Using Software Process Modeling for Building a Case-Based Reasoning Methodology: Basic Approach and Case Study

Ralph Bergmann, Wolfgang Wilke, Jürgen Schumacher

University of Kaiserslautern
Centre for Learning Systems and Applications (LSA)
PO-Box 3049
D-67653 Kaiserslautern, Germany
{bergmann, wilke, jschuma}@informatik.uni-kl.de
Phone: + 49 631/205-3361 or -3363 Fax: +49 631/205-3357

Abstract. Building a methodology for developing and maintaining CBR applications is an important goal currently addressed by CBR researchers and practitioners. Since CBR application development is a special kind of software development, building a CBR methodology can certainly be viewed as a software engineering research and development activity. This paper presents a perspective of how software process modeling, which is a recent approach in software engineering, can be used for building a case-based reasoning methodology. Further we describe a case study to show the applicability of the proposed concepts.

Keywords: CBR methodology, process modeling

1 Introduction

Recently, several activities have been initiated with the goal of establishing a methodology for developing case-based reasoning (CBR) applications. 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). A methodology should give guidelines about the activities that need to be performed in order to successfully develop a certain kind of product, e.g. any kind of software system, a knowledge-based system, or - as in our case - a CBR application. A methodology should make the development an engineering activity rather than an art known by a few experts (cf. Shaw 1990; Gibbs 1994). A methodology should cover:

- the process of project management (cost and resource assessment, time schedules, project plans),
- the process of product development and maintenance: particularly product selection (e.g. of CBR tools), software development and world modeling required to build a running CBR system

- the specification of the different kinds of products or deliverables (including software deliverables) that must be produced,
- the analysis and reorganisation of the environment (e.g. a department) in which the CBR system should be introduced.

One of the main driving forces behind the development and the use of a methodology relates to the need for quality in both the products and processes of the development of computer-based systems. Contemporary organisations can no longer sustain inefficient or ineffectual IT system development. IT systems are increasingly seen as a means of gaining competitive advantage over rivals in the marketplace. The use of an appropriate methodology will provide significant quantifiable benefits in terms of productivity (e.g. reduce the risk of wasted efforts), quality (e.g. inclusion of quality deliverables) and 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). This general observation holds for software development in general and for case-based reasoning application development in particular. Today, a few companies have specialised in developing CBR applications but they still perform this in an ad hoc manner. This will also create serious problems in case of staff departures, particularly when new staff must be trained.

Contributions to the goal of developing a methodology for CBR can be found in books on CBR (Kolodner 1993, chapter 15; Wess 1995, chapter 6) and in papers collecting the experience of people who have successfully developed CBR applications (e.g. Lewis 1995; Bartsch-Sprl 1996a; Curet and Jackson 1996); most valuable experience-based contributions arose from projects where methodology development was explicitly included as one project task, like INRECA[1] (Althoff, Wess et al. 1995; Johnston, Breen, and Manago 1996) or APPLICUS[2] (Bartsch-Sprl 1996b, 1997). The main focus of the INRECA-II[3] Esprit project is the development of such a methodology for CBR applications in the area of diagnosis and decision support. As a first contribution to this task, the ingredients of a case-based reasoning methodology were discussed by Bergmann et al. (1997).

The goal of this paper is to analyze whether software process modeling, which is a recent approach in software engineering (SE), can be used as a basis for developing a CBR methodology. Therefore, we will first describe software process modeling in general and show what must be done in order to use it in the context of CBR. Then, we describe a case study in which we re-modeled the project plan

[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 and 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)

from the APPLICUS project in terms of a particular software process modeling approach.

2 Methodologies in Software Engineering

Software engineering is concerned with aspects of software development and maintenance; planning of software developments projects; performing development and project management, including quality assurance (Rombach and Verlage 1995). Particularly, SE is concerned with principles, methods, techniques and tools that can support these activities. Since CBR application development is a special kind of software development, building a CBR methodology can certainly be viewed as a SE research and development activity.

2.1 Software Process Modeling

One area of SE that can provide such a philosophy also for a CBR methodology is software process modeling (SPM) (Rombach and Verlage 1995). In this area product engineering process models model the engineering of the product, i.e., the software that has to be produced. Unlike early approaches in SE, 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. Product engineering process models include technical SE processes (like requirements engineering, design of the system to be built, coding, etc.), managerial SE 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.

