of intended users. Scenarios are the rationale for the existence of the concept, that is, they describe for what purposes the instances of the concept are needed by whom. In case of abstract concepts, the set of scenarios is empty.

Table 9: Excerpt of a concept glossary

Name	Description	Scenario name	Purpose	Intended user(s)
Artifact	Any printable matter that can be versioned	FindArtifact	finding a relevant artifact	Everybody
Guideline	A suggestion/recommendation for executing a project that (supposably) avoids a problem that occurred in some earlier project (context-dependent)	GetRelevantProjectInfo	informing about recommendations for the project at hand	Project Manager
Improvement Suggestion	A suggestion how to improve a particular *Artifact*; an *Improvement Suggestion* is the result of a problem analysis	FindImprovementSuggestions	finding out how to improve a given artifact	Authors, Artifact owner
IQ Process	Description of a recurring task including general guidelines on how to perform it as well as links to knowledge and experience relevant for this task	FindIQProcess	finding a relevant IQ Process	Everybody
Observation	Something an employee has observed or experienced during a project that is not captured as a guideline or problem/solution statement	GetRelevantProjectInfo	finding interesting comments and notes for the project at hand	Project Manager

Name	Description	Scenario name	Purpose	Intended user(s)
Problem	A negative situation that occurred during a project execution	IdentifyProjectRisks	identifying project risks	Project Manager
		FindSolution	finding a solution for an occurred problem	Project Manager, Project Member
		GetImprovementProgress	informing about the progress regarding the development of a solution for a problem	problem reporter
		FindOpenProblems	finding problems that have not been solved or rejected	improvement squad, creativity team
Project	Something for which Fraunhofer IESE has given a commitment and gets money for	GetProjectDocuments	finding project-related documents and information	Project Managers, Project Members
		FindProjExperience	finding the experience that was gained during a given project	Experience factory team
Solution	A pragmatic way how a *Problem* was solved including its evaluation		modeling	

6.1.2 Terminal Concept Attributes

The intension of a concept is a set of terminal and nonterminal concept attributes (Section 6.1.4). Terminal concept attributes model how software engineering artifacts are specified for storage and retrieval. A terminal concept attribute is specified using a 10-tuple (*name*, *description*, *cardinality*, *type*, *default value*, *mandatory*, *value inference*, *inferred attributes*, *standard weight*, *scenarios*) as described in Table 10.

Table 10: Components of a terminal concept attribute specification

Component	Description
name	unique name used for reference purposes.
description	narrative text defining the meaning of the attribute.
cardinality	range specifying the minimal and maximal number of values the attribute may have. If the cardinality is unequal to 1, the attribute values are specified as a set.
type	Each terminal concept attribute is typed. The type is a construct described in Section 6.1.3.
default value	concerning the insertion of new instances (Section 6.1.6), the default value is used if no specific value for this attribute is specified.
mandatory	specifies whether an attribute value of an instance has to be specified when inserted, i.e., the attribute value may only be undefined if the attribute is not mandatory.
value inference	defines how to calculate the attribute value automatically (if possible) based on other attributes' values (should include an annotation, i.e., a natural language explanation).
inferred attributes	lists all attributes whose value is inferred using a value of this attribute. (There is a mutual dependency between value inferences and inferred attributes. For every attribute used in a value inference, the inferred attribute values of the used attribute must include the attribute whose value is inferred.)

standard weight	may be used by the similarity function of the concept this attribute belongs to. It is a non-negative real number. (For the standard similarity function of concepts, this weight defines the importance of this attribute regarding the global similarity.)
scenarios	set of scenario names. Scenarios are the rationale for the existence of the terminal concept attribute, that is, they describe for what purposes the attribute values are needed by whom. The purpose and viewpoint of the corresponding scenarios can be found in the concept glossary (see Section 6.1.1).

A terminal concept attribute is represented using the concept attribute table which is concept-specific. Table 11 shows as an example terminal concept attributes of a project characterization. The layer column refers to the layer of the intension (see Section 6.1.1).

Concept: Project
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type/ Kind	Default value	Mandatory	Value inference	To infer	Standard weight	Scenario names
artif	proj type	classifies project into equivalence classes	1..10	ProjType	-	yes	-	-	5	IdentifyProjectRisks, FindSolution
	proj funding	source of payment	1	ProjFunding	-	yes	-	-	1	IdentifyProjectRisks, FindSolution
	proj manager	the person managing a project	1	iIESE Employee	-	yes	-	-	1	IdentifyProjectRisks, GetRelevantProjectInfo, FindSolution
	scientific lead	the person responsible for assuring that (a) state-of-the-art is applied in the project and (b) research opportunities provided by the project are exploited	1	iIESE Employee	-	yes	-	-	1	IdentifyProjectRisks, GetRelevantProjectInfo, FindSolution
	proj members	everybody working on the project (project staff)	0..*	iIESE Employee	{}	yes	-	IESE team size	1	IdentifyProjectRisks, GetRelevantProjectInfo, FindSolution
	IESE team size	the number of people on the side of Fraunhofer IESE	1	TeamSize	-	yes	card([proj members])+1	proj team size	1	IdentifyProjectRisks, GetRelevantProjectInfo, FindSolution
	customer team size	team size on customer's side	1	TeamSize	-	yes	-	proj team size	1	IdentifyProjectRisks, FindSolution
	proj team size	total team size	1	TeamSize	-	yes	[IESE team size] + [customer team size]	-	1	IdentifyProjectRisks, GetRelevantProjectInfo, FindSolution
I/F										

Concept: Project
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type/ Kind	Default value	Mandatory	Value inference	To infer	Standard weight	Scenario names
ctxt	customer	the one who pays for the project	1	Customer	-	yes	-	-	1	IdentifyProjectRisks, GetRelevantProjectInfo, FindSolution

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

assertion: TRUE

Table 11 Example of a concept attribute table with terminal concept attributes for a project characterization (excerpt)

6.1.3 Types of Terminal Concept Attributes

All terminal concept attributes are typed. Types model qualities of software engineering artifacts such as lines of code and efficiency, or they are used to categorize instances of a concept, e.g., a type may specify possible programming languages used in a software project.

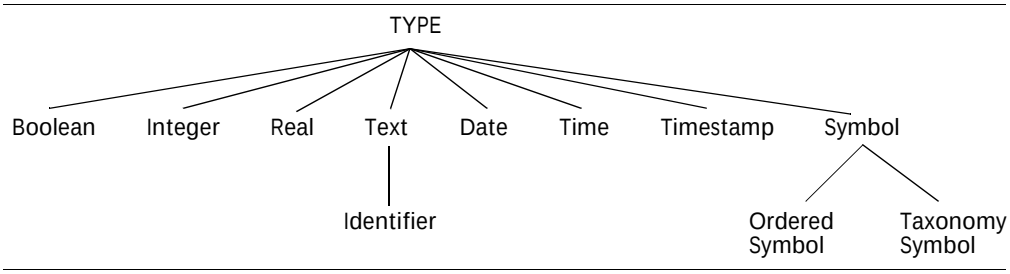
A type is specified using a 5-tuple (*name*, *supertype*, *value range*, *unit*, *sim*) as described in Table 12. Types may be derived from a set of predefined basic types shown in Figure 28. In addition, interval types may be constructed from ordinal types.

Table 12: Components of a type specification

Component	Description
name	unique name used for reference purposes
supertype	specifies the supertype this type is derived from. Types differ from their supertype in one or more of the following ways: value range, unit, sim.
value range	specifies the possible values for all attributes of this type. In addition to the values specified the following special values are allowed: »undefined«, »unknown«, »n/a« (not applicable). »Undefined« means that the value is currently not specified, but will be specified later; »unknown« means that the value is not known and will not be specified later; »n/a« means that the attribute is not applicable for the instance (e.g., an evaluation is not applicable for a general guideline – see example in Table 22 on page 187).
unit	specifies the unit for numerical types. For non-numerical types this component is not applicable.
sim	similarity function associated with the type. It computes the similarity between two possible values of this type. In contrast to the concept's similarity function, this similarity function is of a local nature. For each predefined type of REFSENO, there exists a standard similarity function [TA98].

The comprehensive type system ensures the practical applicability of REFSENO. For example, the symbol types[1] accommodate classification techniques from library and information science (see Section 5.1). For this purpose, the taxonomy

Figure 28: Predefined basic types of REFSENO



symbol type can be used for the enumerated classification technique, the ordered symbol type corresponds to a linear list, and the general symbol type allows the definition of arbitrary relationships among the symbols. Other examples include the text type (which allows the capturing of informal, textual information) and the interval type (which allows the capturing of imprecise information [TA00b]).

Types are represented using a type table as shown in Table 13. For example, the type »ProjType« is defined as a TaxonomySymbol with the possible values »Consulting«, »Seminar«, etc. For symbol types a definition of the possible values is provided in an extra table (Table 15). This is important as it allows to define explicitly the meaning of symbolic values of the attributes corporate-wide.

Table 13: Examples of types of terminal concept attributes

Name	**Supertype**	**Value range**	**Unit of measure**	**Similarity**
Duration	Cardinal	1..3000	days	standard
iIESE Employee	UnorderedSymbol	{..., Carsten Tautz, ...}	n/a	standard
Problem-Status	Symbol	{new, accepted, in progress, mutated, solved, rejected}	n/a	see Table 14
ProjFunding	TaxonomySymbol	see Figure 30	n/a	standard
TeamSize	Cardinal	1..50	n/a	standard

1 The term »symbol« is a synonym for the term »term« used in Section 5.1.

Table 14: Similarity between problem statuses (based on the possible state transitions shown in Figure 29)

	new	**accepted**	**in progress**	**mutated**	**solved**	**rejected**
new	1	0.67	0.33	0	0	0.67
accepted	0.67	1	0.67	0.33	0.67	0.67
in progress	0.33	0.67	1	0.67	0.67	0.67
mutated	0	0.33	0.67	1	0	0
solved	0	0.67	0.67	0	1	0
rejected	0.67	0.67	0.67	0	0	1

Figure 29: Possible state transitions between problem statuses

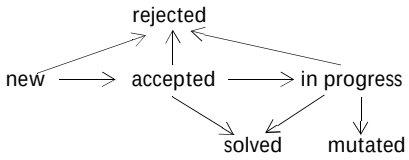


Figure 30: Taxonomy for project funding

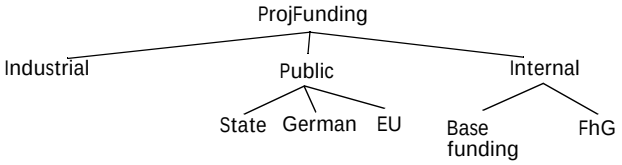


Table 15: Example of a symbol glossary

Type	**Symbol**	**Description**
IlESE Employee	Each symbol corresponds to the name of a person	
Problem-Status	accepted	A problem has been accepted as such; either the improvement squad itself or an appointed creativity team will start working on a solution soon
	mutated	A more fundamental problem is underlying this problem; if the fundamental problem is solved, this problem will be solved as well; therefore, the search for a solution for this problem is abandoned
	new	A new problem has been stored in the problem memory; the improvement squad will decide soon whether to accept or reject the problem
	rejected	A problem was not solved either because its underlying cause was a misunderstanding or because its solution would not yield a positive cost/benefit ratio
	in progress	A creativity team has started to work on the problem
	solved	A solution has been devised and implemented
ProjFunding	Industrial	Projects paid by industry
	Internal	Projects paid by FhG funds
	- Base funding	Projects funded through IESE base funds
	- FhG	Projects funded through ZV Munich (SEF, OEF)
	Public	Projects funded publicly
	- EU	Projects funded by some framework program of the EU
	- German	Projects funded by ministry of German government
	- State	Projects funded by ministry of Rhineland-Palatinate

6.1.4 Nonterminal Concept Attributes

Nonterminal concept attributes as part of the concept's intension associate the concept with other concepts, that is, they define semantic relationships between concepts. For example, a project utilizes guidelines. In contrast to terminal concept attributes, the values of nonterminal concept attributes do not describe characteristics of an instance directly but indirectly through references to related instances.

A nonterminal concept attribute is specified using a 12-tuple (*name*, *kind*, *destination concept*, *reverse attribute*, *description*, *cardinality*, *default value*, *mandatory*, *value inference*, *inferred attributes*, *standard weight*, *scenarios*). Table 16 describes the components name, kind, destination concept, and reverse attribute, whereas the description for the other components can be found in Table 10 on page 178.

Table 16: Components »name«, »kind«, »destination concept«, and »reverse attribute« of a nonterminal concept attribute specification

Component	Description
name	unique name used for reference purposes.
kind	each nonterminal concept attribute is of a particular kind (Section 6.1.5), e.g., »has-part« or »defines«.
destination concept	specifies the concept associated with the concept, the nonterminal concept attribute belongs to. The value of a nonterminal concept attribute is a set of instances of the destination concept.
reverse attribute	each nonterminal concept attribute has a reverse nonterminal concept attribute, i.e., all relationships are bidirectional

Table 17 shows as an example the nonterminal concept attributes of a project characterization. Alternatively to the tabular representation, a graphical representation can be used to give an overview of the relationships between the concepts as shown in Figure 31.

Concept: Project
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type/ Kind	Default value	Mandatory	Value inference	To infer	Standard weight	Scenario name
artif										
I/F										
ctxt	guidelines	guidelines applied in this project	0..*	uses [Guidelines].[evaluation]	{}	yes	-	-	1	FindProjExperience
	observations	observations made during the project	0..*	defines [Observation].[projects]	{}	yes	-	-	1	FindProjExperience

Concept: Project
Super concept: CONCEPT

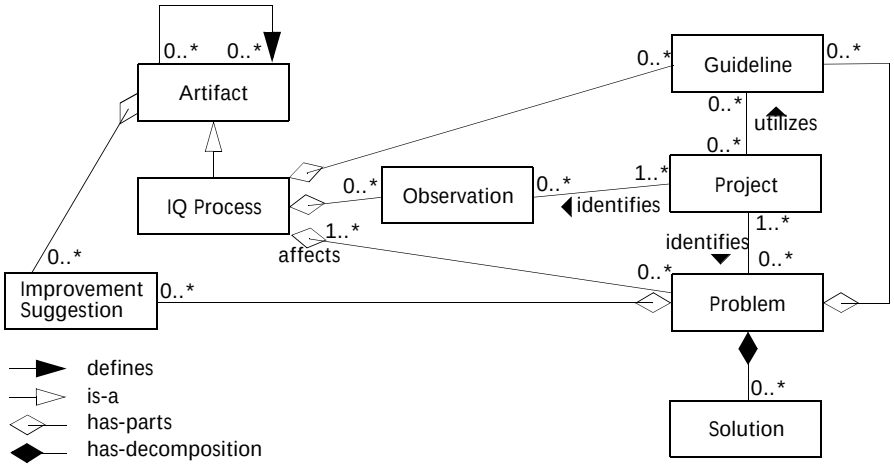
Layer	Name	Description	Cardinality	Type/ Kind	Default value	Mandatory	Value inference	To infer	Standard weight	Scenario name
ctxt	problems	problems occurred during the project	0..*	defines [Problem].[projects]	{}	yes	-	-	1	FindProjExperience

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

assertion: TRUE

Table 17 Example for a concept attribute table with nonterminal attributes for a project characterization (excerpt)

Figure 31: Graphical representation of nonterminal concept attributes



6.1.5 Kinds of Nonterminal Concept Attributes

Each nonterminal concept attribute is of a particular kind. Kinds of nonterminal concept attributes are analog to types of terminal concept attributes. They model the semantic relationships between software engineering artifacts. A kind is specified using a 5-tuple (*name*, *reverse name*, *purpose*, *structure*, *properties*) as described in Table 18.

Table 18: Components of a nonterminal concept attribute specification

Component	Description
name	unique name used for reference purposes
reverse name	As stated in Section 6.1.4, nonterminal concept attributes come pairwise. This is illustrated by Figure 31. The relationship can be read in one direction (the direction shown by the arrow), or in the other direction using the reverse name.
purpose	reason for defining a kind of nonterminal concept attributes by describing how it can be used effectively.
structure	Instances (Section 6.1.6) are related through kinds of nonterminal concept attributes resulting in a structure of the instances. This structure can be characterized as a set of trees, DAGs (directed acyclic graphs), or graphs.
properties	Besides the structure property, additional properties may hold true, e.g., symmetry and transitivity.

For example, in the context of recording lessons learned, a specific organizational interdependency between a project and problems/observations has been identified, the »identifies« relation, as shown in Table 19 and Figure 31. This relation describes the fact that a problem is identified in the course of a project. Likewise, observations are made during a project execution. The explicit modeling of this interdependency guarantees the traceability from a problem or observation to the context they originate from. In contrast to these documented »outputs« of a project or organizational unit, the »utilizes« relationship documents the »inputs« of a project or organizational units (e.g., guidelines that were employed in a project).

Table 19: Example of a kind table

Kind	Reverse name	Description	Structure	Properties
identifies	identified-by	documents what new insights have been gained by a project or organizational unit	DAG	the source of a »identifies« relationship must be either a project or an organizational unit (i.e., a »doer«)
utilizes	utilized-by	points to utilized artifacts; the reverse direction documents the validity of the artifact	DAG	the source of a »utilizes« relationship must be either a project or an organizational unit (i.e., a »doer«)

In addition to the kinds with user-defined semantics, REFSENO provides a set of predefined kinds (see Table 20), which is a superset of the kinds defined in the Unified Modeling Language [Rat97].

Table 20: Predefined kinds of REFSENO

Kind	Reverse name	Description	Structure	Properties
is-a	has-specialization	Denotes a specialization of a concept.	tree (single inheritance)	Transitivity Every concept listed in the concept glossary is a specialization of exactly one concept (single-inheritance). There is one predefined concept »CONCEPT« which is the most general concept. It has only one terminal concept attribute with the name »Id« of the type »Identifier«. *Id* corresponds to the name of the instance. Let c_1 and c_2 be concepts. If c_2 is-a c_1 then the following properties (inheritance) hold: • The intension of c_1 is a subset of the intension of c_2 • The extension of c_2 is a subset of the extension of c_1 • If the assertion of c_1 is true, then the assertion of c_2 must also be true
instance-of	has-instances	Denotes a special is-a relation. An instance is an element of the extension of a concept.	tree with no intermediary nodes	Let *i* be an instance of the concept *c*. Then the following properties hold: • The intension of *i* and *c* are the same • *i* is an element of the extension of *c*
has-parts	part-of	Denotes a decomposition. Subparts may be shared among concepts.	DAG	Transitivity
has-decomposition	decomposition-of	Denotes a decomposition where the subparts exist only if the surrounding part (aggregate) exists.	tree	Transitivity A concept may have at most one nonterminal concept attribute of the kind »decomposition-of«. The cardinality of a nonterminal concept attribute of the kind »decomposition-of« is always 1. If a concept has a nonterminal concept attribute of the kind »decomposition-of«, it may not have any nonterminal concept attributes of kind »part-of«.
defines	derived-from	Denotes a dependency between two artifacts that may be inconsistent (cf. requirement R24 (tolerance of inconsistent information) on page 55).	DAG	Transitivity The relationship has exactly one of the following statuses: • **Unconfirmed.** It is not (yet) known whether the two artifacts are consistent or not. • **Inconsistent source.** The target artifact is up-to-date, but the source artifact needs to be updated. • **Inconsistent target.** The source artifact has been changed, but the target artifact still needs to be updated. • **Consistent.** Source and target artifacts correspond to each other.

6.1.6 Instances of Concepts

The extension of a concept is a set of instances. Usually, only a few instances (if any) are part of the conceptual information because most instances are part of the characterization information (and are thus not considered as part of the schema). However, there is knowledge which is of importance to the modeled domain as a whole. For instance, general guidelines could be specified this way. Instances are specified using a 3-tuple (*name*, *concept*, *values*) as described in Table 21 and represented using an instance table. Table 22 gives an example.

Table 21: Components of an instance specification

Component	Description
name	unique name used for reference purposes.
concept	specifies from which concept the instance is instantiated from, i.e., the intension of the instance is defined by the intension of the specified concept.
values	attribute values of the instance. *Values* is a list of attribute/value pairs. There must be a pair for every attribute of the instance's intension.

Table 22: Example of a general guideline

Concept: Guideline; Instance: Speedy	
Attribute	**Attribute value**
description	IESE employees should respond quickly to customers´ wishes and questions
justification	shows customers that Fraunhofer IESE cares for them
originator	»unknown«
date	10-26-1999
evaluation	»n/a«
…	…

6.1.7 Formulas

Formulas are used for similarity functions, assertions, and value inferences. As such they model:

- The similarity between software engineering artifacts
- The similarity between software engineering quality values
- Dependencies between software engineering quality values

They are specified using constants, variables, operators, and functions [TA98]. Simple examples for dependencies are shown in the »value inference« column of Table 11 on page 180.

6.2 Semantics

The retrieval and maintenance of characterization information (cf. Table 1 on page 45) can be guided if the conceptual information of the experience base is described explicitly. For this purpose, the conceptual information must be formal so it can be interpreted by a computer system. The notation of REFSENO out-

lined in the previous section was conceived with this objective in mind. Technically, characterization information is represented as instances of concepts.

The operations presented in this section must be viewed as »elementary« with respect to the performance of »logical« operations which require domain knowledge not specified as part of the schema. For example (cf. Figure 31 on page 184), to remove a *Problem* with all information related to it (i.e., *Guidelines*, *Improvement Suggestions*, and *Solutions* where *Guidelines* and *Improvement Suggestions* are not referenced by other *Problems*), first the set of instances to be removed must be determined before the elementary operation »removal of characterization information« can be performed. The merging of two or more similar *Problems* to a single generalized *Problem* is another example of such a logical operation.

In the following, an overview is given of how characterization information can be retrieved (Section 6.2.1) and maintained (Section 6.2.2 through Section 6.2.4) based on REFSENO. A precise definition with examples can be found in [TG98a, Chapter 4].

6.2.1 Retrieval of Characterization Information

Characterization information is retrieved using a *query specification*. A query specification is a set of one *main instance* and optionally several related instances. The main instance is an instance of any concept that does not have an empty set of scenarios associated with it (cf. Table 8 on page 176). Related instances are specified using nonterminal concept attributes. In addition, filters can be specified for attribute values that must lie in a given set or range.

For example (cf. Figure 31 on page 184), a query can be specified to find open *Problems* (i.e., problems which have the status »accepted« or »in progress« (cf. Table 15 on page 182) – specified using a filter) that were identified in EU funded *Projects* (specified using the attribute *project funding*). In this example, the specified problem is the main instance and the specified project a related instance. The retrieval is similar to a join operation of traditional relational databases, however, in this case projects with similar funding (e.g., *German* or *State* funding) are also considered if there are not enough problem/project pairs matching exactly the query.

The retrieval is performed as described by the method of the task »retrieve (T3, page 89)« including the task »specify (T4, page 90)« and employing the task »search« as the method for »identify (T8, page 92)« (cf. Figure 14 on page 126).

All subtasks of »retrieve« are automated except for »collect descriptors (T5, page 91)«, »examine (T16, page 97), and »select (T19, page 98)«, which cannot

be automated by their very definition. For two subtasks of »retrieve« the behavior shall be specialized at this point, because the original method description in Chapter 4 gives room for various implementations. The subtask »interpret problem (T6, page 91)« is automated by checking the entered values for the query specification against the value range of the attributes, whereas the subtask »infer descriptors (T7, page 92)« infers values using the specified value inferences (cf. Table 10 on page 178).

6.2.2 Insertion of Characterization Information

Characterization information is inserted using a set of instances. The insertion operation:

- Extends the characterization information already stored in the experience base by the new instances
- Updates the references of related instances to the new instances (references are always bidirectional)
- Performs value inferences that are associated to attributes of inserted or affected instances[1]

The new instances must fulfill the following consistency rules:

1 **Mandatory attributes.** For every mandatory attribute of an instance's intension a value must be specified.

2 **Cardinality.** The number of values supplied for an attribute must be in the cardinality range as defined in the concept attribute table. If nonterminal concept attributes are specified, the cardinality range of the affected nonterminal concept attributes of the referenced instances must also be observed.

3 **Terminal concept attributes.** Values of terminal concept attributes must be within the value range of the corresponding type of the attribute. In addition the special values »unknown« and »n/a« (and »undefined« if the attribute is not mandatory) may be used.

4 **Nonterminal concept attributes.** After the operation, all elements of a nonterminal concept attribute's value (in case of cardinality > 1) or the value itself (in case of cardinality = 1) must refer to existing instances (i.e., dangling references are not allowed). In addition, the special values »unknown« and »n/a« (and »undefined« if the attribute is not mandatory) may be used.

1 Affected instances are all instances that are changed through value inferences or due to an update of their references to one or more of the new instances.

5 **Assertion.** After the operation, all applicable assertions must be true. Applicable assertions are the assertions of the concepts of all affected instances as well as all of the concepts' super concepts.

6.2.3 Removal of Characterization Information

The information to be removed is specified using a set of stored instances. The operation:
- Deletes the specified instances
- Updates the references of related instances (i.e., removes references to the deleted instances)
- Performs necessary value inferences

The deletion of instances may not violate the consistency rules stated in Section 6.2.2.

6.2.4 Change of Existing Characterization Information

The set of instances to be changed is specified by references to the old instances (stored in the experience base) and the set of new values for the instances. The operation:
- Updates the values of the specified instances
- Updates the references of related instances (i.e., removes/adds references according to the changes of nonterminal concept attribute values)
- Performs necessary value inferences

The change may not violate the consistency rules stated in Section 6.2.2.

6.3 Practical Application of REFSENO

The constructs of REFSENO draw from ideas from several areas such as database system mechanisms (e.g., relationships between concepts and implied consistency rules) [Che76, Dit97, GR95], case-based reasoning mechanisms (e.g., similarity-based retrieval with incomplete information) [Alt97], and knowledge-based mechanisms (e.g., inference rules) [Ric89]. In addition, REFSENO is object-centered similar to the meta modeling in UML [Rat97]. The integration of the approaches results in a formalism that unites many of the advantages of the individual approaches while avoiding most of their drawbacks.

6.3.1 Outstanding Characteristics of REFSENO

In the following, some of REFSENO´s outstanding characteristics from a practical point of view are discussed.

Natural representation

The object-centered representation of schemas allows natural modeling. Concepts defined in schemas correspond to real world objects.

Schemas as communication vehicles

Informal parts (e.g., descriptions of concepts and attributes, definition of symbols in symbol glossaries, annotations of value inferences and assertions) not only help in understanding the schema, but are an aid for documenting a corporate-wide vocabulary – a prerequisite to ensure that all users interpret the contents of the experience base in the same way.

The tabular representation of REFSENO can be comprehended better and faster by humans than a representation using formulas (as most knowledge-based representations do – cf. Section 5.4), because information that belongs together is represented physically close together. The tabular representation also guides the schema modeler regarding the kind of information that needs to be specified (through the table columns). Templates for schemas can be easily provided. This ensures precise and complete schema descriptions.

Moreover, the graphical notation of REFSENO for nonterminal concept attributes (see Figure 31 on page 184) is based on UML´s class diagrams. Because software engineers are often acquainted with UML nowadays, schemas can be easily understood by software engineers.

In combination, the tabular and graphical representation allows to deal with the complexity of schemas often encountered in practice [FT98].

Schemas as executable specifications

The formal parts of REFSENO allow tool support for retrieving and characterizing/maintaining characterization information. The semantics of REFSENO imply a system behavior that guides the user for these tasks by providing the relevant attributes (together with informal definitions of their meanings) for which values must be specified.

On the retrieval side, REFSENO´s nonterminal concept attributes allow navigation along stored references between instances. This unique combination of similarity-based retrieval (to find an adequate starting point) and navigation (to find further relevant information) not only aids in the decision which artifact to utilize, but also eases the understanding of retrieved artifacts. Hence, REFSENO also supports the task »understand (T24, page 101)«.

On the learning side, REFSENO´s consistency rules (e.g., value ranges, mandatory attributes, and assertions) reduce the effort for maintaining by relieving the user of having to check these manually. Automatic computation of attribute values

through value inferences and bidirectional references reduce the maintenance effort further. In this respect, REFSENO supports the maintenance explicitly – something that is essential for complex repositories, but which has been mostly neglected by existing approaches (cf. Chapter 5).

Modularity of schemas

Consistency rules can also be used to combine schemas. Schemas can be extended easily by other schemas using nonterminal concept attributes that express the semantic relationships between the concepts of the schemas. If schemas overlap, that is, if they contain concepts that describe the same real world artifacts, the concepts´ intensions can be merged by uniting their attributes. If the attributes are not independent of each other (e.g., if the same characteristic of the artifact is modeled in different ways), the dependency between the attributes can be expressed as either a value inference or an assertion.

Tailorability to organization-specific needs

The modularity, the natural representation, the easy comprehensibility, and the rationales of the schemas (documented in form of scenarios) make schemas tailorable to organization-specific needs with a comparably low transfer effort. The large variety of predefined types and kinds result in high flexibility and expressive power of schemas. For example, the predefined types allow to »imitate« the techniques from library and information science (see Section 6.1.3). Nonterminal concept attributes allow to »imitate« hypertext systems where the instances are the nodes and the references are the links.

Schema grows with organizational needs

Once an initial schema has been defined for an organization, it can grow on an as-needed basis. For example growing collections of artifacts can be managed by formalizing existing attributes (e.g., a text attribute is replaced by one or more attributes of numeric and/or symbol types) or by adding additional attributes that allow to distinguish the artifacts of the larger collection. Because the retrieval mechanism associated with REFSENO is able to deal with incomplete information, artifacts already stored in the experience base do not need to be re-characterized as long as »only« new attributes are added (the new attributes will be ignored for the old artifacts by the similarity computation, while they will be considered for all new artifacts). This feature allows artifacts to be characterized on various levels of abstraction.

Another type of growth often encountered is the inclusion of new artifact kinds. This can be easily managed by adding new concepts corresponding to the new kinds.

Consideration of peculiarities of software engineering experience

Besides these general characteristics for knowledge management, REFSENO has some characteristics that are tailored specifically toward the needs of managing software engineering experience. They include support for:

- Three-layered characterization (cf. Figure 7 on page 68)
- Retrieval of experience relevant in the current context (context-sensitive retrieval)

- Incompleteness, imprecision, inconsistency, and uncertainty of software engineering experience
- Quantitative attributes (e.g., to express quality dimensions of artifacts)

6.3.2 Trade-Offs and Limitations of REFSENO

According to Henninger, »the intuitive and widely held assumption is that up-front investments in structuring a repository result in a proportional increase in the ease with which components [artifacts] can be reused.« [Hen96, p. 280] In general, the investments in structuring are proportional to the formality of the schema and the modeling depth. The more effort is put in, the more formal and detailed the schema will be.

Both the detailed and less detailed schema have their advantages. Informal information (e.g., expressed by a text attribute) requires low observation/acquisition effort, but high analysis effort [Ste90]. In contrast, formal information allows (partly) automated analysis (e.g., in form of value inferences), but usually requires more observation/acquisition effort (although cases have been reported that acquisition effort was actually reduced by prescribing a more structured schema [KS96]).

With increasing detail and more technical artifacts, the distance between the project organization and the experience factory shrinks in terms of time intervals between two accesses. For example, if the experience base contains only general information about project planning and execution, the experience base will be consulted once or twice during a project. However, if the experience base also contains work products, it will be accessed quite frequently in the course of a project. Thus, there is a continuum from organizational information (focusing mainly on ideas and experiences) to technical information (focusing mainly on work products and experiences about them).

Although REFSENO has been designed to cover the whole spectrum, it should not be used to store information which can be represented more efficiently otherwise. For example, information needed for the creation of an artifact should not be stored as part of the experience base but as part of an »intelligent assistant« (an artifact-specific tool that allows the creation and editing of an artifact). Mylopoulos et al. identified typical characteristics of experience base systems and intelligent assistants (see Table 23) [MBY97].

Table 23: Differences between experience base systems and intelligent assistants based on [MBY97]

Criterion	Experience Base	Intelligent Assistant
Type of information captured	descriptive information about the artifact and the environment of the artifact	environment is less important; domain-independent heuristics
Coverage of software knowledge base	quite broad	narrow, specific to the task of the assistant

Criterion	Experience Base	Intelligent Assistant
Completeness of the software knowledge base	useful even if incomplete	sufficiently complete to support inferences required for performance of task
Criteria for success of information capture activity	• time saved in chasing for information • accuracy and completeness of information	how well the knowledge-based system performs intended task
Type of representation used	mix formal, declarative, and informal information	formal, often procedural information

6.4 Fulfilled Requirements

REFSENO fulfills most requirements listed in Chapter 3. These are in particular:

- **R4 (administrative information).** Administrative information can be attached as attributes to the top concept »CONCEPT«.
- **R7 (storage of various kinds of artifacts).** For each kind of artifact, a corresponding concept can be defined using REFSENO.
- **R8 (explicit representation of the conceptual knowledge).** An experience base schema captures the conceptual knowledge.
- **R9 (separation of characterization and conceptual information).** Just as in database systems, characterization information (instances) is separated from the experience base schema (conceptual information).
- **R10 (rationalized conceptual information).** Retrieval goals and the corresponding reuse scenarios rationalize both concepts and attributes. If a reuse scenario is no longer to be supported, the corresponding concepts and attributes can be easily identified and be removed from the experience base schema. All other parts of the schema depend on the concepts and attributes. For instance, if a type is no longer needed, because there exists no attribute of this type any longer, the type definition can be removed from the schema.
- **R11 (modularity of conceptual information).** Two schemas that are complementary in their contents can be connected via nonterminal concept attributes between related concepts (see also page 192).
- **R17 (artifact recording).** The semantic for recording an instance is defined precisely (see Section 6.2).
- **R18 (various characterizations of one artifact).** Since artifacts are linked to a characterization via a reference, several characterizations can reference the same artifact.
- **R19 (tolerance of different levels of abstraction).** An artifact can be characterized by a single text attribute value or by an elaborate system of related instances. In combination with requirement R18, an artifact can be described on different levels of abstraction.
- **R20 (tolerance of incomplete information).** Attribute values can be set to »undefined«, »unknown«, or »not applicable«. During retrieval these

attributes will not be used for similarity computation. Therefore incomplete information is tolerated.

- **R21 (tolerance of uncertain information).** Because the defined retrieval operation is able to find artifacts that have a characterization similar to the query, it is likely that artifacts with »slightly wrong« characterization are also returned as the result of the query. Thus, uncertain information is tolerated.
- **R22 (tolerance of imprecise information).** Imprecise information can be expressed using sets, intervals, and taxonomies.
- **R23 (transparency of duplicated information).** Through the construct of nonterminal concept attributes, no characterization information needs to be duplicated. Therefore, no duplicated information is stored in the experience making this requirement »obsolete«.
- **R24 (tolerance of inconsistent information).** The kind »derived-from« can be used to document the artifacts that led to the development of a new artifact. A value of a nonterminal concept attribute of this kind has consistency information attached (see Table 20).
- **R25 (integrity constraints).** Integrity constraints can be specified using assertions, value inferences, and nonterminal concept attributes.
- **R27 (artifacts' status).** An artifact´s status can be specified as an attribute of the top concept »CONCEPT«. During retrieval, valid statuses may be specified using a filter on the status attribute.
- **R28 (versioning).** A version number can be specified as an attribute.
- **R29 (configurations).** All artifacts belonging to a single configuration can be captured as a set of values for a »configuration« attribute using a nonterminal concept attribute with a cardinality > 1.
- **R31 (artifact preference information).** The preference information can be defined as a regular attribute of a concept.
- **R32 (browsing).** Browsing is explicitly supported through navigation along nonterminal concept attributes.
- **R33 (textual search).** Textual search is supported through keywords: Either using terminal concept attributes of type »text« or terminal concept attributes of symbol types.
- **R34 (similarity-based retrieval).** This is inherent to REFSENO´s retrieval semantics (see Section 6.2.1).
- **R35 (tolerance of incomplete query information).** REFSENO´s retrieval semantics will ignore all attributes which have a special value (»undefined«, »unknown«, or »not applicable«).
- **R36 (tolerance of uncertain query information).** Analogous to reason given for R21.
- **R37 (tolerance of imprecise query information).** Analogous to reason given for R22.
- **R38 (context information).** Context information can be specified using regular attributes. REFSENO supports context information explicitly by providing a special context layer in the concept´s intension.

- **R39 (context-sensitive retrieval).** This is inherent to REFSENO retrieval semantics (see Section 6.2.1) by allowing related instances as part of a query specification.
- **R41 (interface information).** Interface information can be specified using regular attributes. REFSENO supports interface information explicitly by providing a special interface layer in the concept´s intension.
- **R42 (application history).** The application history can be represented using a concept for the description of applications and linking the application instances to the instance characterizing the applied artifact.

6.5 Summary

In this chapter, REFSENO (representation formalism for software engineering ontologies) has been described. It enables the definition of experience base schemas. REFSENO´s central construct is the »concept«. The concept models software engineering artifacts. The specification of a concept includes a set of attributes which either model how software engineering artifacts are specified for storage and retrieval (terminal concept attributes) or how they relate to other artifacts (nonterminal concept attributes). Both terminal and nonterminal concept attributes are typed. Formulas are used to specify similarity computation, value inferences (automatic computation of attribute values), and assertions (conditions that must always be true). REFSENO´s notation is complemented by semantics that define how characterization information is retrieved, inserted, removed, or changed.

REFSENO draws from ideas from several areas including database systems (e.g., semantic relationships between artifacts and implied consistency rules) [Che76, Dit97, GR95], case-based reasoning mechanisms (e.g., similarity-based retrieval with incomplete information) [Alt97], and knowledge-based mechanisms (e.g., value inferences and assertions) [Ric89]. In addition, REFSENO is object-centered similar to the meta modeling in UML [Rat97]. The integration of the approaches results in a new formalism that unites many of the advantages of the individual approaches while avoiding most of their drawbacks.

Among the (practical) benefits of schemas described using REFSENO are the following:
- Software engineering artifacts and their properties can be modeled naturally.
- Schemas can be used as a communication vehicle to (a) discuss the structure underlying the characterization information of an experience base and (b) aid in standardizing a corporate-wide vocabulary.
- Schemas are modular, that is, they can be extended and/or combined by/with other schemas.
- Schemas are tailorable to organization-specific needs.
- Schemas grow with organizational needs regarding both growing number of artifacts and growing diversity of artifacts.

- Schemas consider peculiarities of software engineering experience (e.g., context-sensitive retrieval, consideration of both qualitative and quantitative information as well as the consideration of incomplete, imprecise, inconsistent, and uncertain information).
- Schemas are formal enough to allow automated support for retrieving and characterizing artifacts (semantics of REFSENO).
- The notation and semantics of REFSENO fulfill most of the requirements stated in Chapter 3.

Based on REFSENO´s semantics, a generic tool for retrieving and characterizing artifacts can be specified. The tool »interprets« an experience base schema and guides the user by asking for attribute values of a previously (user) selected concept. This generic tool is called the *General Purpose Browser*, which is part of the technical infrastructure of an experience base. The the architecture will be described in the next chapter.

7 Technical Infrastructure of an Experience Base

Knowledge management is much more than technology, but »techknowledgy« is clearly a part of knowledge management.
[DP98, p. 123]

So far, requirements for the technical infrastructure of an experience base (Chapter 3) and the envisaged usage of such an infrastructure (Chapters 4) have been described. The functionality regarding the retrieval and manipulation of characterizations has been listed in Section 6.2. However, it has not been described how a technical infrastructure could look like that fulfills the elicited requirements and supports the described tasks. This is the aim of this chapter. It summarizes the discussions about the technical infrastructure of an experience base by presenting a practically feasible architecture.

7.1 Scalable Technical Infrastructure for an Experience Base

To support the reuse and learning of software engineering experience effectively and efficiently, the installation of the technical infrastructure of an experience base must focus on practical problems. Therefore, it has to meet various requirements (see Chapter 3). These requirements can be fulfilled by a three-tier client-server architecture as shown in Figure 32 [ABvWT98, ABT98, ABH+99]:

1 The **top or application tier** consists of the tools the user interacts with. There are two types of tools:
   - **Artifact-Specific Tools** allow the development and maintenance of certain kinds of artifacts (usually one kind). Examples include a Measurement Tool for editing measurement plans, a Process Modeling Tool for editing process models, and a text editor or text processing system for editing text documents. Typically, a set of Artifact-Specific Tools already exists in an organization when an infrastructure for an experience base is to be installed. Thus, these tools need to be integrated into the infrastructure. In addition, most of these Artifact-Specific Tools use their own proprietary, native data format. This fact has to be considered by the storage tier of the architecture (see below).
   - **EB-Specific Tools** allow the manipulation and analysis of the characterizations stored in the experience base. Because the behavior of these tools is defined by the schema of the experience base, these tools are artifact-independent. Examples include the General Purpose Browser to find, insert, change, and delete characterizations (see Section 6.2) as well as

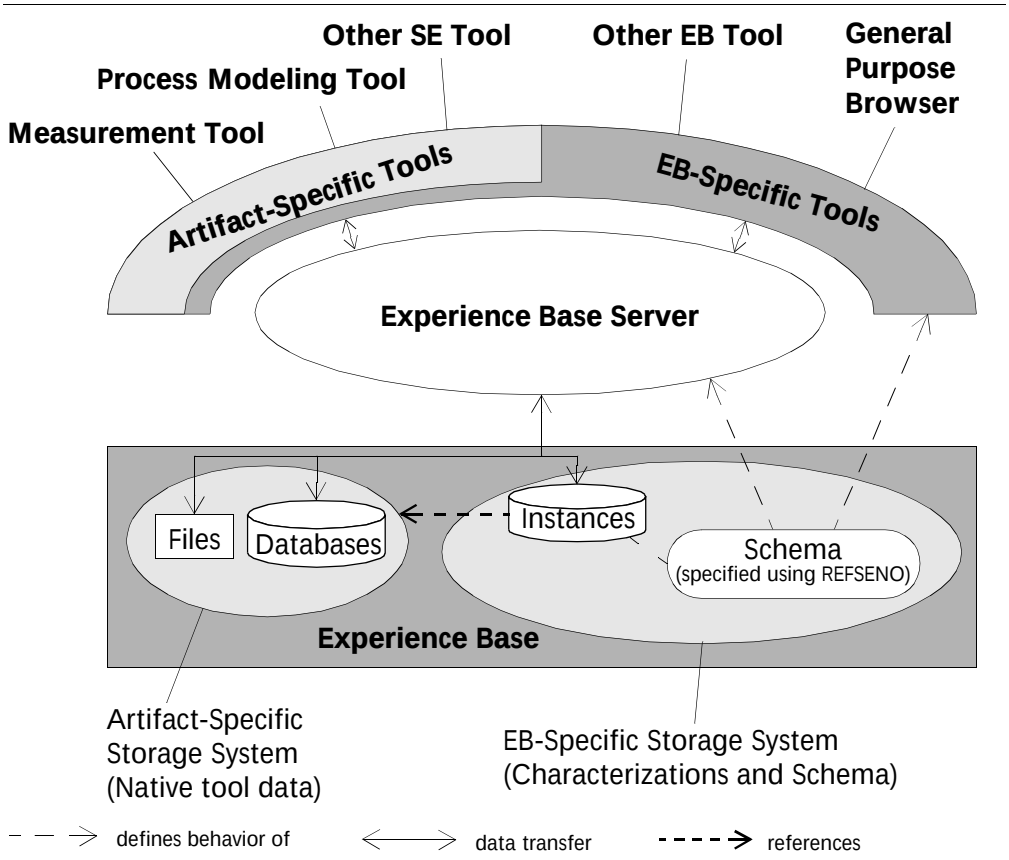


Figure 32: Architecture of the technical infrastructure of an experience base

analysis tools (e.g., to check the consistency among artifacts using the status of all nonterminal concept attributes of the kind »defines« – see Table 20 on page 186).

2 The **middle or access tier** consists of the Experience Base Server which synchronizes the access of the application tier tools to the experience base, captures feedback for improving the infrastructure, creates access statistics, and enforces the access rights for the information stored in the experience base.

3 The **bottom or storage tier** consists of the Artifact-Specific Storage System and the EB-Specific Storage System. The Artifact-Specific Storage System stores the artifacts in the native data format of their corresponding Artifact-Specific Tool. Thus it must be able to store files as well as provide access functions for various database management systems (for those tools that use a database management system to store their data). The EB-Specific Storage System stores the artifacts´ characterizations and the underlying schema. It provides the basic semantics of REFSENO defined in Section 6.2. A reference, which is part of an artifact´s characterization, points to the artifact belonging to the characterization and contains a link to the appropriate Artifact-Specific

Tool. This allows the General Purpose Browser to invoke the right Artifact-Specific Tool once the user requests to view a particular artifact. The reference is also needed by the Experience Base Server to ensure the consistency between an artifact and its characterization(s).

7.2 Usage of the Infrastructure

This section explains how the proposed infrastructure is utilized using a (fictive) scenario. The explanation refers to the steps of the quality improvement paradigm (see Figure 8 on page 77) and the maturing tasks defined in Chapter 4.

QIP 1: Characterize

In the beginning of a project, the General Purpose Browser is started and the new project is (partially) specified guided by the experience base schema for project characterizations, techniques, product models, etc. (tasks »specify (T4, page 90)« and »identify (T8, page 92)«). The General Purpose Browser then returns a list of similar project characterizations by using the EB-Specific Storage System (tasks »evaluate (T11, page 94)« and »select (T19, page 98)«). Starting from the project characterizations, the user can navigate to characterizations of relevant artifacts such as techniques employed, quantitative experiences collected, etc. The navigational search is supported through nonterminal concept attributes, that is, for each instance the user navigates to, the EB-Specific Storage System is used to retrieve it.

QIP 2: Set goals

Next, project goals (including measurement goals to enable strategic learning for the software development organization) are set based on the results of the step before. This is done in a similar manner: The General Purpose Browser is invoked to find similar measurement goals from the past. However, these measurement goals and the respective measurement plans have to be adapted to project-specific needs. For this purpose, a reference, which is stored as part of the artifact's characterization, is passed to the Measurement Tool. The Measurement Tool thus invoked loads the old measurement plan (using the data retrieval services of the Experience Base Server) and allows the project manager to tailor it to the project's needs (task »utilize (T22, page 100)«). The new measurement plan is saved using the data storage services, but its characterization is not specified yet. That is, no instance is stored for it and, therefore, it is not considered as part of the experience base. Thus, it is not available to other projects (yet). This avoids the usage of unvalidated artifacts.

QIP 3: Choose artifacts

In the same way, process models and other artifacts needed by the project are retrieved and tailored to specific needs of the project at hand.

QIP 4: Execute

The project is then executed.

QIP 5: Analyze

Once the project is finished, the project's artifacts (which are deemed worthwhile to keep) are stored in the experience base for future projects. For this pur-

pose, the quality of the artifacts is determined through careful analysis with the help of the Artifact-Specific Tools (task »collect (T34, page 108)«). For those artifacts to be stored (task »copy (T36, page 109)«), the export interface of the Artifact-Specific Tools compute the attributes' values of the artifact's characterization automatically as far as possible (task »characterize initially (T38, page 110)«). This is necessary because the generic EB-Specific Tools are not able to read the native data format of the Artifact-Specific Tools. The procedure may involve inserting several semantically related characterizations (e.g., for a code module each function may be characterized separately in addition to the characterization of the whole module). For this purpose, the artifact has to be split into chunks (task »split (T37, page 109)«). Attribute values, which cannot be computed automatically, must be entered manually. This is realized through invoking the General Purpose Browser, which prompts the user for the missing values. This procedure is followed for all artifacts to be stored.

QIP 6: Package

The newly inserted instances have to be validated and disseminated. Therefore, they are initially marked as »to be validated« at the time of their insertion. To guarantee a minimal quality, artifacts are qualified (tasks »analyze quality (T41, page 111)« and »review (T43, page 112)«). In case of minor quality deficits, the artifacts are corrected by invoking the respective Artifact-Specific Tool (task »evolve (T46, page 114)«). During the review, the reviewers also assess the reuse potential of the artifacts by using the respective Artifact-Specific Tool (task »analyze reuse potential (T44, page 113)«). As a result, the artifact may be modified to increase its reuse potential (task »evolve (T46, page 114)«). Usually this requires modification of the artifact's characterization (using the General Purpose Browser). In case of major quality deficits, the artifacts will be rejected. In case the artifacts pass the review, their characterizations are completed by the quality properties of the artifacts (task »complete characterization (T47, page 114)«). Finally, the corresponding instances are marked »validated« (task »publish (T48, page 115)«) and people interested in the new artifacts are informed (task »inform (T49, page 115)«). After this, it is possible for other projects to access the new artifact.

7.3 Outstanding Characteristics of the Infrastructure

In the following, some of the technical infrastructure´s outstanding characteristics from a practical point of view are discussed.

