# 1. Methodology for Building CBR Applications

**Ralph Bergmann and Klaus-Dieter Althoff**

## 1.1 Introduction

As the previous chapters of this book have shown, case-based reasoning is a technology that has been successfully applied to a large range of different tasks. Through all the different CBR projects, both basic research projects as well as industrial development projects, lots of knowledge and experience about how to build a CBR application has been collected. Today, there is already an increasing number of successful companies developing industrial CBR applications. In former days, these companies could develop their early pioneering CBR applications in an ad-hoc manner. The highly-skilled CBR expert of the company was able to manage these projects and to provide the developers with the required expertise.

Today, the situation has changed. The market for CBR has started to increase significantly. Therefore, these companies have to face the fact that the market demands companies executing more and larger CBR projects than in these early days. It is required that they develop software that fulfills current quality standards. Consequently, contemporary IT companies can no longer sustain inefficient or ineffectual CBR application development. What is required is a *methodology for building CBR applications*. Such a methodology should make CBR application development a systematic engineering activity rather than an art known by a few experts (Shaw, 1990; Gibbs, 1994). A methodology usually combines a number of *methods* into a philosophy which addresses a number of phases of the software development life-cycle (e.g. (Booch, 1994), chapter 1). It should give guidelines (recipes) about the activities that need to be performed in order to successfully develop a certain kind of product, that is, in our case, a CBR application.

The use of an appropriate methodology should provide significant quantifiable benefits in terms of

- *productivity*, e.g., reduce the risk of wasted efforts,
- *quality*, e.g., inclusion of quality deliverables,
- *communication*, a reference for both formal and informal communication between members of the development team, and
- will provide a solid base for *management decision making*, e.g., planning, resource allocation, and monitoring.

Currently, there are several activities with the goal of establishing a methodology for building case-based reasoning applications. Contributions can be found in books on CBR (Kolodner, 1993; Wess, 1995) and in papers collecting the experience of people who have successfully developed CBR applications (Kitano and Shimazu, 1996; Lewis, 1995; Bartsch-Spörl, 1996; Curet and Jackson, 1996); most valuable experience-based contributions arose from projects where methodology development was explicitly included as a project task, like INRECA[1] (Althoff et al., 1995b; Johnston et al., 1996) or APPLICUS[2] (Bartsch-Spörl, 1996; Bartsch-Spörl, 1997).

In this chapter we present a new methodology for building case-based reasoning systems which is based on the Esprit projects INRECA and INRECA-II [3], particularly on the experience gained by all of the project partners. While the main objective of the INRECA project was the development of the core CBR technology with methodology development being a minor issue, the major focus of the INRECA-II project is the development of a methodology for building and maintaining CBR applications in the area of diagnosis and decision support (Bergmann et al., 1997a; Bergmann et al., 1998).

The approach presented in this chapter covers the two major aspects that are important for CBR development to become an engineering activity. First, it presents an analytic framework for describing and classifying CBR systems and applications. This is necessary to structure the large spectrum of different systems that have already been developed. Part II of this book gives a good impression of the variety of CBR applications ranging from analytic tasks such as classification and diagnosis to synthetic tasks like design and planning. Any application developer must first analyze a new application field to decide which type of CBR approach is most appropriate. The analytic framework described in section 1.2 supports this analysis task.

Second, the application developer must determine the specific development steps she/he has to follow in order to come to a CBR system of the desired kind. These development steps also depend very much on the particularities of the current client, like the existing organizational structure, existing IT environment, etc. Section 1.3 describes an experience-based approach to systematically develop a process model of how a particular CBR application should be built for a certain client. Finally, section 1.4 concludes by stating future directions for about how the presented CBR methodology for building and maintaining CBR applications should evolve.

[1] INRECA: Esprit Project P6322. Partners: Acknosoft (prime contractor, France), TECINNO (Germany), Interactive Multimedia Systems (Ireland), University of Kaiserslautern (Germany)

[2] APPLICUS: Esprit Trial Application P20824. Partners: Acknosoft (prime contractor, France), BSR Consulting (Germany), Sepro Robotique (France)

[3] INRECA-II Esprit Project P22196. Information & Knowledge Reengineering for Reasoning from Cases. Partners: Acknosoft (prime contractor, France), Daimler Benz AG (Germany), TECINNO (Germany), Interactive Multimedia Systems (Ireland), University of Kaiserslautern (Germany)

## 1.2 Analytic Framework for Developing CBR Systems

[4]

A particular strength of CBR over most other methods is its inherent combination of problem solving with sustained learning through problem solving experience. This is therefore a particularly important topic of study, and an issue that has now become mature enough to be addressed in a more systematic way. To enable such an analysis of problem solving and learning, in the following we describe first steps towards the development of an analytic framework for studying CBR methods. It provides an explicit ontology of basic CBR task types, domain characterizations, and types of problem solving and learning methods. Further, it incorporates within this framework a methodology for combining a knowledge- level, top-down analysis with a bottom-up, case-driven one. In this section, we present the underlying view and the basic approach being taken, the main components of the framework and accompanying methodology as well as some applications of the framework (one application has been described in section **??**.7). Examples of studies recently done and how they relate to the framework have been described in (Althoff and Aamodt, 1996). A detailed description can be found in (Althoff, 1996).

### 1.2.1 Introduction

Over the last few years substantial progress has been made within the field of CBR. The problems we are facing have become more clearly identified, research results have led to improved methods for case retrieval as well as improved approaches to the harder problems of adaptation and learning (see, e.g., the collection of papers in ICCBR-95: (Veloso and Aamodt, 1995)). In the course of this development it has also become clear that a particular strength of CBR over most other methods is its inherent combination of problem solving with sustained learning through problem solving experience. This is therefore a particularly important topic of study, and an issue that has now become mature enough to be addressed in a more systematic way. To enable such an analysis of problem solving and learning, a unified framework for describing, analyzing, and comparing various types and aspects of CBR methods is needed.

Integration of learning and problem solving may in general start out from different goals, and be viewed from different perspectives. One example is "concept formation" as a goal, and the formation and utilization of operationalization criteria related to the problem solving task, as the perspective. Another example is "improved performance" as a goal, and the improvement of total problem solving speed - for computer and human together - as the perspective. A third example is "sustained learning", i.e., continuous learning

[4] Parts of this section are adapted from (Althoff and Aamodt, 1996).

through problem solving experience, as a goal, and the impact of the application problem task on the learning method as a perspective. Many more examples may be given, and for each of them a particular area of overlap, an "intersection space" between machine learning (ML) and problem solving (PS) methods can be identified. Within this space, dependencies and other relations between specific ML and PS methods may be described and analyzed in a systematic way, provided we have a suitable means to structure the space.

In the following we describe first steps towards the development of a framework and a methodology which defines and makes use of such a structure. Since we are studying CBR methods, the natural focus is on the third of the above goals: Sustained and (continuous) learning from each problem solving experience as a natural part of a problem solver's behavior. Within a broader perspective of integrated learning and problem solving, it is also natural to start a study of learning as close as possible to the source of learning, namely a concrete experience. Our work is related to some earlier suggestions for analytic CBR frameworks, such as the similarity-focused framework by Richter and Wess (Richter and Wess, 1991), Althoff's analysis of systems for technical diagnosis (Althoff, 1992), Aamodt's comparison of knowledge-intensive CBR methods (Aamodt, 1991), Armengol's and Plaza's analysis of CBR system architectures (Armengol and Plaza, 1993), and the framework for systems comparison and evaluation as described in (Auriol et al., 1994). Our framework extends these previous suggestions in several respects. It provides an explicit ontology of basic CBR task types, domain characterizations, and types of problem solving and learning methods. Further, it incorporates within this framework a method- ology for combining a knowledge-level, top-down analysis with a bottom-up, case-driven one. In this section, we present the underlying view and the basic approach being taken, the main components of the framework and accompanying methodology, and refer to some applications of the framework carried out so far.

### 1.2.2 Basic Approach

**Knowledge-Level Analysis.** A potentially useful way to describe problem solving and learning behavior is in terms of the *goals* to be achieved, the *tasks* that need to be solved, the *methods* that will accomplish those tasks, and the *knowledge of the application domain* that those methods need. A description of a system along these lines is often referred to as a knowledge level description, and more recent research in knowledge acquisition (Steels, 1990; Wielinga et al., 1993) has clearly demonstrated the usability of this approach. The original knowledge-level idea has undergone some modifications over the years, from Newell's highly intentional, purpose-oriented way of describing a system (Newell, 1982), to a more structured and usable type of description. An incorporation of the CBR perspective into knowledge-level modeling is discussed in (Aamodt, 1995).

Adopting a knowledge-level perspective to the analysis of integrated PS-ML systems enables the description of methods and systems both from a general (intentional) and case-specific (extension) perspective. A general description relates a method to descriptive terms and relationships within a general model of descriptive concepts - i.e., an ontology of task types, domain characteristics, and method types. Through a case-driven description, methods can be understood by relating them to already known methods within already described/implemented systems (e.g., CBR is combined with rule- and model-based reasoning in the same way as in Creek (Aamodt, 1993); a decision tree is generated as in Inreca (Manago et al., ); the similarity measure is adapted as in Patdex (Wess, 1993); partial determination rules are generated and used like the so- called "shortcut rules" in Moltke (Althoff, 1992); etc.). After having developed/described a certain number of systems, we will be able to select/instantiate a system description at the knowledge level by a combined use of general and case-specific descriptors. What we are aiming at is an effective combination of top- down and bottom-up analysis and modeling methods, based on an integration of these two perspectives.