Using SPM for managing a software project involves four related levels (see figure 1):

- Experience base level. At this level generic process models and product models are described that have a certain degree of generality. These models are collected in an experience base for (re)use during the planning of a new software project.
- Project planning level. At this level, a project plan is constructed that combines a particular set of processes required for a certain project. Primarily, the models contained in the experience base are reused to construct a project plan. However, one cannot expect that the experience base always contains process models that are completely appropriate or at a sufficient level of detail. If we do not find appropriate process models, the existing models must be extended or refined, or even new models must be constructed. Such new or revised models may enter the experience base at some later point.

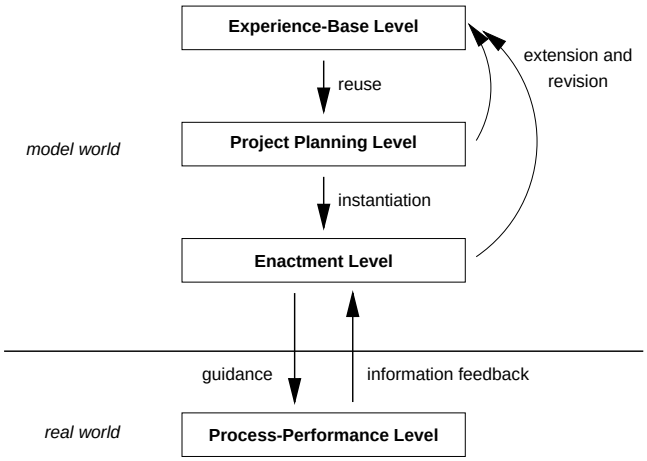

Fig. 1. Different levels in software process modeling

- Enactment level. At this level, a project plan is instantiated and information about the current state of enactment of the processes are included.
- Process-performance level. While the three previous levels deal with representations of the real world, the process-performance level is the real world in which agents are concerned with the development and maintenance of the software, performing the processes defined in the project plan. The project plan at the enactment level guides the real-world project by providing information about the project's state, next meaningful steps, and expected qualities of products and processes. The process performance domain provides feedback about results, events, or changes in order to allow updating the representation.

Managing a software project does not necessarily mean to follow these four levels in a sequence from the experience-base level to the process-performance level. Most likely, there will be changes at the project planning level during the enactment. Sometimes, certain detailed decisions in the project plan can only be drawn after some early parts of the project plan have already been enacted so that certain products (e.g. a requirements analysis document) are available as a basis for more detailed decisions.

2.2 Representing process models

Several representation formalisms and languages have been already developed for representing process models. Goal of the MILOS project (Dellen, Maurer, Verlage, Mnch 1997) is the development of a representation language and of tools supporting the modeling, enactment and storing of process models. Using MILOS, a process model is defined in terms of processes, methods, products, goals

and resources. Other representation languages often contain similar concepts, but we chose MILOS as a base for our terminology because it seems to be one of the most flexible approaches.

Process Types 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 process type. A process type describes a process 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 (see section 2.2)
- 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. These are specified by defining their types (see section see section 2.2).
- A set of resources (agents or tools) that are required to perform the step. Here the necessary qualifications or specifications are defined that a agent or tool respectively must have so that he can be assigned to the process (see section 2.2).
- Additional attributes can be declared, e.g. for measuring the effort spent on the enactment of the process.

Methods Methods contain a detailed specification of a particular way of reaching the goal of a process. A method can be either atomic or complex. While an atomic 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 and the flow of products between the subprocesses. This is one of the most important features of MILOS, because it 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. Further it is possible to instantiate a subprocess more than once.

Product Types 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. In MILOS products are modeled by defining product types, which declare attributes a product of this type may have, e.g. the current state of the product. Further a object oriented representation is used. Therefore product types can be derived from others to define specializations. Besides a product can be decomposed in subproducts so it is possible to pack products belonging together into a single product to make the process interfaces more understandable.