Tailorability to organization-specific needs

When introducing a new infrastructure in an organization, already existing tools must be coped with. The presented technical infrastructure supports the integration of already existing tools using the Experience Base Server. A synchronization module allows to retrieve and store artifacts in the experience base [ABH+99].

Infrastructure grows with organizational needs

The integration with already existing (Artifact-Specific) tools can be done on various levels. The simplest integration does not allow for (partly) automated characterization. On this integration level, artifacts are characterized manually – usually using the General Purpose Browser. Because the characterization of artifacts is effort-intensive, extensive characterization schemas are prohibitive. However, as more and more artifacts of the same kind accumulate, more elaborative schemas are needed to distinguish among the artifacts. The wish for (partly) automated characterization of the artifacts arises. This can be accomplished by using a specialized Artifact-Specific Tool that is able to extract the characterization information from the artifact and to transfer this information to the Experience Base Server (and thus into the experience base). Since each artifact is associated with an appropriate viewer, the Experience Base Server invokes the right viewer – transparent for the user – if the viewing of an artifact is requested (see storage tier in Section 7.1) – even if various artifacts of the same kind are developed using different Artifact-Specific Tools.

Ease of use

Both the possibility to integrate existing tools and the intelligent Experience Base Server reduce the training effort for using the technical infrastructure to a minimum. In fact, the experience base can be populated and utilized using the World-Wide Web (WWW) – something most users are familiar with.

Distributed learning environment

The use of WWW technology for the General Purpose Browser makes the infrastructure scalable from local to global use supporting world-wide learning in a company that is distributed around the globe (see Figure 33 for an exemplary screenshot).

Document Management System

The Experience Base Server supports check-in (= recording) and check-out (T23, page 101) of artifacts. Through support for versioning [NT98a], the Experience Base Server registers when an artifact is checked-out and when it is checked-in. In contrast to code versioning systems (e.g., RCS [Tic85]), artifacts may be checked-out for a long period of time – in the extreme case for the whole duration of a project (which can be years for a large project), because often artifacts are recorded at the end of a project. Therefore, lock mechanisms such as those of RCS are not appropriate. Instead, users who checked-out a particular artifact are informed (task »inform (T49, page 115)«) if someone else has improved the artifact in some way (e.g., removed a defect or extended its contents). In addition, the experience base ensures the consistency between an artifact and its characterizations by providing atomic operations for artifacts (e.g., insertion, deletion, update of artifacts).

7.4 Fulfilled Requirements

The presented technical infrastructure fulfills all requirements listed in Chapter 3 not closed so far:

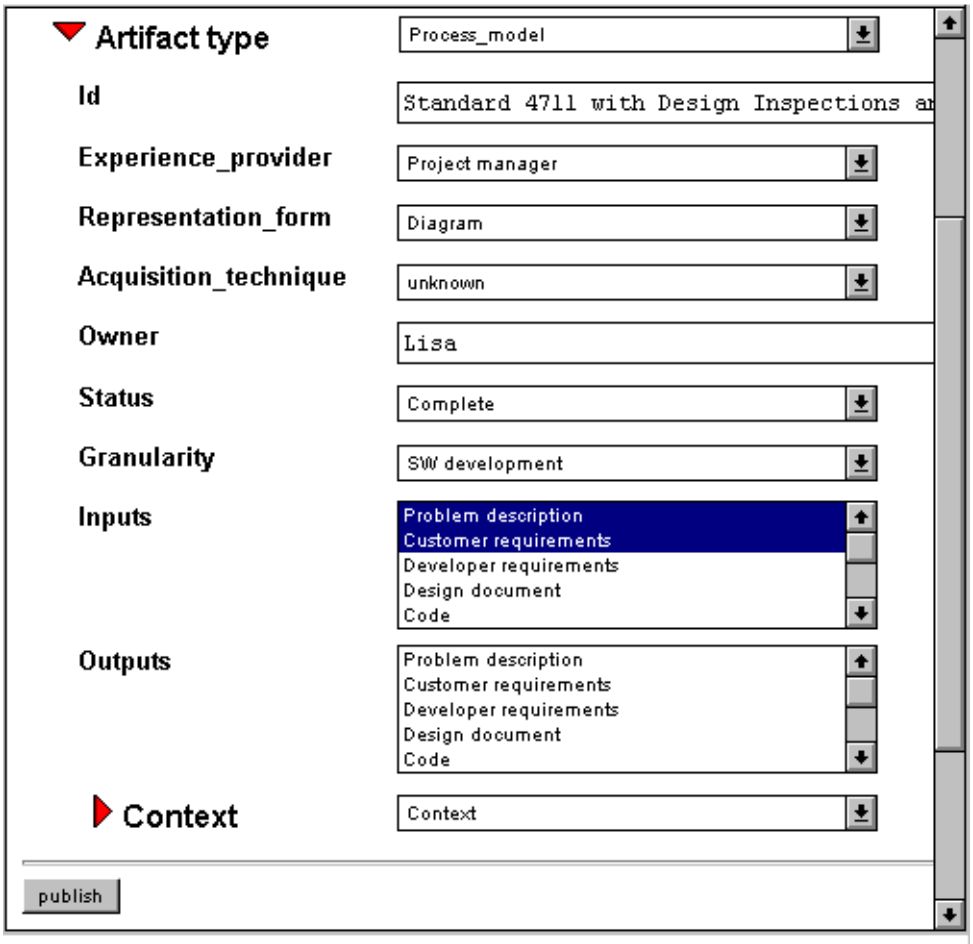


Figure 33: An Example for characterizing and publishing a process model using the WWW interface

- **R1 (ease of use).** Users are guided by the conceptual information for both recording and retrieval of artifacts. Nonterminal concept attributes allow navigation among artifacts or parts thereof. The form-based interface of the General Purpose Browser can even be used over intranets and internets.
- **R3 (tool integration).** The Experience Base Server allows any number of tools to communicate with the experience base over a network. Special synchronization modules allow the integration of arbitrary Artifact-Specific Tools.
- **R5 (access rights).** The access rights are handled by the Experience Base Server.
- **R6 (network access).** The Experience Base Server is accessed over a network. This feature is used to support distributed organizations.
- **R12 (accommodation of new artifact kinds).** As new Artifact-Specific Tools can be added and the experience base schema can be extended by new concepts at any time, new kinds of artifact can be readily incorporated.

- **R13 (maintenance of conceptual information).** This is supported by the EB-Specific Storage System. For example, CBR-Works (an EB-Specific Storage System by tec:inno GmbH, Germany [tec00]) supports the evolution of the experience base schema by adapting the instances accordingly [HW98].
- **R15 (data collection for evaluation).** Data collection is supported by the Experience Base Server. The data is stored in the EB-Specific Storage System.
- **R16 (data analysis for improvement).** Specialized analyses can be supported by an EB-Specific Tool. The Experience Base Server supports any number of EB-Specific Tools.
- **R26 (accommodation of growing collection).** As maintenance of the conceptual information is supported (see above), the characterization schema of an artifact can be expanded when needed. As pointed out in Section 7.3, this may involve the integration of a new Artifact-Specific Tool that is able to (partly) characterize artifacts automatically. Since REFSENO allows to characterize artifacts incompletely (requirement »R20 (tolerance of incomplete information)«) and artifacts of the same kind on different levels of abstractions (requirement »R19 (tolerance of different levels of abstraction)«), the characterizations of the old artifacts do not have to be changed. New artifacts can be characterized using the new schema.
- **R30 (change notification).** Change notification is implemented through the check-in/check-out mechanism of the Experience Base Server.
- **R40 (check-out of artifacts).** The check-out of artifacts is supported directly by the Experience Base Server as an atomic operation.

7.5 Validation of the Infrastructure

At Fraunhofer IESE, the core parts of the architecture described in Section 7.1 have been implemented as a prototype [ABH+99, Sno99, Nic98]. The Intelligent Retrieval and Storage System (INTERESTS) consists of the General Purpose Browser, a rudimentary implementation of the Experience Base Server, and CBR-Works[1] as the EB-Specific Storage System (see Figure 34). So far, INTERESTS has been used successfully for many applications including CBR-PEB (an experience base for case-based reasoning systems) [ANT99a], KM-PEB (an experience base for knowledge management approaches) [Sno99], IPQM (an experience base supporting TQM for hospitals) [ABMN99], a lessons learned experience base at a large German insurance company [ABH+99], and COIN, Fraunhofer IESE's Corporate Information Network (see Chapter 10).

1 CBR-Works is a case-based reasoning tool by tec:inno GmbH, Kaiserslautern, Germany [tec00]. CBR-Works has been extended to provide the functionality needed for an EB-Specific Storage System in the context of a cooperation between Fraunhofer IESE and tec:inno.

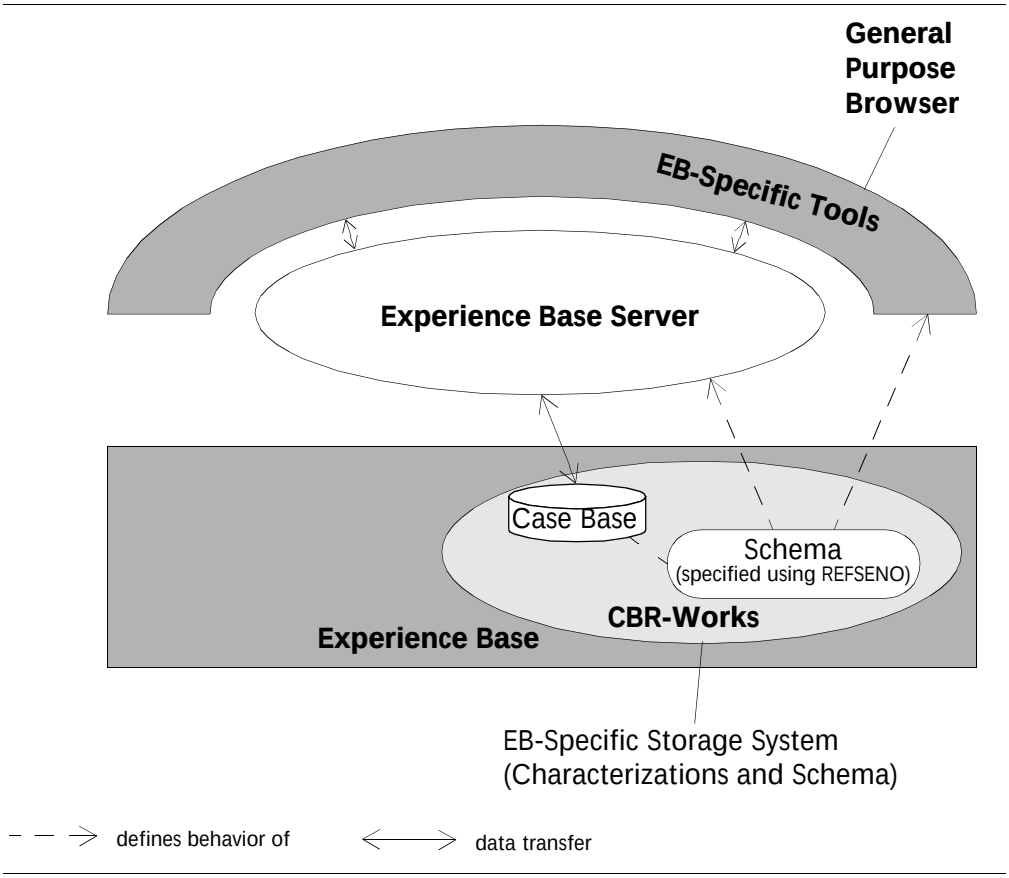


Figure 34: INTERESTS (Intelligent Retrieval and Storage System)

All these applications underline the practical applicability of the architecture. At the time this dissertation was being written, INTERESTS was being continuously extended towards the envisioned architecture of Figure 32.

7.6 Summary

This chapter presents both a vision and a prototype for the technical infrastructure of an experience base. The technical infrastructure has a three-tier architecture consisting of an application, an access, and a storage tier (see Figure 32 on page 200). The application tier consists of all tools the user interacts with, whereas the access tier consists of the Experience Base Server which synchronizes the access of the application tier tools to the experience base. In addition, the Experience Base Server captures feedback for improving the infrastructure, creates access statistics, and enforces the access rights for the information stored in the experience base. Finally, the storage tier stores the artifacts (in their native data format) and their characterizations as well as the experience base schema.

The application and storage tiers consist of an artifact-specific and an EB-specific part. This is necessary to allow the integration of Artifact-Specific Tools (application tier) that already exist in an organization when the technical infrastructure is installed. The Artifact-Specific Storage System (storage tier) reflects this by providing access functionality to allow the storage of artifacts in the native data format of their respective Artifact-Specific Tool. Tools operating on characterizations belong to the EB-Specific Tools (application tier). The most basic EB-Specific Tool is the General Purpose Browser which allows the retrieval and manipulation of characterizations over intranet and internet. Its behavior is defined by the experience base schema. Both characterizations and the schema are stored in the EB-Specific Storage System (storage tier) which provides the basic functionality described in Section 6.2 (semantics of REFSENO).

The (practical) benefits of the presented technical infrastructure include:
- Tailorability to organization-specific needs (already existing Artifact-Specific Tools – usually CASE tools – can be integrated)
- Growth with organizational needs (both growing number and growing diversity of artifacts are supported)
- Ease of use (integration of already existing tools and the use of WWW technology for EB-Specific Tools provide a familiar look-and-feel for the users)
- Support for distributed learning (usage of the WWW technology supports learning for organizations distributed around the globe)

In INTERESTS (Intelligent Retrieval and Storage System), a prototype of the technical infrastructure, key elements of the infrastructure have been implemented. INTERESTS has already proved the practicability of the architecture in a number of industrial-strength applications (see validation in Section 7.5).

One of the core components of the architecture is the EB-Specific Storage System. In INTERESTS, it is implemented using the commercially available case-based reasoning tool CBR-Works from tec:inno GmbH, Germany [tec00]. CBR-Works provides all essential constructs of REFSENO (with the defined semantics) and a graphical user interface for defining the experience base schema.

Since the experience base schema plays such a vital role for the usefulness of the experience base, it must be developed goal-oriented and systematically, that is, it must be engineered. Task methods matching the schema (e.g., methods describing where the attribute values for the artifacts´ characterizations can be acquired) also need to be engineered. These task methods will instantiate the task framework described in Chapter 4. Finally, the generic architecture described in this chapter needs to be tailored to organization-specific needs. These three activities (engineering an experience base schema, instantiating the generic task framework, tailoring the generic architecture) are subject of the task »structure« and will be explained in more detail in the next chapter.

8 Engineering SE Experience Management Systems

The structure of a repository is generally regarded as a key to obtaining good retrieval results. No matter how »intelligent« the matching algorithm, if components are indexed or otherwise structured poorly, it will be difficult to achieve good retrieval performance.

[Hen96, p. 280]

Organizational needs for an experience factory vary from organization to organization. The information stored in the experience base depends not only on the problems the organization wants to solve with the help of the experience factory, but also on already available knowledge and the reuse potential of stored information. All these factors determine what is actually stored in the experience base (cf. requirement »rationalized conceptual information (R10, page 47)«), how it is populated and utilized, and how (far) these tasks can be supported by the technical infrastructure. Hence, it is a necessity to be able to develop and tailor experience base schemas and methods for maturing.

This chapter details the task »structure« by presenting GOODSEE, an industrial-strength method for the systematic, goal-oriented development of an adequate experience base schema and effective methods for populating and utilizing the contents [TA00a]. In addition, it tailors the generic architecture to organization-specific needs. GOODSEE (goal-oriented ontology development for software engineering experience) starts with the identification of stakeholders and their interests (Section 8.1) thus eliciting the goals of the experience factory. It continues with the identification of subject areas (Section 8.1) as well as reuse and knowledge acquisition scenarios (Section 8.2) before it details the information needs for these scenarios in form of the experience base schema using REFSENO's constructs (Section 8.3). Based on both the knowledge acquisition scenarios and the experience base schema, prescriptions for how to record new information are developed (Section 8.4). Finally, a plan for actually implementing the change is created which includes the tailoring of the architecture (Section 8.5).

Thus, the presented technique goes beyond the pure definition of a schema by:

- Involving people who have an interest in the experience factory
- Not merely listing the sequence in which the tables (presented in Chapter 6) are to be filled out, but by providing guidelines and questions tailored specifically to the definition of a repository for software engineering experience

- Including the planning for how to fill the experience base with information

The subtasks of »structure« are described using a notation similar to that of Chapter 4 (see page 83), however, the type (all tasks in this chapter are »generic«) and requirements (all tasks of this chapter require R12 (accommodation of new artifact kinds), R13 (maintenance of conceptual information)) are not listed.

Task T56: structure

Objective: change the conceptual knowledge of the experience base and adapt the technical infrastructure accordingly

Input: state of the practice; improvement suggestions

Output: changed schema of the experience base (part of the improved state of the practice); feedback indicators; training material; change plan; set of defined objectives

Decomposition: design SEEMS (T61, page 210), infuse SEEMS (T107, page 238)

Method: The subtasks are performed sequentially.

Task T61: design SEEMS

Objective: define the conceptual information of the experience base in a goal-oriented manner

Input: state of the practice; improvement suggestions

Output: changed schema for the experience base (part of the state of the practice); feedback indicators; set of defined objectives; minimal quality requirements; application policies; knowledge collection plan; »record« methods; knowledge source adaptation plan; reuse scenarios

Decomposition: set objectives (T62, page 212), set subject areas (T67, page 215), identify scenarios (T70, page 217), conceptualize (T76, page 220), define ›record‹ (T95, page 231)

Method: The subtasks are performed sequentially. The goal-oriented definition of the conceptual information requires as a first step to reach a consensus on the objectives to be achieved by the experience factory in general (task »set objectives«). Each of these objectives has success criteria associated with it allowing to assess the progress toward the objective. To achieve these objectives, information must be collected, stored, disseminated, and utilized across the organization. In the beginning, concrete information needs may not be known; however, subject areas can be identified and selected (task »set sub-

ject areas«). Once the objectives and the subject areas are known, concrete information types are identified using scenarios (task »identify scenarios«). Scenarios also suggest how to acquire and utilize the information items. The information needs identified through the scenarios are further detailed (identification of characterization schemas including the definition of similarity; task »conceptualize«) to allow for (partial) automatic support as described in Chapter 4. Finally, the scenarios and the detailed information needs are used to come up with a precise definition of the »record« methods (task »define ›record‹«).

Note: If the experience base schema is *not* developed for the first time, improvement suggestions lead the way how the design of the experience base should proceed. Improvement suggestions may be on different levels of detail and thus pertain to any of the subtasks. For instance, the (strategic) improvement suggestion to add another objective to be addressed by the experience factory would start the subtask »set objectives«. All remaining subtasks would have to be performed in sequence. However, the (technical) improvement suggestion to add another attribute for a particular concept would trigger the subtask »conceptualize« (and within that subtask the subtask »identify reuse information«). Previous subtasks (i.e., »set objectives«, »set subject areas«, and »identify scenarios«) would not be performed.

Because *improvement suggestions* may be relevant for any of the subtasks, they *are not mentioned explicitly as inputs below*.

8.1 Identification of Competencies

According to O'Leary, the choice of a conceptualization is a »political process that requires ›negotiation‹« among the stakeholders [O'L97, p. 327].Furthermore it is impossible to find a conceptualization that is preferred by all stakeholders. In this context, a stakeholder is defined as follows:

Definition 14: Stakeholder

> *A* stakeholder *with regard to a given issue, decision or initiative is someone who has a definable interest in the outcome of that issue, decision or initiative. [SCK+96, p. 70]*

The stakeholders do not only include supporters, but also strategic opponents or competitors [SCK+96]. They can have multiple interests in the issue at hand, including interests which may conflict.

O'Leary's observation has two consequences:

1 Schemas cannot be transferred between organizations on a one to one basis (although it is expected that schemas are transferable in parts).

2 The »political process« among the stakeholders of a particular experience factory must be performed for each experience factory organization regardless of how much experience has been gained with the setting up and running of experience factories in the world at large.

The importance of involving the stakeholders in the design of the experience base has already been stated in Section 5.1.1 and has been recognized by many others (e.g., [ASR99, BSLC96, SCK+96, Sku95]). This section takes the importance of stakeholders into account by showing how to identify the stakeholders, their interests and their objectives regarding the experience factory.

Task T62: set objectives

Objective: define objectives and associated success criteria for the experience factory

Input: none

Output: set of defined objectives (consisting of selected objectives and corresponding success criteria to measure the progress towards the selected objectives)

Decomposition: identify stakeholders (T63, page 212), identify objectives (T64, page 213), select objectives (T65, page 214), define success criteria (T66, page 214)

Method: The subtasks are performed sequentially. As a first step, the stakeholders and their interests are identified (task »identify stakeholders«). Then, the stakeholders are interviewed for candidate objectives (task »identify objectives«). From these candidate objectives, the final objectives are selected (task »select objectives«). Finally, success criteria are defined for all selected objectives (task »define success criteria«).

Task T63: identify stakeholders

Objective: identify all stakeholders of the experience factory

Input: none

Output: list of stakeholders with their interests

Decomposition: –

Method: First, stakeholders and candidates for critical roles are listed. A checklist may be used to aid the identification (adapted from [SCK+96]):

- Who are potential customers for the information stored in the experience base?
- Who is funding the experience factory?

- Who will be working for the experience factory (project supporters, experience engineers, technology experts, etc.)?
- Who will be supplying artifacts for the experience base?
- Who is providing the technical infrastructure?
- Who might be interested in seeing the experience factory *not* succeed?
- Who were stakeholders in previous process improvement programs and reuse projects?

Second, the interests of the stakeholders are identified [SCK+96]. The interests can be either expressed by the stakeholders directly or they can be inferred by the planners of the experience factory. For each interest, enablers and barriers are listed. Examples for interests are:

- Increase collaboration among project teams
- Relieve experts of recurring tasks
- Avoid making the same mistake twice
- Increase employees' job satisfaction
- Increase customer satisfaction

Third, potential conflicts, trade-offs, and dependencies among the stakeholders' interests are identified and recorded (but not resolved).

Task T64: identify objectives

Objective: determine a candidate list of objectives

Input: list of stakeholders with their interests

Output: set of objectives

Decomposition: –

Method: The stakeholders are asked for existing problems and how these problems could be solved. A questionnaire such as the one in Appendix A may be used to identify the stakeholders' problems, their boundaries, and potential solutions. The questionnaire also includes questions pertaining to the opinions of other stakeholders to identify potential conflicts among the stakeholders. In contrast to the stakeholders' interests, objectives should have a clear connection to the concept of an experience factory.

Exemplary objectives are:

- Support for lessons learned that record past mistakes and ways to avoid these mistakes
- Technical support for Total Quality Management
- Knowledge maps for the identification of experts
- Technical support for templates

Task T65: select objectives

Objective: select the objectives to be met by the experience factory

Input: set of objectives

Output: set of selected objectives

Decomposition: –

Method: Each objective is evaluated and prioritized. Evaluation criteria include:

- How important is the objective with regard to the whole experience factory?
- Are there potential conflicts with other objectives?
- Are there potential conflicts with the interests of any stakeholder?
- What is the time perspective: short-term, mid-term, or long-term?

Additional evaluation criteria are given by Akkermans, Speel, and Ratcliffe [ASR99]:

- What are the expected benefits for the organization from the considered solution?
- How large is this expected added value?
- What are the expected costs for the considered solution?
- How does this compare to possible alternative solutions?
- Are organizational changes required?
- To what extent are economic and business risks and uncertainties involved regarding the considered solution direction?

With the answers to these questions, the objectives, which are to be pursued further, are selected. Selected objectives should range from short-term (yielding acceptance/motivation on part of the software development organization) to long-term (usually yielding higher payoffs than short-term objectives).

Task T66: define success criteria

Objective: define success criteria that allow to measure progress towards the selected objectives

Input: set of selected objectives

Output: success criteria

Decomposition: –

Method: Define success criteria (e.g., using the GQM technique [vSB99]). It must be clear what the success measures are and how validity, quality, and satisfactory performance can be measured.

Task T67: set subject areas

Objective: determine the competence areas of the experience base

Input: set of defined objectives (= set of selected objectives + success criteria)

Output: set of selected subject areas

Decomposition: identify subject areas (T68, page 215), select subject areas (T69, page 215)

Method: The subtasks are performed sequentially. First, potential subject areas are identified (»identify subject areas«). Then, the potential subject areas are evaluated and selected (»select subject areas«).

Task T68: identify subject areas

Objective: list potential subject areas of the experience base

Input: set of defined objectives

Output: set of subject areas

Decomposition: –

Method: The kinds of information needed to reach the objectives are identified.

Task T69: select subject areas

Objective: evaluate and select the actual subject areas of the experience base

Input: set of subject areas

Output: set of selected subject areas

Decomposition: –

Method: First, the subject areas are evaluated. Exemplary evaluation criteria are (adapted from [WD99]):

- Potential:
  - How many of the objectives can be reached by storing information for this subject area? How important is the explicit storage of this information for the objectives?
  - How many users are interested in the particular subject area?
  - Are the tasks to be supported complex enough to justify recording of information for this subject area?
  - How do existing and planned standards (organizational or industry-wide) affect reuse potential?
- Existing artifacts:

  - Are people available who are experienced in the subject area (e.g., for developing and qualifying artifacts of the subject area)?
  - Have they developed artifacts of high quality for this subject area?
  - Are these artifacts available?
  - Can the experts decide whether two artifacts are similar?
  - Is it possible to collect a set of artifacts that represents a good coverage of the subject area within a few months?
- Stability and maturity: Are needs or technology changing slowly enough to allow recovery of an investment to store the information in the subject area explicitly?

If it is not feasible to store information explicitly for a particular subject area due to a low potential or quickly changing needs, a knowledge map may be created telling people whom to ask for information about a given segment of the subject area.

8.2 User Requirements

Objectives and subject areas describe the experience base on an abstract level that needs to be refined. However, the abstract description cannot be refined to a REFSENO description directly, because for the following reasons.

- **Tacit stakeholders' knowledge.** The knowledge on how to process information is often tacit. Therefore, this knowledge needs to be elicited by adequate techniques.
- **Stakeholders think in examples.** For many people, it is difficult to articulate general process descriptions, such as the sequence of work, intermediate artifacts, and roles participating in the process [GHD99]. Instead, it is much easier for them to recall examples from the past (using concrete contexts and people) or state how information processing should be done in the future (again using concrete contexts and people).
- **Formal notation.** The formal notation may be difficult to understand for some stakeholders.

The use of *scenarios* has proven to overcome some of these problems [Jar99, GHD99]. Scenarios are a narrative description of events in a given organizational setting [BSLC96]. A scenario includes descriptions of the organizational context, character profiles for the individuals appearing in the scenario, and a sequence of events.

In this section two kinds of scenarios are advocated: reuse scenarios describing how to reuse the contents of an experience base and record scenarios describing how to populate the experience base. The scenarios are the basis for developing a REFSENO description of the experience base and for defining the reuse and record tasks.

Task T70: identify scenarios

Objective: develop narrative descriptions for using the technical infrastructure of the experience base

Input: set of defined objectives; set of selected subject areas

Output: reuse scenarios (consisting of informal descriptions and retrieval goals); record scenarios (consisting of potential knowledge sources and informal descriptions of knowledge acquisition)

Decomposition: identify reuse scenarios (T71, page 217), define retrieval goals (T72, page 218), identify record scenarios (T73, page 218)

Method: First, typical reuse situations are described informally (task »identify reuse scenarios«). Each of these scenarios is then characterized using one or more retrieval goals (task »define retrieval goals«). They describe what information needs to be stored and for what purpose this information will be used by whom. Consequently, the needed information needs to be acquired. The task »identify record scenarios« elicits informal descriptions of potential acquisition processes.

The tasks are performed sequentially; however, if acquisition of the needed information is not possible, the reuse scenarios may have to be changed, resulting in an iteration of the subtasks.

Task T71: identify reuse scenarios

Objective: describe reuse situations informally

Input: set of defined objectives; set of selected subject areas

Output: informal description of reuse scenarios

Decomposition: –

Method: Users of the experience base are asked to describe how they would use the technical infrastructure. The description should include how to locate artifacts (known to the user) and/or how to search for artifacts (unknown to the user at the time of retrieval) as well as how the artifacts are actually utilized. Special attention should be paid to critical aspects involved such as those relating to time, quality, and needed resources.

The retrieval of artifacts in a scenario may be either direct (i.e., the user finds the suitable artifact(s) without expert help) or indirect (i.e., the user finds a contact person in the experience base who in turn suggests suitable artifact(s)).

Task T72: define retrieval goals

Objective: summarize the informal descriptions by characterizing the corresponding retrieval goals

Input: informal descriptions of reuse scenarios

Output: retrieval goals

Decomposition: –

Method: All relevant retrieval goals are extracted from the informal descriptions and augmented with a unique name and the informal descriptions they were extracted from. A retrieval goal has four facets:

1 **Artifact kind.** The kind of artifact to be retrieved.
2 **Purpose.** The purpose for which the artifact is supposed to be retrieved.
3 **Viewpoint.** The role of the user (e.g., project manager, software developer, experience engineer).
4 **Environment.** The environment in which the retrieval attempts will be made.

Note 1: Retrieval goals are the primary means of rationalizing the experience base schema. They motivate why a particular concept or attribute is necessary. Therefore, it is important that stakeholders agree to support the defined set of retrieval goals.

Note 2: Retrieval goals are very similar to GQM goals [BCR94b]. Similar to GQM goals, retrieval goals can be stated using the following template:

Retrieve <artifact type>
for the purpose of <purpose>
from the viewpoint of a <role>
in the context of <context>

The facet »quality focus« has been omitted because it is always »reusability/applicability« and is further refined by the characterization schema for the artifact type (which is subject of the task »conceptualize«). Applicability criteria may also include contextual information which is, however, on a finer level than the »context« of the above template. For example, the »context« of the above template may be an organizational unit such as »Webonautics' software development center«, whereas the context information »application domain« could be an applicability criterion.

Task T73: identify record scenarios

Objective: devise potential ways of acquiring needed information

Input: reuse scenarios (consisting of informal descriptions and retrieval goals)

Output: record scenarios consisting of potential knowledge sources and informal descriptions of knowledge acquisition

Decomposition: identify knowledge sources (T74, page 219), describe knowledge acquisition (T75, page 219)

Method: For each kind of artifact, potential knowledge sources are identified (task »identify knowledge sources«) and experts are interviewed to find ways of making the knowledge available (task »describe knowledge acquisition«).

Task T74: identify knowledge sources

Objective: identify knowledge sources for needed information

Input: reuse scenarios

Output: knowledge sources

Decomposition: –

Method: In a brainstorming session, the stakeholders identify potential knowledge sources. Knowledge sources can be existing documents and databases, but also human experts.

Task T75: describe knowledge acquisition

Objective: describe acquisition informally

Input: reuse scenarios; knowledge sources

Output: informal knowledge acquisition descriptions

Decomposition: –

Method: Experts (i.e., for human knowledge sources the knowledge sources themselves; for other knowledge sources the people responsible for managing the knowledge sources and/or the people providing the knowledge) are presented the reuse scenarios and asked how the needed information can be made available. There may be several (alternative) ways for acquiring artifacts of a particular kind. There should be record scenarios for both the initial development of artifacts and the maintenance/improvement of existing artifacts.

The scenarios should be checked for their feasibility [ASR99]:

- Technical feasibility:
  - Are there critical aspects involved, relating to time, quality, needed resources, or otherwise? If so, how to go about them?

  - How complex is the interaction with other information systems and possible other resources (interoperability, systems integration)? Are state-of-the-art methods and techniques available and adequate?
  - Are there further technological risks and uncertainties?
- Project feasibility:
  - Is there adequate commitment from the stakeholders for further project steps regarding the solution?
  - Can the needed resources in terms of time, budget, equipment, staffing be made available?
  - Are the required knowledge and other competencies available?
  - Are there further project risks and uncertainties?

Note: Not all knowledge has to be recorded explicitly as information. It may suffice to create a knowledge map to reduce the effort for recording. Consequently, the corresponding reuse scenarios (including the retrieval goals) may have to be revised. This is also why the term »knowledge« instead of »information« (as in the previous tasks) is used in this method description.

8.3 Definition of Conceptual Knowledge

To support the maturing of knowledge technically, the informal scenarios need to be formalized. There are two formalization aspects: One aspect concerns the structure of the contents and the behavior of the retrieval system. The other aspect concerns the precise definition of processes ensuring the availability of needed information and high reuse benefits. The latter is subject of Section 8.4, whereas the former is subject of this section.

Task T76: conceptualize

Objective: conceptualize the contents of the experience base

Input: current schema of the experience base (part of the state of the practice); reuse scenarios

Output: changed schema of the experience base (part of the improved state of the practice); feedback indicators; minimal quality requirements; application policies

Decomposition: define concepts (T77, page 221), define nonterminal attributes (T78, page 221), identify reuse information (T79, page 222), define global similarity (T93, page 230), define dependencies (T94, page 230)

Method: The subtasks may be performed sequentially or iteratively. In the latter case, only some of the concepts are defined (task »define concepts«) before their nonterminal concept attributes with their kinds (task »define nonterminal attributes«), terminal concept attributes with their types and local similar-

ities (task »identify reuse information«) as well as the concepts' global similarity (task »define global similarity«) and the dependencies among them (task »define dependencies«) are defined based on the previously defined attributes. In further iterations, more groups of concepts with their subordinate parts are defined.

Task T77: define concepts

Objective: define the concept glossary

Input: retrieval goals

Output: concepts (part of the changed schema of the experience base) in form of a concept glossary (cf. Table 9 on page 177)

Decomposition: –

Method: All kinds of artifacts stated in the retrieval goals are listed as concepts together with a reference to the retrieval goals (using the name of the retrieval goal). For each concept, a short textual definition is given. If the same concept is used by different users, a consensus among the users on the definition must be reached to ensure that all users mean the same thing. If no consensus can be reached, separate concepts must be defined.

In a second step, additional concepts for modeling purposes (abstract concepts) are defined where appropriate. Concepts for modeling purposes are »artificially« created concepts that do not have a counterpart in the real world. They are typically introduced to express commonalities among existing concepts. For example, context information of artifacts (such as characterizations of projects or organizational units) is often expressed as a separate concept. This allows to share a single context description (e.g., that of a project) among many artifacts stored (e.g., all artifacts developed by the particular project). Such a single context description can lower the maintenance effort. For example, if the project characterization needs to be adapted for some reason, only a single instance needs to be changed.

Task T78: define nonterminal attributes

Objective: define semantic relationships between the concepts

Input: concept glossary

Output: nonterminal concept attributes (part of the changed schema of the experience base) in form of concept attribute tables (cf. Table 17 on page 184) and (optionally) as a graphic (cf. Figure 31 on page 184); kinds (part of the changed schema of the experience base) in form of the kind table (cf. Table 19 on page 185)

Decomposition: –

Method: If one or more of the following questions is answered positively for any pair of concepts, a semantic relationship using nonterminal concept attributes should be modeled:

- Will the relationship be used potentially to navigate between the characterizations (e.g., it may be of interest to a user to navigate from a code module to its specification)?
- Will the relationship be used potentially to formulate a query (e.g., all artifacts should be related to the projects they were developed/used in)?
- Is the relationship needed to define assertions or value inferences?

For each relationship the corresponding kind is determined. If the kind is new, it also needs to be defined in the kind table. The complete definitions are inserted into the attribute tables or kind table respectively.

Note 1: Although it is possible to create as many kinds as there are semantic relationships, one should be very careful in doing so. As a rule of thumb, there should be a clear semantic for each kind (e.g., all predefined kinds differ in their interpretation by the retrieval and storage system). The definition of new kinds should only be done if technical support for some new task (e.g., some reporting chore) or a new interpretation, that is not covered by the existing kinds, is needed. Nonterminal concept attributes can be identified alone by their name and the concept they belong to. Therefore, the kind is not needed for identification purposes, but only for interpretation purposes. For interpretations done by humans, good naming of the nonterminal concept attributes usually suffices.

Note 2: In general, the number of nonterminal concept attributes should be kept at a minimum. Defining nonterminal concept attributes simply because the relationships can be documented is not advisable, because with each nonterminal concept attribute added to a concept, the record and maintenance effort for the concept's instances increases. Therefore, only nonterminal concept attributes with a clear benefit (see questions above) should be modeled.

Note 3: During the detailed analysis of the relationships among the concepts, it is possible that commonalities among the concepts are found which (for modeling purposes) should be extracted to a new concept. In this case, the task »define concepts« needs to be performed again.

Task T79: identify reuse information

Objective: define terminal concept attributes and their types

Input: concepts in form of a concept glossary; reuse scenarios

Output: informal description of the global similarity; feedback indicators; terminal concept attributes (part of the changed schema of the experience base) in form of concept attribute tables; types (including value ranges and local similarities; part of the changed schema of the experience base); minimal quality requirements; application policies

Decomposition: determine reusability factors (T80, page 224), classify reusability factors (T83, page 225), determine minimal quality requirements (T84, page 226), determine application boundaries (T85, page 226), define application policies (T86, page 226), acquire distinguishing characteristics (T87, page 227), define terminal attributes (T88, page 227)

Method: For each concept, attributes are defined that are necessary to utilize candidate experience. This is done as follows:

1 All factors that are necessary to identify and evaluate candidate artifacts are determined (»determine reusability factors«).
2 The thus identified reusability factors are divided into two groups during the subtask »classify reusability factors«: Those that can be determined at the time a new artifact is recorded (»record attributes«) and those that can only be determined *after* an artifact has been utilized (»feedback indicators«).
3 Minimal quality requirements for utilizing artifacts of the respective concept are determined in terms of the »record attributes« (»determine minimal quality requirements«).
4 For each »record attribute«, application boundaries are determined (»determine application boundaries«). Application boundaries describe under what conditions (in terms of the »record attribute« values of the query and the characterizations in the experience base) a given artifact can be utilized.
5 During the subtask »define application policies« artifact preferences (e.g., applicable completed documents should be preferred over templates or standard procedures should be preferred over exceptional procedures) are identified and »encoded« as »preference attributes«.
6 Property classes that have not been identified in earlier steps, but are perceived as »typical« characteristics of an artifact class, are defined as attributes during the subtask »acquire distinguishing characteristics«. They are needed to ensure that each instance will have a unique characterization.
7 Finally, the identified attributes (»reusability factors«, »preference attributes«, and »distinguishing characteristics«) are formally documented using REFSENO (»define terminal attributes«).

Task T80: determine reusability factors

Objective: determine all information needed to choose the best suited artifact (if any) to utilize

Input: concepts in form of a concept glossary; reuse scenarios

Output: reusability factors; informal description of global similarity

Decomposition: define meaning of similarity (T81, page 224), identify reusability factors (T82, page 224)

Method: For each concept and in context of the reuse scenarios for which the particular concept (whose reusability attributes are to be determined) is needed, an informal definition of the global similarity is developed (subtask »define meaning of similarity«). Then, all reusability factors needed for the calculation of the global similarity are identified (subtask »identify reusability factors«).

Task T81: define meaning of similarity

Objective: define global similarity informally

Input: concept; reuse scenarios

Output: informal description of global similarity; informal description of reusability

Decomposition: –

Method: First, an informal description of reusability is developed. For example, for a code component, reusability may be defined as »the inverse of the effort needed to adapt the code component«. This means the less effort is needed, the more reusable the code component will be.

In the second step, the informal reusability description is transformed into an informal description of the global similarity. In the example above the global similarity would be a function of the adaptation effort.

Task T82: identify reusability factors

Objective: identify all factors needed to determine the reusability of an artifact

Input: informal description of the global similarity; informal description of the reusability; reuse scenarios

Output: reusability factors

Decomposition: –

Method: It is important to recognize that the global similarity is an a priori estimation of the a posteriori reusability value [Wes95]. Hence, it may have to be calculated using the *variation factors* of the reusability *quality factors*, that is, factors that determine the reusability indirectly (for a precise definition of these terms see the GQM technique, e.g., [BDR96]). For example, the adaptation cost of a code component may be estimated using a difference in functionality (needed functionality versus provided functionality) as well as size and complexity of the module.

The *abstraction sheet* of the GQM technique is a means for acquiring such variation factors. The practicability of this approach has been validated in practice [NT99].

The identified reusability factors (consisting of both quality and variation factors) can be validated using the general characterization scheme presented in [BR91]. According to Basili and Rombach, each artifact must be characterized using attributes from each of the following three classes: About the artifact itself, about its interface, and about its context (see Figure 7 on page 68).

Task T83: classify reusability factors

Objective: divide reusability factors into record attributes and feedback indicators

Input: reusability factors

Output: feedback indicators; record attributes

Decomposition: –

Method: For each reusability factor, it is checked whether a value for it can be determined at the time the corresponding artifact is recorded in the experience base (»record attribute«) or whether the value can be determined only after the artifact has been utilized (»feedback indicator«). The latter are highly context-dependent. Consequently, values for feedback indicators are a kind of experience that must be recorded with the context in which it was gained. Examples are adaptation effort, reliability, etc.

Typically, record attributes are used to compute the similarity between an artifact´s characterization and a query (task »recommend (T12, page 95)«), whereas feedback indicators are considered by the user to examine artifacts further (task »examine (T16, page 97)«). There are two reasons for this:

1 Values for record attributes are available for every recorded artifact, whereas values for feedback indicators are not available before the artifact has been utilized after its initial recording. (The development and the first utilization before the recording of an artifact does not matter in this

regard, because feedback indicators are usually concerned with the modification of an already existing artifact.)

2 Feedback indicators are often text attributes (e.g., a short description of why and how the artifact was modified), that is, their values are often difficult to formalize, because utilization is a creative process that cannot be foreseen in its entirety. However, in general, meaningful similarity functions can only be defined for formalized attributes.

Task T84: determine minimal quality requirements

Objective: determine minimal quality of artifacts to be stored

Input: record attributes; reuse scenarios

Output: minimal quality requirements

Decomposition: –

Method: For each record attribute, the lowest acceptable value is determined. If a value lower than this cutoff value is assigned to the corresponding attribute during the recording of an artifact, the artifact is rejected or evolved until it meets this quality requirement.

Task T85: determine application boundaries

Objective: determine nonmatching values

Input: record attributes; reuse scenarios

Output: application boundary rules

Decomposition: –

Method: For each record attribute, the value range is defined and which values (of an artifact´s characterization with one of a query) do not match at all. For instance, if a project is specified with a team size of 12, a project with 15 people may still be acceptable (although it does not match exactly), but a project with a team size of 100 might not be. For an integer, an acceptable corridor may be defined (e.g., query value ± 25%). For symbol types, nonmatching terms may be defined (e.g., if someone looks for a process model template, product model templates are not appropriate).

Task T86: define application policies

Objective: install technical support for strategic decisions regarding reuse

Input: concepts in form of a concept glossary

Output: application policies; preference attributes

Decomposition: –

Method: For each concept, classes are identified that require different treatments. Then, preference attributes are defined for all identified classes.

For example, assume that it has been decided to reuse requirements documents (i.e., requirements are a subject area). Within an organization, the structure of requirements documents may differ from one application area to another. In this situation, an application policy could be to prefer:
1 Completed requirements documents of systems with very similar functionality *over*
2 Exemplary requirements documents *over*
3 A template

In this case, the preference attribute would have three possible values: »Complete document«, »exemplary document«, and »template«.

What exactly »very similar functionality« means is determined by the application boundary rules. The same is true for »acceptable« application domains. For example, it is not helpful to use a requirements template that has been designed for toaster software if a ground control system for satellites must be specified.

Task T87: acquire distinguishing characteristics

Objective: ensure unique artifacts´ characterizations

Input: concepts in form of a concept glossary

Output: distinguishing characteristics

Decomposition: –

Method: For each concept, experts are asked for property classes they typically associate with the concept and that characterize instances of the concept uniquely.

Note: Distinguishing characteristics help understanding artifacts recommended by the system, because they represent the dimensions in which the users think. In addition, they can be used during »examine (T16, page 97)« if two or more artifacts have the same values for their record attributes.

Task T88: define terminal attributes

Objective: define the terminal concept attributes

Input: reuse scenarios; reusability factors; preference attributes; distinguishing characteristics; application boundary rules

Output: terminal concept attributes (part of the changed schema of the experience base) in form of concept attribute tables (cf. Table 11 on page 180); types (part of the changed schema of the experience base) in form of the type table (cf. Table 13 on page 181) and symbol glossary (cf. Table 15 on page 182)

Decomposition: classify attribute (T89, page 228), define type (T90, page 229)

Method: For each reusability factor, preference attribute, or distinguishing characteristic a row in the appropriate concept attribute table is added (»classify attribute«). The corresponding type for each of these attributes is determined. If the type is new, it also needs to be defined in the type table (»define type«).

Task T89: classify attribute

Objective: add an entry in a concept attribute table for a terminal concept attribute

Input: reuse scenarios; attribute (either a reusability factor, a preference attribute, or a distinguishing characteristic)

Output: terminal concept attribute (part of the changed schema of the experience base)

Decomposition: –

Method: First it is determined whether the attribute describes the artifact itself, its interface, or its context (cf. Figure 7 on page 68). The attribute is added in a new row in the concept attribute table of the corresponding concept (cf. Table 11 on page 180). Then, all other columns except for the »type«, »value inference« and »to infer« columns are filled out.

Of particular importance is the column »scenario«. Here, the rationale for the attribute is documented. If there is no entry in this table cell it means that the attribute is not needed.

Note: Often, empty table cells in the »scenario« column result from reusing a list of attributes from schemas. In this case, it has to be analyzed whether the attribute is really superfluous or whether an important reuse scenario has been forgotten. In the later case, it will become necessary to go back to the task »identify scenarios (T70, page 217)«.

Task T90: define type

Objective: determine the type of an attribute and create a new type entry if necessary

Input: reuse scenarios; terminal concept attribute; application boundary rules

Output: type

Decomposition: define value range (T91, page 229), define local similarity (T92, page 229)

Method: If an appropriate type for the attribute is already defined, the type is entered in the »type« column of the attribute row in the concept attribute table. Otherwise a new type name is created, entered in the »type« column, and defined in the type table by defining the attribute´s value range (»define value range«) and the meaning of similarity regarding the attribute (»define local similarity«).

Task T91: define value range

Objective: define possible values for an attribute

Input: terminal concept attribute; application boundary rules

Output: value range (part of type)

Decomposition: –

Method: The possible values are derived from the application boundary rules and entered in the »value range« column of the type table (cf. Table 13 on page 181). The unit of measure is also entered. In case of a symbol type, the meaning of each possible value is defined separately in the symbol glossary (cf. Table 15 on page 182).

Note: Through the definition of the values in the symbol glossary, the terms become part of the official corporate-wide vocabulary. Care must be taken in defining these terms precisely and unambiguously. Also, someone in the organization must be responsible for maintaining the type (in case the experience base is restructured at a later point in time). A more detailed process for this aspect can be found in [Leh99].

Task T92: define local similarity

Objective: define meaning of similarity regarding a single attribute formally

Input: reuse scenarios; terminal concept attribute; value range (part of type); application boundary rules

Output: local similarity function (part of type)

Decomposition: –

Method: Based on the reuse scenarios, the available knowledge is classified according to the dimensions presented in Section 2.2. In a second step, needed properties of the similarity function are defined (first informally, then mathematically). Finally, the similarity function is validated. This process is detailed in [TA00b].

Task T93: define global similarity

Objective: define meaning of similarity regarding a kind of artifact formally

Input: informal description of global similarity; terminal and nonterminal concept attributes in form of concept attribute tables

Output: global similarity (part of the changed schema for the experience base)

Decomposition: –

Method: Using the defined attributes, the informal description of the global similarity is expressed formally. This will allow the retrieval system to calculate the similarity automatically (task »calculate similarity (T13, page 95)«).

Task T94: define dependencies

Objective: define assertions and value inferences formally

Input: terminal and nonterminal concept attributes in form of concept attribute tables

Output: assertions (part of the changed schema for the experience base); value inferences (part of the schema for the experience base)

Decomposition: –

Method: Conditions that must always be true are expressed using the defined attributes. They are defined as assertions.

Also during this task, automatically computable attributes are identified. Their computation is defined via value inferences.

Note: The definition of assertions and value inferences may require additional nonterminal concept attributes to be defined. In this case, the task »define nonterminal attributes« needs to be performed again.

8.4 Definition of Methods

While the formal conceptual knowledge as defined during »conceptualize« defines the retrieval precisely through its semantics (Chapter 6), it does not define when and how the characterization knowledge (for a definition see Table 1 on page 45) is recorded. Therefore, the »record« task needs to be defined precisely, that is, for each attribute it has to be defined when and how its value is recorded by whom. This is the objective of this section.