From an engineering point of view, this will enable a particular symbol-level architecture to be chosen, and/or a chosen architecture to be instantiated, based on a thorough understanding of the real world application task and its domain characteristics. However, a knowledge-level description in itself will not provide a language detailed enough to describe or analyze system designs, or to arrive at a symbol-level architecture specification. Our approach therefore incorporates a focusing perspective and an analytic "tool" to help in the more detailed description that guides the architectural specification based on a knowledge-level model.

**Similarity as a Focusing Mechanism.** The focus provided by this mechanism leads to a view of - in principle - all CBR methods as operations related to *similarity*, in one sense or another. That is, problem solving can be described as a process of initial assessment of similarity (case retrieval) followed by a more deliberate assessment of similarity (case adaptation), and learning (case extraction, case indexing, and possibly updates of general knowledge) can be described by relating it to later similarity assessment - i.e., to a pragmatic learning goal. Along with (Richter and Wess, 1991) we view similarity as an *a posteriori* criterion, and any attempt to assess similarity before a retrieved case has been found useful will only result in a hypothesized similarity assessment. Our retrieval methods should of course try to minimize the difference between the hypothesized similarity measure and the actual similarity determined after the attempt has been made to use the case. The general domain knowledge can then be seen as a means to reduce this uncertainty of the initial similarity assessment with respect to the final similarity assessment made after having evaluated the success of the (possibly modified) case in finding a solution to the input problem.

**Tasks and Domain Characterizations.** The types of application domains we address cover a wide spectrum, ranging from strong-theory domains with rather well-defined relationships between domain concepts (e.g., diagnosis of purely technical systems), to weak-theory and open domains with uncertain domain relationships (e.g., medical diagnosis). This is an important feature of the framework, since we particularly want to relate characteristics of the task (the what-to-do) and the knowledge on which per- formance of the task is based, to the methods (how-to-do's) that enable the problem solver to accomplish the task by use of the knowledge. The starting point is always the real world setting in which the system is to operate. Medical diagnosis, for example, in a real world setting, is far away from a pure classification task. If a system shall cover the major tasks involved in practical diagnosis, it will have to include planning tasks (e.g., setting up and continuously revising an examination protocol), as well as prediction tasks (assessing the consequences of a treatment). The next section outlines the core components of the framework, with a focus on the knowledge-level description and analysis and the combined top-down and bottom-up oriented methodology. The incorporation of the similarity assessment mechanism is part of ongoing research.

### 1.2.3 Framework and Methodological Issues

**Basic Framework Components.** At the highest level of generality, a general CBR cycle may be described by four tasks (Aamodt and Plaza, 1994): Retrieve the most similar case or cases, reuse the information and knowledge in that case to solve the problem, revise the proposed solution, and retain the parts of this experience likely to be useful for future problem solving. See chapter **??**, particularly fig. **??**.2. Note that these tasks are internal reasoning tasks, and different from the application problems tasks (diagnosis, planning, etc.) referred to earlier. Each of the the four CBR tasks involves a number of more specific sub-tasks. An initial description of a problem (top of the CBR cycle) defines a new case. In the retrieve task this new case is used to find a matching case from the collection of previous cases. The retrieved case is combined with the input case - in the reuse task - into a solved case, i.e., a proposed solution to the initial problem. The revise task tests this solution for success, e.g. by applying it to the real-life environment or have it evaluated by a teacher, and repaired if failed. This task is important for learning, since the system needs a feedback of how successful its proposed solution actually was. Retain is the main learning task, where useful experience is retained for future reuse, by updating the case base and possibly also the general domain knowledge. As indicated in the figure, general knowledge usually plays a role in this cycle, by supporting the CBR processes. This support may range from very weak to very strong, depending on the type of CBR method. By general knowledge we mean general domain knowledge, as opposed to the specific domain knowledge embodied by cases. For example, in diagnosing a patient by retrieving and reusing the case of a previous patient, a model of anatomy together with causal relationships between pathological and other physiological states may constitute the general knowledge used by a CBR system. A set of rules may play the same role.

Knowledge-level analysis, as previously described, is a general approach to systems analysis. It is therefore applicable to the analysis of application tasks and domains - as manifested in the knowledge acquisition methodologies referred to earlier - as well as internal reasoning tasks of a problem solver and learner. In our framework we therefore take a "task - method - domain knowledge" approach both to the analysis of real-world application tasks, and to the analysis of the CBR reasoning tasks themselves. The mapping between the two is as follows: a method from the application analysis (how to solve a technical diagnosis problem, or how to determine the next test to be done) either decomposes an application task into subtasks, or it solves the task directly. In both cases these *methods* set up *tasks* at the reasoning level (problem solving and learning from experience). In the following, we concentrate on the reasoning tasks.

The tasks from the CBR cycle are further decomposed in figure 1.1 [5]. The tasks are printed in bold letters, while methods are written in plain text. The links between task nodes (bold lines) are task decompositions, i.e., part-of relations. The links between tasks and methods (stippled lines) identify alternative methods applicable for solving a task. The top-level task is problem solving and learning from experience and the method to accomplish the task is a case- based reasoning method. This splits the top-level task into the four major CBR tasks of the CBR cycle. Each of the four tasks is necessary in order to perform the top-level task. The retrieve task is, in turn, partitioned in the same manner (by a retrieval method) into the tasks identify features, search (to find a set of past cases), initially match (the relevant descriptors to past cases), and select (the most similar case). All task partitions in the figure are considered complete, i.e., the set of subtasks of a task are intended to be sufficient to accomplish the task, at this level of description. The figure does not show any control structure over the subtasks. The actual control is specified as part of the problem-solving method. The actual retrieval method, for example (not explicitly indicated in the figure), specifies the sequence and loop-backs for the subtasks of retrieve. A method specifies the algorithm that identifies and controls the execution of subtasks, or solves the task directly, while accessing and utilizing the domain knowledge needed to do this. The methods shown in the figure are high level method classes, from which one or more specific methods should be chosen. The method set as shown is incomplete, i.e., one of the methods indicated may be sufficient to solve the task, several methods may be combined, or there may be other methods that have not been mentioned.

[5] In chapter ?? the same CBR task-method decomposition has been used for the purpose of detailing selected models of software knowledge reuse.

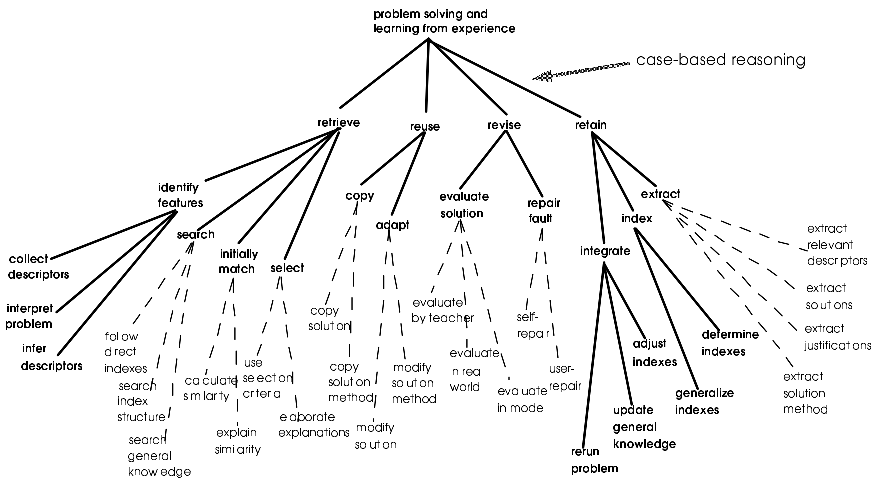


**Fig. 1.1.** Task-method decomposition of CBR

The above structure provides the basis for the analytic framework. It needs to be elaborated and described in more detail, characterizations of domain knowledge types need to be added, and dependencies between the various knowledge types need to be identified.

**Methodology.** As previously stated, the basic methodological approach is to combine a top-down oriented analysis of application tasks, domain knowledge descriptions, and methods with a bottom-up, case-driven method of studying existing systems. The aim is to arrive at a coherent framework and description language that specializes from the high-level analysis and generalizes from the example systems studied. The baseline of the approach is as follows. We *describe* CBR systems as well as domains and application tasks using two different kinds of criteria, namely criteria characterizing the domain and task at hand (domain/task criteria) and criteria describing the abilities and limitations of existing systems and system components (technical/ergonomic criteria). Examples for domain/task criteria are *size* [6], *theory strength* [7], *openness* [8], *change over time* [9], and *complexity* [10] , while *case and knowledge representation, similarity assessment, validation, and data acquisition and maintenance* exemplify important technical/ergonomic criteria. We

[6] The size of a domain is characterized by the amount of different items representing the explicit knowledge.