Resources Resources are entities necessary to perform the tasks and are divided into agents and tools. Agents are models for humans or teams (e.g. designers or programmers), which can be designated to perform the real-world processes. The most relevant properties of agents are their qualifications. Tools (e.g. editors or modeling tools) 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 process type, it is possible to compute the set of agents and tools which can be assigned to a certain process.

3 A CBR Methodology Based on Software Process Modeling

Now, the question arises what the impact of using SPM for a CBR methodology is and what benefits can be expected.

3.1 Components of the Methodology

Using software process modeling for developing a CBR methodology basically means building an experience base (see Figure 1) of generic process types, methods, product types, resources. So, a methodology is an experience base in the sense of section 2.1. The experience base should contain CBR specific managerial, technical and organizational processes at different levels of abstraction organized in a hierarchical way. Such an experience-base should be based upon the experience of successfully enacted CBR projects. From such a CBR project a concrete project plan can be extracted and the processes involved can be identified. An experience base can then be built by abstracting process types as well as the other required entities. A CBR methodology built using SPM should therefore consist of:

- a set of process types, each of which denotes a complex or elementary step required for
  - CBR application development (technical process) or
  - managing the CBR application development (managerial process) or
  - introducing the CBR application into the new environment (organizational process).

  Examples of such complex tasks can be: define the scope of the CBR project, characterize the sources of information available in the organization, define the content of a case, select a CBR tool and platform, case modeling, define similarity measures, define general knowledge and adaptation knowledge, execute initial case acquisition process, test and validate the CBR application, etc.
- a description for each such process type in the way sketched in section 2.2.1 (input- and output products, resource consumption, methods). For example, the process case modeling would require as input a documentation describing the content of a case and the user manual of the selected CBR tool and it

delivers as output a descriptive model, e.g. in CASUEL (Manago, Bergmann et al. 1994) format. Concerning resource consumption, this task requires a CBR specialist using a model-editor-tool. A complex method may decompose the process into the four subprocesses analysis of available information, modeling of types, modeling of concepts and model validation.
- a set of product types that describe the products like descriptive models, case-bases, CBR prototype, technical environment (user-interfaces, hard- and software), documentation, etc.
- a set of resource descriptions that describe the required human resources like domain expert, problem owner, software developer, CBR specialist, etc. and tools like descriptive model editor, case manager, validation tool, etc.

3.2 Applying and Maintaining the Methodology

The methodology (which is in fact the experience base) provides a set of basic components that can be reused or in order to plan and monitor a CBR project. Applying this methodology for a concrete CBR project means to:

1. select appropriate CBR processes and methods from the experience base.
2. extend or refine the selected processes
3. combine the processes to a project plan
4. enact and monitor the project plan
5. review the project plan after execution to detect flaws and generalize new generic elements that should enter the experience base.

These five points can be seen as processes that realize an experience feedback (at the level of CBR application development) which is similar to the general idea of CBR as captured in the CBR cycle by Aamodt and Plaza (1994). This issue is further discussed by Althoff and Wilke (1997).

3.3 Benefits of Process Modeling for a CBR Methodology

We expect that SPM lead to several benefits within the scope of a CBR methodology. First of all, they provide a clear philosophy in combination of a particular notation for writing down a methodology. Moreover the following important benefits can be identified:

- It supports communication within team members developing a CBR application (which is a general benefit of the methodology).
- It supports to control and monitor the CBR development.
- Because of its hierarchical nature, it allows one to first start with an abstract process model and to refine the important or difficult steps when the methodology grows, also based on the experience gained from its application. It also allows to have a general part of the methodology and several more specific parts each of which is suited for a particular kind of application (vertical platform).

- It allows one to share CBR development experience (experience factory (Basili et al. 1994; Althoff and Wilke 1997)).
- Workflow management techniques and tools (Workflow Management Coalition 1996) can be used to organize and coordinate technical and business processes. They provide tools and methods to support the management of projects. One major benefit that is expected to result by using these techniques is the coordination of resources and tasks in a CBR development project. More advanced techniques (see e.g. Maurer 1996) can also assist when re-planning the project becomes necessary during the execution of the project.