Task T95: define ›record‹

Objective: define precisely who has to record which attribute values when and how

Input: record scenarios consisting of potential knowledge sources and informal descriptions of knowledge acquisition; concepts in form of the concept glossary (cf. Table 9 on page 177); nonterminal and terminal concept attributes in form of concept attribute tables (cf. Table 11 on page 180 and Table 17 on page 184); minimal quality requirements; reuse scenarios

Output: knowledge collection plan; »record« methods; knowledge source adaptation plan

Decomposition: define knowledge collection (T96, page 232), define ›split‹ (T103, page 236), define ›analyze quality‹ (T104, page 236), define ›analyze reuse potential‹ (T105, page 237), define ›inform‹ (T106, page 237)

Method: For each kind of artifact to be stored, the record tasks are defined sequentially.

Note: The task »collect (T34, page 108)« plays a special role in this context. Because it is possible that more than one kind of artifact is collected at the same event, the collection events are not artifact-specific. However, at each of these collection events, each kind of artifact may be treated differently. Moreover, additional knowledge about the artifact may have to be acquired when a collected artifact is stored (task »characterize initially (T38, page 110)«). Therefore, the task »define knowledge collection« has three outputs: The knowledge collection plan defining who collects which artifacts when, a set of »collect« methods defining how the artifacts are collected (and, if necessary, how they are developed), and a set of »characterize initially« methods defining how the additional knowledge about the artifacts is acquired. A »collect« method may involve making tacit knowledge explicit, for example, by means of holding an interview, workshop, or feedback session.

Task T96: define knowledge collection

Objective: define precisely who has to record which artifacts when and how

Input: record scenarios consisting of potential knowledge sources and informal descriptions of knowledge acquisition; concepts in form of the concept glossary (cf. Table 9 on page 177); nonterminal and terminal concept attributes in form of concept attribute tables (cf. Table 11 on page 180 and Table 17 on page 184); minimal quality requirements

Output: knowledge collection plan; »collect« methods; knowledge source adaptation plan

Decomposition: survey knowledge sources (T97, page 232), identify overlapping contents (T98, page 233), select knowledge sources (T99, page 233), plan adaptation (T100, page 234), define knowledge collection plan (T101, page 235), define ›characterize initially‹ (T102, page 235)

Method: The potential knowledge sources identified during »identify knowledge sources (T74, page 219)« are further examined as to what knowledge they can actually supply (»survey knowledge sources«). Based on this information, overlapping contents of the knowledge sources are identified (»identify overlapping contents«). The results of both analyses is used to select the most appropriate knowledge sources (»select knowledge sources«). Consequently, some of the knowledge sources may need to be adapted to the new needs. How this is done is defined during »plan adaptation«. Finally, the envisioned knowledge acquisition is documented during »define knowledge collection plan« and »define ›characterize initially‹«.

Task T97: survey knowledge sources

Objective: document contents of knowledge sources

Input: potential knowledge sources (part of record scenarios); concepts in form of the concept glossary (cf. Table 9 on page 177); nonterminal and terminal concept attributes in form of concept attribute tables (cf. Table 11 on page 180 and Table 17 on page 184)

Output: contents of knowledge sources

Decomposition: –

Method: The contents to be documented are limited to those necessary to satisfy the information needs as defined by the concepts and their attributes. The actual method employed depends on the type of the knowledge source being surveyed:

- **Documents.** Exemplary documents are scanned.
- **Databases.** The database schema is investigated.

- **Human experts.** In case a particular person is the knowledge source, the person can be interviewed. If the knowledge source is described by a role (e.g., project manager), the role definition can be taken as a basis. In addition, representatives of the role can be interviewed.

Task T98: identify overlapping contents

Objective: identify contents that can be supplied by more than one knowledge source

Input: contents of knowledge sources

Output: overlaps

Decomposition: –

Method: The contents of all knowledge sources are compared. Contents that can be supplied by more than one knowledge source are documented together with the alternative knowledge sources.

Task T99: select knowledge sources

Objective: select the knowledge sources to be used

Input: record scenarios consisting of potential knowledge sources and informal descriptions of knowledge acquisition; contents of knowledge sources; overlaps; concepts in form of the concept glossary (cf. Table 9 on page 177); nonterminal and terminal concept attributes in form of concept attribute tables (cf. Table 11 on page 180 and Table 17 on page 184)

Output: actual knowledge sources (including the knowledge to be acquired from them)

Decomposition: –

Method: The informal descriptions of knowledge acquisition are analyzed:

- Is it described how all the knowledge needed (= concepts and their attributes) can be acquired? (*Completeness of scenarios*)
  If not, the scenarios need to be extended.
- Are the assumptions about the contents of the knowledge sources correct, that is, do the knowledge sources supply the assumed knowledge (using the documented contents of the knowledge sources)? (*Correctness of scenarios*)
  If not, alternative knowledge sources have to be sought.
- Are the proposed knowledge sources optimal? (*Efficiency of scenarios*)
  There are several optimization criteria possible including the following:
  - Prefer nonhuman knowledge sources over human knowledge sources to relieve experts of superfluous tasks. This will limit knowledge acqui-

sition from humans to knowledge that has not been documented yet and cannot be derived automatically from available, documented knowledge. To limit the number of times a human expert needs to be interviewed, other knowledge sources (documents and databases) may be adapted to include more knowledge in the future (see »plan adaptation« below).
– Use the smallest number of knowledge sources possible. For instance, if a particular knowledge item can be supplied either by a database or a document, the document may be preferred in case the document contains needed knowledge that cannot be found elsewhere.

If not, the informal descriptions of knowledge acquisition should be optimized.

During the analysis, needed knowledge sources are listed. These are the actual knowledge sources.

Task T100: plan adaptation

Objective: tailor knowledge sources to needs of knowledge acquisition

Input: actual knowledge sources (including the knowledge to be acquired from them); contents of knowledge sources; minimal quality requirements

Output: knowledge source adaptation plan

Decomposition: –

Method: Nonhuman knowledge sources can be adapted to better suit the reuse needs. For example:

- Documents can be extended by including a new item in the structure of the document.
- Databases can be extended by extending their schema and defining how the new data will be entered.

In addition, technical solutions for integrating already existing knowledge sources (such as databases) with the experience base are part of the knowledge source adaptation plan. For instance, this might be an agent which queries a database on a regular basis and stores this data in the experience base.

For acquiring knowledge from human knowledge sources, questionnaires and (structured) interviews are the main knowledge acquisition instruments [BDR96]. To support these kinds of knowledge acquisition, questionnaires need to be designed. These questionnaires as well as instructions for filling them out are part of the knowledge source adaptation plan. Questionnaires and the fill-out instructions should ensure that the minimal quality requirements are met.

Task T101: define knowledge collection plan

Objective: document when which artifact has to be collected how by whom

Input: actual knowledge sources (including knowledge to be acquired from them); knowledge source adaptation plan; informal descriptions of knowledge acquisition

Output: knowledge collection plan; »collect« methods

Decomposition: –

Method: First, the artifacts corresponding to the needed knowledge are identified. They depend on the employed knowledge source:

- **Documents.** The documents themselves are the artifacts.
- **Databases.** Usually queries with subsequent analysis must be performed. The exact procedure is part of the informal descriptions of knowledge acquisition. The knowledge is documented in an analysis report (= artifact).
- **Human experts.** The knowledge is documented in a questionnaire (= artifact) or as minutes of an interview (= artifact).

For each artifact a row is entered in the knowledge collection plan. For each artifact, it is described when (event and point in time) the artifact is developed/collected, by whom it is developed and collected, and how it is developed (knowledge acquisition instrument).

For example, lessons learned (= artifacts) can be collected at the end (event) of a project (point in time). They can be acquired in a postmortem meeting (knowledge acquisition instrument) where all project team members (= artifact developers) participate. An experience engineer (= artifact collector) ensures that the postmortem meeting takes place and collects the lessons learned after the meeting. (The way lessons learned are structured is not documented in the knowledge collection plan, but in the knowledge source adaptation plan.)

Task T102: define ›characterize initially‹

Objective: document how to characterize artifacts initially

Input: actual knowledge sources (including knowledge to be acquired from them); knowledge source adaptation plan; concepts in form of the concept glossary (cf. Table 9 on page 177); nonterminal and terminal concept attributes in form of concept attribute tables (cf. Table 11 on page 180 and Table 17 on page 184); informal descriptions of knowledge acquisition

Output: »characterize initially« methods

Decomposition: –

Method: Based on the informal descriptions of knowledge acquisition, it is documented how the attributes are filled in by either using the artifacts to be collected (documented in the knowledge source adaptation plan) or by acquiring the knowledge from the knowledge sources directly (i.e., databases or human experts).

Task T103: define ›split‹

Objective: document how artifacts are split into reusable parts

Input: informal description of knowledge acquisition (part of record scenarios); concepts in form of the concept glossary (cf. Table 9 on page 177); knowledge source adaptation plan

Output: »split« methods

Decomposition: –

Method: Based on the informal descriptions of knowledge acquisition, the knowledge source adaptation plan, and the concepts of the experience base schema, a procedure is developed that will split the appropriate artifact into its reusable parts, so that each part can be characterized by a single instance.

For example, during a postmortem meeting, minutes are recorded. These minutes can be split into problem-solution statements, improvement suggestions, and observations. Problem-solution statements can be split into a problem and (possibly) several solution parts.

Task T104: define ›analyze quality‹

Objective: document how to analyze the parts´ quality

Input: concepts in form of the concept glossary (cf. Table 9 on page 177); nonterminal and terminal concept attributes in form of concept attribute tables (cf. Table 11 on page 180 and Table 17 on page 184); minimal quality requirements

Output: »analyze quality« methods

Decomposition: –

Method: For each reusable part (characterized by an instance), it is documented how its quality can be determined. The relevant quality factors are given by the minimal quality requirements.

Task T105: define ›analyze reuse potential‹

Objective: document how to analyze the reuse potential of an artifact or its reusable parts

Input: concepts in form of the concept glossary (cf. Table 9 on page 177); non-terminal and terminal concept attributes in form of concept attribute tables (cf. Table 11 on page 180 and Table 17 on page 184); minimal quality requirements; reuse scenarios

Output: »analyze reuse potential« methods

Decomposition: –

Method: Based on the available information (concepts and their attributes), its minimal quality and the intended reuse scenarios, guidelines for analyzing the reuse potential are documented.

Note: Guidelines may be based on access statistics (cf. requirement »data collection for evaluation (R15, page 50)« and task »analyze artifact (T53, page 118)«) and thus evolve over time.

Task T106: define ›inform‹

Objective: document whom to inform when new knowledge is recorded

Input: concepts in form of the concept glossary (cf. Table 9 on page 177); non-terminal and terminal concept attributes in form of concept attribute tables (cf. Table 11 on page 180 and Table 17 on page 184); reuse scenarios

Output: »inform« methods

Decomposition: –

Method: Based on the intended reuse scenarios, checklists are created that document who should be informed in case a new artifact is recorded or updated.

8.5 Infusion

Before the new methods can be employed, they have to be infused into the organization. Typically this means that organizational changes need to be made. These have to be prepared carefully. This section points out some important aspects for the implementation.

Task T107: infuse SEEMS

Objective: infuse the software engineering experience management system (SEEMS) into the organization

Input: knowledge source adaptation plan; minimal quality requirements; application policies; knowledge collection plan; »record« methods; reuse scenarios

Output: training material; change plan (including the knowledge source adaptation plan)

Decomposition: develop technical infrastructure (T108, page 238), prepare training material (T109, page 238), plan change (T110, page 239)

Method: The subtasks are performed sequentially.

Task T108: develop technical infrastructure

Objective: instantiate the architecture of the technical infrastructure

Input: knowledge source adaptation plan; »record« methods

Output: technical infrastructure

Decomposition: –

Method: The architecture of the technical infrastructure of an experience base (see Section 7.1) is instantiated according to the needs of the organization. These needs are specified through the knowledge source adaptation plan and the »record« methods.

Note: If the knowledge source adaptation plan includes the implementation of agents that automatically transfer data sets from existing databases into the experience base, the agents are also implemented as part of this task.

Task T109: prepare training material

Objective: develop material to train all users and the experience factory personnel

Input: minimal quality requirements; application policies; knowledge collection plan; »record« methods; reuse scenarios; technical infrastructure

Output: training material

Decomposition: –

Method: The training material should have two parts:

1 **Motivation for reuse.** This part is derived from the reuse scenarios and the application policies. Screenshots and demos should make the users understand the technical infrastructure of the experience base and its benefits.
2 **How knowledge is acquired.** This part is derived from the knowledge collection plan, the minimal requirements (tells the knowledge suppliers what is expected of them), and the »record« methods.

Note: It may be useful to define the training in building blocks. One building block could be the general overview and motivation for the technical infrastructure. More detailed building blocks (divided according to the roles involved) could train how to record knowledge. While the general building block should be attended by everybody who is expected to use the experience base, the more detailed building blocks need to be attended only by those that enact a role the respective building block addresses.

Task T110: plan change

Objective: plan how to introduce the new technical infrastructure and methods into the organization

Input: training material; knowledge source adaptation plan

Output: change plan

Decomposition: –

Method: A project plan for introducing the new technical infrastructure and methods is developed. Aspects to consider include:

- As far as the changes documented in the knowledge source adaptation plan have not been taken care of by the task »develop technical infrastructure«, their implementation should be scheduled now.
- The training material should be validated. The recording part should be trialed as documented. This will sample the identified knowledge sources and validate whether the experience base can be populated using the developed experience base schema.
- Often, existing knowledge in the organization can be used to fill the experience base initially. Therefore, the initial recording should be scheduled. Note that the initial recording may require different methods than those defined for capturing knowledge from running projects. Hence, deviations from the standard »record« task need to be defined and documented.
- A small group of users should be trained to trial the initially filled experience base (= pilot experience base). Both the training and the trials need to be scheduled. Bartsch-Spörl gives some guidelines on selecting pilot users [BS99, page 5]:

  - The users of the pilot experience base more or less decide about the future of the experience base. »Therefore, they should be selected carefully, very well informed, involved, and prepared.«
  - »Try to select persons who are good ›key users‹ within their group and who are able to share their knowledge and experience with their colleagues.« Experts are usually not a good choice, because they mostly do not really need the experience stored. Instead, look for people who are open and want help.
- Based on the feedback of the pilot users, an iteration of »structure« may become necessary. This iteration needs to be scheduled.
- The roll-out involves training more people. Therefore, the additional training sessions need to be scheduled.

8.6 Related Work

The development of information systems in general and the development of conceptual information as part of it is subject of past and ongoing research. Related research can be traced back to those research areas enumerated in Chapter 5 (library and information science, hypertext systems, database management systems, and knowledge-based systems).

library and information science

In library and information science, term vocabularies need to be developed for systems operating with controlled term vocabularies. As already pointed out in Section 5.1, the development of adequate classification schemas is difficult not only for technical reasons, but also because different people have diverse views on the subject matter. In practice, first the type of term vocabulary is selected depending on the degree of consensus among the (future) users and the amount of effort one is willing to invest to reach a certain degree of consensus. For instance, if there is no consensus at all and no effort shall be spent, an uncontrolled term vocabulary can be used. If the users agree on the descriptors, but not on the relationships between them, controlled terms can be used. If there is a limited set of views, faceted classification can be used. And finally, enumerated classification can be used if there is a strong consensus among the users. In addition, thesauri may be used to connect similar descriptors. For reaching a consensus, knowledge acquisition techniques, such as those used for the development of knowledge-based systems (see below), can be employed (mostly various interview techniques). Since all types of term vocabularies are supported by REFSENO´s type hierarchy, the strategy described above can be used to define symbol types (task »define type«).

hypertext

In contrast, development guidelines and methodologies for the design of hypertext systems (e.g., [BMY95, BI95, GMP95, THH95]) are typically not applicable for query-based systems for two main reasons. On one hand, hypertext systems design has to consider not only conceptual knowledge (i.e., node types and relationships between them), but also (and foremost) screen design issues, naviga-

tion patterns, etc. [ISB95], that is, aesthetic and cognitive aspects [NN95]. On the other hand, a query interface is not designed as part of hypertext design methods.

DBMS

The development of query interfaces corresponds to the definition of views in database management systems (DBMS); however, DBMS development methods are domain-independent. This means that these methods describe what needs to be defined in which order (including guidelines), but not how to acquire the relevant concepts and attributes in the first place [Dit97, GR95].

knowledge-based systems

The identification of knowledge and the development of methods for solving problems requiring this knowledge has a long tradition in knowledge-based systems. An overview is given in [dH98]. One particular topic addressed is the »knowledge acquisition bottleneck«, that is, knowledge extraction from an expert. A recent overview of available knowledge acquisition techniques can be found in [Lio98].

Another attempt to tackle the knowledge acquisition bottleneck is to share and reuse knowledge. This is the primary objective of *ontologies* [CJB99]. Ontologies are explicit representations of conceptualizations [Gru95]. As such the term *ontology* can be seen as a synonym for the conceptual information of an experience base. In fact, ontologies have been used for knowledge management systems [BFG98]. There have been some approaches on developing ontologies, but their descriptions are often limited to a set of guidelines [Gru95, UG96] or an enumeration of necessary development activities [Gua97, Sku95] without detailing the activities to a level that is needed for their application in practice. The most serious attempt for the development of ontologies to date is METHONTOLOGY [BFGPGP98, GP98]. The tabular notation of METHONTOLOGY for documenting ontologies was the basis for the notation for REFSENO (see Chapter 6), although extensive tailoring to the needs of the software engineering domain (especially with respect to defining similarity) was necessary. An overview of methodologies for building ontologies can be found in [Lóp99].

While ontologies focus on (sharing and) reusing conceptual knowledge, problem-solving methods codify reusable procedural knowledge (for a discussion of the knowledge types refer to Section 2.1.4). One of the best known attempts at building up a generic library of problem-solving methods and how to utilize such a library is the CommonKADS approach [SWdH+94]. Generally, ontologies and problem-solving methods are seen as complementary parts for building knowledge-based systems [Stu99]. A recent overview of ontologies and problem-solving methods emphasizes this [GPB99]. This integration idea is also fundamental for this thesis, although the tasks described in Chapter 4 are in part carried out by humans, whereas problem-solving methods are typically associated with automatable algorithms.

Other approaches specialize the methodologies for particular kinds of knowledge-based systems. For instance, Kitano and Shimazu present a life cycle model for case-based reasoning systems [KS96]. The most complete methodology for building and maintaining case-based reasoning systems has been the result of INRECA II, a project funded by the European Union [BA98, BBG+98, BBG+99]. Stolpmann and Wess emphasize that technology design is not enough [SW99]. Nontechnical aspects have to be considered as well.

Along this line, GOODSEE is a specialization of case-based reasoning systems for the domain of software engineering experience reuse. Consequently, GOODSEE is in parts more detailed, especially regarding the identification of attributes and the definition of the recording tasks.

In addition to the methodologies described above, other research areas have influenced GOODSEE. For instance, consideration of knowledge management approaches led to the tasks described in Section 8.1 and Section 8.2. Approaches from knowledge management emphasize the importance of non-technical issues such as social and organizational aspects [ASR99, DP98, BSLC96, Wii95 (Chapter 10)]. This has also been recognized by the domain analysis community [SCK+96, WD99]. Although domain analysis approaches also attempt to build up and structure a repository, they are restricted to software development documents [Ara94]. Consequently, their spectrum is too narrow (e.g., they do not consider empirical data or process models to be stored) for the purpose of structuring an entire experience base. Nevertheless, such methods may be used for structuring parts of an experience base as already pointed out in Section 4.5.2.

8.7 Summary

The previous section has shown that existing approaches for developing schemas are either too broad or too narrow. If the application domain is too broad, development approaches are either not detailed enough (i.e., they provide only a set of guidelines and/or enumerate development activities without describing how to perform the activities) or they are too abstract in the sense that the terminology used by the methodology cannot be easily understood by experts from a particular subdomain, for example, software engineers for software engineering experience bases. In both cases, substantial effort is necessary to tailor the methodology to the needs of the particular domain at hand. If the application domain is too narrow, the methodology may be applicable, but will provide only a partial solution (e.g., domain analysis methodologies for software development documents) or will not be applicable at all.

What is needed then, is an approach that is both general enough to cover the whole domain of software engineering experience reuse and specific enough to be able to give meaningful, detailed method descriptions. In this chapter, such

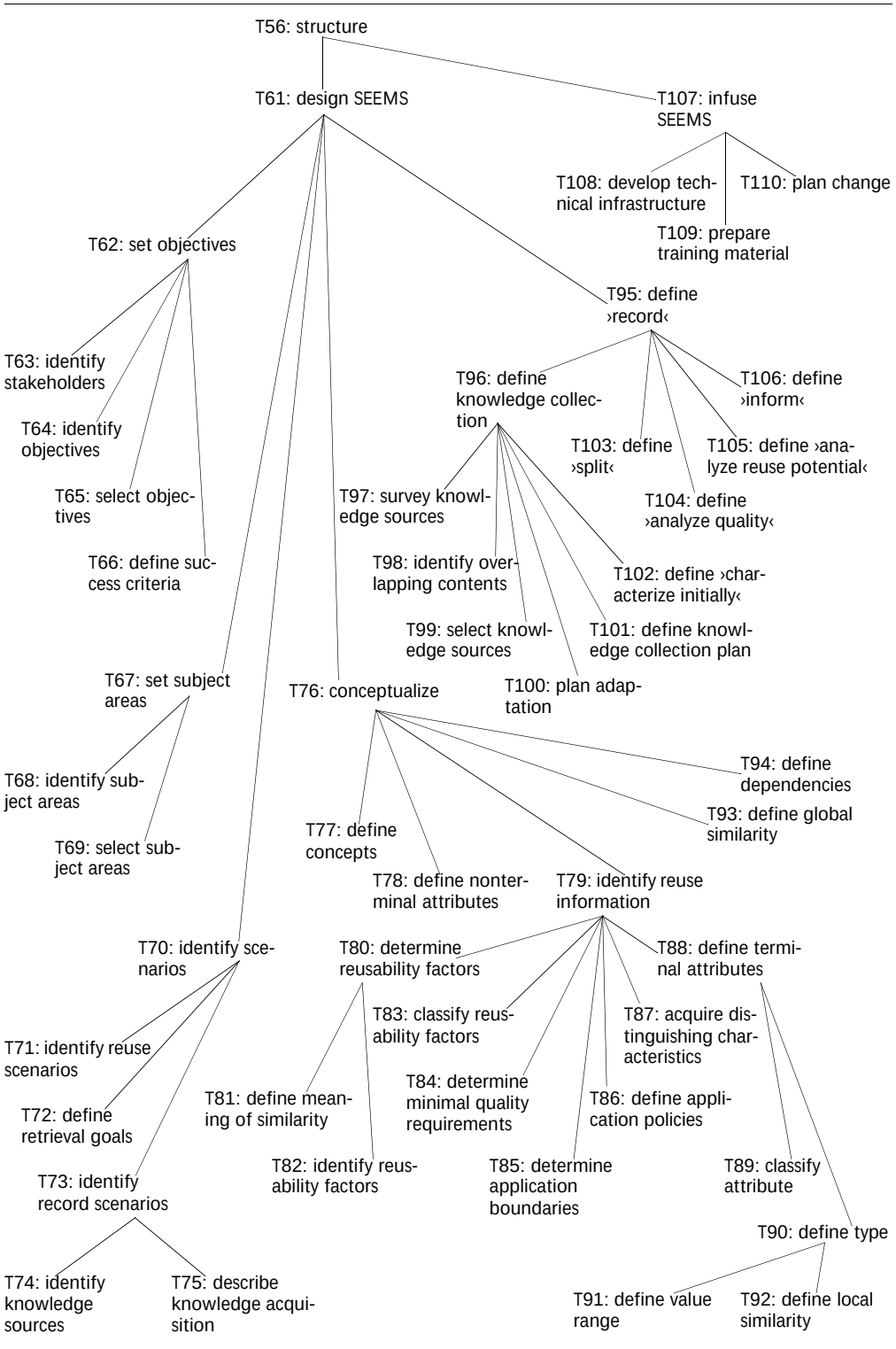
an approach called GOODSEE (goal-oriented ontology development for software experience) has been described (for an overview see Figure 35). GOODSEE is specialized towards the tasks of populating and utilizing an experience base (see Chapter 4), the representation formalism REFSENO (see Chapter 6), and thus to the domain of software engineering experience reuse.

GOODSEE pays particular attention to social and organizational issues by including activities for identifying stakeholders of the experience factory, the stakeholders' interests, scenarios explicating the needs of the users as well as the practical constraints on available knowledge, and methods for populating the experience base. Thus, GOODSEE ensures both the goal-oriented definition of the experience base contents and the definition of processes that smoothly integrate with already established practices at a given organization.

As GOODSEE is applied in practice, it is expected that the methods will be complemented by specific checklists (e.g., for the tasks »select objectives« and »select subject areas«) and predefined conceptualizations for specific kinds of artifacts (e.g., lessons learned, GQM plans, process models, etc.). In the long run, some of the methods may be further decomposed (e.g., the task »select objectives« by subtasks describing how to perform certain analysis tasks such as how to determine business feasibility).

Although a good initial schema for the experience base is vital for its success, it can only be seen as a first step towards a software engineering experience management system that satisfies its users over a longer period of time. The artifacts stored in the experience base are, of course, another important factor for the success of the experience base. The schema, the contents, the methods for populating and utilizing as well as the technical infrastructure of the experience base may have to be improved as the users' needs become clearer or as the environment changes. The next chapter addresses this topic.

Figure 35: Structure tasks



9 Evolving Software Engineering Schemas

In the fast-paced world of changing business needs and technological advances, well-established domains in the computer industry are an increasingly rare commodity. Static and labor-intensive domain analysis methods are ill-suited for these dynamic and evolving domains, making it difficult to have a reasonably complete domain model that reflects the current state of affairs.

[HLR95, p. 95]

Although the initial development of an experience base schema is already a challenging and sometimes time-consuming task (Chapter 8), it cannot be expected that a schema does not need to be evolved [Hen96]. Reasons for evolving a schema include a better understanding of the domain [KS96], a better understanding of the users´ needs, the inclusion of new kinds of artifacts to be stored in the experience base, and a change of the environment an organization operates in (e.g., law changes).

Thus, the improvement of an experience base (both its schema and its characterization information) through user feedback is of vital importance if an experience base shall be a continuous source of benefit to its users. The benefits of an experience base are ultimately determined by the usefulness of the experience base as perceived by its users. Therefore, an improvement of an experience base 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 experience base contents, the quality of the schema used to store knowledge, and the quality of the technical infrastructure employed). Hence, it is difficult to identify good starting points for improvement.

This chapter presents the goal-oriented method OMI (Organizational Memory Improvement) for improving an experience base 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. During each task of the general usage model of OMI, data are recorded to pinpoint improvement potential for increasing the perceived usefulness. If improvement potential is identified, the user is asked for specific improvement suggestions.

Since OMI bases on a continuous, »on-the-fly« diagnosis of the usefulness of the experience base, this method can help to adapt the experience base to the needs of the users – even if the environment, for which the experience base was designed, changes. Thereby the OMI method allows to overcome the general problem that users are often not available during the development of the experience base. The basic idea underlying OMI is to start with an »intelligent guess« (focus of Chapter 8) and to improve the experience base incrementally by systematically collecting and analyzing focused user feedback (focus of this chapter).

The chapter is structured as follows. Based on the description of the maturing tasks (Chapter 4) and REFSENO (Chapter 6), Section 9.1 and Section 9.2 present OMI's cause-effect model for usefulness as perceived by the user and the general usage model, respectively. Section 9.3 describes OMI's diagnostic process for improving an experience base, whereas Section 9.4 gives some information about its validation. Finally OMI is compared to related research work and a summary is given.

9.1 Perceived Usefulness

The success of an experience base 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, EMT98, FT94, Lim96, HS93, Nic98]. Other evaluation criteria, most importantly *recall* and *precision*, come from library and information science [SM83].

However, a measurement program using the GQM technique, where experts participate in the definition of a measurement program [vSB99], showed that the usefulness, as perceived by the user of the experience base, is the most important measure for the success of an experience base [NT99, NAT99]. This is not surprising since an experience base 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«), Cooper [Coo97] (there called »personal utility«), and Müller [Mül99] (there called »interestingness«).

Therefore, based on the perceived usefulness *before* and *after* a change, it can be judged whether the change is an actual improvement. 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, the measurement procedure needs to be simplified. To do so, we recall the meaning of similarity. Ideally, the similarity between an artifact in an experience

base and the needed artifact (specified by the query) is an a priori approximation of the a-posteriori usefulness as experienced by the user [Wes95, ANT99a]. If the retrieval system returns the instances i_1, …, 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, …, 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)[1], a retrieval system 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 it never returns an instance $i \in \{i_1, \ldots, i_m\}$ that is not useful. The more often the retrieval system returns such an instance, the less useful it will be. Using this definition, the user can simply mark all instances i_1, …, i_n as either useful or not useful. Based on this data, the usefulness of a retrieval system can be computed (e.g., relative to the number of queries issued by users). Another important aspect of usefulness is that a retrieval system returns useful instances at all, i.e., m should be greater than 0.

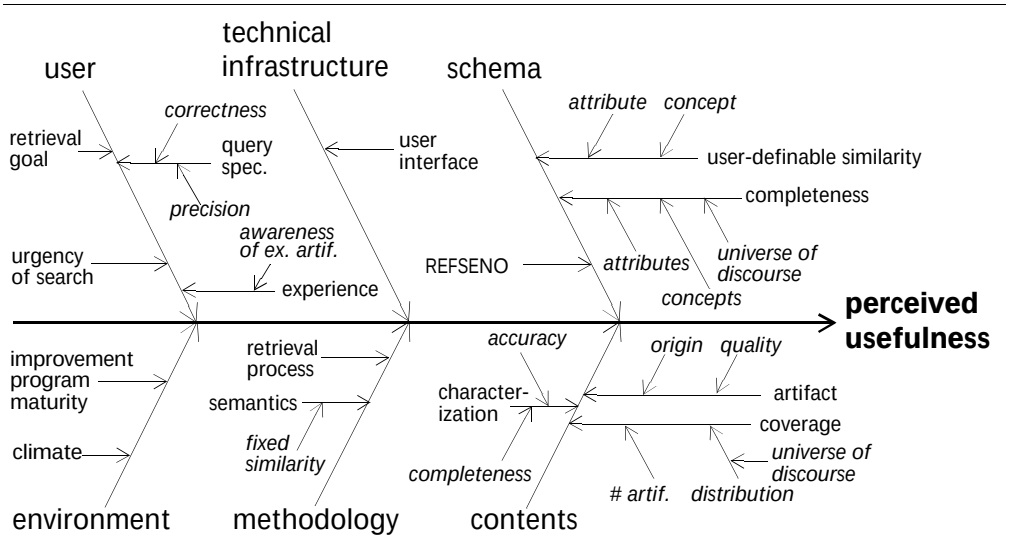
Unfortunately, there is no single parameter with which the usefulness of the experience base can be changed. Rather, there is a large number of variation factors. Figure 36 shows the variation factors the author of this dissertation has identified. However, the author does not claim that the list is exhaustive. In the following, the main branches of the cause-effect model will be detailed.

User

The intentions and abilities of the user will greatly influence the usefulness of the artifacts returned by the retrieval system. For instance, if a user does not specify his needs correctly, the retrieval system might return artifacts adequate for the needed artifact as specified by the query, but not for the actual artifact 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 experience base can be found which matches the query). An overspecification is only possible in a retrieval system that is only capable of retrieving exact matches or a retrieval system computing a cutoff-value (i.e., threshold) for similarities. For a similarity-based retrieval system that returns a fixed number of instances, an overspecification is practically impossible.

1 See [GAB99] for an example where similarity thresholds are used.

Figure 36: OMI's cause-effect model



Also of importance for the usefulness of an artifact is the purpose for which the user retrieves the artifact (denoted as »retrieval goal« in Figure 36) [Bur98]. 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 useless 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 experience factory operates may be mature. In this case the experience base 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, BCR94a, BT98a]. Also, the climate in an organization influences how willingly people are to share their experience with their colleagues over an experience base. 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].

Technical Infrastructure

For the user, the technical infrastructure of an experience base is mainly characterized through its behavior and its user interface. The behavior of the retrieval system, which is the focus of OMI, 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, NT99, NAT99]. And, if a system is not used, it is perceived as not useful and cannot yield any benefit.

Methodology

It is the methodology that is supported by the technical infrastructure. If the underlying methodology (e.g., the retrieval process supported by the technical infrastructure or the semantics of REFSENO's constructs) is not optimal, it cannot be expected that the experience base is perceived as useful as it could be.

Schema

Clearly, the behavior of a technical infrastructure of an experience base is not solely determined by its implementation, but also by the contents and their organization of the experience base. As the schema determines what and how artifacts are stored in the experience base, 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 experience base *can* cover. In contrast to this universe stands the universe of discourse the experience base *shall* cover. The concepts and attributes may not cover all kinds of artifacts 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 constructs of REFSENO.

Contents

Even if the schema is defined optimally regarding the usefulness, the characterization information of the experience base may cause the system to fail. Just as with databases, the information stored in an experience base must be accurate and complete [NT98b]. 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 experience base shall cover is not covered simply by defining the universe in terms of artifact kinds to be stored. The actual artifacts have still to be stored! What the experience base actually *does* cover is influenced by two characteristics: The number of artifacts in the experience base

and their distribution. The more artifacts are stored, the *more likely* it is that the system will return useful artifacts. 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 artifacts offered by the experience base 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 experience base returns artifacts on software inspections whose author is known to the user to be an expert in software testing.[1] The user may now value the usefulness of this particular artifact 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 artifacts regarding software testing.

9.2 Usage Model

As can be seen from the variation factors presented in the previous section, the usage of an experience base 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, first the »ideal« (i.e., sequential) usage scenario for an experience base is described. Later, it is described under which circumstances the user might go back and repeat some of the tasks.

Figure 37 shows the sequential usage model. The usage model is derived from performing the retrieval tasks explained in Section 4.3.1 sequentially.

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 the usage model that the user will not specify 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

1 In practice, software inspections and software testing are often compared regarding their effectiveness and efficiency, because both techniques share the same objective of finding defects.

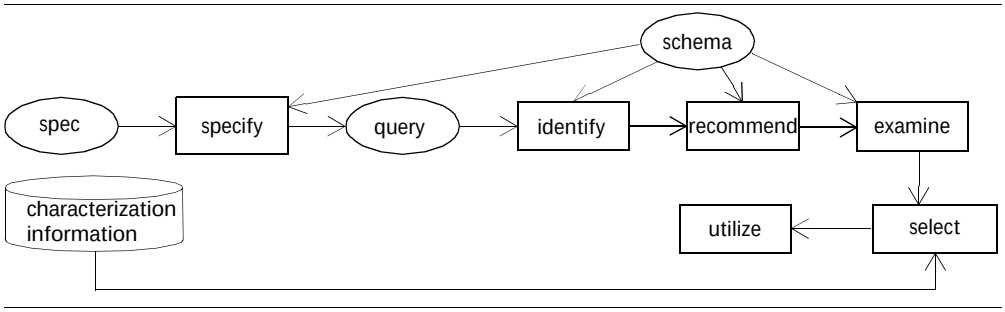


Figure 37: The »ideal«, sequential usage model of an experience base

can do better, he will go back to the task »specify« 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 view the artifacts using some editor to make the decision on whether to utilize them, or not (e.g., Is the artifact well documented? Can it be easily understood?). However, aim of the characterization information is to limit the number of artifacts that have to be inspected in this way as well as to reduce the effort needed to inspect 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 viewing 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* utilizing the artifact that it was not the best candidate for the purpose. If the task could not be performed by utilizing a retrieved artifact, the user may want to reissue his (possibly refined) query to find more suitable artifacts.

9.3 Diagnosing for Improvement

During each task 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 9.1. Based on the diagnosis, changes can be suggested that (hopefully) will lead to improvements of the experience base. As we have already seen in Section 9.1, not all of the variation factors can be changed by purely technical means. Although some of these variation factors (such as the environment in which the experience base is used) influence the usefulness of an experience base substantially, the diagnosis and change of these factors is beyond the scope of this dissertation (see [FF96] for a discussion on the improvement of the more organizational aspects of reuse).

To ease the understanding of situations that can be changed by tailoring the experience base, Figure 38 shows the usage model extended by change tasks to the experience base. In the following, each task of the usage model is examined in detail.

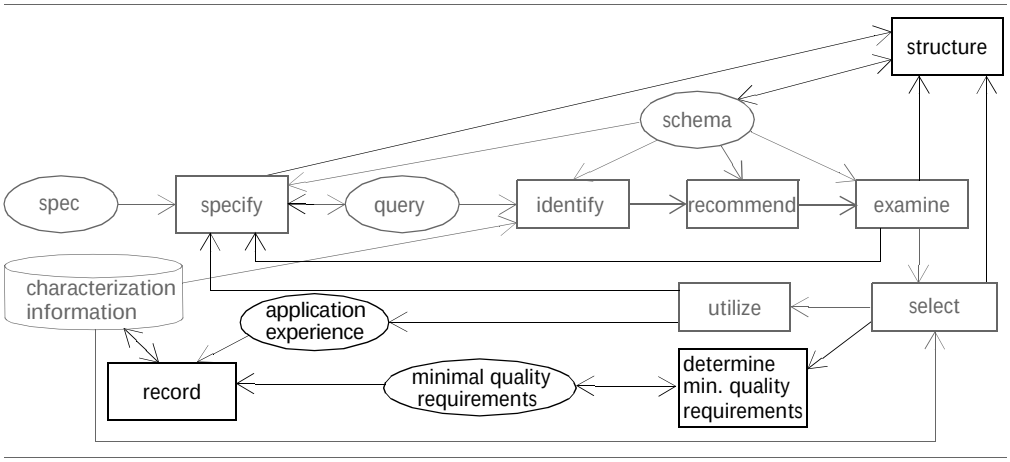


Figure 38: The usage model (shown in grey) can be extended by change tasks to the experience base.

specify

During the task »specify«, it may turn out that the universe of discourse to be covered by the experience base is actually not covered. For instance, if no concept for lessons learned is part of the schema, 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 experience base. Therefore, a new concept for lessons learned should be introduced (task »define concepts (T77, page 221)«). 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 experience base (task »plan change (T110, page 239)«).

identify
recommend

The next tasks »identify« and »recommend« are performed without user interaction. Hence, no situation for improving the usefulness can be identified (by the user) during these tasks.

examine

Considering the extended usage model where the user employs some editor to inspect candidate artifacts to decide on their usefulness, the objective of »examine« is to decide which artifacts to view in which order (artifacts that are deemed useless based on the characterizations are neglected). This is done in task »prioritize«, a subtask of »examine«.

If not enough useful artifacts have been found, the user may decide to reissue a revised query (return to task »specify«).

During »examine«, the user may not be able to decide whether to view an artifact or to declare it as »useless«. As the inspection of an artifact can require considerable effort, the user wants to inspect 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 schema of the experience base in a goal-oriented way by adding new attributes (task »identify reuse information (T79, page 222)«). If the missing information is supplied by the experience base maintenance team (i.e., including the values for new attributes – task »adapt characterization (T59, page 122)«), the user will be able to decide next time he issues a similar query. The usefulness of the experience base will have been improved.

The output of »examine« is the order in which potentially useful artifacts are viewed. If the viewing of the artifacts is invoked under the control of the technical infrastructure of the experience base, the technical infrastructure can record this order. Ideally, the order should be the same as the one determined during »recommend«, subtask »sort« (as pointed out in Section 9.1). 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 (task »explain prioritizing«, a subtask of »examine«). 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 (task »identify reuse information (T79, page 222)«) and the new knowledge should be considered by the similarity functions (tasks »define meaning of similarity (T81, page 224)«, »define local similarity (T92, page 229)«, and »define global similarity (T93, page 230)«)).

select

After prioritizing the artifacts based on their characterizations, the artifacts are viewed in the order of the priorities assigned to them. As soon as one or more artifacts have been viewed, the best suitable artifact is selected to be utilized. If the user shall not be bothered with too many questions (these will arise especially after the initial setup of an experience base), the questions can be restricted to the artifact actually utilized, that is, the task »explain prioritizing« is skipped – only »explain selection« is performed.[1] However, 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 viewing).

1 There are systems that implement this strategy. For instance, KONTEXT is a tool that logs justifications for selections made [BK99].

At any rate, if the selection task 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 experience base 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 minimal quality requirements for the type of the requested artifacts should be analyzed. Do they allow the collection of an artifact similar enough to the requested one? If not, the minimal quality requirements should be changed (»determine minimal quality requirements (T84, page 226)«). If the missing artifact is deemed to have a high reuse 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 recorded after the project has been completed.

utilize

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

During the artifact's utilization, 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 utilizing.

2 Stop utilizing and do nothing more (e.g., if it turns out that a lesson learned is not applicable in the current situation).

3 Stop utilizing and create the needed artifact from scratch (e.g., a project schedule).

4 Stop utilizing 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 of the schema and the contents of an experience base can be pinpointed by automatically collecting data and asking the user for feedback if the system behaves not as expected (based on the usage model). Table 24 shows a summary of the improvement actions described in this section.

Table 24: Indicators, causes of suboptimal usefulness, and corresponding improvement actions

Task	Indicator	Cause	Improvement action
specify	Kind of needed artifact cannot be specified	Schema is incomplete, that is, universe of discourse does not match defined concepts	Extend schema by a new concept
examine	Not enough useful artifacts in the result of the query	Imprecise or incorrect query	Re-specify query
		Coverage too low	Record new artifacts
	User cannot decide whether an artifact is useless or should be considered for (viewing and) utilizing	Schema is incomplete, that is, universe of discourse does not match defined attributes for the respective concepts	1 Extend schema by new attributes 2 Adapt characterizations of stored artifacts
		Characterizations of artifacts are incomplete, that is, not all attribute values are specified	Adapt/complete characterizations of affected artifacts
select	Order in which user views artifacts is not the order recommended by the retrieval tool	Imprecise query	None
		Improper user-definable similarity or there are missing attributes that have not been considered for the similarity computation, but should be (error in schema)	Add attributes (if needed) and redefine similarity
	User does not select any artifact	Coverage too low	1 Ask user why most similar artifact was not selected 2 Check collection criteria (and change them if necessary) 3 Make artifacts similar to the requested one available and record them
utilize	Artifact is not as useful as originally estimated	Wrong user estimation	Ignore and continue utilizing
			Stop utilizing and do nothing more
			Stop utilizing and create artifact from scratch
			Stop utilizing and retrieve another (hopefully more useful) artifact

9.4 Validation of OMI

OMI was validated using CBR-PEB (Case-Based Reasoning Product Experience Base) [NT99], a publicly accessible experience base[1] on case-based reasoning tools and applications [BSAM97]. CBR-PEB has been developed for supporting CBR system development. Emphasis is placed on providing decision support for reusing existing CBR system know-how for the development of new systems.

During its last major evaluation [Nic98], experts (i.e., managers of CBR system developments) were interviewed to find out about their expectations about

1 http://demolab.iese.fhg.de:8080/

CBR-PEB, resulting in an adaptation of the schema underlying the experience base. The experts articulated under what circumstances they would view CBR-PEB as useful. The interviews showed that the characterizations need to be accurate (i.e., up-to-date) and complete (validating the »characterization« branch of »contents« in Figure 36). In addition, the experts listed some information that was essential for providing good decision support. Some of this information was not captured as part of the characterization yet, making the decision which CBR system to use as the basis for a new CBR system very difficult. This corresponds to the »attributes« branch of the »completeness« of the »schema« in Figure 36. The »coverage« was seen as a major influence factor for the usefulness of CBR-PEB. For example, the number of artifacts was named explicitly.

After changing the schema and adapting the characterizations of the CBR systems stored in CBR-PEB (improvement action of second »examine« entry in Table 24), the experts tested the revised experience base. The observations concerning the way they interacted with CBR-PEB confirmed the usage model of OMI (Figure 37) [Nic98].

The indicator/cause/improvement actions listed in Table 24 have been validated partly. For example, the second »examine« entry has been validated as outlined above. Other table entries are either obvious (e.g., the »specify« entry) or can be validated in a similar manner.

9.5 Related Work

OMI can 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 intended to be human-based.

In spirit OMI is also comparable to evolutionary constructions of repositories (e.g., [BDF99, Hen97b]) and approaches based on Failure Mode and Effect Analysis (FMEA) [JK98, PZ96, Pfe96], which also try to avoid failure situations, that is in case of OMI, a decrease of the perceived usefulness.

In the area of software reuse, Frakes and Fox developed a failure modes model [FF96]. It is based on the *reuse success chain*, a temporal sequence consisting of seven main conditions:

1 Try to reuse
2 Part exists
3 Part available
4 Part found
5 Part understood

6 Part valid [i.e., of sufficient quality]
7 Part integrable

Successful reuse requires that all main conditions are satisfied. Frakes and Fox also provide reasons for unsatisfied conditions. Using a Pareto analysis, the most common failures can be identified and suitable corrective actions can be defined. Frakes and Fox suggest to use a continuous improvement technique. Consequently, their approach focuses more on organizational issues. In contrast, OMI is more technical and focuses on the users´ needs of a retrieval system based on REFSENO. As such, OMI details the possible reasons »insufficient representation«, »poor or no search tools«, and »inability to specify search« for the condition »part not found«.

9.6 Summary

The benefits of an experience base are ultimately determined by the usefulness of the experience base as perceived by its users. Therefore, an improvement of an experience base should be measured in the added perceived usefulness. Unfortunately, the perceived usefulness has many impact factors. Hence, it is difficult to identify good starting points for improvement.

This chapter presents the goal-oriented method OMI (Organizational Memory Improvement) for improving an experience base incrementally from the user's point of view. It considers the practical constraints typically encountered in industrial environments (e.g., limited time of users) and consists of a general usage model, a set of indicators for improvement potential, and a cause-effect model. During each task of the general usage model of OMI, the indicators are used to pinpoint improvement potential for increasing the perceived usefulness and asking the user for specific improvement suggestions where feasible.

OMI has been validated using CBR-PEB [NT99], a publicly accessible experience base[1] on case-based reasoning tools and applications [BSAM97]. The validation confirmed the cause-effect model, the general usage model as well as many of the indicator/cause/improvement actions described in this chapter.

Since OMI´s general usage model corresponds to the task »reuse« of Chapter 4, the validation of the usage model also validates this part of the generic task framework.

1 http://demolab.iese.fhg.de:8080/

10 Fraunhofer IESE's Corporate Information Network (Case Study)

The world cannot be understood from a single point of view.

Eleanor Roosevelt

DISER (the methodology developed in Chapters 4 through 7) has been validated in several industrial-strength projects including a large German insurance company and a large aerospace company (Section 9.4 and Section 7.5 list additional projects). Many of these projects cannot be detailed as part of this dissertation due to reasons of confidentiality. Since not all projects can be described, the author decided to present one of the projects (namely, Fraunhofer IESE's Corporate Information Network) in detail. This detailed description serves as a case study for DISER.

The case study presented in this chapter:

- Validates the viability and practicability of the task decomposition described in Chapter 4 and Chapter 8
- Lists first experiences gained with the execution of the tasks
- Gives an example for an experience base schema (this will give the reader an impression of REFSENO's applicability in practice)
- Explains some of the background for the experiment presented in the next chapter

10.1 Reasons for the Corporate Information Network (COIN)

To understand why the project »COIN« (Corporate Information Network) was launched, one has to take a look at the business goals and the situation of Fraunhofer IESE in the middle of 1998. This is the aim of this section.

Fraunhofer IESE was created to transfer innovative software engineering technology into practice through applied research. The software engineering technology is tailored to company-specific needs and is constantly evolving. During the transfer, experience is gained by different people in several departments and groups of Fraunhofer IESE. There are two types of transfer experience:

1 **Experience on the application of the technology.** While applying technology in practice, lessons are learned about the technology itself (e.g., effects and limitations of a technology and how it can be adapted to overcome these limitations).

2 **Experience on setting up and executing a project (project experience).** While interacting with the customer, Fraunhofer IESE learns what its customers want (i.e., what kind of service makes the customers happy), how it can transfer and tailor software technology more effectively and efficiently (in general) as well as how to improve Fraunhofer IESE's internal business processes.

While the former kind of experience is often captured (partly) in form of deliverables and scientific publications, the latter kind of experience is poorly documented (if at all) and thus not accessible to others.

Expectations are that, with time, Fraunhofer IESE will transfer more technology which is increasingly better tailored to companies´ needs (using the same effort). To do so, Fraunhofer IESE must find better ways to tailor technology effectively and establish long-term relationships with its customers (that is, make customers happy). Therefore, the sharing of project experience is a very immediate and obvious problem. In addition, researchers at Fraunhofer IESE have expressed the need for better communication. Finally, the self-application of the experience factory concept increases the credibility of Fraunhofer IESE.

For all these reasons, Fraunhofer IESE decided to launch »COIN«. COIN is both a designation for a project and the name of a repository through which all project-relevant information can be shared. Objective of the project is to build and run the repository.

The rest of this chapter describes how the repository was built by performing the task »structure« as explained in Chapter 8.