[7] The theory strength of a domain depends on the degree of certainty of the involved relationships.

[8] The openness of a domain depends on its environmental dependencies.

[9] The change over time of a domain means that a domain is called static if there are no expected changes and it is called dynamic if it is clear that changes will appear in continuation.

[10] The complexity of a domain means the amount of different taxonomic relations.

analyze the underlying methods and domain/task characteristics by relating domain/task criteria with technical/ergonomic criteria. Figure 1.2 gives an example of how CBR systems can be labeled with domain criteria. Combining such a description with a more general kind of analysis based on the subtask structure of figure 1.1 was shown to be useful for evaluating the final Inreca system (Althoff, 1996).

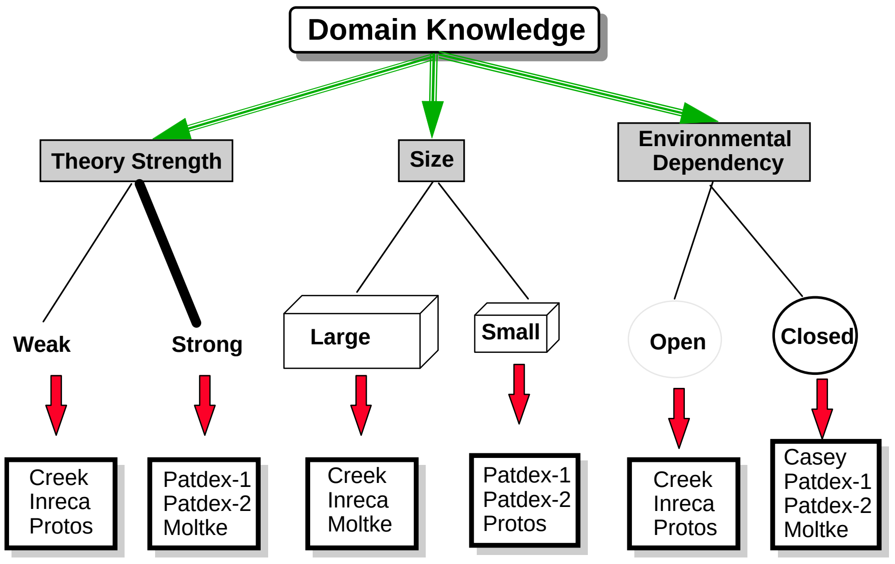


**Fig. 1.2.** An example of domain criteria related to existing systems(Casey: (Koton, 1989); Creek: (Aamodt, 1993); Inreca: (Althoff et al., 1998), see also chapter **??**; Moltke: (Althoff, 1992; Althoff et al., 1990); Patdex-1: (Stadler and Wess, 1989; Richter and Wess, 1991); Patdex-2: (Wess, 1990; Althoff and Wess, 1991); Protos: (Bareiss, 1989))

From a software or knowledge engineering perspective, we try to arrive at non-functional system properties as a systematic means for (CBR) system development. Since we focus on CBR systems we can define more precise criteria than we could for software systems in general. Additionally, we combine these system-oriented, more technical criteria with an analysis of application domains in which we have experience. On the one hand, feedback from applications can be systematically transformed in an evaluation based on domain and application task criteria. On the other hand, methods extracted from CBR-related systems and tools can be labeled with the results of the application of such criteria. Again, the aim is to close the gap between high level characterizations, on the one side, and concrete systems on the other side. General knowledge level analysis and case-driven analysis are merged in order to come up with application frameworks for particular types of systems, based on a common terminology and a unified model. Here technical/ergonomic criteria are combined with domain/task criteria. The intended use of this framework both for analysis and development of integrated problem solving and learning systems, can be described as providing answers to the following questions related to the evaluation of AI research (Cohen, 1989):

- How is the CBR method to be evaluated an improvement, an alternative, or a complement to other methods? Does it account for more situations, or produce a wider variety of desired behavior, or is it more efficient in time or space, or does it model human behavior in a more useful way/detail?
- What are the underlying architectural assumptions? Are all design decisions justified? Does the method rely on other integrated methods? Does it subsume other methods?
- What is the scope of the method? How extendible is it? Will it scale up?
- Why does the method work? Under which conditions would it not work?
- What is the relationship between the class of task, of which the current task is an example, and the method used? Can this relationship be stated clearly enough to support a claim that the method is general to the class of task?

### 1.2.4 Applications of the Framework

Up to now there are a number of applications of the introduced framework. The original application (within the Inreca Esprit project) was a comparison of commercially available CBR tools (Althoff et al., 1995a). The underlying goals for the Inreca project were achieving deeper insights for the development of the final Inreca system, finding a reasonable and realistic combination of research and application issues, and getting a more concrete estimation of what can be expected from CBR technology in the near and the far future. Based on this comparison the framework was extended and used for the evaluation of the final Inreca system (Althoff, 1996) consisting of a comparison with commercial tools, a comparison with a number of CBR-related research systems as well as an in-depth analysis of the system using the introduced framework. The application of the framework to the validation of CBR systems has been described in (Althoff and Wilke, 1997). Parts of the framework have been used for gathering information about existing CBR systems by means of two structured questionnaires (Meissonnier, 1996) that have been collected from CBR system developers from 14 countries (see also section **??**.7 for more details). An analysis of the collected information has been described in (Bartsch-Spörl et al., 1997). Using the described framework enabled us to formally describe the collected questionnaires as cases and to build a CBR system for accessing the included information via similarity based retrieval (Meissonnier, 1996). This will be made available via WWW (see section **??**.7 for further details including the URL).

## 1.3 Building CBR Applications for Analytic Tasks

When building a CBR application that should be used in the daily practice within an existing client organization a large variety of different kinds of processes have to be considered:

- the process of *project management* (cost and resource assessment, time schedules, project plans),
- the *analysis and re-organization of the environment* (e.g., a department) in which the CBR system should be introduced, and of course,
- the process of *technical product development* and *maintenance*: particularly tool selection, customer-specific software development and domain modeling.

All these processes must be defined and tailored according to the needs and circumstances of the current client. This is an activity that requires a lot of practical experience. Although this experience is available in the minds of experienced application developers, it is usually not collected and stored systematically. This creates serious problems, e.g., in case of staff departures or when the companies grow and new staff must be trained. The approach presented here addresses this problem in the line of the experience-based construction of software. It combines two recent approaches from software engineering: the *experience factory* ((Basili et al., 1994); see also section **??**.3) and *software process modeling* ((Rombach and Verlage, 1995)) approach. The approach to a CBR development methodology is itself very "CBR-like". In a nutshell, it captures previous experience from CBR development and stores it in a so-called experience base (a term from the experience factory approach). The entities being stored in the experience base are software process models, or fragments of it, such as processes (managerial, organizational, and technical), products being produced or consumed by the different processes, and methods that can be executed to realize a process. Although this approach is applicable to any kind of CBR development experience, the CBR applications covered so far are analytic tasks only. The current experience base was built up by analyzing several successful industrial applications developed by or in cooperation with the INRECA-II consortium partners (Acknosoft, Daimler-Benz, Interactive Multimedia Systems, TECINNO, and University of Kaiserslautern). Since at the date of printing this book, this project is still on-going, further refinements of this approach are very likely and further experience will be entered into the experience base.

### 1.3.1 Experience Factory

We now briefly introduce the experience factory idea ((Basili et al., 1994); see also section **??**.3). This approach is motivated by the observation that any successful business requires a combination of technical and managerial solutions which includes a well-defined set of product needs to satisfy the customer, assist the developer in accomplishing those needs and create competencies for future business; a well-defined set of processes to accomplish what needs to be accomplished, to control development, and to improve overall business; a closed-loop process that supports learning and feedback. The key technologies for supporting these requirements include: modeling, measurement, the reuse of processes, products, and other forms of knowledge relevant to the (software) business. An 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 (see Figure 1.3). An experience factory packages experience by building informal, formal, or schematized models and measures of various software processes, products, and other forms of knowledge via people, documents, and automated support. The main product of an experience factory is an *experience base*. The content and the structure of an experience base varies based upon the kind of experience clustered in the base. See chapter **??** of this book for a more detailed discussion of the experience factory approach, the associated Quality Improvement Paradigm, related work from software engineering as well as relationships to CBR.