4 A Case Study

In the APPLICUS project a sample project plan for the development of a CBR-based customer support application is discussed (Bartsch-Sprl 1996b, 1997). In the following we set up a process model based on this project plan to examine if it is possible to model the concepts and processes occurring in a real-world development project by means of a process representation language like proposed in the MILOS project (see section 2.2). A second goal of this study is to test whether all concepts provided by MILOS are required for our purposes.

4.1 Building a process model based on a project plan

The development of the CBR system described in the APPLICUS report took place in two phases: First a prototype was developed to test the feasibility of the project and get feedback from the future users. In a second cycle the system was revised, tested, installed and introduced to the users.

For the sake of visualisation we have built our model with the workflow management tool CoMo-Kit (Maurer 1996), which is one of the predecessors of MILOS and allows a graphical representation of the product flows. The following figures are screenshots from this system.[4]

The model starts with an initial process named "CBR Application Development", which can be satisfied by using the method "APPLICUS Method". In a more complex model there could be several alternative methods - chosen with respect to the application domain of the CBR system - to reach the goal of the process.

The structure of "APPLICUS Method" is shown in Figure 2. In this view ellipses denote the subprocesses of the method, while rectangles denote the products which are produced and consumed by the processes. The kind and number of products were not described precisely in the APPLICUS report, so we introduced only a few sample output products per process. Since MILOS allows an object oriented representation of the products, a single product can contain all

[4] Though it supports not all features which MILOS provides, it gave us a good insight into working with a process modeling and workflow management tool.

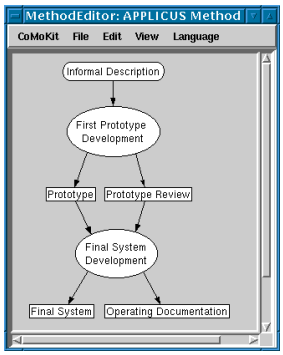

Fig. 2. Structure of the top-level method

necessary components, which otherwise would have to be modeled as single entities. For example the prototype and the final system, which are both instances of the concept "CBR System", consist each of a case base, a similarity measure and a user interface, which in turn each may consist of the implementation and the design documents and so on. A positive side-effect of this packaging of products is a more understandable product-flow diagram.

As stated above, two development phases are enacted. The first one produces a prototypical system along with a documentation of the test results and reviews which describe what is to be done in the second phase. In this phase all components of the system are revised, the final system is tested and installed, and the users are trained. Each phase is modeled as one process, the structures of these two processes are refined in the following.

Figure 3 shows the subprocesses and the product flow in the method for the first phase. Some processes like "System Integration" had no explicit equivalent in the original project plan, but had to be introduced, when a complex product was to be constructed from several parts, as for example the prototype, which consists of the three input products to the "System Integration" process.

The structure of the method for the second phase is also displayed in 3. Note that some processes in this method are very similar to processes in the first phase's method. For example in both methods a "System Integration" process has to be enacted to produce an executable system. In terms of our terminology these processes have the same process type, i.e. products of the same type are required and produces and the same methods are available for the enactment, but the process type has to be defined only once and can be used during the construction of a project plan.

Figure 4 gives a more detailed model for the "Case Acquisition" process which is part of the first phase. There are two subprocesses, one of them describes the actual collection of cases. For this process the APPLICUS report proposes three alternative methods. During the enactment of the process the agent who works on this process can choose one of the methods that seems to be appropriate