10.2 Setting the Objectives

identify stakeholders (T63, page 212)

Once the idea for COIN was born, the potential users of the project-relevant information were identified:

- Director of the institute
- Department heads
- Project managers
- Project members

This first step corresponds to the task »identify stakeholders«, although not all stakeholders were represented by the identified users. The users include the stakeholders who are funding the COIN, but some persons responsible for providing project-relevant information (e.g., library information service, documentation service, project controlling), persons providing the technical infrastructure of COIN, and the persons working for COIN were not considered at this point. One reason for this was that it was not clear yet who would be working for

COIN. Moreover, the interests of the users were not articulated or analyzed. This fact made it difficult to set objectives for COIN suitable to all potential users.

identify objectives (T64, page 213)

To alleviate this situation, it was decided to interview the director, all department heads, and representatives for project managers and members (i.e., employees with extensive experience).

In the middle of 1998 two interviews regarding the interests of the users were performed using the questionnaire of Appendix A: One with the deputy director and one with the department head responsible for coordinating the industrial projects. From these two interviews it became clear that the interests of the users were not only diverse (although both agreed on the level that project-relevant information must be shared), but also very demanding.

Based on the results of the first two interviews, it became obvious that all expectations of the users could not be met on a short-term basis. This called for a stronger focus of the project. Consequently, the decision was made to abandon the interviews and to pursue the following strategy:

- Develop a vision for COIN (i.e., what can (potentially) be part of COIN?)
- Pick a small subset of project-relevant information
- Implement COIN for the subset
- Get feedback from the users
- Expand COIN gradually based on the feedback

The vision was documented at the end of 1998 in form of a knowledge architecture for COIN depicted in Figure 39 and in March 1999 as a set of goals (see Appendix B).

Thus, an alternative method for the task »identify objectives« was performed. This method can be used whenever there are many stakeholders which have diverse and demanding objectives. However, the method bears the danger that some stakeholders may boycott the initiative, because they were not asked. Therefore, this risk has to be taken into account if the simpler method for identifying the objectives is to be applied.

select objectives (T65, page 214)

Based on the vision, two initial objectives were selected for COIN:

1 Install a quality management system at Fraunhofer IESE

2 Share project experience

The first objective aims at giving process guidance to project teams. Thus it corresponds to the provision of process guidance of COIN's goal 1 (see Appendix B). The latter objective corresponds directly to COIN's goal 2. The rest of this chapter details this objective.

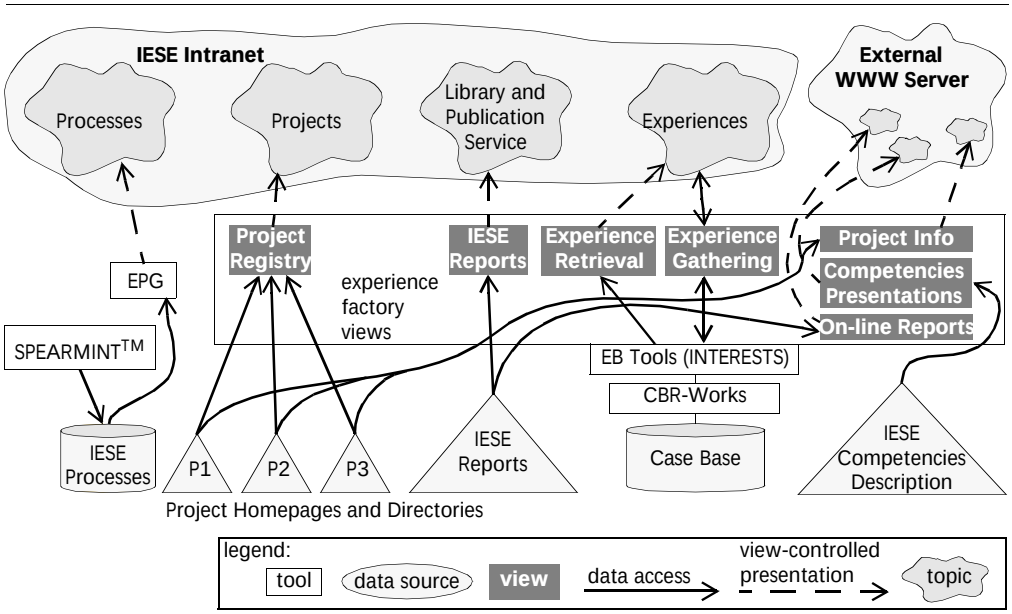


Figure 39: Envisioned Knowledge Architecture of COIN

At the time the objectives were selected, the criteria listed in the method of the task »select objectives« were not known. Overall it can be said that the missing systematic approach to derive the objectives led among other factors (such as the sheer complexity of the undertaking) to the long duration of about nine months until the initial objectives were chosen. This shows how important a goal-oriented, systematic approach for designing an experience base is – especially if it can be foreseen that the structure of the experience will be complex.

define success criteria (T66, page 214)

To be able to judge whether the selected objectives have been accomplished, it is necessary to define success criteria. For example, it was expected that the explicit recording of project experience would improve the state of the practice at Fraunhofer IESE both in terms of time (i.e., useful experience items can be acquired faster by the project manager or member) and quantity (of useful experience items).

At the time the success criteria were defined, the current state of the practice for sharing project experience was to ask one or more colleagues. A project registry in form of an HTML page in the intranet listed all of Fraunhofer IESE's past, current, and upcoming projects including their project title, customer, project manager and members. Thus, the project registry could be used as a knowledge map to find interview partners based on project title and customer. However, the project registry was not always up to date, for example, changes in project staffing were often not recorded.

10.3 Setting the Subject Areas

identify subject areas (T68, page 215)

After it had been decided to share explicitly stored project experience, three questions came up:

1 What kind of experience should be shared (templates, reports, lessons learned)?

2 For which project phases should experience be shared (acquisition, project planning, project execution)?

3 On what should experience be shared (internal processes of Fraunhofer IESE, technical experience regarding baselines, limitations, and tailorability of transferred software engineering technology)

select subject areas (T69, page 215)

To answer these questions, the evaluation criteria of the »select subject areas« method were used. In the following, the killer criteria are listed for the excluded subject areas.

1 **Kind of experience.** Templates were rejected, because (a) they are already accessible to project teams over the intranet, (b) their comparably low number does not warrant to retrieve them using similarity-based retrieval, and (c) it is not expected that the collection of templates will grow significantly in the future. Reports were rejected, because finding useful reports would require an elaborate schema. Such a schema should not only classify the reports by their title, author, etc., but also by the way they can be reused (e.g., an introduction to a software engineering technology may be reused in other reports[1]). Given the large number of reports available (about 250 at the time the decision was taken) and the vague idea about what »similar« reports really mean, the risk of expending a lot of recording effort for a possibly low success rate was assessed to be too high.

2 **Project phases.** The acquisition phase was rejected, because (a) only a few persons were involved actively in acquisition, (b) the persons doing the acquisition were the most experienced, and (c) the persons doing the acquisition had little time to record their experience.

3 **»Reference objects« for the experience.** Technical experience was rejected, because it was felt that (a) person-to-person communication

1 Note that such reuse opportunities cannot be deduced from the title. A report entitled »Experience gained during the improvement program at company X« may contain any number of such introductory texts (including none).

worked quite well in this area and (b) technical experience was already documented partly in IESE reports and other scientific publications.

These rejections left only lessons learned for the internal processes »Set-Up Project« and »Execute Project«. Table 25 shows the evaluation of this subject area. The evaluation shows that the selected subject area is almost a perfect candidate (»almost« because the experience was not available in a written form yet).

Table 25: Evaluation of the selected subject area (lessons learned on project planning and execution)

Category	Question	Answer
Potential	How many of the objectives can be reached by storing information for this subject area? How important is the explicit storage of this information for the objectives?	It addresses both selected objectives (process guidance and sharing of experience), because the experiences can be interpreted as hints for performing the processes »Set-Up Project« and »Execute Project«. It is expected that the explicit recording of experience improves the sharing of experience and supports the project guidance. In addition, capturing the experience explicitly counteracts fluctuation of personnel.
	How many users are interested in the particular subject area?	Everybody who is on a project (which is the majority of Fraunhofer IESE's employees)
	Are the tasks to be supported complex enough to justify recording of information for this subject area?	Yes. For example, the process for executing a project (which may last for over one year) is described on five pages. During a project, a lot of situations can be encountered which are not covered by the process description. Therefore, additional experience is very helpful.
	How do existing and planned standards affect reuse potential?	Positively. The existing process descriptions for »Set-Up Project« and »Execute Project« can be used as references. The process »Wrap-Up Project« will ensure the capturing of the lessons learned.
Existing artifacts	Are people available who are experienced in the subject area?	Yes. Many IESE employees have already been on several projects.
	Have they developed artifacts of high quality for this subject area?	No. However, tacit knowledge for the required artifacts is available. Project teams need to be interviewed to develop the needed artifacts.
	Are these artifacts available?	No. See previous question.
	Can the experts decide whether two artifacts are similar?	Yes. There is already know-how available concerning lessons learned (see [BT98a]).
	Is it possible to collect a set of artifacts that represents a good coverage of the subject area within a few months?	Yes. Teams of current projects and projects that ended within the last 12 months can be interviewed. This will cover all project types.
Stability and maturity	Are needs or technology changing slowly enough to allow recovery of an investment to store the information in the subject area explicitly?	Yes. Project planning and performance are generic tasks that will not be ceased.

The evaluation results of the subject area are backed up by success stories about lesson learned programs described in the literature (e.g., [BWP98, Hen95, KS96, KR97, Sar95]). In fact, lessons learned are a very powerful kind of experience as

the effort for recording them is acceptable (see Section 10.7) and their benefits are very high. Benefits include [Sar95]:

- Prevention of the recurrence of undesirable events
- Promotion of the recurrence of good practice
- Improvement of current practices
- Increase in efficiency, productivity, and quality by reducing costs and meeting schedules
- Increase in communication and teamwork

In addition, lessons learned complement more formal measurement programs by allowing persons to articulate insights (usually informally) not captured in the measurement programs. As such they can also be a vehicle for recording the data interpretations given during feedback sessions that are part of a GQM-based measurement program [vSB99].

10.4 Identifying Scenarios

After the subject area had been selected, reuse and record scenarios were developed.

10.4.1 Reuse Scenarios and Retrieval Goals

identify reuse scenarios (T71, page 217)

The reuse scenarios of COIN are based on the experience gained with the documentation of lessons learned in other projects [BT98b]. Each scenario is described by the following sections:

- **Trigger.** The trigger specifies under which condition(s) the scenario is expected to be executed.
- **Actions.** The actions describe how the scenario is executed. The descriptions explain what information is typically entered and what information is sought after. Because the lessons learned captured will cover a large spectrum (different types of projects, customers, domains, etc.), it is difficult to describe explicitly how to utilize the lesson learned in detail. Therefore, the action section mainly focuses on the retrieval part.
- **Expected benefits.** The expected benefits justify why the scenario should be supported. This allows to assess the consequences if the decision is taken to no longer support one of the scenarios.

The scenarios consequently use a terminology similar to the one in [BT98a]. The major terms used have the following meaning:

- **Lesson Learned.** A *lesson learned* is a semiformal (mostly textual) description of an *observation* or a *problem-solution/guideline/improvement suggestion* statement. If the noticed phenomenon has known negative effects, it is recorded as a *problem-solution/guideline/improvement suggestion* statement. Otherwise it is recorded as an *observation*.

- **Observation.** An *observation* is something which has no known negative effects. *Observations* documenting positive effects are utilized by repeating the actions or trying to achieve the project situation described as the cause (in the *observations*) for the positive effects. All other *observations* describe situations which do not allow positive/negative judgement. They are typically utilized as baselines for some actions. For example, the *observation* »it took us two days to do *X*« can be utilized by planning two days if an action similar to *X* needs to be performed in another project.
- **Problem.** A *problem* documents negative effects together with the project situation that led to the negative effects. Project situations that led to negative effects can be utilized as risk factors. Since the occurrence of *problems* is undesirable, they should not recur. To avoid *problems* from recurring, *guidelines* and *improvement suggestions* are worked out and attached to the *problem* description. It is also possible to attach information concerning the problem-solution controlling to the *problem* description. This allows for the systematic development of *guidelines* and *improvement suggestions.* In addition, the *solutions*, which were applied in the project where the *problem* occurred, are attached to the *problem* description.
- **Solution.** A *solution* is an attempt to take care of a *problem* that has occurred. Its description always contains the actions taken as well as the outcome. Optionally, it may also contain a justification, that is, the reason why the particular *solution* was applied. Recorded *solutions* are utilized by considering them for the development of *guidelines* and *improvement suggestions*. In addition, *solutions* of a *problem* can be utilized if the *problem* recurs. In this case, the user can find out which *solutions* work and which do not.
- **Guideline.** A *guideline* is a suggestion of what to observe when performing some process step and is a (hypothetical) way to prevent an occurred *problem*. It is attached to the process step during which it is to be observed. A reference to its corresponding *problem* gives its user the possibility to decide whether to utilize (i.e., observe) the guideline or not. If the *problem* has a potential of recurring in a new project (project risk), the *guideline* should be utilized (risk mitigation strategy).
  Guidelines and *observations* are similar in the way they are utilized. Both should be observed during the execution of a particular process step. They differ, however, in the way they were developed. While *observations* result directly from project experience, *guidelines* do not. *Guidelines* are the result of a systematic problem solving process. Consequently, they contain a reference to the problem from which they were derived, whereas *observations* do not. Another difference concerns the degree of validation. While *observations* are validated at least once (the project in which they have been observed), *guidelines* can be purely hypothetical (i.e., unvalidated) – although they are derived from a *problem* that occurred in a real project. *Guidelines* can be validated by utilizing them in projects and checking whether the *problems*, which the *guidelines* were trying to prevent, were actually avoided.
- **Improvement suggestion.** An *improvement suggestion* is a suggestion of how to change some artifact and is a (hypothetical) way to prevent an

occurred *problem*. *Improvement suggestions* are utilized by constructively integrating them into the artifact they refer to. After the integration, the improvement suggestion is marked obsolete.
Improvement suggestions are similar to *guidelines* in the way they are derived. However, they differ in their target group. While *improvement suggestions* are written for the author of some artifact, *guidelines* are written for someone performing a particular process step. Consequently, *improvement suggestions* may refer to a document that does not describe a process step.

define retrieval goals (T72, page 218)

Both the reuse scenarios and the corresponding retrieval goals can be found in Appendix C, Section C.2.

In COIN, many discussions took place on what should be stored as part of COIN up to September 1999. These discussions were basically brainstorming sessions eliciting available and needed experience at Fraunhofer IESE. As part of these discussions, a diploma thesis was performed with the objective to document the state of the practice regarding knowledge management and to create a concept for Fraunhofer IESE's future knowledge management [Dec99].

The results of these discussions were summarized in September in form of a concept glossary (see Appendix E, Table E.2 on page 345). For each concept, relevant reuse scenarios were identified by characterizing the retrieval goals for them (retrieval goals are listed without their context as part of the concept glossary; the context is the same as the one of the experience base schema – see Appendix E, Table E.1 on page 345). However, only those concepts relevant for the selected subject area were further detailed. This means that only reuse scenarios relevant for the lessons learned were developed.

This contrasts with the suggestion of the »identify scenarios (T70, page 217)« method which calls for a sequential performance of »identify reuse scenarios« and »define retrieval goals« prior to »conceptualize«. This sequence was deliberately broken, because the discussions centered around contents.

In the case of COIN, it was easy to define the retrieval goals, because most of the experience to be stored already existed at Fraunhofer IESE or was already well defined through existing experience base schemas (as in the case of lessons learned). Hence, the development of the reuse scenarios was more of a documentation of existing experience base schemas rather than the basis for a new schema. For the development of a totally new schema, the sequential performance is probably preferable, because the used terms cannot be defined as clearly (at the time the reuse scenarios are developed) as those resulting from the discussions.

10.4.2 Record Scenarios

After the documentation of the reuse scenarios, a scenario for recording lessons learned was devised. At the end of 1998, a project analysis interview was conducted as a test. This helped a lot in assessing the feasibility of the scenario.

identify knowledge sources (T74, page 219)

As the project experience was not explicitly documented otherwise, the project members were identified as the only available knowledge sources.

describe knowledge acquisition (T75, page 219)

Using project members as knowledge sources puts several constraints on the knowledge acquisition:

- The effort for »supplying« the lessons learned should be low in comparison to the »real« project work, because providing knowledge for other projects is typically not an objective of the project.
- The number of disturbances should be kept at a minimum, that is, everything that can be asked in a single event (interview), should be asked in one event and not in a series of events.
- The interviewer should use a terminology well understood by the project team. The terminology defined in Section 10.4.1 turned out to be difficult to understand.

Therefore, it was decided to use a questionnaire similar to the one used during the project analysis interview at the end of 1998. The new questionnaire can be found in Appendix D, while the complete record scenario is outlined in Appendix C, Section C.3. For COIN, there is only one record scenario because only one collection event (the project analysis interview) takes place. The result of the project analysis interview is the project analysis report (PAR). All lessons learned and the project characterization (for describing the context for the lessons learned) are derived from the PAR.

To ensure minimal effort on the side of the project teams, a member of the COIN team organizes and conducts the project analysis interviews. In this way, the project team does not need to spend the time to write the answers – oral answers suffice. However, project teams are asked to prepare for the project analysis interview by sending them the PAR template (see Appendix D) before the interview.

10.5 Developing the COIN Schema

In October 1999, the experience base schema for the selected subject area was developed based on the identified reuse scenarios (see Appendix E).

conceptualize (T76, page 220)

The task »conceptualize« was performed essentially as described in Chapter 8. Therefore, a detailed reflection of how the task was performed is not necessary here. However, it is worthwhile to note the following:

- Although the modeled subject area is only a small part of the envisioned experience base, the schema description is already quite extensive. However, the tabular representation of the conceptual information and the graphical representation of the relationships among the concepts (see Section E.7) still allow a clear presentation of the schema. This shows that REFSENO is able to cope with industrial-strength experience base schemas.
- In addition, the schema can be used as a communication vehicle for discussing the necessity, meaning, completeness, etc. of schema parts. In fact, discussion is possible on several abstraction levels. For example, in COIN the concept and symbol glossary were discussed between COIN's project manager and the author, whereas discussions down to the attribute level took place with other colleagues.
- The complete definition of the concept glossary allows the documentation of expected schema extensions. A typical schema evolution is to »expand« a simple type (usually a symbol type) to a concept later. For instance, for the selected subject area a detailed characterization of Fraunhofer IESE employees is not necessary (none of the scenarios rationalize this!). Therefore, an enumeration type of all employees suffices. Because future scenarios may require such a detailing, the »IESE employee« type is marked with a leading »i« (which stands for »interface«) documenting a possible future expansion of the schema at this expansion interface. This shows that REFSENO supports modularity (requirement »modularity of conceptual information (R11, page 47)«), because another schema dealing with employees in detail (i.e., a schema which defines »IESE Employee« as a concept) can be integrated into COIN at exactly this interface (i.e., all terminal concept attributes of type »iIESE Employee« are replaced by nonterminal concept attributes with the destination concept »IESE Employee«).
- The attributes listed in Section E.3 were derived from previous industrial transfer and publicly funded projects dealing with lessons learned. Schemas used in those projects were merged and tailored to the needs of COIN. The complete development of the experience base schema as shown in Appendix E took about 5 person days (including the reviews by colleagues). This confirmed the positive experience the author had at a large German insurance company with reusing already existing schemas [ABH+99]. It also confirms the required »modularity of conceptual information (R11, page 47)« of REFSENO.
- No (objective) minimal quality requirements regarding lessons learned were known. Therefore, no minimal quality requirements were produced during »determine minimal quality requirements (T84, page 226)«. Meaningful minimal quality requirements will have to be acquired during actual operation.
- All stored characterization information is rationalized (requirement »rationalized conceptual information (R10, page 47)«) by the documented scenarios. If, for some reason, it would be decided not to support the scenario »FindAr-

tifact« many concept attributes of the concept »Artifact« would become superfluous (e.g., »type«, »owner«, and »author«). Consequently, other definitions (such as the type definition for »ArtifType«) would become obsolete. This eases the maintenance (requirement »maintenance of conceptual information (R13, page 49)«) and thus the transfer of schemas to other organizational contexts which require a different set of scenarios.

- The reuse of (not rationalized) schemas led to attributes which could not be rationalized using the initial set of scenarios. However, some of these attributes (e.g., the author of an artifact) seemed to be important for storing. Analysis showed that the initial set of scenarios was incomplete. In this way, three (important) additional scenarios were identified and documented during the conceptualization phase. This emphasizes the iterative character of the structuring task. Thus, the reuse of schemas can also lead to a completion of the set of reuse scenarios.
- The division of the similarity computation in global (on the concept level) and local similarity (on the attribute level) eases the definition of an adequate similarity. In this way, standard similarities can be used as a first »intelligent guess« which can be improved later using OMI (see Chapter 9). Often, the adequacy of standard similarity functions on the attribute level can be judged by the modeling expert and corrected if necessary. For example, the similarity for the type »ProblemStatus« is not computed using a standard similarity function. Instead, a table is defined for the similarity computation (see Table E.4 on page 362), which is based on the corresponding state transition diagram (see Figure E.4 on page 364).
- The schema can also be used to illustrate the use of concepts introduced for modeling purposes. As pointed out in Section 10.4, observations and guidelines are very similar. In fact, they are used for the same scenario »GetRelevantProjectInfo« (see Section C.2.2 on page 332). To support this scenario adequately, an abstract concept called »Process Info« was introduced (for a definition see Table E.2 on page 345). It subsumes both observations and guidelines (see Section E.7 on page 365). Thus, the reuse scenario can be realized by starting a query using the concept »Process Info« as a starting point.

10.6 Defining the Recording of Experience

define ›record‹ (T95, page 231)

The record scenario stated in Appendix C, Section C.3 restricts the set of possible record methods substantially. Since project managers, scientific leads, and project members are the only knowledge sources (i.e., human experts), the task »define knowledge collection (T96, page 232)« is straight forward (see Table 26). But since the »reusable parts« are not collected in the way they are used later on, the splitting of the project analysis report into its reusable parts is a bit complex. Table 27 defines the remaining methods of the record subtasks.

Table 26: Results of performing the task »define knowledge collection (T96, page 232)«

Subtask	Result
survey knowledge sources (T97, page 232)	Project managers, scientific leads, and project members provide project experience
identify overlapping contents (T98, page 233)	Unknown/cannot be decided a priori
select knowledge sources (T99, page 233)	Every project manager, scientific lead, and project member should be interviewed in a project team meeting. This will allow the team to reflect on the project execution (social aspect) in addition to the provision of project experience.
plan adaptation (T100, page 234)	The template for project analysis reports (see Appendix D) is used for questioning.
define knowledge collection plan (T101, page 235)	Project analysis interviews take place at the end of a project. For long running projects, intermediate project analysis interviews may be scheduled (e.g., every six months). If possible, additional interviews shall be conducted at milestones of a project. A member of the COIN team (experience engineer) arranges and moderates the interview. The template of Appendix D is used for questioning. After the interview, the experience engineer completes the report based on the answers and asks the project manager, the scientific lead, and project members to review it. **Note 1:** It is useful to send the template to the project manager, scientific lead, and project members *before* the interview so they have a chance to prepare for the meeting. **Note 2:** For the later recording it is of high importance that each answer is marked with the name of the person who gave the answer. Alternatively, only the project members may be interviewed. After the completion of the project analysis report (PAR) based on the interview results (and review results of the project members), the project manager and the scientifc lead are asked to complement the PAR with their views and opinions.
define ›characterize initially‹ (T102, page 235)	All characterization information is extracted from the corresponding PAR.

The explicit definition of these methods makes the record task not only repeatable, but also transferable among people. In COIN, this is a very crucial aspect, because COIN team members are not assigned full-time to the COIN project. An experience engineer (someone from the COIN team) is assigned for a project analysis on an »as needed« basis. Therefore, project analyses can be executed even if one COIN team member is on vacation, on a conference, or involved temporarily in some other project. Before the precise definition of the splitting, the recording of the project analysis reports was confusing to and not repeatable for persons other than the author of this dissertation.

10.7 Infusing COIN

After designing COIN and the processes around it, the next step was to collect the project experience, that is, to conduct the project analysis interviews. However, at this point there was still the risk of recording »unsuccessfully« the

Table 27: Methods of »record« subtasks for lessons learned

Task	Method
split	The project analysis report (see Appendix D) is split into the project characterization, observations, improvement suggestions, guidelines, problems and their solutions as follows: **Section D.1, questions 1 and 2:** Instance of *Project* **Questions 3 and 4:** Discarded for selected subject area (acquired for later recording of project acquisition experience) **Question 5:** Instance of *Problem* (possible counteractions given are recorded as instances of *Guideline*, *Improvement Suggestion*, or *Solution* according to their intention – see Section 10.4.1 for a definition) **Note 1:** Newly entered problems have the status »new«. The following attribute values are not entered at this point: »decision date«, »priority«, »contact person«, »creativity team«. They are filled out during the problem solving process. **Note 2:** Newly entered improvement suggestions have the status »new«. The following attribute values are not entered at this point: »closed«, »comments«. **Questions 6:** (a) Positive or neutral rating (1–3) ⟹ Instance of *Observation* (b) Negative rating (4 or 5) ⟹ see Question 5 **Questions 7:** Instance of *Guideline* for the problem »effort too high« (may be a »do not …« guideline if rating is negative) **Questions 8:** (a) Positive or neutral remarks ⟹ Instance of *Observation* (b) Negative remarks ⟹ Instance of *Problem-Solution* or *Problem-Guideline* pair (depending on whether the suggested solution was actually applied or not) **Questions 9 and 10:** see Question 6 **Question 11:** Instance of *Observation* **Question 12:** see Question 5 **Questions 13 and 14:** (a) Positive discrepancies ⟹ Instance of *Observation* (b) no discrepancies ⟹ nothing is recorded (c) Negative discrepancies ⟹ see Question 5 **Questions 15 and 16:** Instance of *Guideline* or *Improvement Suggestion* (cf. Question 5) **Note:** The problem has already been recorded as part of Questions 9 and 10 **Question 17:** see Question 5 **Question 18:** Instances of *Observation*
analyze quality	Since no minimal quality requirements are known, this task is skipped.
analyze reuse potential	Project characterizations are always retained to be able to specify experience contexts. Observations, improvement suggestions, guidelines, problems, and solutions are only retained if they are nontrivial and concrete enough to be useful in future projects. For example, the guideline »do good planning« for a project with effort overrun would violate both criteria. Therefore, it would be rejected. The review is performed by someone of the COIN team who was *not* involved in the writing of the project analysis report.
inform	The authors, reporters, and originators are informed of their improvement suggestions, problems, observations, and guidelines.

project experience. Here, the term »successful« refers to the »conversion« of the contents of the project analysis report into lessons learned using the developed experience base schema. To mitigate this risk, a prototype using INTERESTS (see Section 7.5) was built early in November. Using this prototype, the defined method for the »split« task was trialed with the report produced as the result of the project analysis interview at the end of 1998 (see Section 10.4.2) *before* any further project analysis interviews were conducted.

For the initial population of COIN with project experience, a representative set of projects had to be selected. To do so, the following constraints were identified:

C1 Only projects completed in the past twelve months or still running should be considered, because people may not be able to recall all the experience they had in projects completed over a year ago.

C2 For those projects still running, some significant effort should have been spent on the project (that is, at least two person months). Otherwise, there might not yet be enough project experience to record.

C3 At least one project of each type (cf. definition of »ProjType« in Appendix E, Table E.6 on page 363) should be considered to ensure the representativeness of COIN.

C4 Each researcher (that is, those persons acting as project manager, scientific lead, or project members) should at least be interviewed once (to ensure that everybody contributed something to COIN) and at most three times (to keep the effort for each individual in acceptable bounds).

The selection was done as follows:

1 The project registry was updated.

2 All projects in the project registry fulfilling constraint C1 were put together in a list.

3 All running projects which started only shortly ago were discarded (approximating constraint C2). For all projects for which it was not clear whether they meet constraint C2, the project managers of the respective projects were asked (fulfilling constraint C2). During these »pre-interviews« it turned out that the project registry was not up to date in parts (e.g., concerning project staffing, some internal projects were not listed, some project titles did not reflect what was actually done in the projects). Furthermore, some separately listed projects were really handled as a single project. Although there existed separate contracts, project experience would not have been assignable to the single contracts. Therefore, it was decided to merge inseparable projects by preparing a single project analysis report for each such project grouping.

4 From the remaining 45 projects, the most important projects of each type were selected (fulfilling constraint C3). For some industrial project types, quite a few projects stood on the list. From these, the department head responsible for coordinating the projects chose a representative set using the criterion »something unusual happened in the project«.

5 The 30 selected projects were checked against constraint C4. Only three researchers had to be interviewed four times. It was decided to make these three exceptions. Newly employed researchers (who were not project manager or member in one of the projects fulfilling constraints C1 and C2) would not be interviewed.

6 The projects were prioritized. First those projects that were executed by the same department COIN belongs to were scheduled, then those projects of other departments. This allowed to get internal feedback first.

From the project analysis interview conducted at the end of 1998, a preliminary baseline was known:

- 15–30 min. preparation effort for each project member and the experience engineer (who had to schedule the interview and send out the PAR template to the participants before the meeting)
- Duration of 30–45 min. for the interview
- 2.5 person hours for the experience engineer to write the project analysis report (PAR) and to send it out for review
- 15–30 min. for each project member to review the PAR
- 30 min. to merge the review comments of the project members for the experience engineer
- 2 person hours to record the PAR in the experience base for the experience engineer

This is a total of 1 to 1.75 hours for each project member. Since there were three members on the team, this amounted to roughly half a person day which is an acceptable number compared to the effort of 80 person days put into the project.

For the experience engineer, it took 5.75 hours to collect the experience from this project. To reduce the effort for the experience engineer, two changes in the procedure were made:

1 A laptop was used during the interviews and the answers of the participants were directly typed into the PAR. This worked out well, because (a) the experience engineer was fast enough in typing and (b) there was little jumping between the questions during the interviews. This improvement action saved about 2 person hours to prepare the PAR.

2 The participants of the interview were asked to review the PAR sequentially one after the other. Before, the participants reviewed in parallel and the experience engineer had to merge the changes. This saved about 15 min. for the experience engineer.

Thus, a saving of about 40% (2.25 hours out of 5.75 hours) or nine person days in total (2.25 hours for 30 projects) was achieved.

As it turned out, the scheduling took 45-60 min. per interview. This resulted from the fact that:

- Several persons (that is, the project members of a project) needed to be coordinated
- Participants were asked face to face to give the experience collection a personal touch
- Participants were informed about the objectives of COIN and what they had to expect from the interview

The interviews themselves were conducted from the end of November 1999 until January 2000 and took 45-90 min. each. This was longer than expected and can be explained as follows:

- **Number of participants.** The more persons participate in an interview, the longer it takes, because everybody is asked to voice his opinion. Data shows that an interview takes at least 15 min. »per participant«. The test interview was conducted with two project members. Consequently, interviews with more project members took longer.
- **Preparation of participants.** In general, those participants who prepared themselves for the interviews were able to provide more structured answers which could be recorded directly as part of the project analysis report.
- **Size of project.** The more effort is spent in a project, the more likely are lively discussions during the project analysis interviews regarding potential causes for identified problems. This is especially true for projects that ran for more than 1.5 years, which demands a strict moderation of the interview session.
- **Kind of people participating.** Some people are very open and are willing to share all their experience. Others are very careful about sharing their personal opinions. In the extreme case, one interview took 60 minutes although only one person was interviewed.
- **Survey.** As part of the first interviews, participants were asked to give feedback on the project analysis activities. This survey took about 15 min.

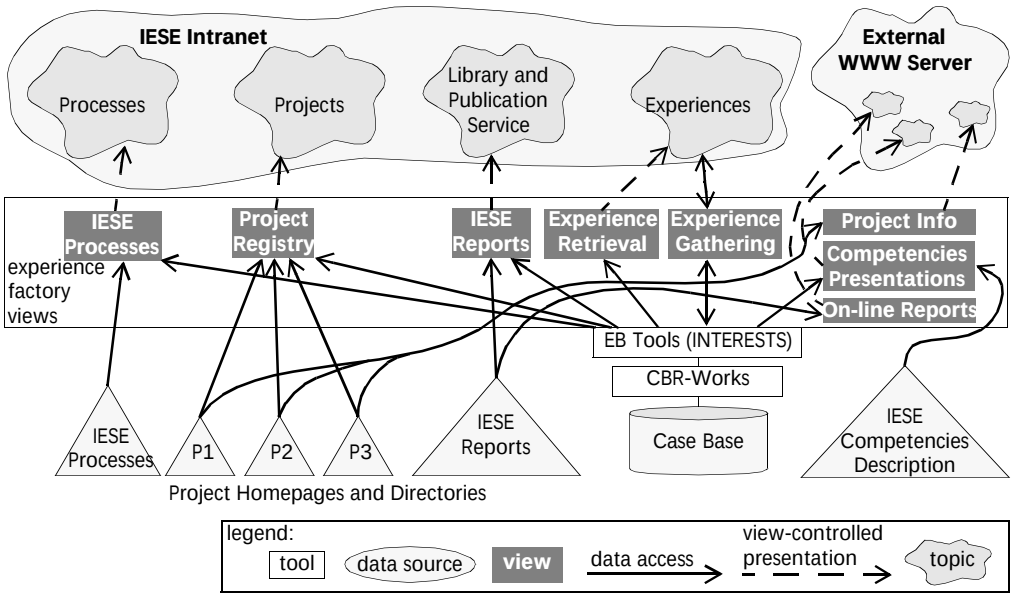
infuse SEEMS (T107, page 238)

In addition to the initial population, the further proceeding for COIN was planned:

- The initial knowledge architecture of COIN (see Figure 39 on page 262) was revised (task »develop technical infrastructure (T108, page 238)«). In the revised version, INTERESTS plays a more prominent role because documenting the relationships between the various artifact types was viewed as being

important. The revised version is depicted in Figure 40. The figure shows that

Figure 40: Revised Knowledge Architecture of COIN



INTERESTS acts as the EB-Specific Storage System, whereas the file system is used for the Artifact-Specific Storage System (cf. Figure 32 on page 200).

- Based on the gained project experience, develop a realistic »reuse situation« to be able to demonstrate the functionality (and usefulness) of COIN (task »prepare training material (T109, page 238)«). Comprehensive training material was not prepared, because:
  - **Reuse part.** It was felt that usage of COIN is straight forward due to its WWW interface. Therefore, a good demonstration should suffice for motivating people to use COIN and for showing how to use it.
  - **Acquisition of experience.** Recording was planned to be done only by the COIN team. Therefore, personal tutoring was preferred over conducting formal training sessions with formal training material. Furthermore, each PAR could be used as training material.
- The introduction of COIN was planned as follows (task »plan change (T110, page 239)«):
  1 After the initial population of the experience base, an experiment shall be conducted in which as many of the researchers as possible participate. The experiment will compare the »traditional« method of »ask your colleague« with the repository-based reuse of project experience. The experiment will give the researchers an impression of what COIN can do for them. (This experiment is described in Chapter 11.)
  2 The feedback from the experiment shall be used to improve the technical infrastructure.
  3 COIN shall be put into daily use.
  4 Project analysis interviews shall be continued on a regular basis.

In the case of COIN, it seemed unrealistic to include only a small group of pilot users (cf. »plan change« method), because all researchers had contributed their project experience and were fond of using the experience from others (this was one result of the survey conducted as part of the interviews).

10.8 Results of the Case Study

COIN has taught many lessons learned on:

- The development of a technical infrastructure of an experience base and the definition of processes to populate and utilize the experience base
- The population of an experience base
- User expectations regarding the explicit documentation of project experience

This section summarizes the most important points.

Applying DISER

A systematic, goal-oriented method for structuring complex experience bases is a must

One of the most important lessons we had to learn in COIN is that it is hard to »just do« the project. Too many stakeholders and interests were involved and the potential contents of COIN were highly interrelated. Thus, it was hard knowing where to start and how to proceed in an orderly fashion.

GOODSEE (see Chapter 8) did not become available until the end of August 1999. At that point in time, the project picked up momentum. Although the slow progress of the project in the beginning was also due to some other factors (e.g., little resources), it was easy to put COIN aside, because it was not clear how to proceed. The progress made since September 1999 shows the viability of the method presented in Chapter 8.

The task hierarchy can be used to analyze strengths and weaknesses

The detailed account of COIN in this chapter illustrates how the task hierarchy can be used to explain what was going on in the project. Especially in the beginning of the project – when the »structure« method was not available yet –, some discrepancies have been described and analyzed. Some of these discrepancies had most likely a negative impact on the progress (e.g., considering only the potential users, but not all stakeholders at first). For others, meaningful justifications were found (e.g., the decision not to ask all potential users for their expectations).

The task hierarchy is a framework for organizing method libraries

(Positive) deviations of COIN with respect to the method defined in Chapter 8 lead to new alternative methods for the tasks in the hierarchy. Thus, they can be used to build up method libraries. Such a library grows with each method explicitly documented and classified with the task it achieves and the practical constraints for which it is applicable. Future projects structuring experience bases can utilize those methods of the library which were applied in projects with similar practical constraints.

Therefore, the case study validates GOODSEE (task »structure (T56, page 121)«) for tailoring the technical infrastructure of an experience base to organization-specific needs. It also emphasizes some of the outstanding characteristics of DISER:

- Concerning GOODSEE (see Chapter 8):
  - Evaluation criteria are very helpful in selecting objectives and subject areas (see Section 10.2 and Section 10.3).
  - Scenarios are an important prerequisite for developing the experience base schema. They document what is expected of the future experience base. They also show the gap between the needed information for performing the (reuse) scenarios and the available information (or information that can be made available). Informal descriptions of how to acquire the needed information illustrate how this gap can be closed.
  - The documentation of reuse scenarios using the three-part structure of Section 10.4 was found to be useful by the COIN project team to decide whether or not to support a particular scenario. Therefore, COIN extended the method for the task »identify reuse scenarios (T71, page 217)«.
  - The explicit documentation of how to record experience makes this task not only repeatable, but also transferable to other people (see Section 10.6).
- Concerning REFSENO (see Chapter 6 and Section 10.5):
  - Schemas defined using REFSENO are comprehensible – even with the complexity of industrial-strength projects. Thus, alternative schemas can be discussed regarding their advantages and disadvantages.
  - The retrieval goals associated with the concepts and concept attributes trace back each piece of information stored in the experience base to the scenario for which it is needed. Thus, the schema is rationalized. If a scenario is no longer to be supported, the corresponding concepts and concept attributes can be deleted from the schema (provided that they are not needed for other scenarios).
  - REFSENO can be used to develop modular schemas.
- Concerning the generic architecture of a technical infrastructure – especially the prototype INTERESTS (see Chapter 7): A prototype can be built fast using the suggested architecture. In case of COIN, it was a matter of two days once the schema had been developed.

Populating COIN

The initial population of COIN validated the »record« method. The subtasks could be applied as documented in Section 4.4.1. The case study showed some outstanding characteristics of the method:

- The collection of project experience follows a very specialized method which is neither transferable to other organizations (as is) nor applicable for artifacts other than project analysis reports. It will be a challenge to find »patterns« in collection methods that would allow a more detailed decomposition. This

also justifies the elaborate method for the task »define knowledge collection (T96, page 232)«.

- During the initial project analysis interviews, participants were asked if they find project analyses important and – if so – for what reason. Among the reasons given were the following:
  - Project analyses as performed are quite abstract – »stories« (history) do not come across; however, it is important to do project analysis
  - Through self-reflection, you get hints what to watch out for in future projects
  - Due to the self-reflection, important insights are remembered longer
  - Other viewpoints on the project are interesting
  - People get feedback they would never have gotten otherwise
  - One can learn a lot from it

  Thus, the collection alone can be worthwhile for project teams. This is somewhat surprising since the actual benefits are usually associated with the reuse of experience and not with the recording.
- The »split« task is the key for »converting« available information into needed information. Just as the collection method, this method is also highly specialized.
- The task »check for existing parts (T42, page 112)« is necessary to find similar or matching lessons learned. This allows to merge the lessons learned and to record the actual validity of the lessons learned. For example, without this check, it would be possible to record the same observation twice, each referring to another context in which it had been gained. This would result in a validity of »1« (each observation was evaluated in one project), although the same phenomenon has actually been observed twice and should have a validity of »2«.
- The task »analyze reuse potential (T44, page 113)« can be performed even if no objective minimal quality requirements are known.

Lessons Learned on Selected Subject Area

Finally, we gained some insights on the selected subject area (lessons learned on planning and executing projects) which will be discussed in the following.

Baseline for recording

As already described in Section 10.7, a first baseline for the recording effort was obtained.

Positive effects of the improvement program

The new improvement program has a positive effect on Fraunhofer IESE:

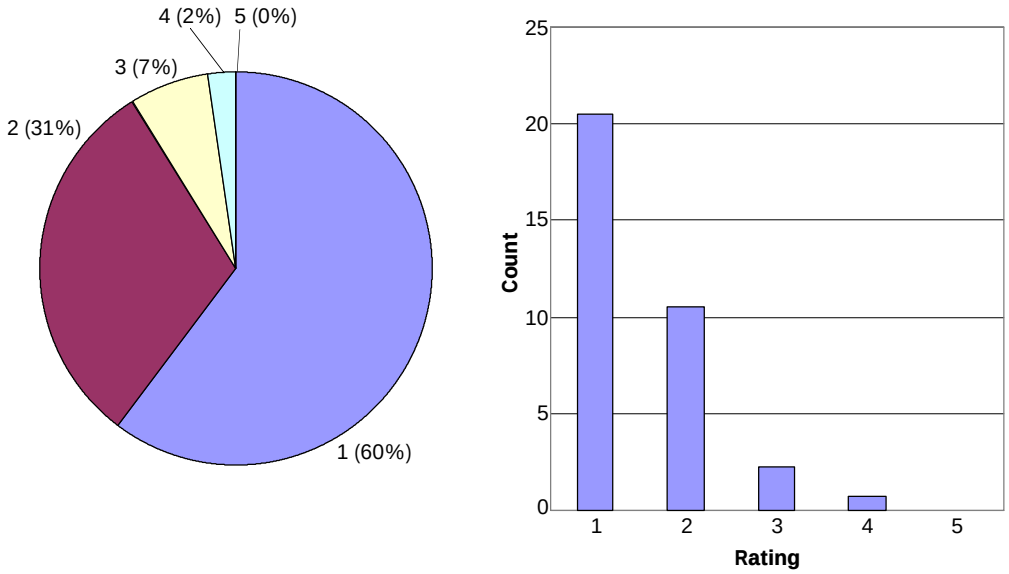
- Before COIN, many improvement initiatives were started (e.g., putting up an improvement suggestion box, setting up informal meetings). All of them are based on the assumption that the IESE employee takes the initiative. However, it is typical human behavior to go through the hassle of taking the initiative only if (a) something is really superb or (b) something is really bad. Therefore, COIN follows a pull approach. Through the project analysis interviews, experience is captured *actively* instead *passively*. During the project

analysis interviews, everybody is asked for his experience/opinion. In this way, (a) detailed information is captured and (b) less »spectacular« feedback is articulated.
- As the project characterization is recorded as part of the project analysis, an update of the project registry after project completion can be done. This resulted in a more up to date registry.

Importance rating of the participants

During the initial interviews, 34 participants rated how important they view project analyses on a scale from 1 (meaning »very important«) to 5 (meaning »very unimportant«). Figure 41 shows the result of the survey.

Figure 41: How important do project members of Fraunhofer IESE rate project analysis?



Many participants argued that the importance depends on whatever will happen with the results of the project analyses interviews. Those rating the importance with »1« did so because they assumed that something useful would happen with the interview results. Most of those rating the importance with »2« and all rating the importance with »3« or »4« had a pessimistic view, that is, they assumed that little or nothing useful would happen with the analysis results. However, they also said that they would change the rating to the better if they see something useful happening with the results. Two participants gave two ratings: One if something happens with the results and one if nothing happens. In the statistics, each of the ratings was counted as a half naming (e.g., for this reason, the rating »1« was named 20.5 times). Statements from participants that support this interpretation include:
- »If some improvements happen based on these project analyses, I would rate them ›1‹. In this case, I would also be willing to continue participating in project analysis interviews. If, however, nothing changes, I am not willing to participate in further interviews.«

- »I expect that my contributions trigger future improvements.«
- »My prediction: The results of the initial interviews will be placed in the intranet (but not maintained) and nothing else will happen.«
- »It is important what you make of it. If it is possible to extract a few guidelines, I would rate the importance as ›1‹. If project analysis reports land in a drawer, then project analysis is not important (rating ›4‹).«

Those rating the importance 3 or better, used the following arguments:

- Experience transfer is important, because projects are the main income of Fraunhofer IESE.
- One should learn from his or her own faults. Why should others not benefit from this?
- People do not have to fall into the same trap as their colleagues did.
- It is important for Fraunhofer IESE to get insights: What effort needs to be put into the projects and what does Fraunhofer IESE get back for the effort?
- Project analyses are important for those people who are responsible for the organization at Fraunhofer IESE.
- The interviews give a chance to say things people always wanted to say, but never did.
- Talking to people, one gets the impression that there is a lot of improvement potential.

Those not rating the importance as 1, argued:

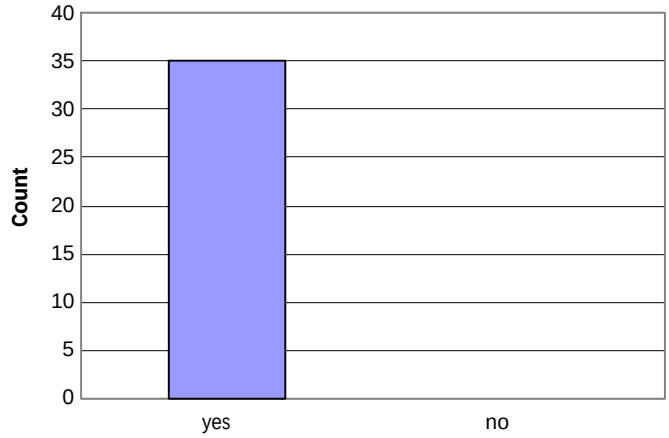
- It takes time to discover/recover the experience of others. And this may take too long.
- Although project analyses are important, there are other more important issues that should be addressed first (e.g., orderly project management).
- It might be difficult to transfer experience from one project to another, because many things look similar when they are really not (e.g., things may look similar due to the used terms, however, the terms may be used in dissimilar ways).
- It is more effective to use mentors.
- It is more effective to talk to people.
- It is more important to build up a personal network. The experience base cannot replace this network.

The argument that it is more effective to talk to colleagues will be addressed in the next chapter.

Was the effort okay for the participants?

In addition to the importance rating, 35 participants were asked whether the effort for the project analyses was okay. The effort included the effort for preparing for the interview, giving the interview, and reviewing the project analysis report. Figure 42 shows the results of the survey.
The figure shows that the project analysis effort was okay for 100% of the participants. In fact, some participants argued that the project analysis is not thorough enough under some conditions: For longer running projects where project

Figure 42: Was the effort for project analysis okay?



members are not satisfied with the way the project was performed, longer project analyses should take place (i.e., several person days should be spent on capturing project experience).

Future expectations

Future expectations and hopes of the participants are:

- Improvements and other positive changes take place based on the project analyses.
- Planning and execution of projects will be improved.
- People can find quickly contact points for similar projects and customers.
- People can get experience on project risks and planning.
- Tips and tricks will exist that help people in their projects (especially regarding project management).
- There will be standardized ways for planning and running projects.
- Project analyses will be standard practice.
- Systematic collection of project experience and provision of it to future projects will take place.
- COIN partly replaces experts who can act as project consultants. (At the time COIN was introduced, these experts were rarely available for giving advice. Thus the demand for project advice could not be met.)
- The reflection done during the project analysis interviews will also be done during (longer) projects.
- Guidelines derived from the project analyses become part of the IESE process descriptions.
- Everybody will be more conscious of what they are doing and can avoid »catastrophes«.
- People are able to avoid uncomfortable situations.
- A project member does not have to gain experience someone else already had.
- Motivation of employees will be improved.
- People can learn more about what is important for projects.

- Somebody takes the project analyses seriously and people want to learn from the experience gained (the experience base should not be a data cemetery).

These statements suggest high expectations. This is surprising since only a very small portion of the envisioned COIN was implemented. The high expectations correlate with the perceived importance (see Figure 41 on page 280) and emphasize the universal character of lessons learned.