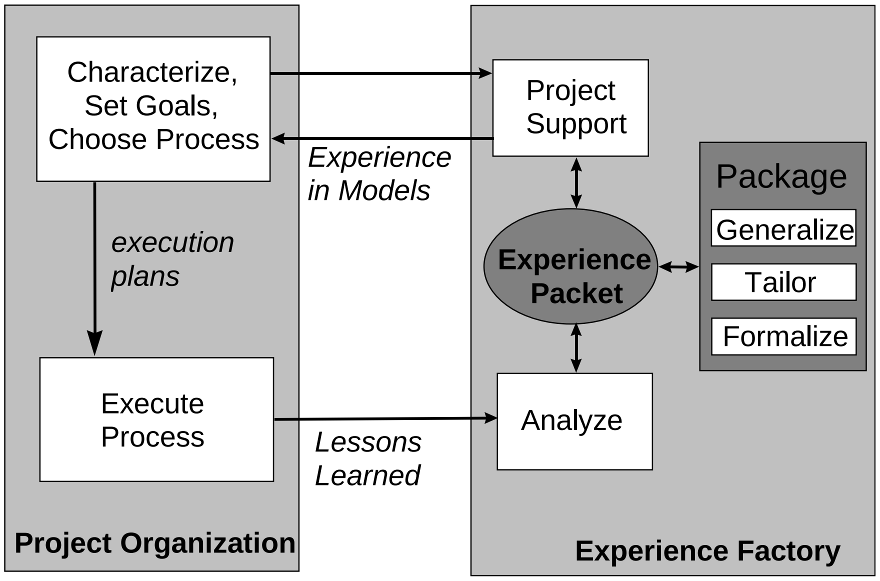


**Fig. 1.3.** The Experience Factory Approach (Basili et al., 1994)

### 1.3.2 Software Process Modeling

*Software process modeling* is an approach that is highly important in the context of the experience factory approach. Software process models describe the engineering of a product, e.g., the software that has to be produced. Unlike early approaches in software engineering, the software development is not considered to follow a single fixed process model with a closed set of predefined steps. A tailored process model particularly suited for the current project must be developed in advance. Software process models include *technical* software engineering *processes* (like requirements engineering, design of the system to be built, coding, etc.), *managerial* software engineering *processes* (like management of product related documentation, project management, quality assurance, etc.), and *organizational processes* (covering those parts of the business process in which the software system will be embedded and that need to be changed in order to make best use of the new software system). From time to time, such a model must be refined or changed during the execution of the project if the real world software development process and the model do not match any longer. Several formalisms and languages have been already developed for representing process models. One such language currently being developed at the University of Kaiserslautern is called MILOS (Dellen et al., 1997a). In MILOS, a process model is defined in terms of *processes, methods, products, goals,* and *resources* (see Fig. 1.4). Within our approach to software process modeling, we adopt the terminology and the concepts from this language to formalize process models for developing CBR applications. Other representation languages often contain similar concepts, but use different terms.

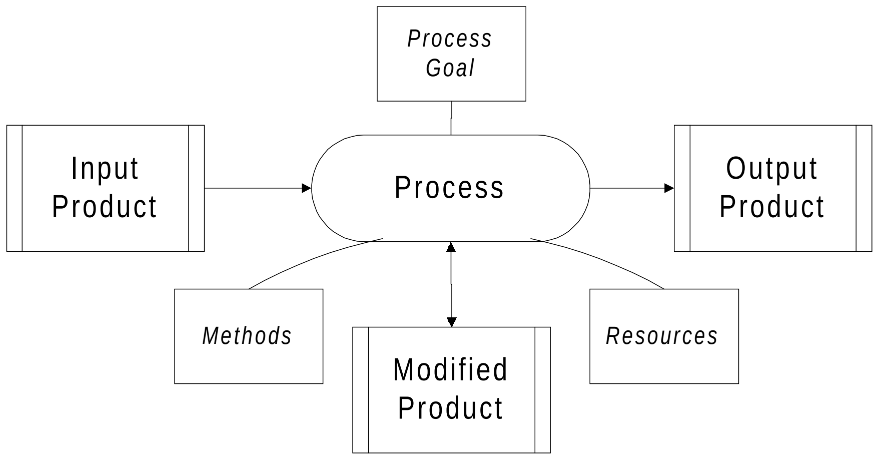


**Fig. 1.4.** Graphical notation for representing process models

**Generic Processes.** A process is a basic step that has to be carried out in a software development project. Each process is defined as an instance of a certain *generic process.* A generic process describes a class of processes by defining the following properties:

- A particular *goal* of such a step specifies *what* has to be achieved.
- A set of different alternative *methods* that can be used to implement the step. Such a method specifies one particular way of carrying out the process, i.e., one way of how to reach the goal of the process.
- *Input, output,* and *modified products* that describe which products are required at the beginning, which products must be delivered at the end of the step, and which products are changed during enactment.
- A set of *resources* (agents or tools) that are required to perform the step. Here the necessary qualifications or specifications are defined that an agent or a tool must have so that he can be assigned to the process.

**Methods.** Methods contain a detailed specification of a particular way of reaching the goal of a process. A method can be either *simple* or *complex.* While a simple method provides only a description of what to do to reach the goal of the associated process, a complex method specifies a set of subprocesses, a set of intermediate products (called *by-products*), and the flow of products between the subprocesses. This allows the definition of very flexible process models in a hierarchical manner, since very different process refinements can be described by utilizing alternative subprocess models.

**Generic Product.** The main goal of processes is to create or modify products. Products include the executable software system as well as the documentation like design documents or user manuals. Products are modeled by *generic product descriptions*, which declare certain properties that all products of a certain kind must have. For example, a generic product can be a domain definition in the CASUEL case representation language. Additionally, a product can be decomposed into several subproducts, e.g. the definition of attribute types, object classes, cases, and similarity measures.

**Resources.** *Resources* are entities necessary to perform the tasks. Resources can be either *agents* or *tools.* Agents are models for humans or teams (e.g. managers, domain experts, designers, or programmers), which can be designated to perform a processes. The most relevant properties of agents are their qualifications. Tools (e.g., a modeling tool, a CBR tool, or a GUI builder) are used to support the enactment of a process and can be described by a specification. Therefore, by using the required qualifications and specifications defined in the generic process, it is possible to determine available agents and tools which can be assigned to a certain process.

### 1.3.3 The Experience Base for Developing Analytic CBR Applications

In our CBR methodology, software process models are used to represent the CBR development experience that is stored in the experience base. Software processes that must be represented can be either very abstract, i.e., they can just represent some very coarse development steps such as: domain model definition, similarity measure definition, case acquisition. But they can also be very detailed and specific for a particular project, such as: analyze data from Company X's product database, select relevant product specification parameters, etc. The software process modeling approach allows to construct such a hierarchically organized set of process models. Abstract processes can be described by complex methods which are themselves a set of more detailed processes. We make use of this property to structure the experience base.

The experience base is organized on three levels of abstraction: a *common generic level* at the top, a *cookbook-level* in the middle, and a *specific project level* at the bottom (see Figure 1.5).

**Common Generic Descriptions.** At this level, processes, products, and methods are collected that are common for a large spectrum of different CBR applications. These descriptions are the basic building blocks of the methodology. The documented processes usually appear during the development of most CBR applications. The documented methods are very general and widely applicable and give general guidance of how the respective processes can be enacted. At this common level, processes are not necessarily connected to a complete product flow that describes the development of a complete CBR application. They can be isolated entities that can be combined in the context of a particular application or application class.

**Cookbook-Level**: Experience Modules. At this level, processes, products, and methods are tailored for a particular class of applications (e.g., help desk, technical maintenance, product catalog). For each application class, the cookbook-level contains a so-called experience module. Such an *experience module* is a kind of recipe describing how an application of that kind should be developed and/or maintained. Thereby, process modells contained in such a module provide specific guidance for the development of a CBR application of this application class. Usually, these items are more concrete versions of items described at the common level. Unlike processes at the common level, all processes which are relevant for an application class are connected and build a product flow from which a specific project plan can be developed.

**Specific Project Level.** The specific project level describes experience in the context of a single particular project that had already been carried out in the past. It contains project specific information such as the particular processes that were carried out, the effort that was spent for these processes, the products that have been produced and methods that have been selected to actually perform the processes and the people that had been involved in executing the particular processes. It is a complete documentation of the project which is today more and more required anyway to guarantee quality standards required by industrial clients.

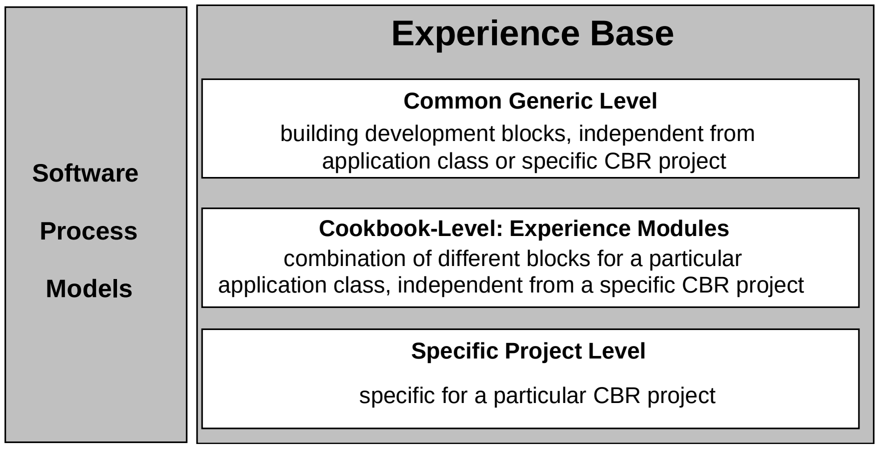

Software
Process
Models
Experience Base
Common Generic Level
building development blocks, independent from application class or specific CBR project
Cookbook-Level: Experience Modules
combination of different blocks for a particular application class, independent from a specific CBR project
Specific Project Level
specific for a particular CBR project

**Fig. 1.5.** Structure of the Experience Base

### 1.3.4 Documentation of the Experience Base

Processes, products, methods, agents, and tools being stored in the experience base are documented using a set of different *types of sheets*. A sheet is a particular form that is designed to document one of the items. It contains several predefined fields to be filled as well as links to other sheets. We have developed four types of sheets (for products, processes, simple methods, and complex methods) for documenting generic processes that occur on the top and the middle layer of the experience base and six types of sheets (four sheets for products, processes, simple methods, and complex methods, and two additional sheets for tool and agent descriptions) for documenting specific processes for the specific project level of the experience base. Figure 1.6 shows the four generic description sheets. One kind of sheet is used to describe generic processes. Generic process sheets contain references to the respective input, output, and modified products of the process. Each product is documented by a separate generic product description sheet. Each process description sheet also contains links to one or several generic methods. A generic method can either be a generic *simple* method (which is elementary and does not contain any references to other description sheets) or it can be a generic complex method. Such a generic complex method connects several subprocesses (each of which is again documented as a separate generic process description) which may exchange some by-products (documented as separate generic product descriptions). Figure 1.7 gives an example of a sheet that documents generic complex methods for defining similarity measures.

