Fraunhofer Einrichtung
Experimentelles
Software Engineering

Using Case-Based Reasoning for Reusing Software Knowledge

Authors:
Carsten Tautz
Dr. Klaus-Dieter Althoff

A shortened version of this report was accepted for the International Conference on Case-Based Reasoning ICCBR-97

IESE-Report No. 004.97/E
Version 1.0
22 August 1997

A publication by Fraunhofer IESE

Fraunhofer IESE is an institute of the Fraunhofer Gesellschaft.
IESE transfers innovative software development techniques, methods and tools into industrial practice, assists companies in building software competencies customized to their needs, and helps them to establish a competetive market position.

Fraunhofer IESE is directed by
Prof. Dr. Dieter Rombach
Sauerwiesen 6
D-67661 Kaiserslautern

Abstract

Reuse of software knowledge is a principle for improving productivity and reliability of software development. To achieve this, reuse must be done systematically. This means that processes for retrieving, reusing, revising, and retaining have to be defined. At the same time organizational issues (such as the establishment of a separate organizational unit responsible for organizational learning) must be considered. In this paper we compare software knowledge reuse models to the CBR cycle of Aamodt and Plaza [AP94a] and show that the approaches are very similar. We suggest to extend the CBR cycle by including organizational issues explicitly and present an organizational "implementation" of CBR. We conclude that CBR is a promising technology for realizing software knowledge reuse if our suggested organizational extensions are considered.

Keywords: Organizational View on CBR, Organizational Learning, Experience Factory, Quality Improvement Paradigm, Software Knowledge Reuse

1 Introduction

> Reuse practice appears to exhibit considerable potential, far more than other ongoing activities, to enhance the software development process and to restructure not only the process of software construction, but also the actual software development departments. To accommodate for and to exploit software reuse, the management and organization of these departments are to be restructured not just locally and in isolation, but also in the context of the entire organization. [ZS95, p. 167]

The reuse of all kinds of software knowledge is one of the main pillars of the approach used for transferring software technologies by our institute. The transfer is based on the Quality Improvement Paradigm (QIP) describing the activities of continuous improvement, the Goal/Question/Metric (GQM) approach for goal-oriented measurement and evaluation, and the Experience Factory (EF) concept describing the organizational structure for implementing a process improvement program which includes an experience base where all knowledge relevant to software development is stored [BCR94].

> QIP, GQM and EF have been applied successfully in several environments. Their successful application within NASA's Software Engineering Laboratory (SEL) has been recognized with the first IEEE/SEI Process Achievement Award. Currently, these approaches are the basis of improvement programs in many companies covering all branches of industry and ranging from small to midsize and large companies. [Rom96, p. 13]

At this point, we are looking for technical support for realizing the experience base. We decided to explore case-based reasoning (CBR) for this purpose. CBR has already been used for software reuse in the past [BE96, FGGH96], however, in this paper we discuss how well the CBR approach and our reuse approach match with respect to the conceptual knowledge level [AA96] and the organizational structure. There are other approaches which use similarity-based retrieval such as AIRS [OHPB92] and faceted classification [PF87], but the approaches do not offer a model on the conceptual knowledge level for using and learning from past experience.

The work described here has been inspired by a paper of Althoff and Wilke [AW97] where they introduce an organizational view[1] on the "CBR cycle" described by [AP94a]. Such an organizational view is more helpful in order to understand the potential uses of CBR in software development and process modeling support. We strongly support this organizational view on CBR by detailing the correspondences between the CBR cycle and the reuse of software knowledge. We will show on an interesting level of detail that CBR is a very promising technology for developing an EF. Here the importance of CBR goes beyond realizing particular EF mechanisms, but supports the realization of an EF on a more general knowledge level [AA96]. In addition, we want to point out that an organizational structure similar to the EF might be of use for other application areas of CBR, too.

The remainder of the paper is structured as follows. The next section gives a short introduction to the reuse of software knowledge. It is followed by a discussion of the concept of the EF which provides an organizational structure for reusing software knowledge (Section 3). Based on this, we compare the reuse process with the CBR task-method decomposition model as introduced by [AP94a], a detailing of the CBR cycle (Section 4). Since the reuse process is very similar to the CBR cycle and organizational issues have long been considered as part of a successful reuse program, it seems natural to extend the interpretation of the CBR cycle to include organizational issues explicitly (Section 5). Such issues have been considered in ongoing project case studies where CBR is used for realizing EF's (Section 6). We will shortly describe our planned activities in this direction and point to issues that are still open (Section 7).