10.9 Summary

To validate REFSENO as well as the task hierarchies of Chapter 4 and Chapter 8, an industrial-strength project (Fraunhofer IESE's Corporate Information Network) was used as a case study. The case study presented in this chapter has shown that the task decomposition hierarchy – especially the task decomposition of »record (T33, page 107)« and »structure (T56, page 121)« – is a viable framework for:

- Tailoring the technical infrastructure of an experience base (as presented in this dissertation) to organizational needs
- Analyzing the buildup of experience bases for its strengths and weaknesses
- Building a method library

As the subject area for the Corporate Information Network (COIN), lessons learned about the internal business processes »Set-Up Project« and »Execute Project« were chosen. A survey conducted as part of the initial population of the experience base revealed that:

- The collection of experience is seen already as having the benefit of learning what to watch out for in future projects.
- The explicit documentation of project experience is viewed as being very important.
- Project managers, scientific leads, and members have high expectations regarding the effects of project experience management. This reflects the high potential typically associated with lessons learned in the literature (e.g., [BWP98, Hen95, KS96, KR97, Sar95]).

The case study presents a subject area that is relevant for software development projects (in fact, some of Fraunhofer IESE's projects have been mainly software development projects), but also for other kinds of projects. Although the experience base schema presented in this chapter has been tailored to the specific needs of Fraunhofer IESE, it can be easily tailored to other organizations and domains. For example, a similar subject area was chosen for supporting TQM technically in the health care sector using DISER [ABMN99]. This shows that the approach is not limited to software engineering experience, but can be used for repository-based knowledge management in general.

To complete the validation, the next chapter will present an experiment which was conducted in context of COIN. The experiment validates the overall approach by comparing the »traditional« information acquisition method (»ask your colleague«) with the repository-based approach (reuse of explicitly documented project experience).

11 Empirical Investigation (Experiment)

There is the world of ideas and the world of practice.
Matthew Arnold (English essayist)

As pointed out in the introduction, this thesis focuses on sharing software engineering experience using a repository. In practice, the question arises whether this repository-based approach has advantages over the human-based approach (see below) for obtaining experience on a personal level that could justify the investment in recording experience explicitly. This chapter investigates the benefits of the repository-based approach over the human-based approach:

1 **Repository-Based Approach.** In this approach, the experience base is queried for useful information stored in it. The usage proceeds as described by the task »retrieve (T3, page 89)« and summarized by Figure 37 on page 251.

2 **Human-Based Approach.** In this approach, the information seeker asks one or more persons (i.e., »experts« from the information seeker´s point of view) who will provide (more or less useful) information. In addition, experts may refer to other experts who probably can provide further (more or less useful) information. In this case, the information seeker decides whether to consult the expert(s) based on the information he has obtained so far, that is, if the information seeker suspects a possibly large information gain, he will talk to the expert(s). It is important to note that the information seeker always tries to get a good cost/benefit ratio (time it takes to talk to the expert versus expected information gain) [Har92]. Therefore, the information seeker will not ask additional experts if he feels that the information obtained so far is reasonably sufficient. This process continues until no new experts are suggested or the information seeker decides not to ask any other experts.

More specifically, two hypotheses are investigated:

1 **Efficiency hypothesis.** Using DISER for setting up and running a repository on guidelines and observations, the repository-based approach is superior to the human-based approach in the following sense: The information seeker[1] finds more useful guidelines and observations per time period (in terms of

1 An information seeker is a project team member that seeks experience for a problem or task at hand.

both effort and duration) where »usefulness« is defined as the *perceived* usefulness by the information seeker regarding the guideline or observation at the time of retrieval. It is measured on the subjective scale »useful«/»not useful«/»don't know«.

2 **Effectiveness hypothesis.** Using DISER for setting up and running a repository on guidelines and observations, the repository-based approach finds useful guidelines and observations not obtained by the human-based approach.

To confirm the hypotheses, an experiment was conducted by the author of this dissertation. In an experiment we attempt to identify causal relationships between one dependent variable[1] and one or more independent variables[2].

The two hypotheses specialize those stated in Section 1.3.2 as follows:

- The kind of experience stored in the experience base is restricted to guidelines and observations. Extending the experience base to more kinds of artifacts would have required too much effort on part of those participating in the experiment. Nevertheless, guidelines and observations can be viewed as typical forms of qualitative experiences (see page 266). Because it is more difficult to achieve improvements with qualitative information than with quantitative information (see Section 1.3.2), the results of this experiment can be generalized to other kinds of artifacts.
- Only constellations C1 (corresponding to the human-based approach) and C4 (corresponding to the repository-based approach) are considered because:
  - Constellation C4 is an extension of constellation C3. Therefore, all benefits of C3 are »inherited« by C4. However, in constellation C3, tacit knowledge is not considered for the experience base. But since guidelines and observations are represented textually, they may be hard to comprehend for people who do not know the context from which they originate. Therefore, it may become necessary to contact the originators (authors) of the guidelines and observations to obtain *tacit* knowledge (cf. discussion in Section 1.3.2). To allow this, the originators' names are part of the characterizations of the guidelines and observations (cf. attributes of concept »Process Info« in Appendix E).
  - C2 was not considered because at Fraunhofer IESE, where the empirical investigation was conducted, everybody who could play the role of a

1 For example, in software engineering, a *dependent (or response) variable* of interest is a quality, efficiency, or effectiveness measure [Pfl95].

2 An *independent (or state or explanatory or treatment) variable* is a variable which may have a direct influence on the studied dependent variable [Pfl95].

knowledge broker is a scarce resource. Hence, they were not available to the extent needed for conducting the experiment.

- Participants query the experience base directly; a dedicated project supporter is not available for mediating between the information seeker and the experience base. The reasons for this choice are:
  - If a dedicated project supporter would be employed, the information seeker would have to explain both the project situation and the information need to the project supporter. Since guidelines and observations are meant to be sought after often, the project supporter would become a scarce resource – information seekers would »line up« at the experience factory. Consequently, many people would do without the experience factory.
  - There was no unbiased person available who could take on the role of the project supporter. The only person who knew the contents of the experience base was the author of this dissertation because he performed 23 out of 24 project analysis interviews and recorded the corresponding guidelines and observations in the experience base. However, as the developer of DISER, the author of this dissertation is biased. As a second choice, people who have an overview of Fraunhofer's projects could have acted as project supporters. However, they are scarce resources and were not available to the extent needed for conducting the experiment.
  - A learning effort on part of the project supporter (i.e., the last participants of the investigation would have been consulted more effectively) would have biased the results of the investigation.

The validation of both hypotheses depends on the quality of the experience base. Only if the guidelines and observations (and their characterizations) exhibit a minimal quality, they will be perceived to be useful (see Section 9.1). In addition, the adequacy of the experience base schema will influence the perceived usefulness. DISER provides an »intelligent guess« for the schema (by applying GOODSEE) and ensures a systematic recording of experience with a minimal quality (by applying its task framework). Therefore, the phrase »using DISER« is essential for the hypotheses. An unsystematic collection and arbitrary characterization of the experience will yield a validation of the hypotheses by chance.

The way experience is obtained also plays an important role for the validation of the effectiveness hypothesis. If an information seeker asks an expert, the information seeker will obtain experience that is tailored to his project situation. In contrast, experience stored in a repository reflects the project situation in which it was gained. Since two project situations are rarely exactly the same, it is likely that the experience retrieved by the repository-based approach complements the experience obtained by the human-based approach. Further discussions on the effectiveness and efficiency hypothesis can be found in Section 1.3.2.

11.1 The Experiment

The experiment was run once at the Fraunhofer IESE as part of a trial run of Fraunhofer IESE´s COIN (Corporate Information Network). 29 researchers of Fraunhofer IESE (including 7 group leaders but no department heads) participated in the experiment.

11.1.1 Experimental Design

Because standard designs do not exist for the evaluation of an experience base, the experimental design had to be tailored to the specific needs of this empirical investigation. To avoid misinterpreting the data it is important to understand the experiment and the reasons for certain elements of its design.

Variables

The experiment manipulates the following independent variables:
- The approach for obtaining information (repository-based vs. human-based)
- Background experience of the participants

The approach for obtaining information is the treatment variable. The other variable allows the assessment of several potential threats to the experiment´s internal validity.

For each treatment the following dependent variables are measured:
- Effectiveness (number of useful guidelines and observations)
- Effort needed to obtain the guidelines and observations
- Duration of obtaining the guidelines and observations

Design

The design treats the repository-based and the human-based approach as alternative approaches for obtaining information. Otherwise learning effects would bias the results (i.e., if the same participant applies first the human-based approach alone and then the combination of human-based/repository-based approach or vice versa). During analysis, the results of both approaches need to be combined to confirm the effectiveness hypothesis.

While the repository-based approach is based on actual project experience available at Fraunhofer IESE (i.e., experience not tailored to the project situation asked for in the experiment), the human-based approach is based on guidelines and observations recommended by experts confronted with the project situation of the experiment (i.e., experience tailored to the project situation asked for in the experiment).

Designs where the human-based approach is actually performed during the experiment are unacceptable for this study, because the human-based approach cannot be performed repetitively. This can be understood by analyzing the two possible scenarios:

1 **The same expert gives different answers** to the participants of the experiment (the expert might have gotten new insights or might have forgotten information). In this case, the outcome of the experiment depends on the number (and sequence) of participants that choose the particular expert. Thus, the application of the human-based approach will not be repeatable (learning effect on the expert´s side). This will dictate the outcome of the experiment, that is, important context characteristics change during the experiment.

2 **The expert gives the same answer** to the participants of the experiment. In this case, the expert »imitates« a repository and there is no point in performing the actual human-based approach as part of the experiment.

Furthermore the actual execution of the human-based approach would take too long as not all experts would be available when needed (meetings, travel, vacation, etc.). In addition, it would require too much effort on the side of (some) experts (e.g., some experts are department heads).

Therefore, this experiment compares the results of the repository-based approach with the answers experts gave the first time they were asked for guidelines and observations (during the experiment´s preparation). Thus, the human-based approach is »simulated«. This is done as follows:

Simulation of the human-based approach

1 The participant chooses the experts he would have asked for assistance.

2 The answers of the chosen experts (i.e., the guidelines and observations given by the expert during the experiment´s preparation) are presented to the participant. These may include the hint to consult other experts.

3 The participant may choose to consult some or all of the recommended experts.

4 Steps 2 and 3 are iterated until no new experts are recommended or until the participant decides not to consult any further experts.

5 At the end, the participant is asked
   - to estimate effort and duration it would have taken if he would have actually talked to all selected experts in the chosen sequence and
   - to rate the usefulness of the guidelines and observations provided to him.

The repository-based approach is performed as described in the introduction of this chapter. After the retrieval, the participant determines the usefulness of all retrieved guidelines and observations up to a chosen cutoff point $m \leq n = 30$ beyond which the participant expects no further useful guidelines or observations (see discussion in Section 9.1). Before and after the retrieval, the time is recorded. Breaks and interruptions are recorded as well. This allows to determine the duration and effort needed to perform the repository-based approach.

Both rounds are performed by the same participants (»within-subject« design [BDF96]). Half the participants perform the human-based approach first, the other half do the repository-based approach first. The assignments of participants to groups is done randomly, however, if two participants are sitting in the same office, they are in different groups.[1]

Threats to Internal Validity

A potential problem in any experiment is that some factor may affect the dependent variable without the researcher´s knowledge. This possibility must be minimized. This experiment considered four such threats:

1 **Selection effects** are due to natural variation of human performance. For example, random assignments of participants may accidentally create an elite team. This experiment uses the same treatment for all participants. Hence, selection effects are minimized.

2 **Maturation effects** are due to a naturally occurring process causing a change in the participant´s behavior as the experiment proceeds, e.g., boredom, fatigue, and learning effects. This experiment divides the participants into two groups which perform the two approaches in opposite order (i.e., half the participants perform the human-based approach first, the other half do the repository-based approach first). Hence, maturation effects are leveled out to some degree. To check for boredom and fatigue, participants are asked for their motivation for applying the approaches as part of the experiment´s debriefing.

3 **Replication effects** are due to difference in materials, participants, or executions of multiple replications. For example, less experienced participants may not judge correctly the usefulness of the guidelines and observations returned by the retrieval system. To minimize the replication effects, this experiment measures the background experience of the participants and allows to mark guidelines and observations not only as »useful« and »not

1 At Fraunhofer IESE, at most two researchers sat in one office at the time this experiment was conducted.

useful«, but also as »don't know«. This enables the assessment of how critical the background experience is. As all participants are confronted with the same assignment, replication effects due to difference in materials are not possible.

4 **Instrumentation effects** are due to differences in the measurement procedures. For example, in the human-based approach the participants are asked to estimate the time it would have taken to talk to the selected experts, whereas the time for the repository-based approach is measured directly. In this experiment, these instrumentation effects are minimized by validating the estimates using the actual effort it took to initially obtain the guidelines and observations (during the preparation of the experiment) and the estimates of other participants.

   Another aspect concerns the measurement of the »usefulness« of the presented guidelines and observations. »Usefulness« is a subjective measure, which is not clearly defined, since it depends among others on the experience and background knowledge of the participants (see Section 9.1). In this experiment, these instrumentation effects are minimized by analyzing the degree of reliability of the »usefulness« measure.

Threats to External Validity

Threats to external validity limit our ability to generalize the results of the experiment to industrial practice. For this experiment, the following external threats were identified:

1 Fraunhofer IESE is not representative for software development organizations.

2 The project experience asked for may not be representative for real information needs of Fraunhofer IESE employees (i.e., they may be too hypothetical or too old).

3 The project experience asked for in the experiment may not be representative for projects performed at Fraunhofer IESE.

The first two threats are real. To overcome these, the experiment should be replicated.[1] The first threat can be overcome by replicating the experiment in a software development organization, the second one can be overcome by replicating the experiment with other treatments (i.e., retrieval of other kinds of artifacts).

1 The replications of the experiment lie outside the scope of this dissertation.

To avoid the third threat, a project actually conducted by Fraunhofer IESE was used as the basis for the treatment of the participants. Project members of the respective project were excluded from the experiment.

11.1.2 Experiment Instrumentation

The following instruments were developed for the experiment:

- A textual description of a situation (including a task description) which requires to get guidelines and observations from past projects.
- A questionnaire asking the participants for their background knowledge, the results of obtaining guidelines and observations, and debriefing information (see Appendix F).

Document Presented to the Participants

The textual description of the situation was:

> *Imagine you have to participate (as a team member) in writing a study for X[1]. Although you are not the project manager, you are responsible for making sure that the contents of the deliverable (a Fraunhofer IESE report) will satisfy X. The objective of the project is to survey existing methods in the area of reuse (subject of the study), choose a viable method in accordance with X, and to develop an integration concept for the chosen method for one of X's business units. The project is staffed with one project manager and two researchers from Fraunhofer IESE and one project manager on the side of X. The planned duration is six months. Furthermore, X pays an equivalent of three person months for the effort.*
>
> *What useful project experience exists at Fraunhofer IESE for assuring X's satisfaction regarding the study?*

Methods to be Used by the Participants

The participants were asked to use INTERESTS for finding the information (repository-based approach) and to provide information and estimates for the human-based approach (simulation). For a more in-depth description see the experiment´s design above.

1 The exact organizational unit of the customer was known to the participants of the experiment.

Reports to be Used by the Participants

Each participant was asked to fill out a questionnaire which contained questions to be answered during the experiment (see Appendix F).

11.1.3 Experiment Preparation

35 researchers of Fraunhofer IESE were interviewed for their project experience using the questionnaire of Appendix D (cf. Section 10.7) on a project-by-project basis. The answers were recorded as 7 improvement suggestions, 84 problem solution pairs, 77 observations, and 57 guidelines in the experience base. The latter two made up the information pool for the experiment. The interviewed employees were selected as follows:

- Each researcher who participated in some project was interviewed at least once and at most four times. On the one hand, this ensured that every researcher of Fraunhofer IESE (who participated in a project) had a chance to be represented in the experience base. On the other hand, it avoided overburdening the researchers with too much effort for project analyses.
- Department heads were not interviewed personally, but were asked to complete the project analysis report with their ideas.
- The researchers were members of a representative project for a particular project type (transfer, study, seminar, …). In total, the analyzed projects were representative for all project types of Fraunhofer IESE.

In addition, all researchers of Fraunhofer IESE (including department heads and researchers not previously involved in a project) were asked once what they would have suggested if some new colleague would have approached them with the project situation outlined on page 292. Suggestions could also include the referral to other colleagues (experts). Since all researchers were asked, any researcher could be selected by the participants as an expert (for the human-based approach).

11.1.4 Conducting the Experiment

Training

The usage of INTERESTS was explained briefly to the participants on an individual basis. Other than that, the participants were not trained. However, to learn more about the shortcomings of the user interface and to avoid that »misuse of INTERESTS« influences the results of the experiment, participants were watched during the experiment and were allowed to ask questions about the usage (see below).

Experimental Phase

During this phase, the participants were asked to find useful guidelines and observations for the project situation outlined in the textual description (see page 292) by filling out the questionnaire (see Appendix F) and applying both the human-based and the repository-based approach. The human-based approach was simulated as described in the experiment´s design. The repository-based approach made use of a prototype of INTERESTS. The prototype consisted of CBR-Works, a rudimentary implementation of the Experience Base Server, and a WWW-based implementation of the General Purpose Browser (see Chapter 7). During the experiment, the participants were watched and had the opportunity to ask questions about:

- The textual description of the situation (however, they did not get any information in more detail)
- How to use INTERESTS
- The meaning of the used attributes

There was no time limitation for the experiment.

11.2 Results

This section presents and discusses the results of the experiment. In the first three subsections, questions regarding the reliability of the analysis results are answered:

- Is the simulation of the human-based approach realistic in comparison to the »real« human-based approach? Only a realistic simulation allows the generalization from the experiment´s results to reality.
- Were the participants motivated? Only if all participants are motivated, biases such as boredom and fatigue can be discarded as an influential factor on the results.
- Is the »usefulness« measure reliable? Only if participants do not classify observations and guidelines by chance, the analysis results will be meaningful.

After answering these questions, the effectiveness and efficiency of both approaches are analyzed resulting in a validation or rejection of the hypotheses stated in the beginning of this chapter. Finally, the meaning of the analysis results is discussed for the complete approach presented in this dissertation (i.e., DISER).

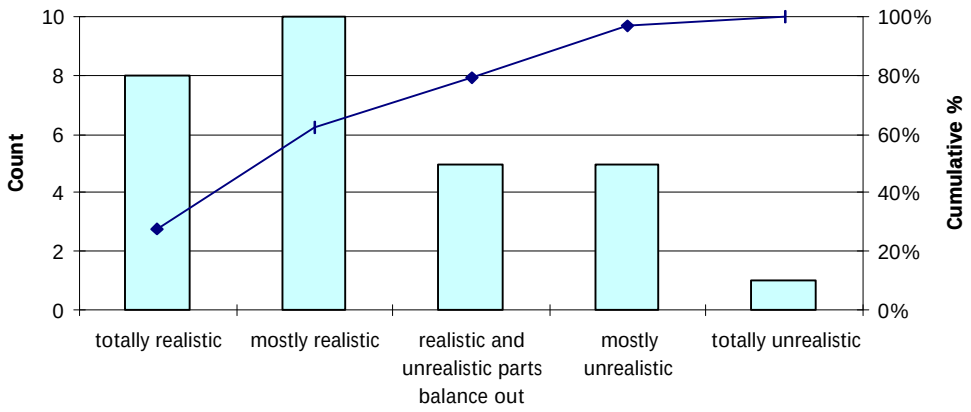
11.2.1 Simulation of Human-Based Approach

As described in Section 11.1.1, the performance of the »real« human-based approach (i.e., actually going to the colleagues) was not feasible for the experi-

ment as it would have strongly biased the results. Therefore, a simulation was used to ensure that all participants received the same answers from the same expert with the same wording. However, this raises the question whether the simulation is close enough to reality so that the results of the experiment can be generalized to the »real« human-based approach.

To analyze how realistic the simulation is in comparison to the »real« human-based approach regarding the gained information, the participants were asked for their estimations as part of the experiment's debriefing. Figure 43 shows the result. The number of times each of the categories »totally realistic«, »mostly realistic«, »realistic and unrealistic parts balance out«, »mostly unrealistic«, and »totally unrealistic« was named, is shown as bars. The line shows the cumulative count starting from the category »totally realistic« in percent. For example, 79.3% rated the simulation as either »totally realistic«, »mostly realistic«, or »realistic and unrealistic parts balance out«.

Figure 43: How realistic is the simulation in comparison to the real human-based approach?



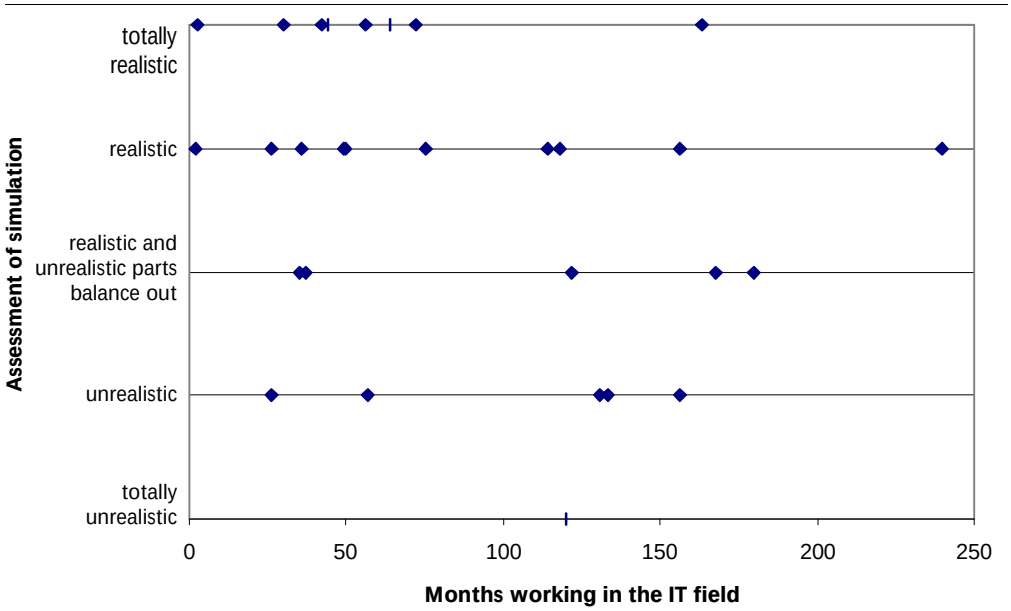
The data indicates that 62.1% of the participants view the simulation as realistic, 17.2% are undecided, and only 20.7% view the simulation as unrealistic. 44.4% of those who find the simulation realistic, view it even as totally realistic, whereas only 1 participant judges the simulation to be totally unrealistic.

Although there are no accepted threshold values at which the simulation can be regarded as realistic, it can be argued that it is the more realistic the higher the percentage of »not unrealistic« ratings is. In the case of the experiment, the simulation can be regarded as quite realistic, because almost 80% of the participants view the simulation not to be unrealistic.

One could postulate that unexperienced participants tend to view the simulation as more realistic due to a lack of experience. However, Figure 44 shows only a slight correlation between the number of months a participant has been working in the IT field and the assessment of the simulation (squared correlation coefficient: 0.0813 for a linear trend analysis where the value 0 means no corre-

lation). Moreover, none of the participants who have been working for more than 13 years (156 months) in the IT field assessed the simulation to be unrealistic.

Figure 44: Do experience and assessment of the participants correlate?



This analysis allows to generalize the data from the simulation to the »real« human-based approach with a considerable degree of confidence.

11.2.2 Motivation of Participants

As pointed out in the section on internal threats (see page 290), fatigue and boredom can bias experiment results. Therefore, the participants were asked for their motivation for performing each of the approaches. The results are shown in Figure 45 and Figure 46.

For the human-based approach, the data indicates that 82.8% of the participants were motivated. Of those motivated participants, 38% were even extremely motivated. None of the participants were unmotivated and only 17.2% were neither motivated nor unmotivated.

For the repository-based approach, 79.3% of the participants were motivated. 30.4% thereof were even extremely motivated. Only one participant was unmotivated and only 17.2% were neither motivated nor unmotivated (as in the human-based approach).

Figure 45: How motivated were the participants for applying the human-based approach?

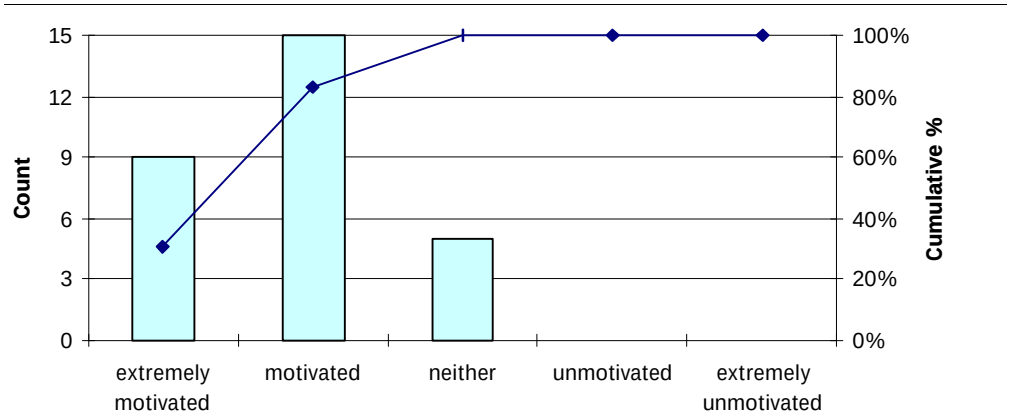
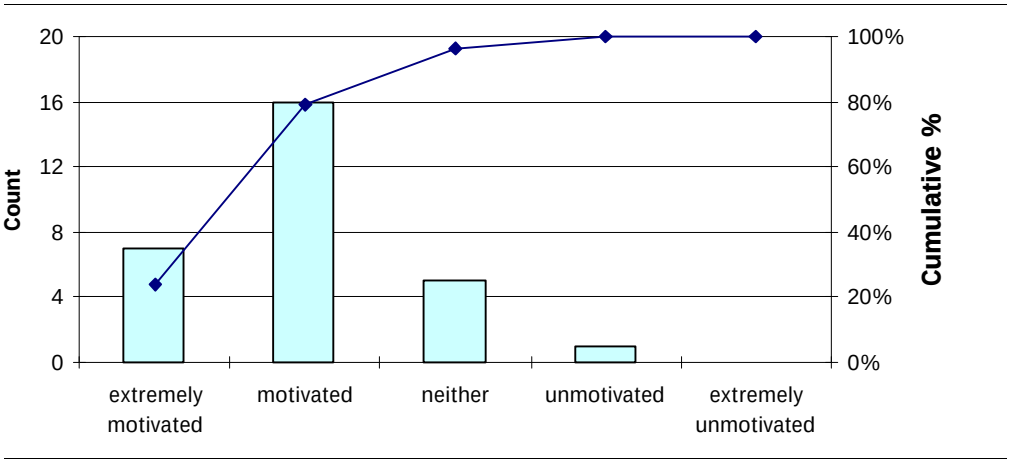


Figure 46: How motivated were the participants for applying the repository-based approach?



From this data (none of the participants were unmotivated for the human-based and only one out of 29 participants was unmotivated for the repository-based approach), it can be concluded that the participants were sufficiently motivated. Therefore, the probability that fatigue and boredom biased the experiment results is small.

11.2.3 Reliability of the »Usefulness« Measure

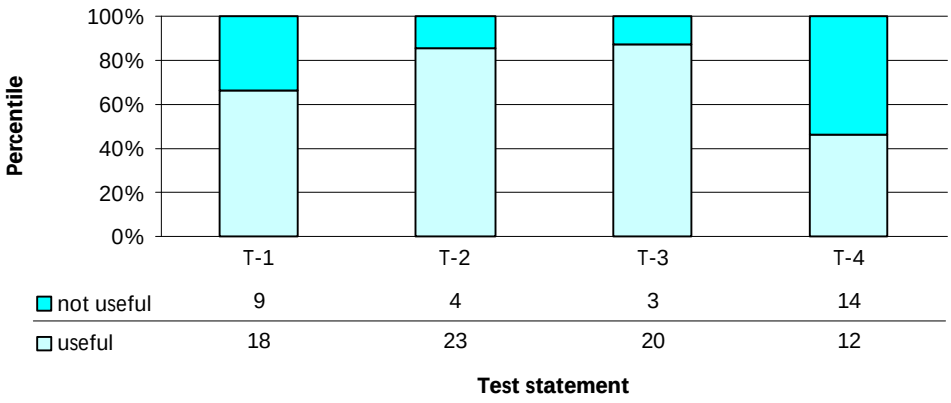
To ensure the reliability of the analyses, it is necessary to prove that the »usefulness« measure is internally consistent. The internal consistency of the »usefulness« measure describes whether each of the participants classified a guideline or observation as »useful« or »not useful« purely by chance or not. For this purpose, all participants were asked to rate four test statements regarding various aspects and to provide information on their background experience. Table 28 shows the answers given by the participants.

Table 28: Answers given by the participants for the test statements T-1 through T-4 and their background experience

	T-1				T-2				T-3				T-4				background experience					
no.	**unexpected/not unexpected**	**known/partially known/unknown**	**sensible/not sensible/?: don't know**	**useful/not useful/?: don't know**	**unexpected/not unexpected**	**known/partially known/unknown**	**sensible/not sensible/?: don't know**	**useful/not useful/?: don't know**	**unexpected/not unexpected**	**known/partially known/unknown**	**sensible/not sensible/?: don't know**	**useful/not useful/?: don't know**	**unexpected/not unexpected**	**known/partially known/unknown**	**sensible/not sensible/?: don't know**	**useful/not useful/?: don't know**	**involved as project member in public projects 0 projects, 1 project; >1: more than 1 project**	**involved as project member in industrial projects 0 projects, 1 project; >1: more than 1 project**	**involved as project member in study 0 projects, 1 project; >1: more than 1 project**	**involved as project manager in projects 0 projects, 1 project; >1: more than 1 project**	**time working for Fraunhofer IESE (in months)**	**time working in the IT field (in months)**
1	nu	p	s	u	nu	k	s	u	nu	u	s	?	nu	k	s	?	0	0	0	0	2	2
2	nu	k	s	u	nu	p	s	u	nu	u	s	?	u	k	s	u	1	1	0	0	3	3
3	nu	k	s	nu	nu	p	s	u	nu	u	s	?	nu	k	ns	nu	0	1	0	0	3	57
4	nu	k	s	u	nu	u	s	u	nu	u	?	nu	u	k	?	nu	>1	1	>1	>1	9	64
5	nu	k	s	nu	nu	k	s	nu	nu	k	s	u	nu	k	s	nu	1	1	0	1	13	156
6	nu	k	s	u	u	u	s	u	u	u	s	?	u	k	s	nu	1	1	0	1	16	56
7	nu	k	s	nu	nu	k	s	nu	nu	k	s	u	nu	k	s	nu	1	>1	0	1	16	131
8	nu	k	s	u	nu	p	s	u	nu	p	s	?	nu	k	s	u	1	>1	0	>1	17	180
9	nu	p	s	u	nu	p	s	u	u	u	s	?	nu	p	s	u	1	>1	0	0	18	36
10	nu	k	ns	nu	nu	p	s	u	nu	u	s	?	nu	k	ns	nu	>1	>1	0	0	20	122
11	nu	k	s	nu	nu	k	s	nu	u	u	s	?	nu	k	s	nu	1	>1	1	>1	22	240
12	nu	k	s	u	nu	k	s	u	u	u	s	?	nu	k	s	nu	0	0	0	0	23	26
13	nu	p	s	u	nu	p	?	?	u	u	s	?	u	k	s	nu	1	1	1	0	25	50
14	nu	k	s	u	u	p	s	u	nu	p	s	?	nu	k	s	u	1	>1	0	0	26	26
15	nu	k	s	u	nu	p	s	u	nu	u	s	?	nu	k	?	u	0	>1	>1	>1	27	168
16	nu	k	s	nu	nu	p	s	u	u	u	?	nu	nu	k	s	nu	1	>1	0	1	30	30
17	nu	p	?	u	nu	p	s	u	nu	p	s	?	nu	k	s	u	1	1	1	0	30	42
18	nu	p	s	u	u	p	?	nu	nu	p	s	?	nu	k	s	u	1	>1	0	0	35	35
19	nu	k	s	u	nu	p	s	?	u	u	?	nu	u	k	s	nu	1	1	1	0	36	49
20	u	u	?	?	nu	p	s	u	u	u	s	?	u	u	?	?	>1	>1	>1	0	37	37
21	u	k	?	?	u	p	s	u	u	u	s	?	u	k	s	nu	0	>1	>1	>1	37	118
22	nu	k	ns	nu	u	p	s	u	u	u	s	?	u	p	?	u	1	>1	>1	>1	37	163
23	nu	k	s	nu	u	p	?	u	nu	p	s	nu	nu	k	s	nu	1	>1	>1	1	40	75
24	nu	k	s	u	nu	k	s	u	u	u	?	nu	u	k	s	u	1	1	0	0	44	44
25	nu	k	s	nu	nu	k	s	u	u	u	ns	u	u	k	s	nu	1	>1	1	1	44	114
26	nu	k	s	u	nu	k	s	u	u	u	s	?	nu	k	s	u	1	>1	>1	0	46	120
27	nu	k	s	u	nu	k	s	u	u	u	?	nu	nu	k	s	u	>1	>1	1	>1	48	156
28	nu	k	s	u	nu	p	?	u	u	u	s	?	nu	k	ns	?	0	>1	>1	>1	49	72
29	nu	k	s	u	nu	k	s	u	u	p	s	?	nu	k	s	u	>1	>1	1	0	50	134

Figure 47 shows a summary over the usefulness ratings of the test statements. The figure excludes »don't know« answers. Thus, there are less than 29 answers for each of the test statements. The »usefulness« measure is consistent among all participants if all participants give the same answer, i.e., if either the »useful« or the »not useful« portion of a bar is 100% high. In case of total disagreement, both bars have the same height (50%). In the figure, the bars for T-2 and T-3 show a fair amount of agreement with a height of 85% and 87% respectively. However, the bars for T-1 and T-4 show little agreement with a height of 67% and 54% respectively. Overall, there is only a slight agreement among the participants.

Figure 47: Do the participants agree on the usefulness of the test statements?



The slight agreement regarding the usefulness is not surprising since the perception of the usefulness of a guideline or observation depends (among other factors) on the past experience of the participants. To validate this, a classification tree was built on the results of the four test statements using the classification and regression tree (CART) algorithm [BFOS97]. The CART algorithm builds classification trees with minimal impurity of the leafs (called terminal nodes). The impurity of a node *N* is defined as

$$i(N) = 1 - \frac{|n_u - n_{nu}|}{n_u + n_{nu}}$$

where n_u and n_{nu} are the number of useful and not useful statements respectively in a node *N*. A node´s impurity is at its maximum if the node contains an equal number of »useful« and »not useful« statements. It is smallest if only »useful« or only »not useful« statements are part of a node. A node is split into subnodes if the impurity of the subnodes is an improvement over the impurity of the node split. Figure 48 shows the generated classification tree for the experiment (statements rated with a »don't know« in either the usefulness or sensibleness category were not considered). The tree was generated using »useful/not useful« as the dependent variable and the following independent variables: »unexpected/expected«, »known/partially known/unknown«, »sensible/not

sensible«, »project member in public projects«, »project member in industrial projects«, »project member in study«, »project manager in projects«, »time working at Fraunhofer IESE«, and »time working in the IT field«.

In the figure, each node is represented as a box and contains its number in the upper left corner, the number of »not useful« and »useful« statements as bars on the bottom, and whether there are more »useful« or »not useful« statements in the upper right corner. The predicate below nonterminal nodes splits the node into two subnodes. Those ratings fulfilling the predicate are on the left, those not fulfilling the predicate are on the right. For example, the top node (number 1) shows that 95 times a test statement has been rated. In 39 of those ratings, a test statement has not been marked with »known« (meaning they have been marked as either »partially known« or »unknown«). This means that the 56 other ratings rated a test statement as »known«. At terminal nodes, the impurity is given.

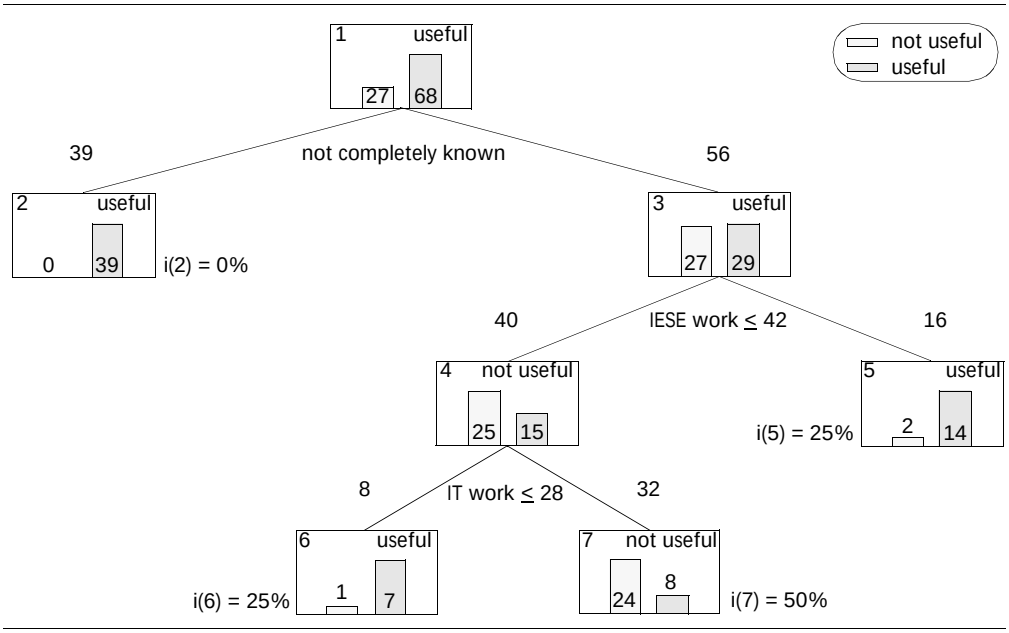


Figure 48: Classification tree for usefulness measure

On the top level, the tree is split by the predicate »are the contents of a statement completely known to the participant?«. If the contents are not completely known, the statement is most likely to be classified as useful (in all 39 cases where a statement was unknown or partially known to the participant, the statement was classified as useful, corresponding to 100% consistency). In case a statement was completely known, the background experience of the participants influenced how the statement was classified. In the experiment, the consideration of experience in the various kinds of projects (public, industrial, etc.) did not lead to a split that would reduce the impurity. Instead, the time participants spent at Fraunhofer IESE and were working in the IT field can be used as a

predictor for classifying statements. In the experiment, those working for more than 42 months at the institute tended to classify (completely) known statements as »useful« (impurity at node 5: 25%). Participants working in the field of information technology for less than 28 months also tended to classify (completely) known statements as »useful« (impurity at node 6: 25%). Intermediate experienced participants (i.e., participants working less than 42 months at the institute, but more than 28 months in the IT field) tended to classify (completely) known statements as »not useful«. This terminal node has an impurity of 50%, meaning that three fourths of the statements classified by intermediate experienced participants are classified as »not useful«.

The classification tree can be used to predict the usefulness of a statement based on the background experience of the participant and whether the statement is known to the participant or not. In a threefold cross validation[1], approximately 81% of the ratings were classified correctly. Interestingly, the other (subjective) ratings (expectancy and sensibleness) did not lead to splits with a lower impurity indicating that the background experience of the participants is a better predictor for usefulness than the participants´ rating of unexpectancy and sensibleness.

In conclusion, all not completely known statements are classified reliably as »useful« and all completely known statements are classified based on the experience of the participants regarding the IT field in general and Fraunhofer IESE in particular. Therefore, in case of unknown or partially known statements the usefulness measure is very reliable and consistent across all participants of the experiment. For known statements the reliability of the usefulness measure is influenced by factors that cannot be controlled such as past experience, which differs on an individual basis. Thus, the perception of usefulness cannot by seen as a by-chance-classification, enabling further analyses based on the »usefulness« measure.

In addition, this analysis provides empirical, quantitative evidence for assumptions formulated purely qualitative in the past, that is, that the perceived usefulness of qualitative experience depends (among others) on the background experience of the information seeker (cf. Figure 36 on page 248).

1 This technique divides all 95 samples into three partitions. Then a classification tree is built using two of those partitions. Finally, the remaining partition is used to predict (and verify) the usefulness. This is done with all pairs of the three partitions.

11.2.4 Analysis of Effectiveness

To analyze the effectiveness, the number of useful guidelines and observations resulting from applying each approach were analyzed. In case of the human-based approach, duplicate guidelines or observations were counted as one information item (i.e., if the statement of one expert included the statement of another). For example if one expert gave the answer »pay 100% attention that it is clear what the customer wants« and another »be in contact with the customer; interview the customer regarding objectives, needs, and expectations«, then the second statement covers the first one, i.e., the first does not provide any additional information. Table 29 shows the data set. One of the participants did not ask any colleagues for the human-based approach. Therefore, for this participant, the human-based approach was not effective. Figure 49 shows the distribution of the number of useful guidelines and observations provided by the two approaches (for the legend of the box plot see Figure 50). For the human-based approach, the outlier with no useful guidelines or observations was not further considered. Therefore, the distribution of the effectiveness of the repository-based approach is based on 29 cases, whereas the effectiveness of the human-based approach is based on only 28 cases.

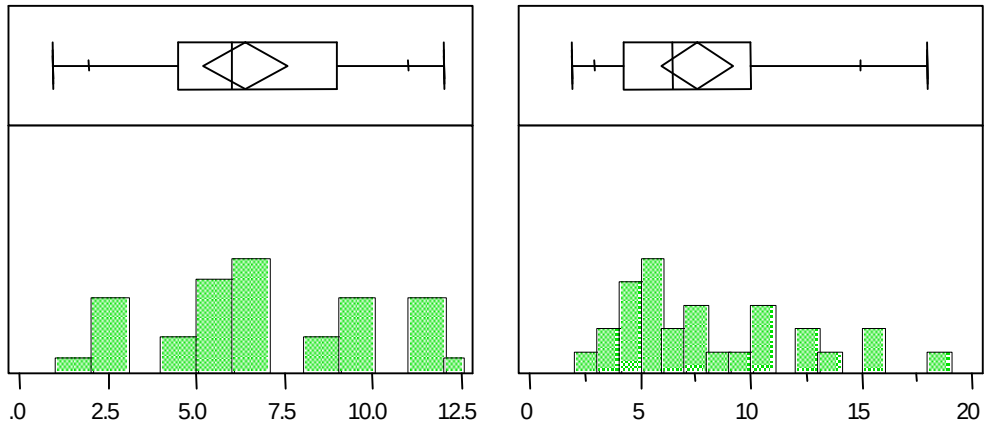


Figure 49: Distribution of the effectiveness (measured in number of useful guidelines and observations) of the repository-based approach (left) and the human-based approach (right)

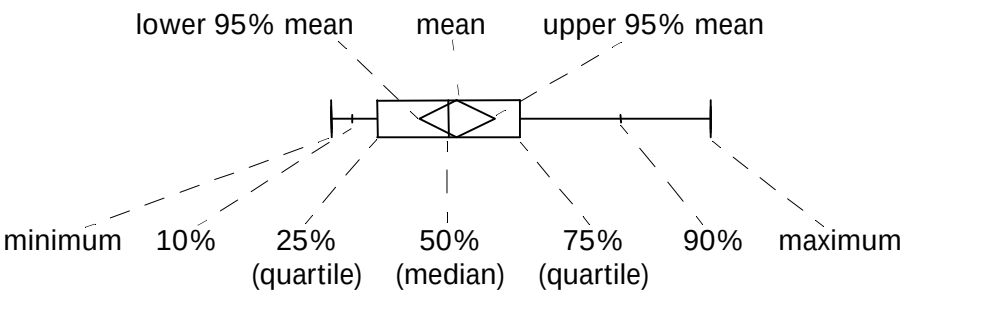


Figure 50: Legend for box plots

Table 29: Effectiveness of the repository-based (rba) and human-based approach (hba)

	number of useful statements		number of unexpected statements		relative improvement (in %)	
no.	**rba**	**hba**	**rba**	**hba**	**rba over hba**	**hba over rba**
1	1	6	1	6	16,7	600,0
2	2	5	1	3	20,0	150,0
3	2	6	2	6	33,3	300,0
4	2	7	2	7	28,6	350,0
5	2	0	2	0	undefined	0,0
6	4	7	3	6	42,9	150,0
7	4	15	4	15	26,7	375,0
8	5	10	3	9	30,0	180,0
9	5	18	4	17	22,2	340,0
10	5	5	5	4	100,0	80,0
11	5	7	5	7	71,4	140,0
12	5	3	5	2	166,7	40,0
13	6	10	4	7	40,0	116,7
14	6	5	4	4	80,0	66,7
15	6	13	5	11	38,5	183,3
16	6	5	5	4	100,0	66,7
17	6	3	5	2	166,7	33,3
18	6	9	6	8	66,7	133,3
19	8	5	6	3	120,0	37,5
20	8	4	7	3	175,0	37,5
21	9	8	6	5	75,0	55,6
22	9	15	7	13	46,7	144,4
23	9	2	8	1	400,0	11,1
24	9	12	8	9	66,7	100,0
25	11	12	8	9	66,7	81,8
26	11	4	9	3	225,0	27,3
27	11	4	10	1	250,0	9,1
28	11	4	10	2	250,0	18,2
29	12	10	12	10	120,0	83,3

In the experiment, the repository-based approach provided between 1 and 12 useful guidelines and observations (average: 6.4), whereas the human-based approach provided between 2 and 18 (average: 7.6). The actual average based on the sample lies between 5.2 and 7.6 for the repository-based approach and between 6.0 and 9.2 for the human-based approach with a probability of 95% (see lower and upper 95% mean of the box plots). This indicates a slightly better effectiveness of the human-based approach over the repository-based approach in the experiment (measured in the absolute number of useful guidelines and observations provided). The effectiveness of the human-based approach is skewed toward the lower end. This means that the effectiveness of the human-based approach is not evenly distributed and that it is more likely that the approach provides less useful guidelines and observations than the average suggests. In the experiment, the human-based approach provided up to 6.5 useful guidelines and observations in half of the cases and between 6.5 and 20 useful

guidelines and observations in the other half of the cases. The repository-based approach is more evenly distributed. It provides up to 6 useful guidelines and observations half of the time and 6 to 12 the other half of the time.

To confirm the effectiveness hypothesis (page 286), the useful but unexpected guidelines and observations[1] were analyzed (Table 29). A guideline or observation is unexpected if (a) the same information is not provided by the other approach and (b) the information was not provided by the participant himself. The distribution of the useful but unexpected guidelines or observations is shown in Figure 51.

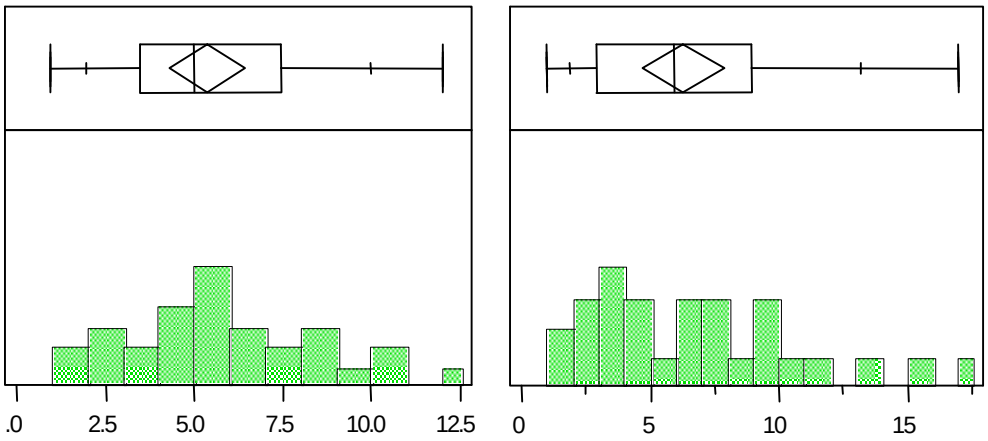


Figure 51: Improvement of one approach by the other, measured in the number of additional useful guidelines and observations (left: improvement of human-based approach by repository-based approach; right: vice versa)

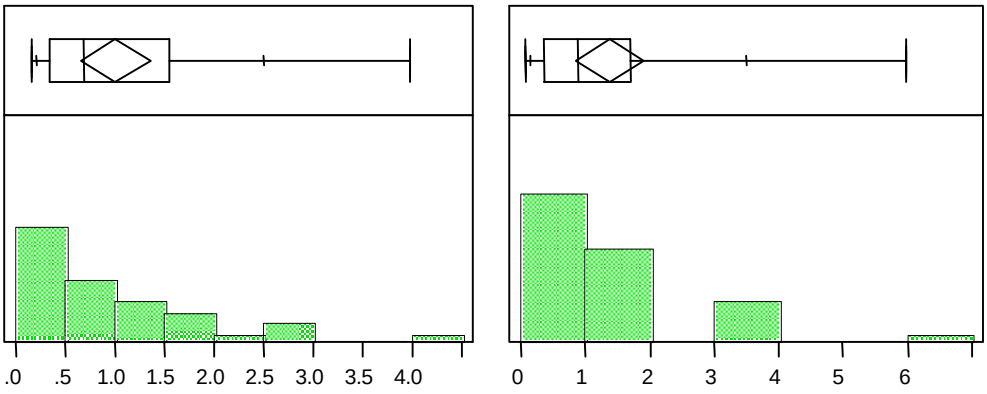
Each useful but unexpected guideline or observation of one approach complements the information provided by the other approach. Therefore, any positive number of such items will enhance the other approach. The results show that in all cases the repository-based approach complements the human-based approach (minimum is greater than 0 in Figure 51). Vice versa, in 28 out of 29 cases (96.6%) the human-based approach also complements the repository-based approach. The only case where the human-based approach did not complement the repository-based approach was due to the fact that the participant did not ask anybody in the human-based approach.