As part of the INRECA-II project, a particular methodology tool was implemented by Interactive Multimedia Systems which supports the management of the experience base and the different modules it consists of (see Fig. 1.8). It supports the filling of the different sheets, checks consistency, and creates the required links. It exports the experience base as HTML network in which each sheet becomes a separate HTML page that includes links to the related pages. Therefore, it is possible to investigate the experience base via intranet/internet using a standard Web browser.

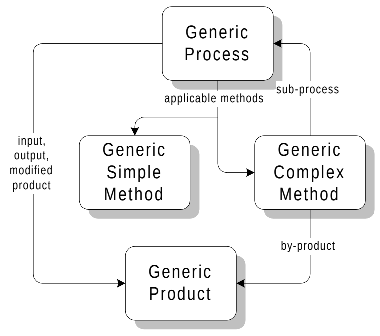


**Fig. 1.6.** Overview of generic description sheets

### 1.3.5 Current Experience at the Common Generic Level

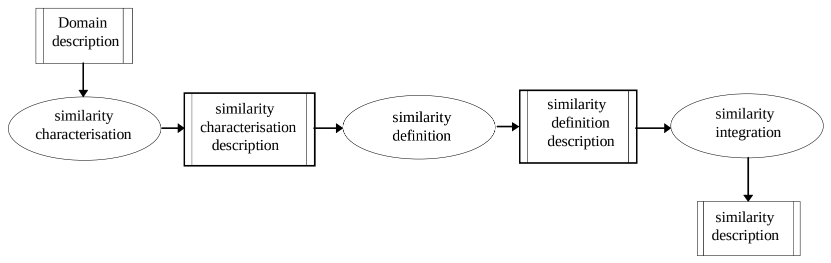
A large number of processes are involved in a CBR project. As introduced, we distinguish between the technical processes (getting the CBR system to work in context), the organizational processes (ensuring that the relationships between the users of the system and their working environment are adapted to take advantage of the power of the system), and the management of the associated process of change (or the project). However, this distinction does not imply that these different kinds of processes can be considered independently. There is in fact a lot of interaction between them. Altogether, the common generic level currently consists of a network of about 150 description sheets. Fig. 1.9 gives an overview of the processes and the products that they exchange. Only the top-level processes and products are shown in this figure. For most of these top-level processes, refinements have been defined through subprocesses which are combined in the context of complex methods or subproducts. Here, we only give an overview of top-level processes we have identified. At this top-level, the processes look very much like processes that might occur in any IT project. While this is in fact true for some of the

**Generic Complex Methods Description Sheet**

| | | | |
|---|---|---|---|
| **Project:** | **Version:** 1.0 | **Date:** 07.04.97 | **Context:** TP |
| **Author:** Armin Stahl | | | |

**Method Name: similarity construction method**

**Product Flow Diagram:**


Domain description
similarity characterisation
similarity characterisation description
similarity definition
similarity definition description
similarity integration
similarity description

**Subprocesses**

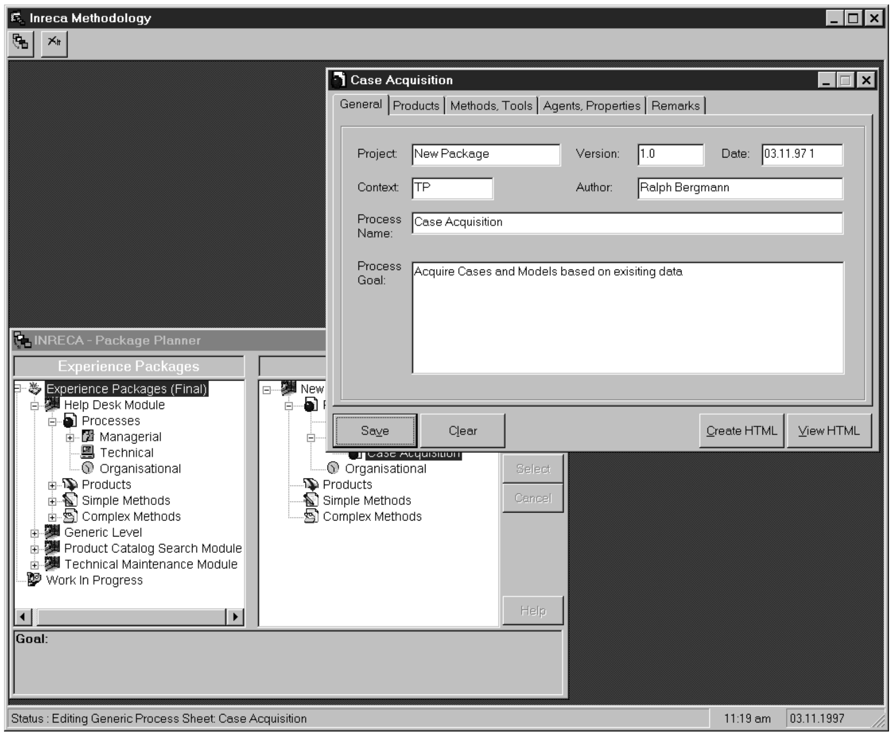
| **Name** | **Generic Process** |
|---|---|
| similarity characterisation | similarity characterisation process |
| similarity definition | similarity definition process |
| similarity integration | similarity integration process |

**By-Products**

| **Name** | **Generic Product** |
|---|---|
| similarity characterisation description | similarity characterisation description document |
| similarity definition description | similarity definition description document |

**Remarks:** In the first subprocess the CBR consultant and the domain expert must characterise the similarity measures for every simple attribute of the domain model. This means that they find out general properties of the similarity measures. In the second subprocess these properties are used to define the concrete similarity measures. In the last step the similarity measures of all simple attributes must be combined to a complex similarity measure for the domain model.

**Fig. 1.7.** Generic complex method description sheet for constructing similarity measures



Project: New Package
Version: 1.0
Date: 03.11.97 1
Context: TP
Author: Ralph Bergmann
Process Name: Case Acquisition
Process Goal: Acquire Cases and Models based on exisiting data

Status : Editing Generic Process Sheet: Case Acquisition 11:19 am 03.11.1997

**Fig. 1.8.** INRECA-II methodology tool Image provided curtsey of IMS

processes or subprocesses (e.g. some of the managerial processes are standard IT processes) most of them are described at the lower levels in a way that is particularly tailored for CBR application development. In the following we give a brief overview of all the top-level processes and there interactions, before explaining them in some more detail.

A CBR project starts with a statement of a client's problem. In the *top-level start-up* process (managerial) a first vision document is created, which preliminary defines the mission of the project. The following *goal definition* process (managerial) is executed in strong interaction with the *preliminary organizational analysis* process (organizational). The organizational problems (that should be addressed by introducing the CBR system) and the respective problem owners are identified. Additionally, a basic goal checklist for the project is created. Based on this goal check list and the general project vision, a *feasibility study* (technical) should be carried out. Depending on the results of this study, it must be decided whether

– to continue the project as planned,
– to revise the main project goals,
– to stop the project.

Thereafter, the *identification of the high-level processes* (managerial) leads to a set of technical and organizational processes that should be addressed within the project. The methodology experience module should be consulted during this process since it gives valuable advice on the processes that are relevant. Additionally, a *detailed analysis of the organization* (organizational process) can be carried out in order to revise and refine the set of organizational processes that should be addressed and to identify possible project teams. Then, the project leader should *plan and schedule the technical and organizational processes* (managerial) and produce an overall project plan. Here, again the experience module should be consulted in order to build on concrete experience from other, similar CBR projects. The resulting project plan is then enacted and monitored during the technical process of *software development for case-based reasoning* and the *organizational development* of the environment into which the CBR system will be introduced. The resulting software implementation and the newly established responsibilities and workflow in the organization will then be *implemented and evaluated.*

Typically, a CBR project requires several development cycles, each of which results in a prototype application that is evaluated by the end-user. The evaluation feedback states whether a new development cycle must be considered or whether a satisfying solution is reached already. At this point, the first living application being in real use has been reached. During the lifetime of this application, some *maintenance* (technical process) will be required during which the application will be modified.