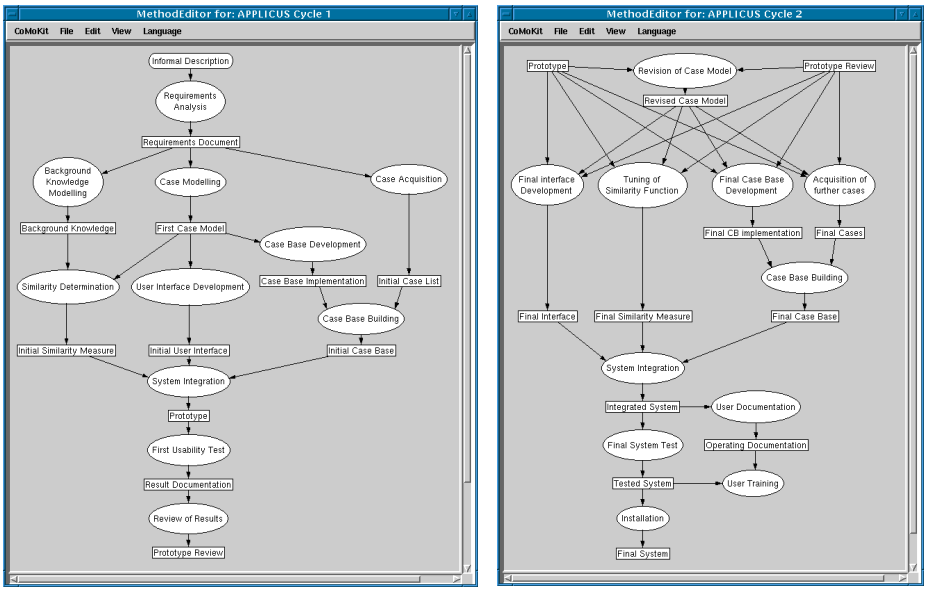

Fig. 3. The two development phases in detail

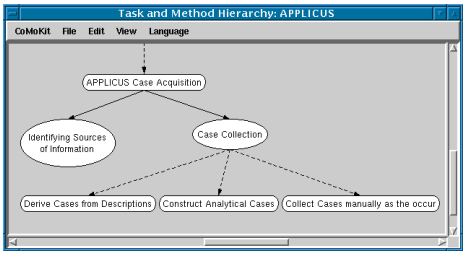

Fig. 4. Alternative methods for one process

for the application domain or the available data. Note that this decision can be delayed until the process is actually to be executed and is not necessary to be made during the project planning. So it should be possible to set up very flexible project plans.

One feature we missed in MILOS was an explicit support for modeling process cycles. These would be useful when modeling processes like the "Tuning of Similarity Function" process in the second phase of our example, where two processes "Tuning" and "Reviewing" could have to be enacted alternating until the review gives a positive result. One possibility to model such a cycle could be to define processes with an unlimited cardinality and to declare proper entry and exit criteria for the single processes to force a execution in the correct sequence.

Many of these features need explicit tool support to be fully usable: Decisions about chosen methods need to be propagated, subprocess delegated, documents traced etc. Some of the main features of CoMo-Kit supporting the execution of such project plans are:

- It provides an agenda for every team member, showing him or her which processes are ready to be started, and updates it according to decisions met by other members. Subprocesses can then be delegated to other agents.
- Decisions which method was chosen for a process are recorded and can be reviewed. Such decisions can even be retracted if parts of the project must be replanned. This leads to invalidating complete products, canceling processes in enactment and informing team members about the new processes which must be enacted now.

4.2 Results

Based on the experiences from this case study we conclude that a process representation language like MILOS is sufficient for our purposes. In particular the following points seem important to us:

- Process types are useful for two purposes:
  1. A project manager who is new to developing CBR systems can be given a set of sample process types and a guide to build project plans from them (e.g. by giving some sample project plans), so he should be able to set up reasonable project plans from the start.
  2. Experiences gained during the enactment of projects can be saved in process types for later reuse thus leading to better plans in future projects. Additionally processes from different plans can be combined to new project plans.
- The definition of different alternative methods for a single process leads to flexible project plans because the details of enactment can be decided at that point in the project when the process has to be started instead of specifying it during planning when parameters may be unknown which may lead to better decisions.

- Moreover, by providing alternative complex methods for one process we can model very different project plans in one model, i.e. depending on the decisions at such choice points the resulting project plans can consist of very different sets of subprocesses. This should guarantee the necessary flexibility to react to problems which could not be foreseen during the initial project planning.
- The object oriented representation of the products leads to very compact process models, because objects which belong together can be packaged into one single product. This avoids having a great number of products as the output of one process.
- Tool support is essential for the enactment of project plans based on these concepts, because in general the plans are not fixed during the project, so a lot of coordination can be necessary during the enactment of such a plan, especially after changing it.

5 Future Work

To reach the goal of having a comprehensive CBR methodology, it is necessary to further generalize appropriate processes types, methods, product types, and resources out of existing plans of CBR projects and thereby build up a larger experience base. Furthermore, already existing workflow management systems that show a high degree of flexibility must be enhanced to support the reuse of process models from the experience base. For this purpose, CBR itself may become useful to support the retrieval of appropriate process models.