Although the usage of both approaches in combination will result in an improvement of effectiveness, it is still of interest by how much the effectiveness is improved. If the approaches complement each other only little, it would not be worth the effort for performing both approaches.

1 These (objectively defined) unexpected guidelines and observations should not be confused with the participants´ subjective ratings whether they did not expect such a guideline or observation (cf. Table 28).

To investigate this further, the relative improvement was computed for all treatments (see Table 29). The relative improvement of an approach A over approach B is defined as the number of useful but unexpected guidelines and observations provided by A over the number of useful guidelines and observations provided by B. Therefore, an improvement of 1 means that A finds as many additional useful guidelines and observations as B has already provided if A is performed after B. The improvement distributions are shown in Figure 52.

Figure 52: Relative improvement of one approach by the other (left: relative improvement of human-based approach by the repository-based approach; right: vice versa)



Both distributions are skewed toward the lower end of the relative improvement. However, an improvement of up to 400% (with the repository-based approach) and 600% (with the human-based approach) are possible. In the experiment, the repository-based approach complemented the human-based approach by 101.7% on the average, whereas the human-based approach improved the repository-based approach by 139.7% on the average. The minimum improvement of the repository-based approach (over the human-based approach) was 17%. Based on the data from the experiment, it can be concluded that the real average improvement of the repository-based approach lies between 66.3% and 137.0% with a probability of 95%. Using a t-Test, the probability that the average improvement is less than or equal to 50% can be computed. It is below 0.5% (p-value 0.0033)[1].

1 The t-Test can be applied for normally distributed samples or samples with a size greater than 30 [BB87, Cap88]. In the latter case, the normal distribution requirement can be dropped. In our case, the sample size was 28. Therefore, a sensitivity analysis for the three missing values with worst case assumptions for the relative improvements (i.e., 0.0% improvement each) was performed. The analysis showed stable results regarding the average improvement of less than or equal to 50% on a significance level of 1% (p-value: 0.0091).

Therefore, it can be concluded that the repository-based approach complements the human-based approach substantially. Hence, the effectiveness hypothesis has been validated.

It is interesting to compare this result to the subjective rating of the participants. For this purpose, the participants were asked which of the approaches they would apply in practice during the debriefing. 28 of the participants (96.6%) would combine the repository-based approach with the human-based approach. Only one participant would apply the human-based approach solely. This clearly supports the analysis results that both approaches should be combined.

11.2.5 Analysis of Efficiency

To analyze the efficiency of both approaches, both effort and duration for applying the approaches were collected. In case of the human-based approach, effort and duration were estimated by the participants since the simulation required less effort than the »real« approach. Table 30 shows the data from the experiment. Figure 53 and Figure 54 show the distribution of the effort and duration per useful guideline or observation respectively.

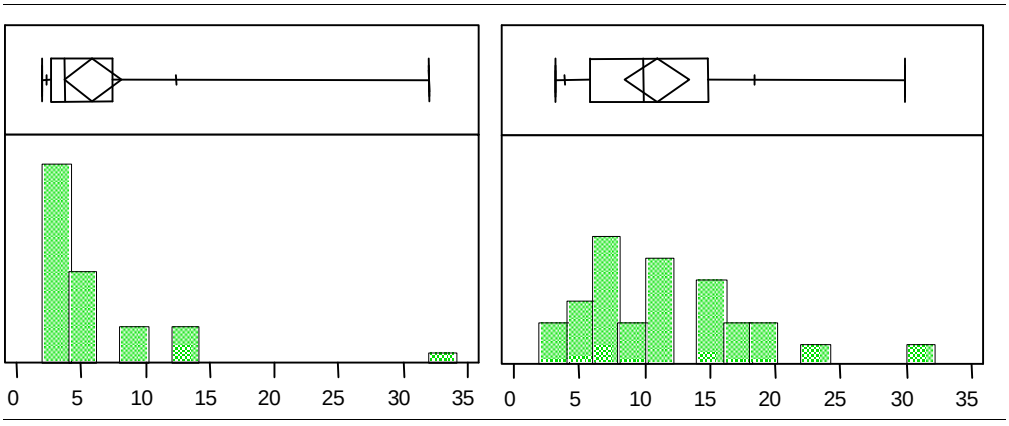


Figure 53: Efficiency of the repository-based (left) and human-based approach (right) measured in effort (minutes) per useful guideline or observation

Regarding the effort per useful guideline or observation, both approaches had an outlier (repository-based approach: 32 minutes, human-based approach: 30 minutes). In case of the repository-based approach, this outlier was due to the fact that the participant found only one useful guideline/observation. However, in 90% of the cases, the effort per useful guideline or observation was 12.5 minutes or less, and in 50% of the cases, the effort was 3.8 minutes or less. In the case of the human-based approach, the outlier was due to a corresponding high estimation of the needed effort (it was the second highest effort estimation for talking with three people and »only« six useful guidelines and observations

Table 30: Efficiency of the repository-based and human-based approach

	repository-based approach					human-based approach				
no.	number of useful guidelines and observations	effort	duration	effort per useful guideline or observation	duration per useful guideline or observation	number of useful guidelines and observations	effort (min)	duration (min)	effort per useful guideline or observations (min)	duration per useful guideline or observation (min)
1	1	32	32	32	32	6	20	2880	3	480
2	2	19	19	10	10	5	30	8640	6	1728
3	2	25	25	13	13	6	60	180	10	30
4	2	25	25	13	13	7	30	2880	4	411
5	2	25	25	13	13	0	0	0	undefined	undefined
6	4	12	12	3	3	7	120	1680	17	240
7	4	37	37	9	9	15	90	1680	6	112
8	5	20	20	4	4	5	150	4320	30	864
9	5	20	20	4	4	10	150	1680	15	168
10	5	25	25	5	5	18	90	10080	5	560
11	5	29	29	6	6	7	60	5760	9	823
12	5	47	47	9	9	3	30	120	10	40
13	6	15	15	3	3	5	70	10080	14	2016
14	6	16	16	3	3	10	60	60	6	6
15	6	20	20	3	3	5	90	4320	18	864
16	6	22	22	4	4	9	30	2880	3	320
17	6	23	23	4	4	3	30	30	10	10
18	6	25	25	4	4	13	90	240	7	18
19	8	28	28	4	4	4	30	240	8	60
20	8	35	35	4	4	5	90	240	18	48
21	9	22	22	2	2	15	60	4320	4	288
22	9	24	24	3	3	2	30	240	15	120
23	9	28	28	3	3	8	90	20160	11	2520
24	9	48	48	5	5	12	75	225	6	19
25	11	23	23	2	2	4	90	1440	23	360
26	11	25	25	2	2	12	210	3120	18	260
27	11	30	30	3	3	4	45	1440	11	360
28	11	40	40	4	4	4	60	2880	15	720
29	12	30	105	3	9	10	90	2160	9	216

resulted). However, the 90% and 50% marks lie higher than for the repository-based approach (18.5 and 10 minutes respectively).

If – instead of the effort – the duration until the wanted information is actually provided to the information seeker is used as a basis, the difference between the repository-based and human-based approach becomes even more obvious. The duration of a repository-based approach is only slightly longer than the time

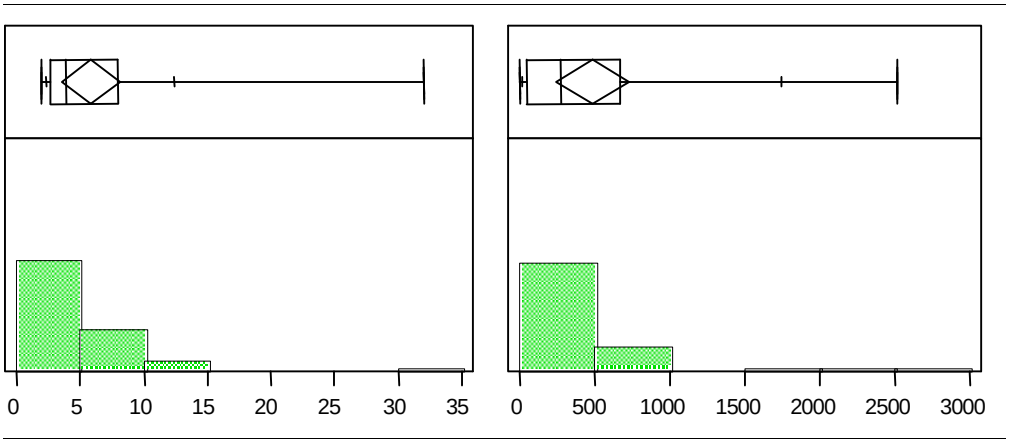


Figure 54: Efficiency of the repository-based (left) and human-based approach (right) measured in duration (minutes) per useful guideline or observation

to actually search for information, because the search is typically interrupted only shortly (e.g., by a telephone call or lunch break). In contrast, the human-based approach requires the availability of the expert to be interviewed. However, experts are often a scarce resource. This is clearly reflected by the data of the experiment. Although both approaches have an outlier again (32 minutes for the repository-based approach and 2 weeks for the human-based approach), the 50% and 90% marks show clearly the »waiting time« for experts (4.6 and 29.3 *hours* respectively per useful guideline or observation for the human-based approach versus 3.9 and 12.5 *minutes* for the repository-based approach).

Since learning effects[1] (see page 290) cannot be ruled out due to the »within-subject« design of the experiment, a one-tailed Wilcoxon matched-pairs signed-ranks test [She97] was used to confirm the efficiency hypothesis. The test can be based on both the effort and the duration. Based on the effort, the Wilcoxon test returns a p-value of 0.0016. This means: If we reject the hypothesis that both approaches have the same average efficiency (in favor of the efficiency hypothesis), there is a probability of 0.16% that the rejected hypothesis is actually true. Typically, significance levels of 1-5% are used, that is, the p-value should be lower than the significance level. Therefore, the hypothesis that both approaches have the same average efficiency can be rejected at a statistical significance level well below 0.5%. The rejection is in favor of the efficiency hypothesis, because the average efficiency of the repository-based approach is better than the average efficiency of the human-based approach. The hypothesis can be rejected at an even lower statistical significance level (p-value $2x10^{-6}$) based on the duration.

1 A more detailed discussion of learning effects in this experiment can be found in [TANO00].

Therefore, the repository-based approach is more efficient than the human-based approach.

11.2.6 Discussion of Results

The results of the experiment clearly demonstrate the benefits of the repository-based approach. These results have to be seen in the following context.

Information provided by hum1an-based approach had longer history

The experience base contained guidelines and observations of 24 projects which were either completed in 1999 or had started in 1999. Projects completed in 1998 or earlier were not analyzed. Although the exact average of the time people have been working at Fraunhofer IESE cannot be easily determined, it can be approximated by the average working time of the participants (27.7 months). Hence, the information provided by the human-based approach was based on a longer history than the information provided by the repository.

Experiment demonstrated benefits in an area where human strengths lie

For the experiment, qualitative experience was used. Guidelines and observations were – to a large degree – general tips and tricks as typically provided by humans (in fact, they were provided by humans as part of project analyses). Although a strength of humans, the experiment showed encouraging results in comparison. Therefore, it can be expected that the repository-based approach provides even more benefits if not only qualitative, but also quantitative experience is sought after.

Contents of experience base have improvement potential

Furthermore, no minimal quality requirements were known initially as described in Section 10.5. Therefore, the only quality measure used was the assessment of the author of this dissertation during recording: Only specific guidelines and observations (e.g., »Plan work packages that can be finished with an effort of two person weeks.«) were recorded; common sense guidelines and observations (e.g., »Improve planning«) were not recorded. Comments by the participants of the experiments revealed some important quality aspects: Comprehensibility and detail. For instance, participants indicated that they do not have a high confidence in observations that are expressed in one or two sentences, because they do not know on what prerequisites the description depends (i.e., they have a hard time deciding on the applicability for their project situation). For guidelines, the confidence is higher in general because the underlying problem describes some of the prerequisites. Therefore, an easy way to raise the quality of the experience base's contents is to have the guidelines and observations reviewed by someone who was neither involved in the interviewed project nor in its analysis.

Another possibility to improve the experience base is to provide better guidance by improving its schema. The schema has been developed using the goal-oriented, systematic method GOODSEE based on lessons learned schemas used in other industrial-strength projects (see Chapter 10). Yet, it can only be seen as an

»intelligent guess« for the environment »Fraunhofer IESE«. During the preparation of the experiment, some interviewed researchers suggested to also include application domains of the industrial partners and competencies for the context characterization of guidelines and observations. An analysis of the queries issued during the experiment suggests similar improvements: Participants often entered a competence like »reuse« as a keyword for the description of the guidelines and observations. Other keywords used included »survey« and »study«. However, these keywords did not match because during project analyses, the term »report« was preferred. Therefore, the introduction of a controlled vocabulary (cf. Section 5.1) will improve both effectiveness and efficiency of the repository-based approach.

Experience base infrastructure has improvement potential

Next to the improvements regarding the contents, the technical infrastructure can be improved. For instance, feedback of the participants suggests that the handling of the General Purpose Browser can be improved.

All these points suggest that the repository-based approach will grow even more beneficial over time if the suggested improvements are implemented and the experience base is kept up-to-date. In addition, the inclusion of the originators' names in the characterization of the guidelines and observations will lead to a continuous improvement: Unless the originators are keen of being asked the same questions all the time (because their guidelines and observations are hard to comprehend), they will improve/extend the description of their guidelines and observations. Hence, the quality of the guidelines and observations will improve over time. The necessity to contact the originators will decrease. This will in turn increase efficiency.

The results of the experiment also validate DISER because the experience base used in the experiment was set up using this methodology. The experience base was structured and populated as described in Chapter 8 and Chapter 4 respectively (see Chapter 10). During the experiment, INTERESTS, a prototype of the architecture as presented in Chapter 7, was used. Finally, the participants followed the usage model of OMI and provided feedback regarding the information they would like to see in the future (cf. Chapter 9). The experiment even provided empirical, quantitative evidence that the user´s experience influences the perceived usefulness of qualitative experience (see Section 11.2.3).

The experiment results also indicate that care has to be taken if they are to be transferred to other environments. First of all, the external threats listed on page 291 have to be considered. Second, the experience (and culture) of people at other organizations may be different. Therefore, experiments such as the one described in this chapter should be part of an experience base´s introduction. It allows to receive early feedback from future users, gives the opportunity to train the people regarding the usage of the experience base[1], and at the same time encourages and motivates people to use the experience base (provided that the experiment´s results are positive).

11.3 Summary

To confirm the hypothesis that the repository-based approach using the infrastructure presented in this dissertation (knowledge representation formalism REFSENO (Chapter 6) and the retrieval tool INTERESTS (Chapter 7)) is both more effective and efficient than the purely human-based approach (i.e., talking to colleagues), an experiment was conducted.

To prepare the experiment, 35 researchers of Fraunhofer IESE were asked for guidelines and observations in case they would be confronted with the project situation *S*[1]. This was the basis for the human-based approach. In addition, the experience base was populated with project experience (including 77 observations and 57 guidelines) of the researchers. This was the basis for the repository-based approach.

In the experiment, the participants were asked to find useful project experience available at Fraunhofer IESE for the project situation *S*. All participants applied both the human-based and the repository-based approach to find useful guidelines and observations and marked all observations and guidelines as either »useful«, »not useful«, or »don't know«.

To validate the effectiveness hypothesis (i.e., the repository-based approach complements the human-based approach by providing additional useful observations and guidelines), it was analyzed how many guidelines and observations were marked as »useful«. The analysis showed with an error probability of 0.4% that the repository-based approach improves the human-based approach (in terms of useful observations and guidelines) by at least 50% on average. In the experiment, the repository-based approach improved the human-based approach by more than 100%. Thus, both approaches should be used in combination. The participants agreed: As part of the debriefing they were asked which approaches they would apply in practice. 28 out 29 participants would apply both approaches in combination.

To validate the efficiency hypothesis (i.e., the repository-based approach is more efficient[2] than the human-based approach), the time needed by the participant to complete the repository-based approach and the estimated time it would have taken to talk to the colleague(s) were collected. The experiment validated

1 This has been a very important aspect of the conducted experiment. Some participants acknowledged that they would not have continued to use the experience base because of its complexity if they would not have received an introduction to it.

1 This chapter contains a detailed description of *S*.

2 Efficiency is measured as the time needed to find a useful guideline or observation (on average).

this hypothesis (based on the effort as well as on the duration) with an error probability of well below 0.5%.

In conclusion, the experiment

- Provided empirical, quantitative evidence that the perceived usefulness of qualitative experience is influenced by the background experience of the information seekers
- Showed that the repository-based approach complements the human-based approach in a substantial way (and vice versa) and thus strongly suggests that both approaches should be used in combination
- Validated the hypothesis that the repository-based approach is more efficient than the human-based approach
- Provided feedback for improving the quality of the stored guidelines and observations, their characterization schema, and INTERESTS

Consequently, the experiment has validated the whole DISER methodology since it was applied to set up and run the experience base used in the experiment.

12 Concluding Remarks

The Ordinary learns from his own faults,
The Smart learns from the faults of others,
The Stupid not even learns from his own faults.

Confucius (551–479 BC)

12.1 Major Results

This thesis presents a methodology for the goal-oriented customizing of software engineering experience management systems to organizational needs. The methodology called DISER (design and implementation of software engineering repositories) is based on the principle »learning from examples« [CMM84] and consists of:

- A representation formalism for experience base schemas (REFSENO; Chapter 6)
- A generic architecture for the technical infrastructure of an experience base (Chapter 7)
- A generic framework for populating, maintaining, and utilizing an experience base in form of a task hierarchy (Chapter 4)
- A method for engineering an experience base schema, customizing the architecture, and instantiating the framework (GOODSEE; Chapter 8)

It is complemented by the Organizational Memory Improvement (OMI) method (Chapter 9) which improves a software engineering experience management system from the user´s point of view.

REFSENO

The representation formalism REFSENO (representation formalism for software engineering ontologies) draws from ideas from several areas including database systems (e.g., semantic relationships among artifacts and implied consistency rules) [Che76, GR95], case-based reasoning mechanisms (e.g., similarity-based retrieval with incomplete information) [AAB+95, Alt97, Wes95], and knowledge-based mechanisms (e.g., value inferences and assertions) [Ric89]. In addition, REFSENO is object-centered similar to the meta modeling in UML [Rat97]. The integration of the approaches results in a new formalism that unites many of the advantages of the individual approaches (see existing approaches in Chapter 5 for an overview) while avoiding most of their drawbacks.

Among the (practical) benefits of schemas described using REFSENO are the following:

- Software engineering artifacts and their properties can be modeled naturally.

- Schemas can be used as a communication vehicle to (a) discuss the structure underlying the characterization information of an experience base and (b) aid in standardizing a corporate-wide vocabulary.
- Schemas are formal enough to allow automated support for retrieving and characterizing artifacts (semantics of REFSENO).
- Schemas are modular, that is, they can be extended by and/or combined with other schemas.
- Schemas are tailorable to organization-specific needs.
- Schemas grow with organizational needs regarding both growing number and growing diversity of artifacts.
- Schemas consider peculiarities of software engineering experience (e.g., context-sensitive retrieval, consideration of both qualitative and quantitative information as well as the consideration of incomplete, imprecise, inconsistent, and uncertain information).

Architecture for the technical infrastructure of an experience base

The semantics of REFSENO define the behavior of the EB-Specific Storage System, which is part of the larger technical infrastructure of an experience base. The (practical) benefits of the presented technical infrastructure include:

- Tailorability to organization-specific needs (already existing Artifact-Specific Tools – usually CASE tools – can be integrated)
- Growth with organizational needs (both growing number of artifacts and growing diversity of artifacts are supported)
- Ease of use (integration of already existing tools and the use of WWW technology for EB-Specific Tools provide a familiar look-and-feel for the users)
- Support for distributed learning (usage of the WWW technology supports learning for organizations distributed around the globe)

Together, REFSENO and the architecture for the technical infrastructure fulfill the requirements listed in Chapter 3 (see Table 31). The requirements are based on insights from the people who coined the term »experience base«, especially those requirements documented in [BCC92, BCR94a, BR91, MR87, RM88]. These were complemented by requirements that follow from the identified practical constraints (Chapter 2) and requirements elicited through recent industrial projects both in the area of incremental, continuous learning [ABT98, KS96] and software reuse [EMT98].

Framework for populating, maintaining, and utilizing an experience base

The usage of the technical infrastructure is described by a generic framework for populating, maintaining, and utilizing its experience base. The tasks are described using Steel´s knowledge-use level approach (components of expertise) [Ste90], an extension of Newell's knowledge level approach [New82]. The knowledge-use level approach describes an intelligent system by arranging the tasks to be performed in a hierarchy and by specifying for each task:

- The goal to be achieved by the task
- A method that either defines in what order to perform the subtasks of the task or how to perform the task if it is not decomposed further
- Contents (not structure!) of knowledge that is needed to perform some task

Table 31: Closed requirements

Requirement	Closed by: Detailed requirements (Chapter 3, Figure 6)	Closed by: REFSENO (Section 6.4)	Closed by: Tool Architecture (Section 7.4)
R1 (ease of use)			X
R2 (support for incremental, continuous learning)	X		
R3 (tool integration)			X
R4 (administrative information)		X	
R5 (access rights)			X
R6 (network access)			X
R7 (storage of various kinds of artifacts)		X	
R8 (explicit representation of the conceptual knowledge)		X	
R9 (separation of characterization and conceptual information)		X	
R10 (rationalized conceptual information)		X	
R11 (modularity of conceptual information)		X	
R12 (accommodation of new artifact kinds)			X
R13 (maintenance of conceptual information)			X
R14 (maintenance of experience packages)	X		
R15 (data collection for evaluation)			X
R16 (data analysis for improvement)			X
R17 (artifact recording)		X	
R18 (various characterizations of one artifact)		X	
R19 (tolerance of different levels of abstraction)		X	
R20 (tolerance of incomplete information)		X	
R21 (tolerance of uncertain information)		X	
R22 (tolerance of imprecise information)		X	
R23 (transparency of duplicated information)		X	
R24 (tolerance of inconsistent information)		X	
R25 (integrity constraints)		X	
R26 (accommodation of growing collection)			X
R27 (artifacts' status)		X	
R28 (versioning)		X	
R29 (configurations)		X	
R30 (change notification)			X
R31 (artifact preference information)		X	
R32 (browsing)		X	
R33 (textual search)		X	
R34 (similarity-based retrieval)		X	
R35 (tolerance of incomplete query information)		X	
R36 (tolerance of uncertain query information)		X	
R37 (tolerance of imprecise query information)		X	
R38 (context information)		X	
R39 (context-sensitive retrieval)		X	
R40 (check-out of artifacts)			X
R41 (interface information)		X	
R42 (application history)		X	

- Pragmatic constraints on the access and availability of the knowledge

In this thesis, the task descriptions are extended by the requirements that should be fulfilled to support the task by the technical infrastructure. Consequences – if the requirements are not fulfilled – are documented in Chapter 3.

The task hierarchy has been synthesized and refined through years of experience in the area of experience factory [ABT98, ABH+99, ANT99b, BR88, Bec96, BT98a, Bro98, Gäß95, GB97, Nic98, Stu95, Tau93, TA97, TG98b]. It is the most comprehensive description for incremental, continuous learning based on a repository to date. The description is systematic in the sense that not only the goal and the way to perform a given task is specified, but also the technical requirements needed to support the task effectively. Inputs and outputs define the product flow between tasks.

The knowledge-use level description serves a similar purpose as the task-method decomposition of case-based reasoning [AP94] in [Alt97] and represents a framework that

- is flexible enough to be tailored to company-specific needs and policies, i.e., increases potential for reuse across organizations [PD91]
- guides the tailoring to company-specific needs, i.e., it facilitates management control
- enables comparisons among processes employed in various organizations (by mapping the processes onto the tasks and comparing them on the task level)
- enables analyses of employed processes to detect strengths and weaknesses (by mapping the processes onto the tasks and checking (a) which requirements for the corresponding tasks are fulfilled and (b) which tasks are performed only implicitly or not at all)
- closes the gap between the tasks to be performed for running an experience base and the functionality provided by a technical infrastructure of an experience base, i.e., it helps in identifying support tools
- enables the demand-driven buildup of libraries for reusable problem-solving methods in the context of running an experience base (a problem-solving method achieves the goal of some task while considering pragmatic constraints on the availability and accessibility of needed knowledge)

More complex methods can be defined using the framework as a reference. For example, the Organizational Memory Improvement (OMI) method (Chapter 9) consists of a general usage model (which is an excerpt of the framework), a set of indicators for improvement potential, and a cause-effect model. OMI improves a software engineering experience management system incrementally from the user's point of view. It considers the practical constraints typically encountered in industrial environments (e.g., limited time of users) and works as follows: During each task of the general usage model of OMI, the indicators are used to pinpoint improvement potential for increasing the perceived usefulness.

If there is improvement potential, the user is guided for supplying specific improvement suggestions.

Method for engineering schemas, customizing the architecture, and instantiating the framework

However, to provide a software engineering experience management system that fulfills organizational needs, it is necessary to develop organization-specific experience base schemas. Existing approaches for developing schemas are either too broad or too narrow. What is needed then, is a systematic, goal-oriented method that is both general enough to cover the whole domain of software engineering experience reuse and specific enough to be able to give meaningful, detailed method descriptions. The schema engineering method proposed in this dissertation (GOODSEE) is specialized towards REFSENO and the presented framework, and thus to the domain of software engineering experience reuse. It pays particular attention to social and organizational issues by including activities for identifying stakeholders of the experience factory, the stakeholders' interests, scenarios explicating the needs of the users as well as the practical constraints on available knowledge, and methods for populating the experience base. Based on the defined knowledge collection procedures, the architecture is customized to the organization´s needs. Thus, the approach ensures the goal-oriented definition of the experience base contents, the extension of the architecture on an as-needed basis, and the definition of processes that instantiate the framework and smoothly integrate with already established practices at a given organization.

Validation

The presented approach has been validated extensively in several industrial-strength projects. For example, OMI has been validated using CBR-PEB [NT99], a publicly accessible experience base[1] on case-based reasoning tools and applications [BSAM97]. The validation confirmed the cause-effect model, the general usage model as well as many of the indicator/cause/improvement actions described in Chapter 9.

INTERESTS (Intelligent Retrieval and Storage System), a prototype of the tool architecture, has been built using the commercially available case-based reasoning tool CBR-Works of tec:inno [tec00]. The prototype uses CBR-Works both as a modeling tool (for the experience base schema) and as the retrieval engine. CBR-Works was selected because it already fulfilled many of REFSENO's requirements (e.g., similarity-based retrieval). The early availability allowed an extensive validation of the generic architecture. INTERESTS has been used in a large German insurance company [ABH+99] and for TQM in the health care sector

1 http://demolab.iese.fhg.de:8080/

[ABMN99], but also for publicly accessible experience bases on case-based reasoning systems [NT99] and knowledge management approaches[1] [Sno99].

Representative for several industrial-strength projects, Fraunhofer IESE's Corporate Information Network was presented as a case study to validate DISER (Chapter 10). The case study has shown that the framework – especially the task for populating the experience base and GOODSEE – is a viable approach for:

- Tailoring the technical infrastructure of an experience base (as presented in this dissertation) to organizational needs
- Analyzing the buildup of experience bases for its strengths and weaknesses
- Building a method library

Three hypotheses were postulated in the beginning of this dissertation (Section 1.3.2) comparing an experience factory organization with an organization without an experience factory:

1 **Efficiency hypothesis.** Using DISER for setting up and running the experience base of an experience factory, the experience factory is superior to an organization without an experience factory in the following sense: The information seeker[2] finds more useful experience items of a given kind per time period (in terms of both effort and duration) where »usefulness« is defined as the *perceived* usefulness by the information seeker regarding the experience item at the time of retrieval. It is measured on the subjective scale »useful«/»not useful«/»don't know«.

2 **Effectiveness hypothesis.** Using DISER for setting up and running the experience base of an experience factory which also manages tacit knowledge, the experience factory improves the project development organization, which shares experience via an informal expert network, in the following sense: The information seeker finds additional useful experience items for those kinds of experience items stored in the experience base where »usefulness« is defined as in the efficiency hypothesis.

3 **Applicability hypothesis.** DISER can be applied in industrial-strength projects with a justifiable amount of effort. DISER is considered to be applicable if people are willing to use the experience base on a regular basis. The amount of effort is considered to be justifiable if an organization is willing to pay for the application of DISER.

1 http://demolab.iese.fhg.de:8080/KM-PEB/
2 An information seeker is a project team member that seeks experience for a problem or task at hand.

An experiment validated the overall approach DISER as well as the comprehensive tool architecture and the framework for utilizing the experience base in particular (Chapter 11) for qualitative information. The experiment

- Provided empirical, quantitative evidence that the perceived usefulness of qualitative experience is influenced by the background experience of the information seekers
- Validated the effectiveness hypothesis both objectively (based on the data) and subjectively (based on the rating of the experiment's participants)
- Revealed that 28 out 29 participants would utilize both the experience factory and their informal expert network for obtaining qualitative information
- Validated the efficiency hypothesis

The applicability hypothesis has been validated as well:

- DISER is applicable because people are willing to use it on a regular basis (28 out of 29 participants of the experiment said they would use the experience base in practice; in addition, CBR-PEB [NT99] and KM-PEB [Sno99] have been used on a regular basis)
- The amount of effort is considered justifiable because organizations are willing to pay for the development of such a repository (e.g., a large German insurance company for its lesson learned program [ABH+99], the Fraunhofer-Gesellschaft for TQM in the health care sector [ABMN99], and Fraunhofer IESE for its Corporate Information Network)

12.2 Impact of Environment Changes

The validation at the end of the previous section has shown the benefits of the Corporate Information Network directly after its introduction and before its widespread use. However, Fraunhofer IESE is not a static organization. Hence, it is of interest to analyze the impact of its changes on the benefits of DISER:

- **Time progression.** Humans tend to forget details of experiences they gained in projects long ago. Experience documented explicitly is still available in all its details – even after years – unless it is deliberately removed. Thus, since the experiment was based on current experience only, it is expected that the repository-based approach grows more beneficial as time progresses.
- **Increased project experience.** As more and more projects are analyzed, project experience increases. The qualification of experience (as part of the experience recording task) searches for already stored similar experiences. If there is no similar experience stored, the new experience is simply added to the already stored experience. This will increase the coverage of the experience base and, consequently, the perceived usefulness. On the other hand, if similar experiences are already stored, the new experience is either merged with the already stored experiences or not recorded at all. If it is merged, the experience´s validity (and thus its usefulness) increases. Hence, an increased number of projects will make the validity a meaningful attribute and support

decision-making. For example, if a particular guideline *G* has been applied in ten different projects successfully and in one project unsuccessfully, we know that the guideline is generally useful. If, however, *G* was applied only once successfully and three times unsuccessfully, one should think about applying it very carefully (or removing it from the experience base). For such statistics to be meaningful, a critical mass of projects and lessons learned must be available.

- **Increased number of employees.** As the number of employees grows, project experience is distributed among more people. This makes it harder for each individual to know what is going on in the organization as a whole. This is especially true for geographically distributed organizations. While such a development does not have an impact on the effectiveness of the repository-based approach, the human-based approach becomes less effective or at least less efficient (i.e., it will take longer to acquire all relevant information regarding a particular topic in a large organization than in a small one).
- **Staff change.** As employees leave an organization, they take their experience with them. For example, during the preparation of the experiment, some information providers said »If Y would still be working for us, I would have recommended to go to him«. Experience documented explicitly does not leave the organization when its originators leave (provided that it is documented in a way that everybody can understand it). Thus, while the effectiveness of the repository-based approach will remain the same, the effectiveness of the human-based approach will decrease.
  On the other side, new employees usually have less experience than those who have been working in the organization for a long time. Usually, new employees (a) do not know whom to ask (therefore, the human-based approach is less efficient for new employees) and (b) hesitate to ask colleagues for every small advice. If new employees utilize the experience base, they (a) do not have to know whom to ask and (b) they can query as often as they like without bothering anybody. Thus, new employees will utilize more available project experience if an experience base on project experience is available.
- **Increased number of artifact kinds.** The more information is stored in the experience base, the more useful the experience base will be perceived (if it is possible to filter the available information effectively, e.g., by the relevant context). For example, during the preparation of the experiment, many information providers suggested to take a look at IESE reports that were deliverables of a project which was of the same type or was completed for the same customer. However, at the time COIN was set up, it was not documented which projects delivered which IESE reports. Furthermore, the information for whom the report was written, was not published within the institute (the exception being if the customer was named in the report title). None of the information providers was able to give a concrete list of IESE reports without using some retrieval means. Therefore, the addition of IESE reports to COIN (together with a reference to the project which delivered it) would yield a significant benefit which cannot be achieved via the human-based approach.

- **Introduction of a systematic problem solving process.** Often problem solutions (especially, locally optimized solutions as they are often derived in daily project business) solve a problem for one party, but create new problems for others. A project-independent problem solving team can work out alternative solutions, analyze the solutions concerning positively and negatively affected people, and finally choose a solution that is optimized for the whole organization, not just for a single project. Such a problem solving process is also able to solve problems recognized only after the project was already completed (and thus out of mind for the project team). These kinds of benefits can only be gained if problems are documented explicitly. Characterizations for problems can guide this systematic problem solving process (cf. Appendix E).

To summarize, all these environment changes will result in an increased effectiveness and/or efficiency of the repository-based approach in comparison to the human-based approach.

12.3 Future Research

This thesis has advanced the state-of-the-art regarding the customizing of software engineering experience management systems to organizational needs. It provides a reference framework for setting up and running an experience base. This section discusses some research issues that will become important in the future as DISER is applied in »daily« practice.

Schema libraries

One research area is the reuse of experience base schemas. For instance, experience gained in developing experience base schemas for a large German insurance company and COIN showed that the reuse (with modification) of schemas can save a lot of effort. One can distinguish between schema patterns (e.g., for versioning and configuring artifacts [NT98a]) and concrete schemas (e.g., for lessons learned – see Appendix E). Open research questions in this area are:

- How to come up with a set of consistent experience base schemas[1], each covering a particular area or method in software engineering (e.g., one schema for requirements engineering, one for goal-oriented measurement, one for process modeling, …)?
- How to identify patterns behind a set of similar concrete schemas (e.g., a problem should always contain the description of some situation, the effects of the situations as well as the cause(s) of the problem – although the attributes of each section may vary from organization to organization)?

1 A consistent set of experience base schemas is a set of schemas that can be easily integrated to one comprehensive experience base schema. A consistent set of schemas requires clearly defined interfaces between the schemas (similar to the integration of code modules to software systems).

- What basic schema patterns besides those for versioning and configuring are needed for experience bases?
- How to characterize schemas so they can be easily found when developing some new schema?

Framework extension

Another research area concerns the refinement and extension of the framework presented in this dissertation. In this context, the degree to which a task can be automated by tools plays a central role. With each refinement of a task it is expected that the degree of automated tool support increases, while the human-based part of the task execution decreases in terms of effort.

The framework can be extended by either providing a method library (i.e., a set of alternative methods for one or more particular tasks) or a detailing of one of the tasks (i.e., a (sub)task hierarchy). For the former, the following questions arise:

- What kind of practical constraints will determine the selection of methods?
- How can these constraints be characterized? (This will allow to use DISER to manage the methods)
- For which tasks does it make sense to build method libraries, that is, for which tasks is it likely that reusable methods can be developed?

For the latter (i.e., detailing of tasks), some examples follow below.

Maintenance

One of these examples is the maintenance of the experience base. Chapter 9 presents OMI, a method that improves a software engineering experience management system based on user feedback. However, there may also be technical reasons to start a maintenance process (preventive maintenance). For instance, if more than ten improvement suggestions are attached to an artifact, they should be worked in constructively. Such maintenance processes can be described as maintenance policies [LW98]. A maintenance policy is described by a:

- **Trigger** telling when to check for a maintenance activity (e.g., periodically once a month)
- **Condition** that needs to be checked after the policy has been triggered (corresponds to a method for the task »analyze SEEMS (T50, page 116)«, e.g., for all artifacts *A*: more than ten lessons learned attached to *A*)
- **Action(s)** telling what to do in case the condition evaluates to true (corresponds to a method for »forget (T54, page 119)« or »package (T55, page 120)«, e.g., aggregate artifact and attached improvement suggestions)

In this context, the following questions arise:

- What kind of maintenance policies are of practical use?
- How to identify the needed maintenance policies and how to derive them systematically (refinement of »structure (T56, page 121)«)?

Requirements elicitation

Another example is the description of scenarios. During the task »structure (T56, page 121)« reuse and acquisition scenarios are identified. Here, the following questions are interesting:

- What kind of information should be contained in a scenario description?
- How can existing methods of requirements engineering, knowledge engineering, and psychology be tailored to the needs of an experience base? (This would result in a refinement of »identify reuse scenarios (T71, page 217)« and »describe knowledge acquisition (T75, page 219)«.)

Automating the »inform« task

As part of the acquisition (recording task), people who might be interested in the newly acquired artifact are informed. To automate this subtask, the experience base schema (and thus REFSENO) needs to be extended. Questions to be answered include:

- What are possible information strategies?
- Given an information strategy, under what conditions (e.g., a new artifact is inserted, an artifact is updated, …) should it be applied?
- How to derive systematically information strategies and conditions when to apply them?

Views

The acquisition of large amounts of information may result in an overburdening of the user during retrieval. Therefore, it should be possible to restrict the available information items to the relevant ones for the task at hand. One way of doing this, is to introduce views.

A view is defined by a retrieval goal. Each retrieval goal defines a particular set of information items (documented in the concept glossary and the concept attribute tables) that are needed for the task specified in the goal. The exploitation of views may lead to improved guidance of the user because the set of visible information items is greatly reduced. However, it is difficult to foresee all retrieval goals the users will have in mind when they use the experience base. Hence, it is hard to answer the following questions:

- How can the user be guided in selecting the right view? (If he selects the wrong view, he will not perceive the retrieval system as useful as it could be.)
- How to ensure that all relevant reuse scenarios have been captured during the design of the experience base? (The case study presented in this dissertation has shown that it is easy to forget important reuse scenarios.)
- To implement views, the definition of a view needs to be detailed: How to extend REFSENO by a view definition, i.e., how do views differ with respect to REFSENO's constructs?

Organizational issues

Besides these technical issues, there are a lot of open research questions regarding the organizational issues »surrounding« the experience base:

- Support for project planning:
  - How many people are needed to attain the objectives?
  - What skills do the people need to have to execute the corresponding methods described in this dissertation?

- How to staff the experience factory?
- How to integrate knowledge management activities (which focus traditionally on non-repository-based methods such as group meetings) with the framework presented in this dissertation?

It is expected that advances regarding the research issues listed above will result in an accelerated setup as well as a more effective and efficient running of a software engineering experience management system.

Appendix A:
Exemplary Questionnaire for Identifying Objectives of an Experience Factory

The following questions have been derived from questions that have been developed for a »systemic first interview« during the course »Psychology for Consultants« held at the Fraunhofer IESE in 1997.

All stakeholders of an experience factory should be interviewed using this questionnaire.

A.1 Expectations Concerning the Buildup of the Experience Factory

A.1.1 How does the experience factory become a flop, that is, what do we have to do to let the experience factory project fail?

A.1.2 How does the experience factory become a success? What do we have to do?

A.1.3 What do the colleagues think about the introduction of an experience factory?

A.1.4 How do you define a successful experience factory?

A.2 Current Problem

A.2.1 What do you think what <Stakeholder X, Y, Z> thinks about the experience factory project?

A.2.2 What is the problem for you right now?

A.2.3 What exactly is bothering you in this situation?

A.2.4 Why is this problem *your* problem?

A.2.5 What effects does the problem have?

A.2.6 Who is affected by this problem?

A.2.7 Who has a share in this problem?

A.3 Causes

A.3.1 What do you think are the causes of the problem?

A.3.2 What facts would you use to support your claim?

A.3.3 Is there an example from the past to support your claim?

A.3.4 What do you think what <XY> (person affected by the problem) would claim as the causes?

A.3.5 Which facts do you think would <XY> use to support his/her claim?

A.3.6 How long does the problem already exist?

A.3.7 When or under what conditions does the problem occur?

A.3.8 How often does the problem occur?

A.4 Solution Attempts

A.4.1 Have you already taken steps towards solving the problem? If so, what steps?

A.4.2 How did it work out?

A.4.3 Where/why got the solution attempt stuck?

A.5 Potential Solution

A.5.1 Why do you think, the experience factory project will solve your problem?

A.5.2 What would you view as a success regarding the problem solution?

A.6 Problem Boundaries and Things to Avoid

A.6.1 If the problem grows worse, what could happen in the worst case?

A.6.2 What has limited the effects of the problem until now?

A.6.3 What could happen that would worsen the problem?

A.6.4 What could be done in the current situation to worsen the problem?

A.7 Risks in Solving the Problem

A.7.1 What would you miss if the problem was solved?

A.7.2 Who is interested in *not* changing the current practice?

A.7.3 Who would lose something if the problem was solved?

A.7.4 What risks are there if the problem would be solved using the experience factory approach (i.e., using an explicit repository to store knowledge)?

A.8 Things Not to Change

A.8.1 What is currently very good (outside the discussed problem area)?

A.8.2 What should not be changed under any circumstances?

A.9 Success Evaluation

A.9.1 When would you consider the problem to be solved?

A.9.2 How could that be measured?

A.9.3 How would the organization work then?
(Description of the problem-free time)

A.9.4 Who will notice it first?

A.9.5 Who would try to regain the old state?

A.10 Hypothetical Questions Regarding the Future

A.10.1 What happens if the problem occurs again (e.g., if colleagues do not document/share their knowledge)?

A.10.2 What could lead to similar problems?

A.10.3 What if the problem cannot be solved?

Exemplary Questionnaire for
Identifying Objectives of an
Experience Factory

Appendix B:
Goals of the Corporate Information Network

This appendix lists the major goals of COIN and their subgoals:

1 Increase contract winnings/renewals and optimize proposals to strategies of Fraunhofer IESE *by supporting people* through the provision of process guidance and the explicit collection, analysis, and packaging of transfer experience that includes (but is not limited to) proposals and reference projects for competencies of Fraunhofer IESE.

External Subgoals:
- Ensure consistent[1] and comprehensive[2] marketing
- Continuously improve quality of our products
- Continuously improve customer satisfaction
- Reach ISO 9000 certification (long-term)

Internal Subgoals:
- Increase efficiency of work procedures by simplifying and clarifying them
- Increase process conformance for selected processes
- Secure existing know-how by operationalizing the collection of existing knowledge (artifacts [checklists, reports, publications, presentations, templates, …] and documentation of important decisions)

2 Support people by increasing transparency of projects and other available knowledge to optimize personal knowledge acquisition.

Subgoals:
- Increase exchange of experiences among IESE employees
- Increase transparency of projects for ILA members
- Increase reusability of existing knowledge
- Reduce search effort in comparison to existing intranet

1 Consistent marketing := (a) for a given purpose/goal, everybody advocates the same product(s) and (b) everybody uses the same product description resulting in a consistent message from Fraunhofer IESE
2 Comprehensive marketing := everybody knows about all business areas and products and markets them actively even if they are outside the individual's scope

3 Measure the benefits, collect user feedback, and use the results to help future COIN improvement changes

Subgoals:
- Validate and continuously improve work procedures
- Ensure that everybody gets the information he needs

4 Coordinate COIN activities with other ongoing activities (improving net security and restructuring internet/intranet)

Appendix C:
Scenarios for the Corporate Information Network

This appendix lists the scenarios for COIN.

C.1 Purpose of Scenarios

Just as for almost every service provider, project work is fundamental to Fraunhofer IESE. Therefore, it is the objective of Fraunhofer IESE to continuously improve project work. One way of doing so is to provide experience from past projects to ongoing projects. From an organizational point of view, this means to reapply successful strategies and to avoid making the same mistake twice.

This appendix describes scenarios for
- reusing experience from past projects
- recording project experience (including its preparation for dissemination)

C.2 Reusing Project Experience

In the context of this appendix, *project experience* is a set of lessons learned (problem descriptions with corresponding solutions, guidelines, and improvement suggestions) and observations.

Project experience is usually described informally, that is, textually. However, for querying, a more formal representation is needed to enable the computation of meaningful similarity values.

The scenarios in this section describe how project experience can be reused (i.e., retrieved and utilized).

C.2.1 Scenario »IdentifyProjectRisks«

Retrieval Goal

Retrieve problems
for the purpose of finding project risks
from the viewpoint of a Project Manager
in context of Fraunhofer IESE

Scenario

Trigger — During »Set-Up Project/Detailed Project Planning«

Actions

1 Based on the characterization of the project at hand, problems that occurred in similar projects are retrieved using COIN. (Actually, the problems suggested by COIN should be limited to those that are likely to recur.)

2 The suggested problems are analyzed further to check which can occur potentially in the project at hand.

3 The identified problems are listed as project risks in the project plan. Corresponding mitigation strategies are documented as guidelines (see Section C.2.2).

Assumption: Problems that occurred in (similar) past projects can occur again. Therefore, they constitute risks unless constructive steps have been taken to prevent their recurrence (e.g., by changing the process description for »Project Execution«).

Expected benefits — Through the explicit availability of problems that occurred in past projects, a more complete list of project risks can be included in the project plan than if the project manager would have to rely solely on his or her own experience. In turn, this will increase project adherence, because »exceptions« are more unlikely to occur.

C.2.2 Scenario »GetRelevantProjectInfo«

Retrieval Goal

Retrieve guidelines and observations
for the purpose of finding recommendations, comments, and notes for the project at hand
from the viewpoint of a Project Manager
in context of Fraunhofer IESE

Scenario

Trigger — Every time one of the following IQ system[1] processes is started: »Acquisition of Industrial Projects«, »Acquisition of Public Projects«, or »Set-Up Project/Detailed Project Planning«

Actions

1 Based on
   - the started process (one of the four listed above)
   - the characterization of the current project (including the customer and the project size)

   potential guidelines[1] and observations[2] are identified using COIN.

2 The guidelines and observations suggested by COIN are further analyzed to decide which to utilize in the project at hand. For guidelines, their corresponding problems are examined. Guidelines related to problems that can occur potentially in the project (see Section C.2.1) are good candidates for inclusion in the project plan (or as something that should be included in the proposal in form of requirements for the customer).

   For observations, no further systematic analysis description can be given. However, if the observation and/or project characterization does not contain all information necessary to make the utilization decision for an observation, this should be noted. The feedback shall be used to improve the characterization schema.

3 The identified guidelines are included in the project plan. Observations are utilized indirectly by considering conclusions drawn by them in the project plan.

Expected benefits

Guidelines can be interpreted as mitigation strategies for identified project risks. Thus, the explicit availability of guidelines improves the project risk management. In turn, this will increase the project adherence, because »exceptions« are more unlikely to occur.

Observations give hints on what worked well in the past or some preliminary baseline (e.g., effort numbers). Thus, the utilization of observations facilitates the development of realistic proposals and project plans. In turn, this will also increase the project adherence.

1 IESE Quality Management System
1 Guideline := a suggestion or recommendation for executing a project that (supposably) avoids a problem that occurred in some earlier project (context-dependent)
2 Observation := something an employee has observed or experienced during a project that is not captured as a guideline or problem/solution statement

C.2.3 Scenario »FindSolution«

Retrieval Goal

Retrieve problems
for the purpose of finding a solution for an occurred problem
from the viewpoint of a Project Manager or Project Member
in context of Fraunhofer IESE

Scenario

Trigger — A Project Member has recognized a problem in the project.

Actions

1 Based on the characterization of the project at hand, problems that occurred in similar projects, are retrieved using COIN.

2 The suggested problems are analyzed further to see whether they occurred in a similar situation.

Assumption: Each problem description includes an informal description of the situation in which it occurred (e.g., the status of the project).