Please note that a separate *monitor & review* process (managerial) is introduced. This process controls the enactment of the software and organizational development. This process does not have a formal output. It also uses the evaluation feedback from the implemented application to allow the project leader to decide whether a new development cycle should be considered to further improve the current system or the organizational structure.

**Managerial Processes.** This subsection is primarily concerned with the management of the process of development and implementation of the system in its context, but it also touches the other two aspects where necessary given that they are impossible to separate. Managing a CBR project (or any AI project in general) differs from managing other IT projects to the extent that the associated concepts of the CBR technology are mostly previously unknown to the users. Therefore, there must be more than usual emphasis on early awareness training, and user-participation in the successive iterations of the prototyping process.

**Top-Level Start-up**

During the top-level start-up process, client and the top-management of the consulting firm agree on a terms of reference, which specifies the problem or opportunity to be addressed. During this process, the project leader is also identified. This person should have the time and motivation to ensure that the project is a success, and should also have some ownership of the project results. The leader then helps to define the overall project vision stated in the vision document in close interactive consultation with the client and the consultant- firm's management. It is also the duty of the leader to set up a system for monitoring the implementation of the project.

**Goal Definition**

The first process after start-up is to define the project goal. The vision of what life will be like when the project is completed has to be written down. At this stage, it is appropriate to introduce a high-level abstract concept of a 'case', from which case-based reasoning can be developed, and to identify what will be different about the people, their tasks, the technology and organization when the project is finished. This description should also elaborate any associated or dependant goals, in a goal checklist. These goals should be agreed amongst all participants in the project, and by anybody affected by the outcomes of the project. When this has been done, it becomes possible to define the high-level processes which support the performance of scheduled technical and organizational tasks during the whole project.

**Identification of High-Level Processes**

Even though it is difficult to define precisely each project process in detail, it is possible first to produce a lot of high-level processes that have to be enacted in order to achieve the project goal. As described, the processes can usefully be subdivided into technical and organizational categories. For each process the project planner should check the resource requirements, input- and output products and the dependencies between processes. The determination of the relevant processes should be done based on existing CBR building experience. For this purpose, the project leader should have access to the experience base of CBR methodology (see section 1.3.8).

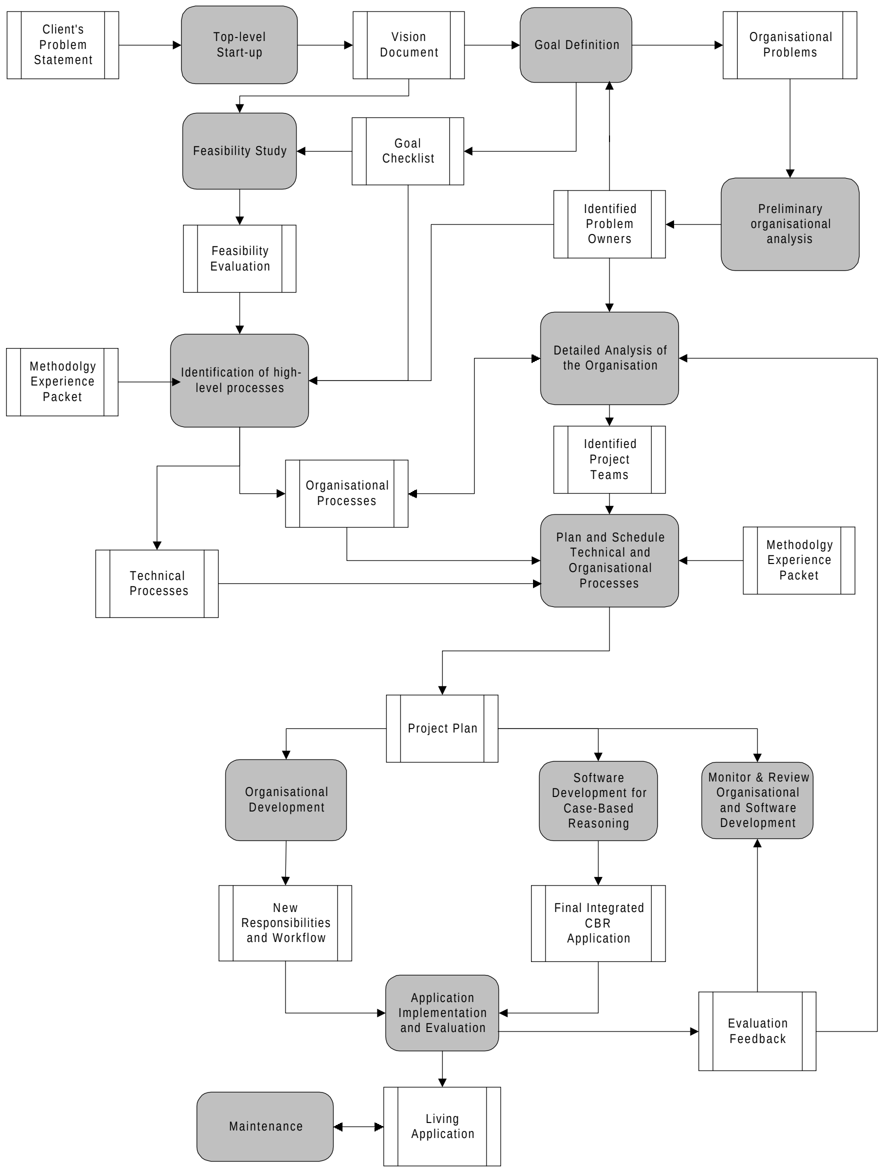

Client's Problem Statement
Top-level Start-up
Vision Document
Goal Definition
Organisational Problems
Feasibility Study
Goal Checklist
Identified Problem Owners
Preliminary organisational analysis
Feasibility Evaluation
Methodolgy Experience Packet
Identification of high-level processes
Detailed Analysis of the Organisation
Identified Project Teams
Organisational Processes
Plan and Schedule Technical and Organisational Processes
Methodolgy Experience Packet
Technical Processes
Project Plan
Organisational Development
Software Development for Case-Based Reasoning
Monitor & Review Organisational and Software Development
New Responsibilities and Workflow
Final Integrated CBR Application
Application Implementation and Evaluation
Evaluation Feedback
Maintenance
Living Application

**Fig. 1.9.** Interaction between top-level processes involved in a CBR project

**Plan and Schedule Processes**

During the detailed planning of the technical and organization processes, a couple of important aspects must be considered by the project leader. She/he must

- ensure that the immediate objectives of each project phase is embedded in strategic objectives of the overall project.
- ensure that in the design of the system there is provision for the participative interactive re-design of the work-practices of those using it.
- ensure that when the developed system is implemented, there is timely provision for the necessary level of training in its use by all who interact with it.
- ensure that the implemented system is evaluated critically, with a view to future improvements, identification of possible weaknesses etc., bearing in mind that development is an ongoing process.

**Monitoring and Review of Organizational and Software Development**

There is no difference in monitoring and reviewing the development processes compared to the management of standard IT projects. The implementation of the project plan in the organizational and software development processes must be monitored by the project leader. He must ensure, that the important project milestones can be reached in time. He must control the overall quality of the development process and of the deliverables. Based on the feedback from the evaluation of the application he finally has to decide whether a new organizational or technical development cycle should be considered.

**Technical Processes: Software Development.** The technical processes described in the subsection are general processes, i.e., such processes that we expect to be a part of most CBR application development projects. A number of the technical processes appear in usual software development projects, too. An example is the GUI development process. Such processes can be handled by standard methods. However, we emphasize on the CBR specific aspects.

**Feasibility Study**

The feasibility study consists of a cost/benefit analysis and should deliver an estimation of the commercial success potential of the intended CBR application. A simple mock-up of the intended application should be developed as a means to further clarify the goals and the vision of the project as well as a means to convince the customer to believe in the success of the project. Figure 1.10 gives a brief overview of the subprocesses involved in the feasibility study.

**Software Development for Case-Based Reasoning**

This is the technical top-level process during which the CBR application itself is developed. A CBR application development process in general is a usual software development process with mostly standard ingredients on

- Cost/Benefit Analysis
- Application Mock-up Development
    - GUI Mock-up Development
    - Case-Base Mock-up Development
    - Retrieval Mock-up Development
    - Integration Mock-up Development

**Fig. 1.10.** Sub-Processes occuring in the Feasability Study Process

its top-level. This includes a large number of different subprocesses listed hierarchically in Figure 1.11. Each of these processes is described in detail in the experience base.