Acknowledgment Funding for this work has been provided by the Commission of the European Union (INRECA-II: Information and Knowledge Reengineering for Reasoning from Cases; Esprit contract no. 22196) to which the authors are greatly indebted. The authors also want to thank Sean Breen and Roy Johnston (both IMS) for helpful discussions on CBR methodology.

References

Aamodt, A. and Plaza, E. (1994) *Case-based reasoning: Foundational issues, methodological variations, and system approaches.* AI Communications 7, No. 1, 39-59.

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

Althoff, K.-D. and Wilke, W. (1997). *Potential uses of case-based reasoning in the experience-based construction of software systems.* In: R. Bergmann and W. Wilke (eds.), *Proceedings of the 5th German Workshop in Case-Based Reasoning*, LSA-97-01E, Centre for Learning Systems and Applications (LSA), University of Kaiserslautern.

Bartsch-Sprl, B. (1996a). *Towards a methodology for how to make CBR systems work in practice.* In: H.-D. Burkhard and M. Lenz (eds.), Proceedings of the 4th German Workshop on Case-Based Reasoning (GWCBR-96)- System Development and Evaluation - Informatik-Bericht No. 55, Humboldt Universitt Berlin, 2-9.

Bartsch-Sprl, B. (1996b). *How to introduce case-based reasoning in customer support.* Applicus Deliverable D3, Applicus Consortium.

Bartsch-Sprl, B. (1997). *How to introduce CBR applications in customer support.* In Bergmann and Wilke (Eds.) In: R. Bergmann and W. Wilke (eds.), *Proceedings of the 5th German Workshop in Case-Based Reasoning*, LSA-97-01E, University of Kaiserslautern, pp. 39-48.

Basili, V. R., Caldiera, G., Rombach, H. D. (1994). *The Experience Factory.* In: J. J. Marciniak (ed.), Encyclopedia of Software Engineering, Vol. 1, 469-476, New York: Wiley

Bergmann, R., Wilke, W., Althoff, K.-D., Breen, S., Johnston, R. (1997). *Ingredients for Developing a Case-Based Reasoning Methodology.* In: R. Bergmann and W. Wilke (eds.), *Proceedings of the 5th German Workshop in Case-Based Reasoning*, LSA-97-01E, University of Kaiserslautern, pp. 49-58.

Booch, G. (1994). *Object-oriented analysis and Design.* With Applications. Second Edition. Benjamin/Cummings Publishing Company.

Breen, S. and Johnston, R. (1996). *The scope of the methodology.* Draft 2 of the deliverable D1 of the Inreca-II Esprit Project P22196.

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

Dellen, B., Maurer F., Mnch, J., Verlage, M. (1996). *Enriching Software Process Support by Knowledge-based Techniques.* SFB 501 Internal Report No. 0/96

Gibbs, W. W. (1994): *Softwares chronic crisis.* Scientific American, September, pp. 86-95.

Johnston, R., Breen, S., Manago, M. (1996). *Methodology for developing CBR applications.* InrecaDeliverable D30.

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

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

Manago, M, Bergmann, R. et al. (1994). *CASUEL: A Common Case Representation language*; Esprit project 6332, Deliverable D1. Version 2.

Maurer (1996). *Modeling the knowledge engineering process.* Second Knowledge Engineering Forum. Bericht 1/96 of the SFB 501. University of Kaiserslautern.

Rombach, H. D. and Verlage, M. (1995). *Directions in Software Process Research.* In: M. V. Zelkowitz (ed.), Advances in Computers, Vol. 41, Academic Press, 1-61

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

Wess, S. (1995). *Fallbasiertes Problemlsen in wissensbasierten Systemen zur Entscheidungsuntersttzung und Diagnostik.* Dissertation Universitt Kaiserslautern. Infix Verlag, DISKI 126.

Workflow Management Coalition (1996). *The workflow management coalition specification.* Doc. WFMC-TC-1011, June 1996. http://www.aiai.ed.ac.uk/WfMC/DOCS/glossary/glossary.html