3 The solutions associated to the problem are analyzed.

Assumption: Each problem is associated with a number of solutions. A solution describes an attempt how the problem was solved (pragmatically) and includes an evaluation. The evaluation states whether the solution solved the problem successfully, or not. If it did not solve the problem successfully, it states the reason why it did not.

4 The most promising solution is chosen and adapted to the needs of the project at hand.

Expected benefits — The reuse of solutions that worked successfully in the past will (a) reduce the effort for devising a new solution and (b) avoid applying a solution that worked unsatisfactory in the past (this avoids making a mistake twice).

C.2.4 Scenario »FindImprovementSuggestion«

Retrieval Goal

Retrieve improvement suggestions
for the purpose of finding out what and how to improve an IQ system process
from the viewpoint of the process owner
in context of Fraunhofer IESE

Scenario

Trigger — One of the following:
- Periodic – every half a year, IQ processes should be updated
- More than five improvement suggestions are pending to an IQ system process.[1]

Actions — Based on the IQ system process to be updated, the corresponding improvement suggestions are retrieved using COIN.

Expected benefits — All improvement suggestions are considered formally, that is, no improvement suggestions are forgotten.

C.2.5 Scenario »FindArtifact«

Retrieval Goal

Retrieve artifacts
for the purpose of finding relevant artifacts
from the viewpoint of an IESE Employee
in context of Fraunhofer IESE

Scenario

Trigger — Anytime someone thinks there might exist a helpful artifact at Fraunhofer IESE for the task at hand.

Actions — 1 Based on
- title
- author
- date
- artifact type (template or document)
- keywords

existing artifacts are retrieved using COIN.

1 This is an initial suggestion. It is based on the fact that more than 7 ± 2 items cannot be processed intellectually at the same time. Here, the pessimistic number is taken. However, experience may show that this number is too large (it takes too long until improvement suggestions are actually worked in) or too small (the process owner has to improve the IQ system process description almost all the time).

2 The suggested artifacts are analyzed further by looking at the artifacts' history (From which artifacts has the currently analyzed artifact been derived? Is there a newer version available?) and the contents themselves.

3 The most appropriate artifact is selected.

Expected benefits

A single query is sufficient to look through all kinds of artifacts. The user does not need to know:
- What kind of artifact (IQ Process, reference document, …) is appropriate
- Where the artifact is stored (as a file in the »~iese« directory, as a file at the publication service, as a file in the WWW, as a BLOB in a database)

This will save time and also find artifacts that would not have been found »traditionally« (i.e., using the »old« intranet).

C.2.6 Scenario »FindIQProcess«

Retrieval Goal

Retrieve IQ Processes
for the purpose of finding relevant IQ Processes
from the viewpoint of an IESE Employee
in context of Fraunhofer IESE

Scenario

Trigger

Anytime someone needs to locate a particular IQ Process or needs a list of related IQ Processes for the task at hand.

Actions

Based on
- title
- process owner
- keywords
- roles involved
- viewpoint

appropriate IQ Processes are retrieved using COIN.

Expected benefits

The user does neither need to know the exact title of the appropriate IQ Process nor have an idea what all IQ Processes are about (which would be the case if the user would have to select the appropriate IQ Process using navigation). Instead, totally new groupings (e.g., all processes where a Project Member is involved or the main actor (= viewpoint)) can be created on the fly.

This will allow a particular IESE Employee to restrict the number of relevant IQ Processes to a minimum. In this way, even a larger number of IQ Processes can be managed without running into the situation that people lose sight of IQ Processes that are relevant to them.

C.2.7 Scenario »GetImprovementProgress«

Retrieval Goal

Retrieve problems
for the purpose of informing about the progress regarding the development of a solution for a problem
from the viewpoint of an IESE Employee (problem reporter)
in context of Fraunhofer IESE

Scenario

Trigger

Every time a problem reporter wants to get a progress report on his problem (i.e., regarding the solutions that have been developed and how far the implementation has proceeded)

Actions

1 Based on
- name of reporter
- date
- affected IQ processes
- affected roles

the problem reporter retrieves matching problems using COIN.

2 The problem reporter selects the problem he or she is interested in.

3 The problem reporter looks/can look at:
- Status of problem
- Priority of problem
- Assigned personnel (contact person, creativity team)
- Identified cause of problem
- Devised guidelines
- Devised improvement suggestion including their status

Expected benefits

The feedback shows IESE Employees whether their problems are taken seriously and are being worked on. If so (and this should be the case!), the feedback will motivate IESE Employees to file further problem reports. In turn, this will lead to a faster recognition of (potential) problems within the organization of Fraunhofer IESE than without this feedback possibility.

C.2.8 Scenario »FindOpenProblems«

Retrieval Goal

Retrieve problems
for the purpose of finding problems that have not been solved or rejected
from the viewpoint of the COIN team, creativity team
in context of Fraunhofer IESE

Scenario

Trigger: Before the COIN team or creativity team meets

Scenario:

1 Based on
- status
- name of creativity team

problems are retrieved using COIN.

2 Problems are sorted in order of their priority and date of reporting.

3 A handout for the team meetings is prepared consisting of the sorted problems together with the guidelines and improvement suggestions worked out so far.

Expected benefits: The systematic management of the solution process for problems (problem controlling) ensures that all problems are considered and feedback is given to their problem reporters.

C.2.9 Final Remarks

While guidelines, problem/solution statements, and observations must be accessible to all IESE employees, improvement suggestions must only be visible to the respective process owners, improvement suggestion authors, and reporters of the corresponding problem.

C.3 Recording Project Experience

Project experience is best captured during the internal project analysis interview (cf. IQ system process »Project Wrap-Up«). The collection of experience in a single event has the following advantage: The Project Manager, the Scientific Lead, and the Project Members do not have to worry about the form (observation, guideline, solution, etc.) in which their experience will be recorded.

It is suggested to use Appendix D for recording the experience. An experience engineer (a member of the COIN team) should support the project analysis interview by moderating it and preparing the experience provided for future utilization. Experience from a trial run of a project analysis interview has shown that it is important that an experience engineer reminds the project team of holding such an event and to follow-up on the action items resulting from this interview.

The preparation for the future utilization is done by entering the experience listed in the project analysis report (see Appendix D) into COIN. To support the reuse scenarios listed in Section C.2, the following information must be collected: guidelines, improvement suggestions, observations, problems, solutions, and project characterizations.

Next, it is described what information can be extracted from the project analysis reports:

- **Guidelines.** Answers to questions 5 and 11–17
- **Improvement suggestions.** Answers to questions 5 and 11–16
- **Observations.** Answers to question 6–10, 18
- **Problems.** Answers to questions 5, 8, 9, and 11–13
- **Project characterizations.** Section D.1 and Section D.2.1
- **Solutions.** Answers to questions 5 and 11–13

The Project Manager, the Scientific Lead, and the Project Members may not provide problems, solutions, guidelines, and improvement suggestions as part of their answers to the questions 5 and 11–13. In this case, the moderator of the project analysis interview must ask explicitly for them by asking questions like:

- Why are you suggesting this? Can you give an example where this would have helped? [Asking for an occurred problem]
- What do you think was the cause for the problem? [Asking for the problem situation]
- What did you do about the problem in the project? [Asking for a solution]
- How do you think can this be avoided in the future? [Asking for a guideline or improvement suggestion]

Appendix D: Template for Project Analysis Reports

In the following, the questions marked with »*« do not have to be filled out if the project has not been completed yet.

Date of project analysis interview:

D.1 Project Characterization

Funding: . . . . . .

Type: . . . . . .

Project manager: . . . . . .

Scientific lead: . . . . . .

Project member(s): . . . . . .

Customers/Partners (names and functions in parentheses): . . . e.g.: Dept. X (Fritz Mayer: Dept. head)
Sponsor: . . . . . .
Information/Technology providers (experts): . . . . . .
Users/Application partners: . .

Experience engineer for project analysis: . . . . . .

Project start:
Kickoff meeting: . . . . . .
Start of project work: . . . . . .

Project end:
Negotiated with customer: . . .
End of project work*: . . . . . .
Final presentation*: . . . . . .

Effort:
Paid (per contract): . . . . . .

Internally planned effort: . . . .
Actual effort*:. . . . . .

D.2 Insights About Project Planning and Execution

Aim of this section is to get feedback about the IQ Processes »Set-Up Project« and »Project Execution« at Fraunhofer IESE.

D.2.1 Questions to the Project Manager: Project Description

The following questions should be answered only by project manager.

1 What was the (official) objective of the project (1 sentence)?
…

2 What were the (internal) goals of the involved project partners (short answer)?
- Fraunhofer IESE:
- Customer:
- …

3 Why did Fraunhofer IESE get the contract? I think, because:
- …

4 Why was the project acquired, that is, what did Fraunhofer IESE hope for (except money and the internal goal stated under 2)?
- …

D.2.2 Questions to the Project Team

The following questions should be answered by all project members including the project manager and the scientific lead.

Project Evaluation

Risk Factors

5 What was especially critical in the project, that is, what would you look out for in future projects?
- …

External Satisfaction

6 How did the customer like the project results (please answer on a scale from 1 (very satisfied) to 5 (very unsatisfied) with a short reason)?
- …

7 In case more effort was put in than internally planned: how would the customer have liked the project results if design-to-cost would have been implemented?*
   - …

Customer Expectations

8 What did you learn about the (wishes and expectations of the) customer in this project?
   - …

Internal Satisfaction

9 How happy were you with the project seen from a scientific point of view (please answer on a scale from 1 (very satisfied) to 5 (very unsatisfied) with a short reason)?
   - …

10 How happy were you with the way the project was performed (please answer on a scale from 1 (very satisfied) to 5 (very unsatisfied) with a short reason)?
   - …

11 What was *the* high point in the project?
   - …

12 What was *the* low point in the project?
   - …

Discrepancies*

13 How do you explain the discrepancies (if any) in the project characterization regarding the duration (negotiated date vs. end of project work)?*
   - …

14 How do you explain the discrepancies (if any) in the project characterization regarding the effort (internally planned vs. actual effort)?*
   - …

Improvement

15 What could have been done to improve your motivation regarding the issues listed under question 9 (scientific satisfaction)?
   - …

16 What could have been done to improve your motivation regarding the issues listed under question 10 (satisfaction with the way the project was performed)?
   - …

17 If you would have to do the same project again: What would you do differently (except for those answers given in 5)?
- …

18 What worked well in the project and for what reason?
- …

Appendix E:
Experience Base Schema for the Corporate Information Network

This appendix lists the experience base schema used for COIN.

E.1 Ontology Specification

Table E.1: Ontology Specification

Domain	Corporate Information Network (COIN)
Context	Fraunhofer IESE
Date	December 1, 1999
Conceptualized by	Carsten Tautz
Reviewed by	Klaus-Dieter Althoff (Nov 15, 1999)
Purpose	Manage all project-relevant knowledge and experience of Fraunhofer IESE
Level of formality	Semiformal (REFSENO)
Remarks	COIN is realized incrementally. Although the concept glossary is intended to be complete to give a vision what COIN will consist of in the end, all other tables are not. They are restricted to the portion of the COIN ontology that has already been implemented or is planned to be implemented in the near future. Completely defined concepts are printed in **boldface** in the concept glossary. Due to the incremental development, some of the concepts are realized initially as »simple types«. For instance, the concept »IESE Employee« will be initially defined as the UnorderedSymbol »iIESE Employee«. The »i« in the beginning stands for »interface« and indicates that the ontology will be extended at this interface later. This means that all terminal attributes of this type will be converted to nonterminal attributes later (their type being a reference to the corresponding concept).

E.2 Concept Glossary

Table E.2: Concept Glossary

Name	Description	Scenario name	Purpose	Intended user(s)
Application Roadmap	Current description of a *Business Area* and envisioned future development of this field	GetMarketingStrategy	informing about marketing strategy	FhG, Institute Director
		ImproveMarketingStrategy	improving/updating/checking marketing strategy	Institute Director, Department Head
Artifact	Any printable matter that can be versioned	FindArtifact	finding a relevant artifact	Everybody
Brochure	A flyer or document given to (potential) customers of Fraunhofer IESE advertising either Fraunhofer IESE work in general or some *Core Product* in particular.	GetProducts	informing about products/ services of Fraunhofer IESE	Customers
		SellProducts	selling products/services	Marketing and Acquisition

Name	Description	Scenario name	Purpose	Intended user(s)
Business Area	A grouping of *core products* in which at least 15% of Fraunhofer IESE's income is realized	GetBusiness-Description	informing about fields in which Fraunhofer IESE offers products/services	Customers
Comment	A remark of an *Employee* regarding an *Artifact* (e.g., questions asked during a presentation)	GetComment	informing about experience gained with a given artifact	All
		ImproveArti-fact	improving/updating an artifact	Author (of artifact)
Contract	A legally binding document describing how Fraunhofer IESE cooperates with a given project *partner organization*	GetAims	informing about goals and deliverables of a project	Project Members, ILA
		GetPublRe-strictions	informing about restrictions regarding publications	Frank Bomarius
		SendInvoice	sending invoices	Project Accounting
Core Competence	A competence that is needed to develop two or more *core products*	FindEx-pertWith-Compentenc e	finding experts	Project Members
		ImproveBusi-nessStrategy	improving business strategy	Institute Director, Department Heads
Core Product	Something Fraunhofer IESE sells and makes money with	FindExpert-ForCoreProd-uct	finding experts	Project Members
		FindBattleKit	finding battle kits	Marketing and Acquisition
		ImproveBusi-nessStrategy	improving business strategy	Institute Director, Department Heads
Domain	A grouping of *partner organizations* that produce similar products (e.g., telecommunication systems)	FindDomain-Coordinator	finding domain coordinator	Project Members
Event	A conference, workshop, or fair	MakeParticipationDecision	finding whether it is worthwhile to participate in a particular event	Scientists
		FindEventInfo	finding pros and cons of an event	Everybody
		GetEventDetails	finding details about a particular event (e.g., date)	Everybody
Guideline	A suggestion/recommendation for executing a project that (supposably) avoids a problem that occurred in some earlier project (context-dependent)	GetRelevant-ProjectInfo	informing about recommendations for the project at hand	Project Manager
IESE Employee	Institute director or someone working at Fraunhofer IESE		modeling (information about employees must be restricted to those necessary to perform some task – to avoid evaluation of people)	

Name	Description	Scenario name	Purpose	Intended user(s)
IESE Organization	An organizational unit of Fraunhofer IESE, i.e., a department or group		modeling	
IESE Report	A report carrying an official IESE Report number	FindReportDetails	finding confidentiality or status information about a particular IESE report	Scientists, LIS-Manager
		FindReportNumber	finding report number of a particular IESE report	Scientists, Project Members
		GetDetailedOverview	getting detailed overview of a particular topic	Scientists, Project Members
Improvement Suggestion	A suggestion how to improve a particular *Artifact*; an *Improvement Suggestion* is the result of a problem analysis	FindImprovementSuggestions	finding out how to improve a given artifact	Authors, Artifact owner
IQ Process	Description of a recurring task including general guidelines on how to perform it as well as links to knowledge and experience relevant for this task	FindIQProcess	finding a relevant IQ Process	Everybody
Observation	Something an employee has observed or experienced during a project that is not captured as a guideline or problem/solution statement	GetRelevantProjectInfo	finding interesting comments and notes for the project at hand	Project Manager
Partner Organization	An organization to which Fraunhofer IESE has a professional relationship	FindOrgInfo	finding background information on a particular organization	External Representation, Marketing and Acquisition, Procurement, Domain Coordinator, Project Manager, Project Members, Public Project Coordinator, Industrial Project Coordinator
Person	An employee of a *Partner Organization*	FindPersonInfo	finding background information on a particular person working in a partner organization	External Representation, Marketing and Acquisition, Procurement, Domain Coordinator, Project Manager, Project Members, Public Project Coordinator, Industrial Project Coordinator

Name	Description	Scenario name	Purpose	Intended user(s)
Presentation	A set of slides	GetQuick-Overview	getting a quick overview on a particular topic (product, competence, scientific topic)	Project Members, Scientists
Problem	A negative situation that occurred during a project execution	IdentifyProjectRisks	identifying project risks	Project Manager
		FindSolution	finding a solution for an occurred problem	Project Manager, Project Member
		GetImprove-mentProgress	informing about the progress regarding the development of a solution for a problem	problem reporter
		FindOpen-Problems	finding problems that have not been solved or rejected	improvement squad, creativity team
Process Info	Complementary information about an *IQ Process*		modeling	
Product	A tool or proceeding/ technique	FindProduc-tExpert	finding an expert for a given product	Project Members, Scientists
		FindProduc-tExperience	finding tips and tricks for a given product	Project Members, Scientists
Project	Something for which Fraunhofer IESE has given a commitment and gets money for	GetProject-Documents	finding project-related documents and information	Project Managers, Project Members
Proposal	An (usually legally binding) offer for cooperation	FindLegalPro-posal	finding proposal phrasing that is legally accepted by Fraunhofer ZV	Project Manager, Contract Management
Publication	A document printed outside Fraunhofer IESE, but whose authors are/were employees from IESE	FindPublInfo	finding details about a given publication (e.g., where or when published)	Scientists, Project Members
Reference Document	A document that may be of interest as input to projects or research work	FindIdeas	finding ideas for a paper or project work	Scientists, Project Members
Research Question	An open issue for which projects should be acquired, so the question can be answered	Find-Research-Issues	finding research issues for new projects	Marketing and Acquisition, Project Manager, Scientists
Review	Experience (positive or negative) an *Employee* had with a particular *Artifact* (in contrast, a *Comment* is neutral)	FindReview	finding assessments of colleagues	Project Members, Scientists
Role	An actor of an *IQ Process*	FindRole-Descr	finding a role description	Everybody
		FindIQProcess	finding *IQ Processes* a particular role is involved in	Everybody

Name	Description	Scenario name	Purpose	Intended user(s)
Solution	A pragmatic way how a *Problem* was solved including its evaluation		modeling	
Technology Roadmap	Current description of a given *Core Competence* and planned future development of the competence		modeling; provides detailed info about *Core Competence*	

E.3 Terminal And Nonterminal Concept Attributes

Concept: Artifact
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight	Scenario name
artif	title	name or title of artifact	1	Text	-	yes	-	-	1	FindArtifact, FindIQProcess
	type	template or complete document	1	ArtifType	document	yes	-	-	10	FindArtifact
	owner	someone who coordinates the changes of the artifact (value = »unknown« if owner is not an IESE employee); owners must be updated if the old owner leaves!	1	iIESE Employee	-	no	-	-	5	FindArtifact
	authors	anybody from IESE who has contributed to the artifact	1..*	iIESE Employee	-	yes	-	-	3	FindArtifact
	date	date of release	1	IESEDate	sysdate()	yes	-	-	1	FindArtifact, FindIQProcess
	version	version number	1	Version	-	yes	-	-	1	FindArtifact
	contents	URL to document	1	Text	-	no	-	-	0	FindArtifact, FindIQProcess
	keywords	keywords describing the contents	1	Keywords	-	no	-	-	1	FindArtifact, FindIQProcess

Concept: Artifact
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight	Scenario name
I/F	follow-ups	artifacts that were derived from this one	0..*	defines [Artifact].[based on]	{}	yes	-	-	0	FindArtifact, FindIQProcess
	improvement suggestions	suggestions on how to improve the artifact	0..*	has-part [Improvement Suggestion].[Object]	{}	yes	-	no of impr sugg	1	FindArtifact, FindIQProcess
	no of impr sugg	number of pending improvement suggestions	1	Cardinal	-	yes	card([improvement suggestions])	-	2	FindArtifact, FindIQProcess
ctxt	based on	artifacts from which this version has been derived	0..*	derived-from [Artifact].[follow-ups]	{}	yes	-	-	1	FindArtifact

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

assertion: TRUE

Concept: Guideline
Super concept: Process Info

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight	Scenario name
artif	justification	why the guideline avoids the corresponding problem	1	Text	-	no	-	-	0	IdentifyProjectRisks, GetRelevantProjectInfo, GetImprovementProgress
I/F	problem	the problem this guideline avoids	1	part-of [Problem].[guidelines]	-	yes	-	-	1	GetRelevantProjectInfo
ctxt	no of successful applications	number of projects in which this guideline has been applied successfully	1	Validity	0	yes	-	no of unsuccessful applications, validation ratio	1	GetRelevantProjectInfo

Concept: Guideline
Super concept: Process Info

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight	Scenario name
ctxt	no of unsuccessful applications	number of projects in which this guideline has not been applied successfully	1	Validity	-	yes	card([evaluation]) - [no of successful applications]	-	1	GetRelevantProjectInfo
	validation ratio	number of successful applications in percent	1	Percent	-	yes	100*[no of successful applications]/ card([evaluation])	-	1	GetRelevantProjectInfo
	failure reasons	for all unsuccessful projects: reasons for failure	1	Text	-	no	-	-	0	GetRelevantProjectInfo

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

assertion: TRUE

Concept: Improvement Suggestion
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight	Scenario name
artif	object	which artifact to improve	1	part-of [Artifact].[improvement suggestions]	-	yes	-	-	1	FindImprovementSuggestions, GetImprovementProgress
	description	what to change	1	Text	-	yes	-	-	0	FindImprovementSuggestions, GetImprovementProgress
	problem	what is to be prevented	1	part-of [Problem].[improvement suggestions]	-	yes	-	-	1	FindImprovementSuggestions
	justification	why this will prevent the problem from recurring; also cost/benefit	1	Text	-	no	-	-	0	FindImprovementSuggestions, GetImprovementProgress

Concept: Improvement Suggestion
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight	Scenario name
artif	author	contact person for questions regarding this improvement suggestion	1	IIESE Employee	-	yes	-	-	0	FindImprovementSuggestions
	date	creation date for this suggestion	1	IESEDate	sysdate()	yes	-	-	1	FindImprovementSuggestions
	closed	date when this suggestion was worked in or rejected	1	IESEDate	-	yes	-	-	1	GetImprovementProgress
	status	status of implementation	1	SolStatus	-	yes	-	-	1	FindImprovementSuggestions, GetImprovement-Progress, FindOpen-Problems
	comments	e.g., how the description was devised, why the improvement suggestion was rejected, or how the improvement suggestion was actually worked in	1	Text	-	no	-	-	1	FindImprovementSuggestions, GetImprovementProgress
I/F										
ctxt										

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

assertion: TRUE

Concept: IQ Process
Super concept: Artifact

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight	Scenario name
artif	viewpoint	the role from whose perspective the IQ process is described (main actor)	1	iRole	-	yes	-	-	5	FindIQProcess
	involved roles	roles involved in executing the process (except the viewpoint)	0..*	iRole	{}	yes	-	-	3	FindIQProcess
	guidelines	context-specific guidelines	0..*	has-part [Guideline].[object]	{}	yes	-	-	1	FindIQProcess
I/F										

Concept: IQ Process
Super concept: Artifact

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight	Scenario name
ctxt	potential problems	problems that can occur during the execution of this IQ process	0..*	has-part [Problem].[affected processes]	{}	yes	-	-	1	FindIQProcess
	observations	observations that were made while applying this IQ process	0..*	has-part [Observation].[object]	{}	yes	-	-	1	FindIQProcess

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

assertion: TRUE

Concept: Observation
Super concept: Process Info

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight	Scenario name
artif										
I/F										
ctxt	validity	number of projects that support this observation	1	Validity	-	yes	card([evaluation])	-	1	GetRelevantProjectInfo

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

assertion: TRUE

Concept: Problem
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight	Scenario name
artif	issue	short title for problem description	1	Text	-	yes	-	-	0	IdentifyProjectRisks, FindSolution, GetImprovementProgress
	reporter	the person reporting the problem	1	ilESE Employee	-	yes	-	-	1	FindSolution, GetImprovementProgress

Concept: Problem
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight	Scenario name
artif	date	date of reporting	1	IESEDate	sysdate()	yes	-	-	1	GetImprovementProgress, FindOpenProblems
	description	problem description	1	Text	-	yes	-	-	0	IdentifyProjectRisks, FindSolution, GetImprovementProgress, FindOpenProblems
	keywords	keywords characterizing the problem	1	Keywords	-	no	-	-	1	IdentifyProjectRisks, FindSolution, GetImprovementProgress, FindOpenProblems
	affected processes	set of processes influenced negatively by this problem	1..*	part-of [IQ process].[problems]	-	no	-	-	5	IdentifyProjectRisks, FindSolution
	affected roles	roles affected by this problem	1..*	iRole	-	no	-	-	5	IdentifyProjectRisks, FindSolution
	status	problem status	1	ProblemStatus	-	yes	-	-	10	IdentifyProjectRisks, FindSolution, GetImprovementProgress, FindOpenProblems
	decision date	date for acceptance/rejection	1	IESEDate	-	no	-	-	1	GetImprovementProgress
	priority	priority for solving the problem	1	Priority	-	yes	-	-	1	GetImprovementProgress, FindOpenProblems

Concept: Problem
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight	Scenario name
artif	contact person	the person responsible for ensuring the progress for solving the problem	1	ilESE Employee	-	no	-	-	1	FindSolution, GetImprovement-Progress, FindOpen-Problems
	creativity team	the team working out a solution for the problem	1..*	ilESE Employee	-	no	-	-	1	FindSolution, GetImprovement-Progress, FindOpen-Problems
	guidelines	guidelines that will avoid this problem in the future	0..*	has-part [Guideline].[problem]	{}	yes	-	-	1	FindSolution, GetImprovementProgress
	improvement suggestions	suggestions that will prevent the problem from recurring if the corresponding artifacts are changed accordingly	0..*	has-part [Improvement Suggestion].[problem]	{}	yes	-	-	1	GetImprovement-Progress, FindOpen-Problems
	solutions	pragmatic solutions used in projects including their evaluation	0..*	has-part [Solution].[proplem]	{}	yes	-	-	1	FindSolution
I/F	cause	real problem underlying the problem description	1	Text	-	no	-	-	0	IdentifyProjectRisks, FindSolution, GetImprovementProgress
	cause keywords	keywords for the cause	1	Keywords	-	no	-	-	1	IdentifyProjectRisks, FindSolution, GetImprovementProgress
ctxt	projects	projects in which this problem occurred (initially when a problem is reported, only one project is specified; as the result of the problem analysis phase, problems with similar descriptions may be merged)	1..*	derived-from [Project].[problems]	-	no	-	-	1	IdentifyProjectRisks, FindSolution

Concept: Problem
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight	Scenario name
ctxt	situation	the situation in which the problem occurred	1	Text	-	yes	-	-	0	IdentifyProjectRisks, FindSolution

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

assertion: TRUE

Concept: Process Info
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight	Scenario name
artif	description	contents of process information	1	Keywords	-	yes	-	-	1	IdentifyProjectRisks, GetRelevantProjectInfo, GetImprovementProgress
	originator	person who originally contributed this process information	1	ilESE Employee	-	yes	-	-	0	GetRelevantProjectInfo
	date	creation date of this process information	1	IESEDate	sysdate()	yes	-	-	1	GetRelevantProjectInfo
I/F	IQ process	process during which this information should be considered	1	part-of [IQ process].[guidelines]	-	yes	-	-	1	GetRelevantProjectInfo
ctxt	evaluation	projects in which this process information has been identified or applied	0..*	identified-by [Project].[guidelines]	{}	yes	-	[Guideline].[no of unsuccessful applications], [Guideline].[validation ratio], [Observation].[validity]	0	GetRelevantProjectInfo

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

assertion: TRUE

Concept: Project
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight	Scenario name
artif	proj type	classifies project into equivalence classes	1..10	ProjType	-	yes	-	-	5	IdentifyProjectRisks, FindSolution
	proj funding	source of payment	1	ProjFunding	-	yes	-	-	1	IdentifyProjectRisks, FindSolution
	proj manager	the person managing a project	1	iIESE Employee	-	yes	-	-	1	IdentifyProjectRisks, GetRelevantProjectInfo, FindSolution
	scientific lead	the person responsible for assuring that (a) state-of-the-art is applied in the project and (b) research opportunities provided by the project are exploited	1	iIESE Employee	-	yes	-	-	1	IdentifyProjectRisks, GetRelevantProjectInfo, FindSolution
	proj members	everybody working on the project (project staff)	0..*	iIESE Employee	{}	yes	-	IESE team size	1	IdentifyProjectRisks, GetRelevantProjectInfo, FindSolution
	IESE team size	the number of people on the side of Fraunhofer IESE	1	TeamSize	-	yes	card([proj members])+1	proj team size	1	IdentifyProjectRisks, GetRelevantProjectInfo, FindSolution
	sponsor	project manager on side of partner	1	Keywords	-	yes	-	-	1	IdentifyProjectRisks, GetRelevantProjectInfo, FindSolution
	partner experts	methodology or information providers	0..*	Text	{}	yes	-	-	1	IdentifyProjectRisks, GetRelevantProjectInfo, FindSolution
	partner users	people who have to use/ apply IESE technology	0..*	Text	{}	yes	-	-	1	IdentifyProjectRisks, GetRelevantProjectInfo, FindSolution

Concept: Project
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight	Scenario name
artif	industrial partners	industrial partner organizations (needed because it is cumbersome to do text search for partners in the attributes sponsor, partner experts, and partner users)	0..*	Customer	-	yes	-	-	3	IdentifyProjectRisks, GetRelevantProjectInfo, FindSolution
	customer team size	team size on customer's side	1	TeamSize	-	yes	-	proj team size	1	IdentifyProjectRisks, FindSolution
	proj team size	total team size	1	TeamSize	-	yes	[IESE team size] + [customer team size]	-	1	IdentifyProjectRisks, GetRelevantProjectInfo, FindSolution
	kickoff meeting	date when the kick off meeting was held with the customer	1	IESEDate	-	no	-	-	1	future analyses (different definition of proj duration)
	start of project	date when the actual project work started	1	IESEDate	-	yes	-	duration	1	calculation of duration
	negotiated project end	date as negotiated with customer	1	IESEDate	-	no	-	-	1	future analyses (different definition of proj duration)
	end of project	last day actual project work was performed	1	IESEDate	-	yes	-	duration	1	calculation of duration
	final presentation	day when the final presentation took place	1	IESEDate	-	no	-	-	1	future analyses (different definition of proj duration)
	duration	length of project in days	1	Duration	-	yes	daydiff([end of project], [start of project])	-	1	IdentifyProjectRisks, GetRelevantProjectInfo, FindSolution
	paid effort	effort paid according to contract	1	Effort	-	yes	-	-	1	IdentifyProjectRisks, GetRelevantProjectInfo, FindSolution

Concept: Project
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight	Scenario name
artif	internally planned effort	effort planned internally to execute the project	1	Effort	-	no	-	-	1	IdentifyProjectRisks, GetRelevantProjectInfo, FindSolution
	actual effort	effort actually spent on the project	1	Effort	-	yes	-	-	1	IdentifyProjectRisks, GetRelevantProjectInfo, FindSolution
	objective	official objective of project	1	Keywords	-	yes	-	-	1	IdentifyProjectRisks, GetRelevantProjectInfo
	goals	(internal) goals of the involved project partners	1	Keywords	-	yes	-	-	1	IdentifyProjectRisks, GetRelevantProjectInfo
	contract reasons	reason(s) why Fraunhofer IESE got contract	1	Text	-	yes	-	-	0	GetRelevantProjectInfo
	acquisition reasons	reason(s) why Fraunhofer IESE acquired project (except for IESE goal stated under [goals])	1	Text	-	yes	-	-	0	IdentifyProjectRisks, GetRelevantProjectInfo
	comments	notes complementing the formal characterization	1	Text	-	no	-	-	0	IdentifyProjectRisks, GetRelevantProjectInfo
I/F										
ctxt	guidelines	guidelines applied in this project	0..*	identifies [Guidelines].[evaluation]	{}	yes	-	-	1	
	observations	observations made during the project	0..*	identifies [Observation].[projects]	{}	yes	-	-	1	

Concept: Project
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight	Scenario name
	problems	problems occurred during the project	0..*	identifies [Problem].[projects]	{}	yes	-	-	1	

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

assertion: TRUE

Concept: Solution
Super concept: CONCEPT

Layer	Name	Description	Cardinality	Type	Default value	Mandatory	Value inference	To infer	Standard weight	Scenario name
artif	problem	the problem this solution tried to solve	1	part-of [Problem].[solutions]	-	yes	-	-	1	
	description	what was done to cope with the problem	1	Text	-	yes	-	-	0	FindSolution
	justification	why this solution mitigates or solves the problem	1	Text	-	no	-	-	0	FindSolution
	result	outcome (did the solution help?)	1	Text	-	no	-	-	0	FindSolution
I/F										
ctxt										

sim_{artif}, $sim_{I/F}$, sim_{ctxt}: standard

assertion: TRUE

E.4 Type Table

Table E.3: Type Table

Name	Supertype	Value range	Unit of measure	Similarity
ArtifType	Symbol	{document, template}	n/a	standard
Customer	TaxonomySymbol	see Figure E.1	n/a	standard
Duration	Cardinal	1..3000	days	standard
Effort	Real	1..1000	person days	standard
IESEDate	Date	1.1.96 ..31.12.2200	n/a	standard

Name	Supertype	Value range	Unit of measure	Similarity
iIESE Employee	Symbol	{Klaus-Dieter Althoff, ...}	n/a	standard
iRole	Taxonomy-Symbol	see Taxonomy »iRole«	n/a	standard
Keywords	Text	inherit	n/a	Treat an attribute value as a set of keywords: Let w_q be the set of words contained in the query and w_i the set of words contained in the instance. Then $\mathrm{sim}(i,q) = \frac{\lvert w_q \cap w_i \rvert}{\lvert w_q \rvert}$
Percent	Real	0..100	n/a	standard
Priority	OrderedSymbol	{low, medium, high}	n/a	standard
ProblemStatus	OrderedSymbol	{new, accepted, in progress, mutated, solved, rejected}	n/a	see Table E.4
ProjFunding	TaxonomySymbol	see Figure E.2	n/a	standard
ProjType	Symbol	{Consulting, Education, Expert Report, Infrastructure, R&D, Seminar, Study, Training, Transfer, Workshop}	n/a	standard
SolStatus	OrderedSymbol	{new, accepted, in progress, worked-in, rejected}	n/a	see Table E.5
TeamSize	Cardinal	1..50	n/a	standard
Validity	Integer	0..100	n/a	standard
Version	Real	0.01..99.99	n/a	standard

Taxonomy »iRole«

Possible values are only leaves, but no intermediary nodes of the taxonomy:

- Project
  - Contract Management
  - Project Manager
  - Project Member
  - External Representation
  - Marketing and Acquisition
  - Public Project Coordinator
  - Industrial Project Coordinator
  - Domain Coordinator
  - Meetings
    - Meeting Coordinator
    - Meeting Leader

    - Recorder
- Research
    - Competence Expert
    - PhD Advisor
    - PhD Candidate
    - Thesis Advisor
    - Thesis Worker
    - HiWi Advisor
    - HiWi
- Support
    - Accounting
        - Procurement
        - Budget Planning and Control
        - Project Accounting
        - Travel Accounting
    - Personnel
        - Application Management
        - Application Reviewer
        - Personnel Administration
        - Personnel Development
    - Process Owner
    - Public Relations
    - LIS Manager
    - IT Services
- Line Management
    - Director
    - Department Head
    - Group Leader
    - Secretary

Figure E.1: TaxonomySymbol »Customer«

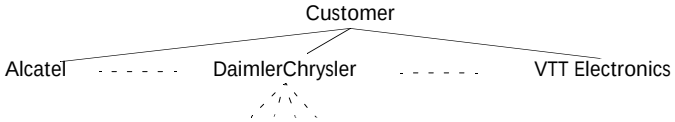


Figure E.2: TaxonomySymbol »ProjFunding«

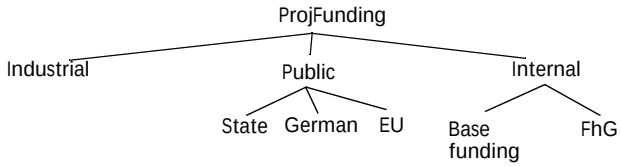


Table E.4: Similarity for »ProblemStatus«

	new	**accepted**	**in progress**	**mutated**	**solved**	**rejected**
new	1	0.67	0.33	0	0	0.67
accepted	0.67	1	0.67	0.33	0.5	0.67
in progress	0.33	0.67	1	0.67	0.67	0.67
mutated	0	0.33	0.67	1	0	0
solved	0	0.5	0.67	0	1	0
rejected	0.67	0.67	0.67	0	0	1

Table E.5: Similarity for »SolStatus«

	new	accepted	in progress	worked-in	rejected
new	1	0.67	0.33	0	0.67
accepted	0.67	1	0.67	0.33	0.33
in progress	0.33	0.67	1	0.67	0.67
worked-in	0	0.33	0.67	1	0
rejected	0.67	0.33	0.67	0	1

E.5 Symbol Glossary

Table E.6: Symbol Glossary

Type	Symbol	Description
ArtifType	document	A complete report or other document; its primarily intended utilization is the dissemination of the contents
	template	An incomplete report or other document, often containing only formatting styles or defining the structure of a report or document; its primarily intended utilization is the guidance for writing a new report or document of the same kind
Customer	Companies and their organizational units	
iIESE Employee	Each symbol corresponds to the name of a person	
iRole	see IESE Roles of IESE Quality Management System	
Priority	low	Solving the problem will boost people's motivation
	medium	Some people will have to put in extra project effort if this problem is not solved
	high	Fraunhofer IESE will lose contracts if this problem is not solved
Problem-Status (see also Figure E.3)	accepted	A problem has been accepted as such; either the improvement squad itself or an appointed creativity team will start working on a solution soon
	mutated	A more fundamental problem is underlying this problem; if the fundamental problem is solved, this problem will be solved as well; therefore, the search for a solution for this problem is abandoned
	new	A new problem has been stored in the problem memory; the improvement squad will decide soon whether to accept or reject the problem
	rejected	A problem was not solved either because its underlying cause was a misunderstanding or because its solution would not yield a positive cost/benefit ratio
	in progress	A creativity team has started to work on the problem
	solved	A solution has been devised
ProjFunding	Industrial	Projects paid by industry
	Internal	Projects paid by FhG funds
	- Base funding	Projects funded through IESE base funds
	- FhG	Projects funded through ZV Munich (SEF, OEF)
	Public	Projects funded publicly
	- EU	Projects funded by some framework program of the EU
	- German	Projects funded by ministry of German government
	- State	Projects funded by ministry of Rhineland-Palatinate

Type	Symbol	Description
ProjType	Consulting	A project where an IESE expert gives advice to a manager of a company; however, IESE is not involved actively in training and/or transfer of technology
	Education	A project where a training course is prepared once, but held repeatedly
	Expert Report	A special type of *study* where special emphasis is put on expert judgment (German: »Gutachten«)
	Infrastructure	A project that aids in running IESE (e.g., the Corporate Information Network)
	R&D	A project where IESE develops a company-specific technology
	Seminar	A project with the objective to train/educate the people of a company
	Study	A project where several alternatives are surveyed/developed, compared, and documented, but not actually applied
	Training	A project where a company receives training once; in comparison to a *seminar*, *training* is longer and may take from several days to several months
	Transfer	A project where IESE technology is introduced into a company with the objective that the company continues to apply this technology – even after the cooperation
	Workshop	A meeting (possibly over several days) where a limited set of topics is discussed; in contrast to *consulting*, a *workshop* emphasizes the exchange of ideas and the demonstration of techniques, skills, etc.
SolStatus (see also Figure E.4)	accepted	A solution has been accepted as such; the artifact owner will work in the improvement suggestion
	new	A new improvement suggestions has been stored as part of the pending improvement suggestions for an artifact; the artifact owner will decide soon whether to accept or reject the suggestion
	rejected	An improvement suggestion was not worked in the change would be too cost-intensive
	in progress	The artifact owner has started to change the artifact and will consider the improvement suggestion
	worked-in	The improvement suggestion has been worked in

Figure E.3: Possible state transitions between problem statuses

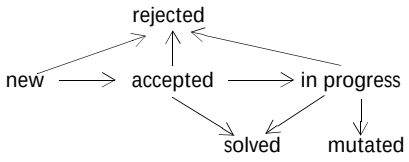
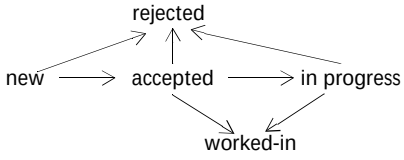


Figure E.4: Possible state transitions between solution statuses

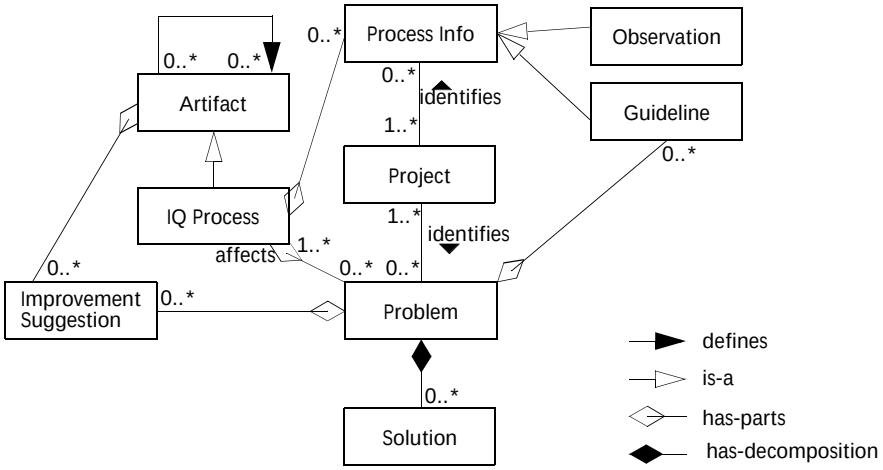


E.6 Kinds

Table E.7: Kinds

Kind	Reverse name	Description	StructureProperties	
identifies	identified-by	documents what new insights have been gained by a project or organizational unit	DAG	the source of a »identifies« relationship must be either a *Project* or an *IESE Organization*

E.7 Graphical Representation of Semantic Relationships



Appendix F:
Questionnaire for the Experiment

The following questions are used for the experiment comparing the human-based and the repository-based approach for acquiring information.

Your answers will be kept confidential and anonymitized for analysis.

F.1 Contact Information for Possible Further Questions

Name: .....

Date: .....

F.2 Background Experience

Please answer the following questions before starting the experiment:

1 Have you been involved as a project member in a public project of Fraunhofer IESE (please mark one answer)?
- O no
- O yes, once
- O yes, more than once

2 Have you been involved as a project member in an industrial project of Fraunhofer IESE (please mark one answer)?
- O no
- O yes, once
- O yes, more than once

3 Have you been involved as a project member in preparing a study for a company (please mark one answer)?
- O no
- O yes, once
- O yes, more than once

4 Have you been acting as a project manager in a project of Fraunhofer IESE (please mark one answer)?
- O no
- O yes, once
- O yes, more than once

5 How long have you been working for Fraunhofer IESE? ..... months

6 How long have you been working in the IT field? ..... months

F.3 Human-Based Approach

Please read the project situation given to you carefully. Then answer the following questions.

7 Whom would you have asked to acquire the needed information (put »none« if you would not have asked anybody)?

.....

.....

.....

.....

8 During the session, please evaluate the information given using the following table:

Info Id	unexpected	not unexpected	useful	not useful	don't know	known	partially known	unknown	sensible	not sensible	don't know

Info Id	unexpected	not unexpected	useful	not useful	don't know	known	partially known	unknown	sensible	not sensible	don't know

At the end of this session, please answer:

9 How long do you think it would have taken (effort) to make an appointment and to actually talk to the selected people in the selected order?

..... person hours (fractions allowed, e.g., 0.5 person hours)

10 How long do you think it would have taken (duration) until you had a chance to talk to all selected people in the selected order?

..... hours (fractions allowed, e.g., 0.5 hours)

F.4 Repository-Based Approach

11 Before reading further, please record the start time: .....

12 Please use INTERESTS to find useful project experience (= guidelines and observations). Then evaluate the presented guidelines and observations using the following table.

	Case Id	unexpected	not unexpected	useful	not useful	don't know	known	partially known	unknown	sensible	not sensible	don't know
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												
11												
12												
13												
14												
15												

	Case Id	unexpected	not unexpected	useful	not useful	don't know	known	partially known	unknown	sensible	not sensible	don't know
16												
17												
18												
19												
20												
21												
22												
23												
24												
25												
26												
27												
28												
29												
30												

13 When you are done, please record the finish time: .....

F.5 Subjective Ratings

14 Please evaluate the following statements:

Info Id	Description	unexpected	not unexpected	useful	not useful	don't know	known	partially known	unknown	sensible	not sensible	don't know
T-1	Talk to people who have been in contact with Daimler-Chrysler: Have they sent documents to Daimler-Chrysler? What was the reaction of DaimlerChrysler?											
T-2	Proceed incrementally, that is, send the parts one after the other to DaimlerChrysler; this means that you have to define increments.											
T-3	Get reports and study on reuse that have been published by DaimlerChrysler. Ask DaimlerChrysler's marketing department.											
T-4	Ask questions politely.											

15 Please specify the reasons why the statements are useful or not useful:

T-1: .....

T-2: .....

T-3: .....

T-4: .....

F.6 Debriefing

16 Is the human-based approach realistic regarding the information gain in comparison to the normal process used in your every day work (please mark one answer)?
- O totally realistic
- O mostly realistic
- O realistic and unrealistic parts balance out
- O mostly unrealistic
- O totally unrealistic

17 In general, which of the two approaches do you think is more effective (please mark one answer)?
- O human-based approach
- O repository-based approach

18 In general, which of the two approaches do you think is more efficient (please mark one answer)?
O human-based approach
O repository-based approach

19 For the experiment, which of the two approaches do you think is more effective (please mark one answer)?
O human-based approach
O repository-based approach

20 For the experiment, which of the two approaches do you think is more efficient (please mark one answer)?
O human-based approach
O repository-based approach

21 Which of the approaches would you apply if you were confronted with new projects (a combination of both is also possible)?
◊ human-based approach
◊ repository-based approach

22 How high was your motivation for applying the human-based approach (please mark one answer)?
O extremely motivated
O motivated
O neither
O unmotivated
O extremely unmotivated

23 How high was your motivation for applying the repository-based approach (please mark one answer)?
O extremely motivated
O motivated
O neither
O unmotivated
O extremely unmotivated

Questionnaire for the
Experiment

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.

[AAB+95] Klaus-Dieter Althoff, Eric Auriol, Ralph Bergmann, Sean Breen, Stefan Dittrich, Roy Johnston, Michel Manago, Ralph Traphöner, and Stefan Wess. Case-based reasoning for decision support and diagnostic problem solving: The INRECA approach. In *Proceedings of the Third German Conference on Knowledge-Based Systems (XPS-95)*, 1995.

[ABH+99] Klaus-Dieter Althoff, Andreas Birk, Susanne Hartkopf, Wolfgang Müller, Markus Nick, Dagmar Surmann, and Carsten Tautz. Managing software engineering experience for comprehensive reuse. In *Proceedings of the Eleventh Conference on Software Engineering and Knowledge Engineering*, pages 10–19, Kaiserslautern, Germany, June 1999. Knowledge Systems Institute, Skokie, Illinois, USA.

[ABMN99] Klaus-Dieter Althoff, Frank Bomarius, Wolfgang Müller, and Markus Nick. Using case-based reasoning for supporting continuous improvement processes. In Petra Perner, editor, *Proceedings of the Twelfth German Workshop on Machine Learning*, pages 54–61, Leipzig, Germany, September 1999. Institute for Image Processing and Applied Informatics.

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

[ABvWT98] Klaus-Dieter Althoff, Andreas Birk, Christiane Gresse von Wangenheim, and Carsten Tautz. Case-based reasoning for experimental software engineering. In Mario Lenz, Brigitte Bartsch-Spörl, Hans-Dieter Burkhard, and Stefan Wess, editors, *Case-Based Reasoning Technology - From Foundations to Applications*, number 1400, chapter 9, pages 235–254. Springer-Verlag, Berlin, Germany, 1998.

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

[AHU83] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *Data Structures and Algorithms*. Computer Science and Information Processing. Addison-Wesley Publishing Company, Reading, Mass., USA, 1983.

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

[AMY88] Robert M. Akscyn, Donald L. McCracken, and Elise A. Yoder. KMS: A distributed hypermedia system for managing knowledge in organizations. *Communications of the ACM*, 31(7):820–835, July 1988.

[AN95] A. Aamodt and M. Nygard. Different roles and mutual dependencies of data, information and knowledge - an AI perspective on their integration. *Data and Knowledge Engineering*, 16:191–222, 1995.

[And88] John R. Anderson. *Cognitive Psychology: An Introduction (in German)*. Spektrum der Wissenschaft, Heidelberg, Germany, 1988.