- GUI Development
    - GUI Requirements Analysis
    - GUI Design
    - GUI Programming
    - GUI Manual Writing
    - GUI Acceptance Testing
- CBR Engine Development
    - CBR Retrieval Requirements Analysis
    - CBR Engine Design
    - CBR Engine Implementation
    - CBR Engine Test
- Case Base Development
    - Case Base Requirements Analysis
    - Descriptive Model Development
    - Similarity Development
        - Similarity Characterization
        - Similarity Definition
        - Similarity Integration
    - Case Acquisition
        - Develop Case Collection Forms
        - Collect Case Data
        - Evaluate Selected Good Cases
        - Case Entry into the Case Base
- System Integration
    - Integration Requirements Analysis
    - Integration Design
    - Integration Implementation
    - Integration Test

**Fig. 1.11.** Sub-Processes occuring in the Software Development Process

**Maintenance Process**
The maintenance process relies heavily on the successful installation of organizational structures that support this process. One important organizational aspect is the installation of a board of users and domain experts that review the case data regularly and that decides what changes and updates to perform. From the technical point of view it should be aspired that updates

of the system should be executed by an application administrator of the user organization in order to ensure some independence from the consultancy company. The application should therefore provide means for easy updating of the case base and simple modifications of the domain model. However, more drastic changes of the system (e.g. restructuring of the domain model) should only be performed by or in co- operation with a CBR consultant. Figure 1.12 gives an overview of the maintenance processes identified so far.

Change Selection
Model Maintenance
Case Base Maintenance
Application Update

**Fig. 1.12.** Sub-Processes occuring in Maintenance Process

**Organizational Processes.** In analyzing the existing organization's structure and related procedures, there is a preliminary scoping of the environment of the project. This includes a listing of the perceived problems and opportunities at the human and organizational levels.

**Preliminary Organizational Analysis**

In the preliminary organizational analysis process the first step is to identify the boundaries of the socio-technical system of which the perceived problems or opportunities are currently or potentially dealt with by introducing a CBR system. Once this has been identified, it can be regarded as being a 'production' unit embedded in a larger organization. We need to enact this preliminary organizational analysis in strong interaction with the managerial goal definition process. Bearing this in mind, we go on to analyze the processes embedded in the socio-technical unit, and to assess the impact of the technological change on the process management. This naturally leads to a synthesis in which the processes are re-organized in the context of the new technology.

**Detailed Analysis of the Organization**

The design of the CBR system will be the result of an iterative process involving all the people in the existing organization who are involved with the cases, the collective of stakeholders who own the material which will end up in the case- file, and who are potential users of the results of the system.

It is useful, having identified the problems and/or opportunities, to set up one or more working groups focused on them, to address the questions a) how is this problem handled elsewhere and b) how does the present situation compare with other CBR applications? The group should consider the processes, the people, the technologies and the structure within the socio-technical unit which deals with the cases. In this way they can begin to tease out the implications for the organization.

**Organizational Development**

The basic processes that must be addressed during the organizational development are the

– adaptation of the structure of the organization, including
– the structuring of the maintenance process for the CBR application,
– the training of the maintenance personal, and
– the training of the users of the system.

### 1.3.6 Current Experience at the Cookbook Level

We draw on the cookbook metaphor in the following sense: like a cookbook that contains a collection of recipes each of which leads to a particular dish, the cookbook-level of the methodology contains a collection of experience modules each of which combines selected generic processes in order to realize a particular kind of CBR application. Till the end of the INRECA-II project, we intend to have three modules for: *Parametric Search in Product-Bases, Complex Help Desk, and Technical Maintenance.*

The most advanced module currently is the parametric search module. The basic action is a search by a potential client, or a sales person in the presence of a client, in a product-base or catalogue. Typically, the set of all available products can be divided into several product categories each of which might contain some 100s of products. Each product can be characterized (or even exactly specified) by a set of parameters. Typically each product might be characterized by some 10s of parameters. The client specifies a need by specifying constraints on certain product parameters, and the search comes up with something approximating to it. Such a service can be provided for example on a CD-ROM or on the World-Wide-Web.

To summarize the catalogue search application class in the form of a checklist, we can say that:

– a catalogue of products must exist, each of which can be characterized by a number of qualitative (real or integer values) or quantitative parameters (symbols).
– the list of products subdivides into a number of subsets (product categories), in each of which the products share a common structure of parameters.
– a sufficient number of products must exist to make the task of searching for a product with a desired set of parametric values onerous.
– a group of potential users of the products must exist who are in a position to specify the parametric profile of the product that they need, making use of expert knowledge.

### 1.3.7 Current Experience at the Specific Level

At the specific project level, the experience from five projects from the industrial INRECA-II partners is captured and stored in about 350 sheets. These projects are documented using the software process modeling approach after the projects were finished. These projects provide the basis for developing experience modules at the cookbook-level through generalization.

### 1.3.8 Using and Maintaining the Experience Base

When a new CBR project is being planned, the relevant experience from the experience base must be selected and reused. The experience modules of the cookbook-level are particularly useful for building a new application that directly falls into one of the covered application classes. We consider the experience modules to be the most valuable knowledge of the methodology. Therefore, we suggest to start the "retrieval" [11] by investigating the cookbook-level and only using the common generic level as fall-back. Furthermore, it is important to maintain the experience base, i.e., to make sure that new experience is entered if required. For using and maintaining the experience base we propose the following procedure:

1. Identify whether the new application to be realized falls into an application class that is covered by an experience module of the cookbook. If this is the case then goto step 2a; else goto step 3.

2a. Analyze the generic processes, products, and methods that are proposed for this application class.

2b. Select the most similar particular application from the specific project level related to this module and analyze the specific description sheets in the context of the current application.

2c. Develop a new project plan and workflow for the new application based on the information selected in steps 2a and 2b. Goto step 4.

3. Develop a new project plan and workflow for the new application by selecting and combining some of the generic processes, products, and methods from the common generic level; make these descriptions more concrete and modify them if necessary.
4. Execute the project by enacting the project plan. Record the experience during the enactment of this project.
5. Decide whether the new project contains new valuable information that should be stored in the experience base. If this is the case, goto step 6, else stop.
6. Document the project using the specific description sheets and enter them into the specific project level of the experience base (supported by the methodology tool).
7. If possible, create a new experience module by generalizing the particular application (together with other similar applications) to an application class and generalize the specific descriptions into generic descriptions.

[11] Up to now, this retrieval is not supported by a tool, but through an index schema. However, support for retrieval (e.g. a CBR approach) is considered important for the future.

Add the new to the current cookbook (supported by the methodology tool).

8. If new generic processes, methods, or products could be identified that are of a more general interest, i.e., relevant for more than the application class identified in step 7, then add them to the common generic level (supported by the methodology tool).

## 1.4 Conclusion

In this chapter, we have presented a methodology for building CBR applications. First, this methodology consists of an analytic framework for describing different kinds of CBR approaches. It provides an explicit ontology of basic CBR task types, domain characterizations, and types of problem solving and learning methods. Further, it incorporates within this framework a methodology for combining a knowledge-level, top-down analysis with a bottom-up, case-driven one. Second, the presented methodology consists of a systematic way to identify the main steps required to build a CBR application, based on collected experience stored as CBR specific software process models. We gave an overview of the main processes involved in building a CBR application for an analytic task, including managerial, organizational, and technical processes.

Up to now, this methodology is supported by two tools:

- A CBR system for retrieving descriptions of CBR approaches, CBR systems, and CBR applications. This retrieval system (implemented by the Fraunhofer Institute for Experimental Software Engineering in cooperation with TECINNO) is currently be made available via WWW (see section **??**.7 for further details including the URL).
- The INRECA-II methodology tool (implemented by Interactive Multimedia Systems) which supports the management of the experience base and the different modules it consists of (see Fig. 1.8). Thereby, this tool currently supports step 6 and partially the steps 7 and 8 from section 1.3.8.

The development of a methodology is an ongoing task which is not finished yet. Future work should address the following two major points:

First, the scope the of the methodology must be enlarged as new kinds of applications are being developed. That is, more kinds of analytic applications must be covered, but in the long-term also synthetic applications should be included. Therefore, the experience base must grow further. More specific experience from successful CBR projects must be collected and generalized into new experience modules (cookbook recipes). New processes at the generic level must be identified.

Second, further tool support is required to make the methodology easier to use. Based on our current experience with the methodology, tools are necessary that also support steps 1 to 5 from section 1.3.8. Particularly, a tool is required that allows to access process models from the experience base more efficiently, either by a flexible navigation approach or by a retrieval system (support of steps 1 to 2b). The application of case-based reasoning to this retrieval task would certainly be a new challenging CBR application itself.

Further, an integration with existing project planning and workflow management tools is desirable. This would allow to select certain processes from the experience base and to combine and tailor them according to the current application being developed (support of steps 2c to 4). A first case study that shows the appropriateness of the flexible workflow management system CoMoKit (Dellen et al., 1997b) for representing and enacting a process model of a CBR application development project has been performed (Bergmann et al., 1997b).

# References

Aamodt, A. (1991). *A Knowledge-Intensive, Integrated Approach to Problem Solving and Sustained Learning.* PhD thesis, University of Trondheim.

Aamodt, A. (1993). Explanation-driven cased based reasoning. In (Wess et al., 1993), pages 274 – 288.

Aamodt, A. (1995). Knowledge acquisition and learning by experience the role of case-specific knowledge. In Kodratoff and Tecuci, editors, *Machine Learning and Knowledge Acquisition - Integrated Approaches*, pages 197–245. Academic Press.

Aamodt, A. and Plaza, E. (1994). Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–59.

Althoff, K.-D. (1992). *Eine fallbasierte Lernkomponente als integrierter Bestandteil der* MOLTKE-*Werkbank zur Diagnose technischer Systeme.* PhD thesis, Dept. of Computer Science, University of Kaiserslautern, Germany.

Althoff, K.-D. (1996). The inreca case study. Postdoctoral thesis, University of Kaiserslautern.

Althoff, K.-D. and Aamodt, A. (1996). Zur Analyse fallbasierter Problemlöse- und Lernmethoden in Abhängigkeit von Charakteristika gegebener Aufganstellungen und Anwendungsdomänen. *Künstliche Intelligenz, Themenheft Fallbasiertes Schließen*, 10(1):10–15.

Althoff, K.-D., Auriol, E., Barletta, R., and Manago, M. (1995a). *A Review of Industrial Case-Based Reasoning Tools.* AI Intelligence, Oxford.

Althoff, K.-D., Bergmann, R., Wess, S., M., M., Auriol, E., Larichev, O. I., Bolotov, A., Zhuravlev, Y. I., and Gurov, S. I. (1998). Integration of induction and case-based reasoning for decision support tasks in medical domains: The inreca approach. *To appear in: AI in Medicine Journal.*

Althoff, K.-D., Maurer, F., and Rehbold, R. (1990). Multiple Knowlegde Acquisition Strategies in MOLTKE. In Wielinga, B., Boose, J., Gaines, B., Schreiber, G., and van Someren, M., editors, *Proceedings of the 4th European Knowledge Acquisition Workshop EKAW'90*, Amsterdam. IOS.

Althoff, K.-D. and Wess, S. (1991). Case-Based Knowledge Acquisition, Learning and Problem Solving for Diagnostic Real World Tasks. In *Proceedings of the 5th European Knowledge Acquisition Workshop EKAW'91.*

Althoff, K.-D., Wess, S., Weis, K.-H., Auriol, E., Bergmann, R., Holz, H., Johnston, R., Manago, M., Meissonnier, A.and Priebisch, C., R., T., and W., W. (1995b). An evaluation of the final integrated system. Technical report, Esprit Project INRECA Deliverable D6.

Althoff, K.-D. and Wilke, W. (1997). Potential uses of case-based reasoning in experienced based construction of software systems and business process support. In (Bergmann and Wilke, 1997), pages 31–38.

Armengol, E. and Plaza, E. (1993). A Knowledge Level Model of Case-Based Reasoning. In (Wess et al., 1993), pages 53 – 64.

Auriol, E., Althoff, K.-D., Wess, S., and Dittrich, S. (1994). Integrating induction and case-based reasoning: Methodological approach and first evaluations. pages 18–32. AcknoSoft Press.

Bareiss, R. (1989). *Exemplar-Based Knowledge Acquisition: A unified Approach to Concept Representation, Classification and Learning.* Academic Press.

Bartsch-Spörl, B. (1996). How to introduce case-based reasoning in customer support. Technical report, Esprit Project APPLICUS Deliverable D3.

Bartsch-Spörl, B. (1996). How to make CBR systems work in practice. Informatik-Berichte, pages 36–42. Berlin.

Bartsch-Spörl, B. (1997). How to introduce cbr applications in customer support. In (Bergmann and Wilke, 1997).

Bartsch-Spörl, B., Althoff, K.-D., and Meissonnier, A. (1997). Reasoning about case-based reasoning systems. In Mertens, P. and Voss, H., editors, *4th German Conference on Expert Systems*, pages 115–128, Sankt Augustin, Germany. infix Verlag. Proceedings in Artificial Intelligence 6.

Basili, V. R., Caldiera, G., and Rombach, H. D. (1994). Experience factory. In Marciniak, J. J., editor, *Encyclopedia of Software Engineering, volume 1*, pages 469–476. John Wiley & Sons.

Bergmann, R., Breen, S., Göker, M., Manago, M., Schumacher, J., Stahl, A., Tartarin, E., Wess, S., and Wilke, W. (1998). The INRECA-II Methodology for Building and Maintaining CBR Applications.

Bergmann, R. and Wilke, W., editors (1997). *5th German Workshop on CBR — Foundations, Systems, and Applications —*, Report LSA-97-01E, Kaiserslautern. University of Kaiserslautern.

Bergmann, R., Wilke, W., Althoff, K.-D., Johnston, R., and Breen, S. (1997a). Ingredients for developing a case-based reasoning methodology. In (Bergmann and Wilke, 1997), pages 49–58.

Bergmann, R., Wilke, W., and Schumacher, J. (1997b). Using software process modeling for building a case-based reasoning methodology: Basic approach and case study. Lecture Notes in Artificial Intelligence, 1266, pages 509–518. Springer Verlag.

Booch, G. (1994). *Object-Oriented Design with Applications.* Benjamin/Cummings.

Cohen, P. R. (1989). Evaluation and Case-Based Reasoning. In (Hammond, 1989), pages 168–172.

Curet, O. and Jackson, M. (1996). Towards a methodology for case-based systems. In *Expert Systems 96: Proc. of the 16th annual workshop of the British Computer Science Society.*

Dellen, B., Maurer, F., and Jürgen Muench, M. V. (1997a). Enriching software process support by knowledge-based techniques. *Int. Journal of Software Engineering and Knowledge Engineering*, 7(2):185–215.

Dellen, B., Pews, G., and Maurer, F. (1997b). Knowledge based techniques to increase the flexibility of workflow management. *Data & Knowledge Engineering Journal.*

Gibbs, W. W. (1994). Software's chronic crisis. *Scientific American*, pages pp. 86–95.

Hammond, K. J., editor (1989). *Proceedings: Case-Based Reasoning Workshop.* Morgan Kaufmann Publishers.

Johnston, R., Breen, S., and Manago, M. (1996). Methodology for developing cbr applications. Technical report, Esprit Project INRECA Deliverable D30.

Kitano, H. and Shimazu, H. (1996). The Experience-Sharing Architecture: A Case Study in Corporate-Wide Case-Based Software Quality Control. In Leake, D. B., editor, *Case-Based Reasoning: Experiences, Lessons, and Future Directions*, chapter 13. AAAI Press/MIT Press, Menlo Park, CA/Cambridge, MA.

Kolodner, J. L. (1993). *Case-Based Reasoning.* Morgan Kaufmann, San Mateo.

Koton, P. (1989). Evaluating Case-Based Problem Solving. In (Hammond, 1989), pages 173–175.

Lewis, L. (1995). *Managing computer networks: A case-based reasoning approach.* Artech House Publishers, London.

Manago, M., Althoff, K.-D., Auriol, E., Traphöner, R., Wess, S., Conruyt, N., and Maurer, F. Induction and reasoning from cases. pages 313–318.

Meissonnier, A. (1996). Ein fallbasiertes informationssystem für fallbasierte anwendungen und systeme auf der basis von inreca. Diploma thesis, University of Kaiserslautern.

Newell, A. (1982). The knowledge level. *Artificial Intelligence*, 18:87–127.

Richter, M. M. and Wess, S. (1991). Similarity, uncertainty and case-based reasoning in PATDEX. In Boyer, R., editor, *Automated Reasoning*, pages 249–265. Kluwer. Essays in Honour of Woody Bledsoe.

Rombach, H. D. and Verlage, M. (1995). Directions in software process research. In Zelkowitz, M. V., editor, *Advances in Computers, Vol. 41*, pages 1–61. Academic Press.

Shaw, M. (1990). Prospects for an engineering discipline of software. *IEEE Software 7*, pages 15–24.

Stadler, M. and Wess, S. (1989). Patdex: Konzept und implementierung eines fallbasierten, analogieorientierten inferenzmechanismus zur diagnose eines cnc-bearbeitungszentrums. Project thesis, University of Kaiserslautern.

Steels, L. (1990). Components of expertise. *AI Magazine*, 11(2):29–49.

Veloso, M. M. and Aamodt, A., editors (1995). *Case Based Reasoning Research and Development: First International Conference, ICCBR-95.* Number 1010 in Lecture Notes in Artificial Intelligence. Springer, Berlin.

Wess, S. (1990). Patdex/2: ein system zum adaptiven, fallfokussierenden lernen in technischen diagnosesituationen. Diploma thesis, University of Kaiserslautern.

Wess, S. (1993). PATDEX – Ein Ansatz zur wissensbasierten und inkrementellen Verbesserung von Ähnlichkeitsbewertungen in der fallbasierten Diagnostik. In Puppe, F. and Guenter, editors, *Proceedings XPS-93*, Hamburg. Springer-Verlag.

Wess, S. (1995). *Fallbasiertes Problemlösen in wissensbasierten Systemen zur Entscheidungsunterstützung und Diagnostik.* PhD thesis, Universität Kaiserslautern. Available as DISKI 126, infix Verlag.

Wess, S., Althoff, K.-D., and Richter, M. M., editors (1993). *Topics in Case-Based Reasoning. Proc. of the First European Workshop on Case-Based Reasoning (EWCBR-93)*, Lecture Notes in Artificial Intelligence, 837. Springer Verlag.

Wielinga, B. J., Van de Velde, W., Schreiber, G., and Akkermans, H. (1993). Second generation expert systems. chapter Towards a unification of knowledge modeling approaches, pages 299–335. Springer.