[ANT99a] Klaus-Dieter Althoff, Markus Nick, and Carsten Tautz. CBR-PEB: An application implementing reuse concepts of the experience factory for the transfer of CBR system know-how. In Erica Melis, editor, *Proceedings of the Seventh Workshop on Case-Based Reasoning during Expert Systems '99 (XPS-99)*, pages 39–48, Würzburg, Germany, March 1999.

[ANT99b] Klaus-Dieter Althoff, Markus Nick, and Carsten Tautz. Improving organizational memories through user feedback. In Frank Bomarius, editor, *Workshop on Learning Software Organizations*

at SEKE'99, pages 27–44, Kaiserslautern, Germany, June 1999. Fraunhofer IESE, Kaiserslautern, Germany.

[AP94] 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.

[AR99] Klaus-Dieter Althoff and Michael M. Richter. Similarity and utility in non-numerical domains. In Wolfgang Gaul and Martin Schader, editors, *Mathematische Methoden der Wirtschaftswissenschaften*, pages 403–413. Physica-Verlag, Heidelberg, 1999.

[Ara89] G. Arango. Domain analysis - from art form to engineering discipline. In *Proceedings of the Fifth International Workshop on Software Specification and Design*, pages 152–159, September 1989.

[Ara94] Guillermo Arango. Domain analysis methods. In Wilhelm Schäfer, Rubén Prieto-Díaz, and Masao Matsumoto, editors, *Software Reusability*, pages 17–49. Ellis Horwood Ltd., New York, USA, 1994.

[ASR99] Hans Akkermans, Piet-Hein Spiel, and Alex Ratcliffe. Problem, opportunity, and feasibility analysis for knowledge management: An industrial case study. In *Proceedings of the Twelfth Workshop on Knowledge Acquisition, Modeling, and Management (KAW '99), Track on "Knowledge Management and Knowledge Distribution throughout the Internet"*, Banff, Alberta, Canada, October 1999. URL: http://sern.ucalgary.ca/KAW/KAW99/papers/Akkermans1/kaw99-jma.pdf.

[ATW96] Klaus-Dieter Althoff, Ralph Traphöner, and Stefan Wess. Efficient integration of induction and Case-Based Reasoning: The INRECA system. In Hans Czap, Peter Jaenecke, and Heinz Peter Ohly, editors, *Analogie in der Wissensrepräsentation: Case-Based Reasoning und Räumliche Modelle*, volume 4 of *Advances in Knowledge Organization*, pages 13–22. INDEKS-Verlag, Frankfurt, Germany, 1996.

[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.

[BA98] R. Bergmann and K.-D. Althoff. Methodology for building case-based reasoning applications. In M. Lenz, B. Bartsch-Spörl, H. D. Burkhard, and S. Wess, editors, *Case-Based Reasoning Technology - From Foundations to Applications*, pages 299–328. Springer-Verlag, 1998.

[BAB+87] Bruce A. Burton, Rhonda Wienk Aragon, Stephen A. Bailey, Kenneth D. Koehler, and Lauren A. Mayes. The reusable software library. *IEEE Software*, 4(7):25–33, July 1987.

[Bas95] Victor R. Basili. The Experience Factory and its relationship to other quality approaches. In Marvin V. Zelkowitz, editor, *Advances in Computers*, volume 41, pages 65–82. Academic Press, 1995.

[BB87] Günter Bamberg and Franz Baur. *Statistics (in German)*. Oldenbourg Verlag, Munich, Germany, fifth edition, 1987.

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

[BBG+98] Ralph Bergmann, Sean Breen, Mehmet Göker, Michel Manago, Jürgen Schumacher, Armin Stahl, Emmanuelle Tartarin, Stefan Wess, and Wolfgang Wilke. The INRECA-II methodology for building and maintaining CBR applications. In Lothar Gierl and Mario Lenz, editors, *Proceedings of the Sixth German Workshop on Case-Based Reasoning*, volume 7 of *IMIB Series*, pages 3–12, Berlin, Germany, March 1998. Institut für Medizinische Informatik und Biometrik, Universität Rostock.

[BBG+99] Ralph Bergmann, Sean Breen, Mehmet Göker, Michel Manago, and Stefan Wess. *Developing Industrial Case-Based Reasoning Applications – The INRECA Methodology*. Springer Verlag, 1999.

[BC95] Victor R. Basili and Gianluigi Caldiera. Improve software quality by reusing knowledge and experience. *Sloan Management Review*, pages 55–64, Fall 1995.

[BCC92] Victor R. Basili, Gianluigi Caldiera, and Giovanni Cantone. A reference architecture for the com-

ponent factory. *ACM Transactions on Software Engineering and Methodology*, 1(1), 1992.

[BCR94a] 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.

[BCR94b] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. Goal Question Metric Paradigm. In John J. Marciniak, editor, *Encyclopedia of Software Engineering*, volume 1, pages 528–532. John Wiley & Sons, 1994.

[BDF96] Lionel Briand, John Daly, and Pierfrancesco Fusaro. Design of empirical studies: Tutorial for IESE project members, 1996. IESE-internal document.

[BDF99] C. Bellettini, E. Damiani, and M. G. Fugini. User opinions and rewards in a reuse-based development system. In *Proceedings of the Fifth ACM SIGSOFT Symposium on Software Reusability*, pages 151–158, Los Angeles, CA, USA, May 1999. ACM Press.

[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.

[BDY94] Victor R. Basili, Michael K. Daskalantonakis, and Robert H. Yacobellis. Technology transfer at Motorola. *IEEE Software*, 11(2):70–76, March 1994.

[BE95] R. Bergmann and U. Eisenecker. Case-based reasoning for supporting reuse of object-oriented software: A case study (in German). In M. M. Richter and F. Maurer, editors, *Expertensysteme 95*, pages 152–169. infix Verlag, 1995.

[Bec96] Stefan Becker. Overview of approaches and tools for comprehensive software reuse (in German). Master's thesis, Department of Computer Science, University of Kaiserslautern, 67653 Kaiserslautern, Germany, January 1996.

[BFG98] V. Richard Benjamins, Dieter Fensel, and Asunción Gómez-Pérez. Knowledge management through ontologies. In *Proceedings of the Second International Conference on Practical Aspects of Knowledge Management (PAKM '98)*, Basel, Switzerland, October 1998.

[BFGPGP98] M. Blázquez, M. Fernández, J. M. García-Pinar, and A. Gómez-Pérez. Building ontologies at the knowledge level using the ontology design environment. In *Proceedings of the Eleventh Workshop on Knowledge Acquisition, Modeling, and Management (KAW '98)*, 1998. URL: http://ksi.cpsc.ucalgary.ca:80/KAW/KAW98/blazquez/.

[BFOS97] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Chapman & Hall, New York, New York, USA, 1997.

[BFP95] Roberto Bellinzona, Maria Grazia Fugini, and Barbara Pernici. Reusing specifications in OO applications. 12(2):65–75, March 1995.

[BI95] Michael Bieber and Tomás Isakowitz. Designing hypertext applications. *Communications of the ACM*, 38(8):26–29, August 1995.

[Bir97] Andreas Birk. Modeling the application domains of software engineering technologies. In *Proceedings of the Twelfth IEEE International Automated Software Engineering Conference*. IEEE Computer Society Press, 1997.

[Bir00] Andreas Birk. *A Knowledge Management Infrastructure for Systematic Improvement in Software Engineering*. PhD thesis, University of Kaiserslautern, Germany, 2000. To appear.

[BK99] Andreas Birk and Felix Kröschel. A knowledge management life cycle for experience packages on software engineering technologies. In Frank Bomarius, editor, *Workshop on Learning Software Organizations at SEKE'99*, pages 115–126, Kaiserslautern, Germany, June 1999. Fraunhofer IESE, Kaiserslautern, Germany.

[BKP94] Christian Bunse, Bernd Krüger, and Andreas Portz. Overview and evaluation of commercially available software quality assurance tools (in German). Technical Report STTI-94-04-D, Software Technology Transfer Initiative, 67663 Kaiserslautern, Germany, 1994. This report is not publicly available.

[BLW90] James C. Browne, Taejae Lee, and John Werth. Experimental evaluation of a reusability-oriented parallel programming environment. *IEEE Transactions on Software Engineering*, 16(2):111–120, February 1990.

[BMY95] V. Balasubramanian, Bang Min Ma, and Joonhee Yoo. A systematic approach to designing a WWW application. *Communications of the ACM*, 38(8):47–48, August 1995.

[Bog92] R. Bogaschewsky. Hypertext/hypermedia systems: An overview (in German). *GI Informatik Spektrum*, 15(3):127–143, June 1992.

[BP84] Ted J. Biggerstaff and Alan J. Perlis. Foreword. *IEEE Transactions on Software Engineering*, 10(5):474–476, September 1984.

[BPS91] Ottavia Bassetti, Daniele Pagani, and Marnie Smyth. Applications navigator: Using hypertext to support effective scientific information exchange. In *Proceedings of the Hypertext 91*, pages 411–416, San Antonio, Texas, USA, December 1991.

[BR87] Ted J. Biggerstaff and Charles Richter. Reusability framework, assessment, and directions. *IEEE Software*, 4(2), March 1987.

[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.

[BR91] Victor R. Basili and H. Dieter Rombach. Support for comprehensive reuse. *IEEE Software Engineering Journal*, 6(5):303–316, September 1991.

[Bro98] Mikael Broomé. Requirements and implementation approaches for an experience base. Master's thesis, Lund University, Lund, Sweden, July 1998.

[Bro00] Oliver Brosda. An integrated experience base for the case-based retrieval of technology experience packages. Diploma thesis, Fraunhofer IESE, University of Kaiserslautern, Germany, May 2000.

[BS96] François Bry and Dietmar Seipel. Deductive database management systems (in German). *GI Informatik Spektrum*, 19(4):214–215, August 1996.

[BS99] Brigitte Bartsch-Spörl. Keys of success for CBR applications in practice. In *Proceedings of the Industry Day at ICCBR '99*, Monastery Seeon, Germany, July 1999.

[BSAM97] Brigitte Bartsch-Spörl, Klaus-Dieter Althoff, and Alexandre Meissonnier. Learning from and reasoning about case-based reasoning systems. In *Proceedings of the Fourth German Conference on Knowledge-Based Systems (XPS97)*, March 1997.

[BSLC96] Sidney Bailin, Mark Simos, Larry Levine, and Dick Creps. Learning and inquiry based reuse adoption (LIBRA): A field guide to reuse adoption through organizational learning, version 1.1. STARS Asset A1098, Lockheed Martin Tactical Defense Systems, February 1996. URL: http://direct.asset.com/.

[BSW96] Brigitte Bartsch-Spörl and Stefan Wess (eds.). Special issue on case-based reasoning (in German). *Künstliche Intelligenz*, 1, 1996.

[BT98a] Andreas Birk and Carsten Tautz. Knowledge management of software engineering lessons learned. In *Proceedings of the Tenth Conference on Software Engineering and Knowledge Engineering*, San Francisco Bay, CA, USA, June 1998. Knowledge Systems Institute, Skokie, Illinois, USA.

[BT98b] Andreas Birk and Carsten Tautz. Knowledge management of software engineering lessons learned. Technical Report IESE-Report No. 002.98/E, Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany, 1998.

[Bur98] Hans-Dieter Burkhard. Extending some concepts of CBR – foundations of case retrieval nets. In Mario Lenz, Brigitte Bartsch-Spörl, Hans-Dieter Burkhard, and Stefan Wess, editors, *Case-Based Reasoning Technologies: From Foundations to Applications*, number 1400 in Lecture Notes in Artificial Intelligence, pages 17–34. Springer-Verlag, Berlin, Germany, 1998.

[Bus45] Vannevar Bush. As we may think. *Atlantic Monthly*, 176(1):101–108, July 1945.

[BWP98] Hans-Jörg Bullinger, Kai Wörner, and Juan Prieto. Knowledge management today: Data, facts, trends (in German). Technical report, Fraunhofer IAO, Stuttgart, Germany, 1998.

[BZM+95] Victor Basili, Marvin Zelkowitz, Frank McGarry, Jerry Page, Sharon Waligora, and Rose Pajerski. SEL's software process-improvement program. *IEEE Software*, 12(6):83–87, November 1995.

[Çak90] A. Çakir. How human is communication? (in German). *GI Informatik Spektrum*, 13(6):331–333, 12 1990.

[Cap88] J. Anthony Capon. *Elementary Statistics for the Social Sciences*. Wadsworth Publishing Company, Belmont, California, USA, 1988.

[CB91] Gianluigi Caldiera and Victor R. Basili. Identifying and qualifying reusable components. *IEEE Software*, 8:61–70, February 1991.

[CC97] Yonghao Chen and Betty H. C. Cheng. Facilitating an automated approach to architecture-based software reuse. In *Proceedings of the Twelfth IEEE International Conference on Automated Software Engineering*, pages 238–245, Incline Village, Nevada, USA, November 1997. IEEE Computer Society Press.

[CDF96] Bonnie Collier, Tom DeMarco, and Peter Fearey. A defined process for project postmortem review. *IEEE Software*, 13(4):65–72, July 1996.

[CHDH94] Les Carr, Wendy Hall, Hugh Davis, and Rupert Hollom. The microcosm link service and its application to the world-wide web. In *Proceedings of the First International Conference on the World-Wide Web*, Geneva, Switzerland, 1994.

[Che76] P. P. Chen. The Entity-Relationship Model - toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[CHEK96] Krzysztof Czarnecki, Reinhard Hasselmann, Ulrich W. Eisenecker, and Wolfgang Köpf. Classexpert: A knowledge-based assistant to support reuse by specialization and modification in Smalltalk. In Murali Sitaraman, editor, *Proceedings of the Fourth International Conference on Software Reuse*, pages 188–194, Orlando, Florida, USA, April 1996. IEEE Computer Society Press, Los Alamitos, California, USA.

[CJB99] B. Chandrasekaran, John R. Josephson, and V. Richard Benjamins. What are ontologies, and why do we need them? *IEEE Intelligent Systems*, 14(1):20–26, February 1999.

[CJR96] C. L. Chee, S. Jarzabek, and C. V. Ramamoorthy. An intelligent process for formulating and answering project queries. In *Proceedings of the Eighth Conference on Software Engineering and Knowledge Engineering*, pages 309–316, Lake Tahoe, Nevada, USA, June 1996.

[CMM84] Jaime G. Carbonell, Ryszard S. Michalski, and Tom M. Mitchell. An overview of machine learning. In Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, chapter 1, pages 3–23. Springer-Verlag, Berlin, 1984.

[Cod70] E.F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, June 1970.

[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.

[Con98] Reidar Conradi. Practical software experience databases: Use the web. In *Proceedings of the Tenth Conference on Software Engineering and Knowledge Engineering*, pages 305–306, San Francisco Bay, California, USA, June 1998. Knowledge Systems Institute, Skokie, Illinois, USA.

[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.

[CRS+92] E. Casais, M. Ranft, B. Schiefer, D. Theobald, and W. Zimmer. OBST: An overview. Technical Report FZI.039.1, Forschungszentrum Informatik (FZI), Karlsruhe, Germany, June 1992.

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

[Das92] Michael K. Daskalantonakis. A practical view of software measurement and implementation experiences within Motorola. *IEEE Transactions on Software Engineering*, 18(11):998–1010, November 1992.

[DBSB91] Premkumar Devanbu, Ronald J. Brachman, Peter G. Selfridge, and Bruce W. Ballard. LaSSIE: a knowledge-based software information system. *Communications of the ACM*, 34(5):34–49, May 1991.

[Dec99] Björn Decker. A system for management of IESE corporate experience. Diploma thesis, Fraunhofer IESE, University of Kaiserslautern, Germany, November 1999.

[DFB99] E. Damiani, M.G. Fugini, and C. Bellettini. A hierarchy-aware approach to faceted classification of object-oriented components. *ACM Transactions on Software Engineering and Methodology*, 8(3):215–262, July 1999.

[DFF97] Ernesto Damiani, Maria Grazia Fugini, and Enrico Fusaschi. A descriptor-based approach to OO code reuse. *IEEE Computer*, 30(10):73–80, October 1997.

[dH98] Robert de Hoog. Methodologies for building knowledge based systems: Achievements and prospects. In Jay Liebowitz, editor, *The Handbook of Applied Expert Systems*. CRC Press, 1998.

[Dit89] K.R. Dittrich. Object-oriented database management systems (in German). 12(4):215–218, August 1989.

[Dit97] K. Dittrich. Database systems (in German). In Peter Rechenberg and Gustav Plomberger, editors, *Informatik-Handbuch*, pages 745–774. Carl Hanser Verlag, Munich, Germany, 1997.

[DP98] Thomas H. Davenport and Laurence Prusak. *Working Knowledge. How Organizations Manage What They Know*. Harvard Business School Press, Boston, USA, 1998.

[DvK95] Liesbeth Dusink and Jan van Katwijk. Reuse dimensions. In Mansur Samadzadeh, editor, *Proceedings of the ACM SIGSOFT Symposium on Software Reusability SSR '95*, pages 137–149, April 1995.

[EB00a] Khaled El Emam and Andreas Birk. Validating the ISO/IEC 15504 measure of software requirements analysis process capability. *IEEE Transactions on Software Engineering*, 2000. To appear.

[EB00b] Khaled El Emam and Andreas Birk. Validating the ISO/IEC 15504 measures of software development process capability. *Journal of Systems and Software*, 51(2):119–149, 2000.

[EMT98] Michel Ezran, Maurizio Morisio, and Colin Tully. *Practical Software Reuse: The Essential Guide*. European Systems and Software Initiative (ESSI), 1998.

[FAFL95] Edward A. Fox, Robert M. Akscyn, Richard K. Furuta, and John J. Legget. Digital libraries. *Communications of the ACM*, 38(4):23–28, April 1995.

[Fel96] Thomas Fellger. Classification and structuring of experience with hypertext-based implementation (in German). Master's thesis, University of Mannheim, Germany, June 1996.

[Fel99] Raimund L. Feldmann. Developing a tailored reuse repository structure: Experience and first results. In Frank Bomarius, editor, *Proceedings of the Workshop on Learning Software Organisations at SEKE'99*, pages 45–55, Kaiserslautern, Germany, June 1999. Fraunhofer IESE, Kaiserslautern, Germany.

[FF96] William B. Frakes and Christopher J. Fox. Quality improvement using a software reuse failure modes model. *IEEE Transactions on Software Engineering*, 22(2):274–279, April 1996.

[FFD96] S. Fäustle, M. G. Fugini, and E. Damiani. Retrieval of reusable components using functional similarity. *Software–Practice and Experience*, 26(5):491–530, May 1996.

[FG90] W. B. Frakes and P. B. Gandel. Representing reusable software. *Information and Software Technology*, 32(10):653–664, December 1990.

[FNP92] M. G. Fugini, O. Nierstrasz, and B. Pernici. Application development through reuse: the Ithaca tools environment. *ACM SIGOIS Bulletin*, 13(2):38–47, August 1992.

[FP94] William B. Frakes and Thomas P. Pole. An empirical study of representation methods for reusable software components. *IEEE Transactions on Software Engineering*, 20(8):617–630, August 1994.

[FPDF98] William Frakes, Rubén Prieto-Díaz, and Christopher Fox. DARE: Domain analysis and reuse environment. *Annals of Software Engineering*, 5:125–141, 1998.

[FT94] William Frakes and Carol Terry. Reuse level metrics. In William B. Frakes, editor, *Proceedings of the Third International Conference on Software Reuse*, pages 139–148, Rio de Janeiro, Brazil, November 1994. IEEE Computer Society Press.

[FT98] 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).

[Gom97] Héctor Gómez. A cognitive approach to software reuse applying case-based reasoning to mutant cases. In *Proceedings of the Ninth Conference on Software Engineering and Knowledge Engineering*, Madrid, Spain, June 1997.

[GAB99] Christiane Gresse von Wangenheim, Klaus-Dieter Althoff, and Ricardo M. Barcia. Intelligent retrieval of software engineering experienceware. In *Proceedings of the Eleventh Conference on Software Engineering and Knowledge Engineering*, pages 128–135, Kaiserslautern, Germany, June 1999. Knowledge Systems Institute, Skokie, IL, USA.

[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.

[Gar98] Stephen R. Gardner. Building the data warehouse. *Communications of the ACM*, 41(9):52–60, September 1998.

[Gäß95] Sven Gäßner. A language for the specification of characterization schemas and similarity measures for the purpose of reuse (in German). Master's thesis, Department of Computer Science, University of Kaiserslautern, 67653 Kaiserslautern, Germany, February 1995.

[Gau95] Wilhelm Gaus. *Documentation and Classification Science (in German)*. Springer-Verlag, Berlin, 1995.

[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.

[GC87] Robert B. Grady and Deborah L. Caswell. *Software Metrics: Establishing a Company-Wide Program*. Prentice Hall, 1987.

[GF91] Dedre Gentner and Kenneth D. Forbus. MAC/FAC: a model of similarity-based retrieval. In *Proceedings of the Thirteenth Annual Conference of Cognitive Science Society*, pages 504–509, 1991.

[GF97] Pedro A. Gonzáles and Carmen Fernández. A knowledge-based approach to support software reuse in object-oriented libraries. In *Proceedings of the Ninth Conference on Software Engineering and Knowledge Engineering*, Madrid, Spain, June 1997.

[GFW94] Martin L. Griss, John Favaro, and Paul Walton. Managerial and organizational issues - starting and running a software reuse program. In W. Schäfer, R. Prieto-Diaz, and M. Matsumoto, editors, *Software Reusability*, chapter 3, pages 51–78. Ellis Horwood Ltd., 1994.

[GHD99] A. D. Griffiths, M. D. Harrison, and A. M. Dearden. Using scenarios to envisage the impact of CBR on decision-making processes. In Sascha Schmitt and Ivo Vollrath, editors, *Challenges for Case-Based Reasoning: Proceedings of the ICCBR '99 Workshops*, Monastery Seeon, Germany, July

1999. Centre for Learning Systems & Applications, University of Kaiserslautern, Germany.

[Gl94] M. R. Girardi and B. Ibrahim. A similarity measure for retrieving software artifacts. In *Proceedings of the Sixth Conference on Software Engineering and Knowledge Engineering*, pages 478–485, Jurmala, Latvia, June 1994.

[Gl95] M. R. Girardi and B. Ibrahim. Using English to retrieve software. *Journal of Systems and Software*, 30(3):249–270, September 1995.

[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.

[Gla94] Wolfgang Glatthaar. Innovation needs wide consensus (in German). *Informatik-Magazin*, pages 10–11, February 1994.

[Glu97] Peter Gluchowski. Data warehouse (in German). *GI Informatik Spektrum*, 20(1):48–49, February 1997.

[GMP95] Franca Garzotto, Luca Mainetti, and Paolo Paolini. Hypermedia design analysis, and evaluation issues. *Communications of the ACM*, 38(8):74–86, August 1995.

[GP98] Asunción Gómez-Pérez. Knowledge sharing and reuse. In Jay Liebowitz, editor, *The Handbook of Applied Expert Systems*. CRC Press, 1998.

[GPB99] Asunción Gómez-Pérez and V. Richard Benjamins. Overview of knowledge sharing and reuse components: Ontologies and problem-solving methods. In V. R. Benjamins, B. Chandrasekaran, A. Gomez-Perez, N. Guarino, and M. Uschold, editors, *Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods (KRR5)*, Stockholm, Sweden, August 1999. URL: http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-18/.

[GR95] Roland Gabriel and Heinz-Peter Röhrs. *Datenbanksysteme*. Springer-Verlag, Berlin, Germany, 1995.

[Gri93] Martin L. Griss. Software reuse: From library to factory. *IBM Systems Journal*, 32(4):548–566, November 1993.

[Gru95] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *International Journal on Human-Computer Studies*, 43:907–928, 1995.

[GS90] Pankaj K. Garg and Walt Scacchi. A hypertext system to manage software life-cycle documents. *IEEE Software*, 7(3):90–99, May 1990.

[Gua97] Nicola Guarino. Understanding, building, and using ontologies. *International Journal on Human-Computer Studies*, 46(2/3):293–310, February/March 1997.

[GvWB98] C. Gresse von Wangenheim, A. von Wangenheim, and R. M. Barcia. Case-based reuse of software engineering measurement plans. In *Proceedings of the Tenth Conference on Software Engineering and Knowledge Engineering*, San Francisco, 1998.

[GW94] Martin L. Griss and Kevin D. Wentzel. Hybrid domain-specific kits for a flexible software factory. In *Proceedings of SAC '94, Reuse and Reengineering Track*, pages 47–52, Phoenix, AZ, USA, March 1994.

[GW95] Martin L. Griss and Kevin D. Wentzel. Hybrid domain-specific kits. *Journal of Systems and Software*, 30(3):213–230, September 1995.

[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.

[Hal96] Thomas J. Haley. Software process improvement at Raytheon. *IEEE Software*, 13(6):33–41, November 1996.

[Har88] David Harel. On visual formalisms. *Communications of the ACM*, 31(5):514–530, May 1988.

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

[Hen95] S. Henninger. Developing domain knowledge through the reuse of project experiences. In Mansur Samadzadeh, editor, *Proceedings of the Symposium on Software Reusability SSR'95*, pages 186–195, April 1995.

[Hen96] Scott Henninger. Supporting the construction and evolution of component repositories. In *Proceedings of the Eighteenth International Conference on Software Engineering*, pages 279–288, Berlin, Germany, March 1996. IEEE Computer Society Press.

[Hen97a] Scott Henninger. Capturing and formalizing best practices in a software development organization. In *Proceedings of the Ninth Conference on Software Engineering and Knowledge Engineering*, pages 24–31, Madrid, Spain, June 1997.

[Hen97b] Scott Henninger. An evolutionary approach to constructing effective software reuse repositories. *ACM Transactions on Software Engineering and Methodology*, 6(2):111–140, April 1997.

[Hen97c] Scott Henninger. Tools supporting the creation and evolution of software development knowledge. In *Proceedings of the Twelfth IEEE International Conference on Automated Software Engineering*, pages 46–53, Incline Village, Nevada, USA, November 1997. IEEE Computer Society Press.

[HK97] Frank Houdek and Hubert Kempter. Quality patterns - an approach to packaging software engineering experience. *ACM SIGSOFT Software Engineering Notes*, 22(3):81–88, May 1997.

[HKV92] Robert Hendrick, David Kistler, and Jon Valett. Software Management Environment (SME): Concepts and architecture. Technical Report SEL-89-103, NASA Goddard Space Flight Center, Greenbelt, Maryland, USA, September 1992.

[HKV94] R. Hendrick, D. Kistler, and J. Valett. Software management environment (SME) components and algorithms. Technical Report SEL-94-001, SEL, February 1994.

[HLR95] S. Henninger, K. Lappala, and A. Raghavendran. An organizational learning approach to domain analysis. In *Proceedings of the Seventeenth International Conference on Software Engineering*, pages 95–103. ACM Press, 1995.

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

[HSW98] Frank Houdek, Kurt Schneider, and Eva Wieser. Establishing experience factories at Daimler-Benz: An experience report. In *Proceedings of the Twentieth International Conference on Software Engineering*, pages 443–447, Kyoto, Japan, April 1998. IEEE Computer Society Press.

[HW98] Frank Heister and Wolfgang Wilke. An architecture for maintaining case-based reasoning systems. In Barry Smyth and Pádraig Cunningham, editors, *Proceedings of the Fourth European Workshop on CBR*, pages 221–232, Dublin, Ireland, September 1998. Springer-Verlag, Berlin.

[IK96] Tomás Isakowitz and Robert J. Kauffman. Supporting search for reusable software objects. *IEEE Transactions on Software Engineering*, 22(6):407–423, June 1996.

[ISB95] Tomas Isakowitz, Edward A. Stohr, and P. Balasubramanian. RMM: A methodology for structured hypermedia design. *Communications of the ACM*, 38(8):34–44, August 1995.

[ISO99] ISO/IEC 9075:1999: Information technology – database languages – SQL – part 2: Foundation (SQL/foundation), 1999.

[Jar99] Matthias Jarke. Scenarios for modeling. *Communications of the ACM*, 42(1):47–48, January 1999.

[JDFM97] Lamia Labed Jilani, Jules Desharnais, Marc Frappier, and Rym Mili. Retrieving software components that minimize adaptation effort. In *Proceedings of the Twelfth IEEE International Conference on Automated Software Engineering*, pages 255–262, Incline Village, Nevada, USA, November 1997. IEEE Computer Society.

[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.

[KBM98] Oh Cheon Kwon, Cornelia Boldyreff, and Malcolm Munro. Software configuration management for a reusable software library within a software maintenance environment. *International Journal of Software Engineering & Knowledge Engineering*, 8(4):483–515, December 1998.

[Kim96] Won Kim. Object-relational: The unification of object and relational database technology. White-paper, UniSQL Inc., 1996.

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

[KKL+98] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, and Moonhang Huh. FORM: A feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering*, 5:143–168, 1998.

[KLG98] Gerd Kamp, Steffen Lange, and Christoph Globig. Related areas. In Mario Lenz, Brigitte Bartsch-Spörl, Hans-Dieter Burkhard, and Stefan Wess, editors, *Case-Based Reasoning Technology: From Foundations to Applications*, number 1400 in Lecture Notes in Artificial Intelligence, chapter 13, pages 327–351. Springer-Verlag, Berlin, Germany, 1998.

[KM93] A. Kemper and G. Moerkotte. Basic concepts of object-oriented database management systems (in German). *GI Informatik Spektrum*, 16(2):69–80, April 1993.

[KR97] Art Kleiner and George Roth. How to make experience your company's best teacher. *Harvard Business Review*, 75(5):172–177, September/October 1997.

[KS96] Hiroaki Kitano and Hideo Shimazu. The experience-sharing architecture: A case study in corporate-wide case-based software quality control. In David B. Leake, editor, *Case-Based Reasoning: Experiences, Lessons, and Future Directions*, pages 235–268. AAAI Press / The MIT Press, Menlo Park, California, USA, 1996.

[KV95] P. Katalagarianos and Y. Vassiliou. On the reuse of software: A case-based approach employing a repository. *Automated Software Engineering*, 2:55–86, 1995.

[Leh99] Peter Lehmann. Defined terminology as critical success factor (in German). *Computerwoche Focus*, (1):24–25, 23.4. 1999.

[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.

[Lio98] Yihwa Irene Liou. Expert system technology: Knowledge acquisition. In Jay Liebowitz, editor, *The Handbook of Applied Expert Systems*, pages 2–1 – 2–11. CRC Press, Boca Raton, USA, 1998.

[LM95] David M. Levy and Catherine C. Marshall. Going digital: A look at assumptions underlying digital libraries. *Communications of the ACM*, 38(4):77–84, April 1995.

[Lóp99] M. Fernández López. Overview of the methodologies for building ontologies. In V. R. Benjamins, B. Chandrasekaran, A. Gómez-Pérez, N. Guarino, and M. Uschold, editors, *Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods (KRR5)*, Stockholm, Sweden, August 1999.

[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.

[Luf99] Jens Lufter. Object-relational database systems (in German). *GI Informatik Spektrum*, 22(4):288–290, August 1999.

[LW98] David B. Leake and David C. Wilson. Categorizing case-base maintenance: Dimensions and directions. In Barry Smyth and Pádraig Cunningham, editors, *Advances in Case-Based Reasoning: Proceedings of the Fourth European Workshop on Case-Based Reasoning*, pages 196–207, Berlin,

Germany, September 1998. Springer-Verlag.

[Mat84a] Yoshihiro Matsumoto. Management of industrial software production. *IEEE Computer*, 17(2):59–72, February 1984.

[Mat84b] Yoshihiro Matsumoto. Some experience in promoting reusable software: Presentation in higher abstract levels. *IEEE Transactions on Software Engineering*, 10(5):502–513, 9 1984.

[MB94] Yoëlle Maarek and Daniel M. Berry. GURU: Information retrieval for reuse. In Patrick A. V. Hall and Lingzi Jin, editors, *Landmarks in Reuse and Reverse Engineering*, pages 55–82. Unicom Seminars Ltd., Uxbridge, Middlesex, UK, 1994.

[MB97] Yasuhiro Mashiko and Victor R. Basili. Using the GQM paradigm to investigate influential factors for software process improvement. *Journal of Systems and Software*, 36(1):17–31, January 1997.

[MBK91] Y. S. Maarek, D. M. Berry, and G. E. Kaiser. An information retrieval approach for automatically constructing software libraries. *IEEE Transactions on Software Engineering*, 17(8):800–813, August 1991.

[MBY97] John Mylopoulos, Alex Borgida, and Eric Yu. Representing software engineering knowledge. *Automated Software Engineering*, 4(3):291–317, June 1997.

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

[Men99] Tim Menzies. Knowledge maintenance heresies: Meta-knowledge complicates knowledge maintenance. In *Proceedings of the Eleventh Conference on Software Engineering and Knowledge Engineering*, pages 396–400, Kaiserslautern, Germany, June 1999. Knowledge Systems Institute, Skokie, Illinois, USA.

[MMM94] A. Mili, R. Mili, and R. Mittermeir. Storing and retrieving software components: A refinement based system. In *Proceedings of the Sixteenth International Conference on Software Engineering*, pages 91–100, Sorrento, Italy, May 1994. IEEE Computer Society Press.

[MR87] Leo Mark and Dieter Rombach. A meta information base for software engineering. Technical Report TR-1765, Department of Computer Science, University of Maryland, College Park, Maryland, USA, July 1987.

[Mül99] Michael Müller. Interestingness during the discovery of knowledge in databases (in German). *Künstliche Intelligenz*, pages 40–42, September 1999.

[NAT99] Markus Nick, Klaus-Dieter Althoff, and Carsten Tautz. Facilitating the practical evaluation of organizational memories using the goal-question-metric technique. In *Proceedings of the Twelfth Workshop on Knowledge Acquisition, Modeling and Management*, Banff, Alberta, Canada, October 1999.

[Nei84] James M. Neighbors. The Draco approach to constructing software from reusable components. *IEEE Transactions on Software Engineering*, SE-10(5):564–574, September 1984.

[Nel67] Ted H. Nelson. Getting it out of our system. In G. Schechter, editor, *Information Retrieval: A Critical Review*, pages 191–210. Thompson Books, Washington, D.C., USA, 1967.

[New82] Allen Newell. The knowledge level. *Artificial Intelligence*, 18:87–41, 1982.

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

[NN95] Jocelyne Nanard and Marc Nanard. Hypertext design environments and the hypertext design process. *Communications of the ACM*, 38(8):49–56, August 1995.

[Non91] Ikujiro Nonaka. The knowledge-creating company. *Harvard Business Review*, 69(6):96–104, November–December 1991.

[NT98a] Markus Nick and Carsten Tautz. Maintenance of experience packages: A collection of patterns with REFSENO and CBR-Works. Technical Report IESE-Report No. 048.99/E, Fraunhofer Institute

for Experimental Software Engineering, Kaiserslautern, Germany, 1998.

[NT98b] 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 *XPS'99: Knowledge-Based Systems - Survey and Future Directions*. Springer Verlag, Würzburg, Germany, March 1999. LNAI Nr. 1570.

[OB92] Markku Oivo and Victor R. Basili. Representing software engineering models: The TAME goal oriented approach. *IEEE Transactions on Software Engineering*, 18(10):886–898, October 1992.

[OHPDB92] Eduardo Ostertag, James Hendler, Rubén Prieto-Díaz, and Christine Braun. Computing similarity in a reuse library system: An AI-based approach. *ACM Transactions on Software Engineering and Methodology*, 1(3):205–228, July 1992.

[Oiv94] Markku Oivo. *Quantitative Management of Software Production Using Object-Oriented Models*. PhD thesis, Faculty of Science, University of Oulu, Finland, March 1994.

[O'L97] Daniel E. O'Leary. Impediments in the use of explicit ontologies for KBS development. *International Journal on Human-Computer Studies*, 46(2/3):327–337, February/March 1997.

[Ort91] Erich Ortner. Information management (in German). *GI Informatik Spektrum*, 14(6):315–327, December 1991.

[Ort99] Erich Ortner. Repository systems, part 1 (in German). *GI Informatik Spektrum*, 22(4):235–251, August 1999.

[Ost92] Eduardo Jenkins Ostertag. *A Classification System for Software Reuse*. Dissertation, University of Maryland, 1992.

[PA97] John Penix and Perry Alexander. Toward automated component adaptation. In *Proceedings of the Ninth Conference on Software Engineering and Knowledge Engineering*, pages 535–542, Madrid, Spain, June 1997.

[PD91] Rubén Prieto-Díaz. Implementing faceted classification for software reuse. *Communications of the ACM*, 34(5):89–97, May 1991.

[PDF87] Rubén Prieto-Díaz and Peter Freeman. Classifying software for reusability. *IEEE Software*, 4(1):6–16, January 1987.

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

[Pfl95] Shari Lawrence Pfleeger. Experimental design and analysis of software engineering. *Annals of Software Engineering*, 1(1):219–253, 1995.

[PH90] C. K. Prahalad and Gary Hamel. The core competence of the corporation. *Havard Business Review*, 68(3):79–91, May-June 1990.

[PRO00] PROFES Consortium. *The PROFES User Manual*. Fraunhofer IRB, Stuttgart, Germany, 2000.

[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.

[Rat97] Rational Software Corporation. *Unified Modeling Language*, 1997. Version 1.1.

[Rei91] Ulrich Reimer. *Introduction to Knowledge Representation: Net-like and Schema-Based Representation Formats (in German)*. Leitfäden der angewandten Informatik. Teubner, Stuttgart, Germany, 1991.

[Rei94] Thomas Reinartz. Case-based reasoning for reusing excerpts of a Smalltalk class library. Master's

thesis, Department of Computer Science, University of Kaiserslautern, 67653 Kaiserslautern, Germany, August 1994.

[Ric89] Michael M. Richter. *Principles of Artificial Intelligence (in German)*. B. G. Teubner, Stuttgart, Germany, 1989.

[Ric98] Michael M. Richter. Introduction. In Mario Lenz, Brigitte Bartsch-Spörl, Hans-Dieter Burkhard, and Stefan Wess, editors, *Case-Based Reasoning Technologies: From Foundations to Applications*, number 1400 in Lecture Notes in Artificial Intelligence, chapter 1, pages 1–15. Springer-Verlag, Berlin, Germany, 1998.

[RM88] Dieter Rombach and Leo Mark. Software process and product specifications: A bases for generating customized software engineering information bases. Technical Report CS-TR-2062, Department of Computer Science, University of Maryland, College Park, Maryland, USA, July 1988.

[Rom96] H. Dieter Rombach. New institute for applied software engineering research. *Software Process Newsletter*, pages 12–14, Fall 1996. No. 7.

[RY97] Kristi Racine and Qiang Yang. Maintaining unstructured case bases. In *Proceedings of the Second International Conference on Case-Based Reasoning*, pages 553–564, 1997.

[Sar95] Charisse Sary. Recall prototype lessons learned writing guide. Technical Report 504-SET-95/003, NASA Goddard Space Flight Center, Greenbelt, Maryland, USA, December 1995.

[SC99] B. Schatz and H. Chen. Digital libraries: Technological advances and social impacts. *IEEE Computer*, 32(2):45–50, February 1999.

[SCK+96] Mark Simos, Dick Creps, Carol Klingler, Larry Levine, and Dean Allemang. Organization Domain Modeling (ODM) guidebook, version 2.0. Informal Technical Report STARS-VC-A025/001/00, Lockheed Martin Tactical Defense Systems, Manassas, VA, USA, June 1996.

[Sen97] Arun Sen. The role of opportunism in the software design reuse process. *IEEE Transactions on Software Engineering*, 23(7):418–436, July 1997.

[She97] David J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. CRC Press, Boca Raton, USA, 1997.

[Sim91] M. Simos. *Domain Analysis and Software Systems Modeling*, chapter The Growing of an Organon: A Hybrid Knowledge-Based Technology and Methodology for Software Reuse, pages 204–221. IEEE Computer Society Press, 1991.

[SK95] Barry Smyth and Mark T. Keane. Remembering to forget: A competence-preserving case deletion policy for case-based reasoning systems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 377–382, 1995.

[Sku95] Douglas Skuce. Conventions for reaching agreement on shared ontologies. In *Proceedings of the Ninth Knowledge Acquisition for Knowledge Based Systems Workshop*, 1995.

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

[Sno99] Björn Snoek. Knowledge management and organizational learning: Systematic development of an experience base on approaches and technologies. Diploma thesis, Fraunhofer IESE, University of Kaiserslautern, Germany, September 1999.

[Spa99] Patricia Spallek. Knowledge management at work: The chemical information network of Arthur D. Little. In Frank Puppe, Dieter Fensel, Jana Köhler, Rudi Studer, and Thomas Wetter, editors, *Proceedings of the Fifth German Conference on Knowledge-Based Systems*, pages 57–60, Würzburg, Germany, March 1999. Würzburg University, Institute for Informatics.

[Ste90] Luc Steels. Components of expertise. *AI Magazine*, 11(2):28–49, 1990.

[Ste95] Andreas Steigner. Foundations of an information system for a software engineering group at a university (in German). Project thesis, Department of Computer Science, University of Kaiserslautern, 67653 Kaiserslautern, August 1995.

[Ste98] Sol Steinmetz et al., editor. *Random House Webster's Unabridged Dictionary*. Random House Inc., New York, USA, second edition, August 1998.

[Sti89] Jon Sticklen. Problem-solving architecture at the knowledge level. *Journal of Experimental and Theoretical Artificial Intelligence*, 1(4):233–247, 1989.

[Stu95] Jens Stummhöfer. A direct manipulatable user interface for maintaining and querying an experience base (in German). Master's thesis, Department of Computer Science, University of Kaiserslautern, 67653 Kaiserslautern, Germany, June 1995.

[Stu99] Rudi Studer. Knowledge engineering: Survey and future directions. In *Proceedings of the Workshop on Knowledge Management, Organizational Memory and Knowledge Reuse during Expert Systems '99 (XPS-99)*, number 2, pages 1–23, Würzburg, Germany, March 1999.

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

[SW99] Markus Stolpmann and Stefan Wess. *Optimization of Customer Relations with CBR Systems (in German)*. Addison-Wesley, Bonn, Germany, 1999.

[SWdH+94] Guus Schreiber, Bob Wielinga, Robert de Hoog, Hans Akkermans, and Walter Van de Velde. CommonKADS: A comprehensive methodology for KBS development. *IEEE Expert*, 6(6):28–37, December 1994.

[SWT89] J. Solderitsch, K. Wallnau, and J. Thalhamer. Constructing domain-specific Ada reuse libraries. In *Proceedings of the Seventh Annual National Conference on Ada Technology*, Ft. Monmouth, NJ, March 1989. U.S. Army Communications-Electronics Command.

[TA97] Carsten Tautz and Klaus-Dieter Althoff. Using case-based reasoning for reusing software knowledge. In D. Leake and E. Plaza, editors, *Proceedings of the Second International Conference on Case-Based Reasoning*. Springer Verlag, 1997.

[TA98] Carsten Tautz and Klaus-Dieter Althoff. Operationalizing comprehensive software knowledge reuse based on CBR methods. In Lothar Gierl and Mario Lenz, editors, *Proceedings of the Sixth German Workshop on Case-Based Reasoning*, volume 7 of *IMIB Series*, pages 89–98, Berlin, Germany, March 1998. Institut für Medizinische Informatik und Biometrik, Universität Rostock.

[TA00a] Carsten Tautz and Klaus-Dieter Althoff. A case study on engineering ontologies and related processes for sharing software engineering experience. In *Proceedings of the Twelfth Conference on Software Engineering and Knowledge Engineering*, Chicago, Illinois, USA, July 2000. Knowledge Systems Institute, Skokie, Illinois, USA.

[TA00b] Carsten Tautz and Klaus-Dieter Althoff. Towards engineering similarity functions for software engineering experience. In Mehmet Göker, editor, *Proceedings of the Eighth German Workshop on Case-Based Reasoning*, pages 87–97, Lämmerbuckel, Germany, March 2000. DaimlerChrysler Research and Technology.

[TANO00] Carsten Tautz, Klaus-Dieter Althoff, Markus Nick, and Michael Ochs. An empirical investigation of the effectiveness and efficiency of utilizing repositories for qualitative experience. Technical Report IESE-Report No. 016.00/E, Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany, 2000.

[Tau93] Carsten Tautz. Design and implementation of a tool for comprehensive reuse of software processes (in German). Master's thesis, Department of Computer Science, University of Kaiserslautern, 67653 Kaiserslautern, Germany, July 1993.

[TBF95] Carsten Tautz, Christian Bunse, and Oliver Flege. Hyper system checks for world-wide web servers. Technical Report STTI-95-07-E, Software Technologie Transfer Initiative Kaiserslautern, Fachbereich Informatik, Universität Kaiserslautern, D-67653 Kaiserslautern, 1995.

[tec00] CBR-Works. URL http://www.tecinno.de/english/products/cbrw_main.htm, 2000. tec:inno GmbH, Germany.

[TG98a] 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.

[TG98b] Carsten Tautz and Christiane Gresse von Wangenheim. A representation formalism for supporting reuse of software engineering knowledge. Technical Report IESE-Report No. 051.98/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.

[THH95] Manfred Thüring, Jörg Hannemann, and Jörg M. Haake. Hypermedia and cognition: Designing for comprehension. *Communications of the ACM*, 38(8):57–66, August 1995.

[Tic85] Walter Tichy. RCS–a system for version control. *Software–Practice and Experience*, 15(7):637–654, July 1985.

[Tra94] Will Tracz. Software reuse myths revisited. In *Proceedings of the Sixteenth International Conference on Software Engineering*, pages 271–272. IEEE Computer Society Press, May 1994.

[Tra95] Will Tracz. *Confessions of a Used Program Salesman: Institutionalizing Software Reuse*. Addison-Wesley, Reading, MA, USA, 1995.

[UG96] Mike Uschold and Michael Gruninger. Ontologies: Principles, methods, and applications. *The Knowledge Engineering Review*, 11(2):93–136, 1996.

[vDD+88] Axel van Lamsweerde, Bruno Delcourt, Emmanuelle Delor, Marie-Claire Schayes, and Robert Champagne. Generic life cycle support in the ALMA environment. *IEEE Transactions on Software Engineering*, 14(6):720–741, June 1988.

[Vis94] Giuseppe Visaggio. Process improvement through data reuse. *IEEE Software*, 11(4):76–85, July 1994.

[vSB99] Rini van Solingen and Egon Berghout. *The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development*. McGraw-Hill, London, England, 1999.

[vvK96] Gertjan van Heijst, Rob van der Speck, and Eelco Kruizinga. Organizing corporate memories. In *Proceedings of the Tenth Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, 1996. URL http://ksi.cpsc.ucalgary.ca/KAW/KAW96/vanheijst/HTMLDOC.html.

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

[WD99] Steven Wartik and Ted Davis. A phased reuse adoption model. *Journal of Systems and Software*, 46(1):13–23, April 1999.

[Web95] Merriam Webster. *Merriam Webster's Collegiate Dictionary*. Merriam Webster Inc., Springfield, USA, 1995.

[WES87] Scott N. Woodfield, David W. Embley, and Del T. Scott. Can programmers reuse software? *IEEE Software*, 4(7):52–59, July 1987.

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

[Wii95] Karl M. Wiig. *Knowledge Management Methods: Practical Approaches to Managing Knowledge*. Schema Press, Arlington, Texas, USA, 1995.

[WW99] Stephan Weibelzahl and Gerhard Weber. User modeling of customer demands through Case-Based Reasoning (in German). In Stefan Wrobel et al., editor, *Proceedings of the Workshop Days on Learning, Knowledge Discovery, and Adaptivity (LWA '99)*, pages 295–300, Otto-von-Guericke-Universität Magdeburg, Germany, September 1999.

[Ygl98] Kathryn Priest Yglesias. IBM's reuse programs: Knowledge management and software reuse. In *Proceedings of the Fifth International Conference on Software Reuse*, pages 156–165, June 1998.

Index

Lebenslauf

Name	Carsten Tautz
Geburtsdatum	25. August 1966
Geburtsort	Celle
Adresse	Buchenlochstraße 66, 67663 Kaiserslautern
Familienstand	ledig
Staatsangehörigkeit	deutsch

Schulbildung

1973–1977	Grundschule, Celle and Stadtallendorf
1977–1979	Gymnasium, Amöneburg
1980–1983	High School, Yuma, Arizona, USA
1983–1986	Gymnasium, Idar-Oberstein (Abschluß: Allgemeine Hochschulreife)

Berufsausbildung und Studium

1986–1987	Wehrdienst
1987–1993	Informatik-Studium mit Nebenfach Wirtschaftswissenschaften an der Universität Kaiserslautern (Abschluß: Diplom)

Berufliche Tätigkeiten

1993–1995	Wissenschaftlicher Mitarbeiter bei der Software-Technologie-Transfer-Initiative Kaiserslautern (STTI-KL)
1996–dato	Wissenschaftlicher Mitarbeiter am Fraunhofer-Institut für Experimentelles Software Engineering (Fraunhofer IESE), Kaiserslautern

Kaiserslautern, 30. März 2000