2 Reuse of Software Knowledge

The benefits of software reuse are manifold. Among them are improved productivity, improved reliability, better estimates, and faster time-to-market [SPM94]. Traditionally, the emphasis has been on reusing code. However, reuse does not have to stop there. All kinds of software-related knowledge can be reused,

1 We subsume two aspects under the term "organizational view": activities performed by humans rather than by machines, and the organizational structure, e.g., an organization may be subdivided into several project organization units and a unit responsible for organizational learning.

including products (documents created by a software development project), processes (activities or actions aimed at creating some product) [BR91], and anything else useful for software development, e.g., effort prediction and productivity models or models of the application being implemented.

Reuse can be applied both to the planning of a software development project and to the performance of the project (see Figure 1).

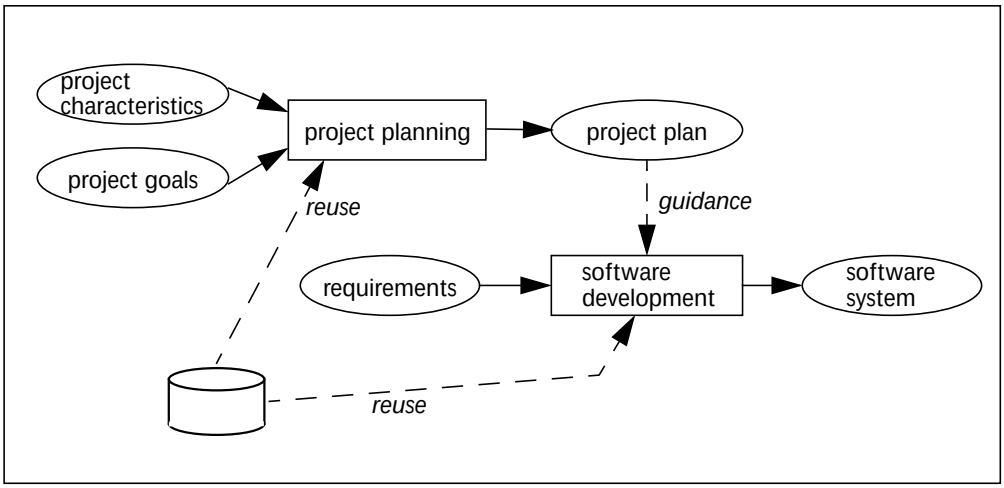


Figure 1: Reuse can be applied both to project planning and performing the project.

2.1 Reuse for Planning a Software Development Project

Before software development can start, the development project has to be planned (with respect to cost, schedule, effort, resource demands, etc.). At the same time, indicators for detecting deviations from the project plan are defined.

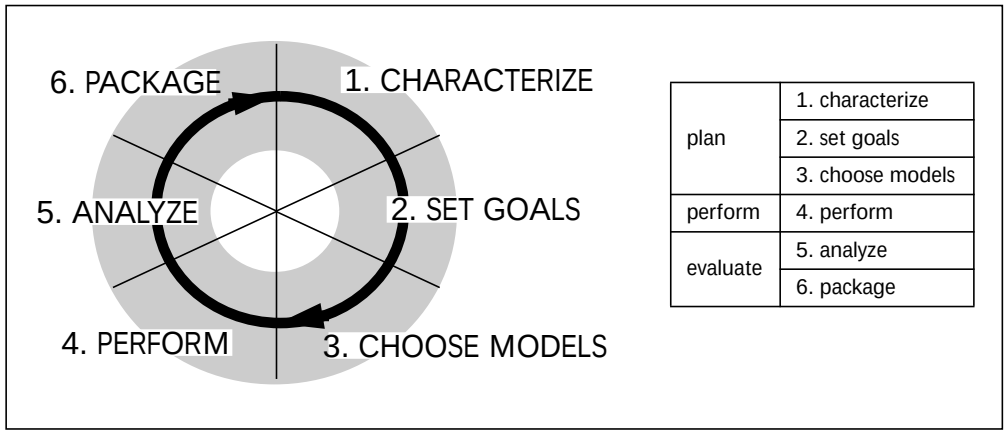
Unlike manufacturing where production is a repetitive task, software development has creative elements. This does not mean, however, that software knowledge reuse cannot be applied. Similar processes (project plans) can be used for similar software systems. Hence, planning a project can be based on plans of projects already terminated. This fact led to the development of the quality improvement paradigm (QIP) [BCR94]. It divides the planning process into three steps, and takes into account that the lessons learned from the performance of the project (were the predicted cost, effort, resource demands, etc. correct?) have to be saved in order to improve the planning of future projects (see Figure 2).

The planning steps are as follows:

- **Characterize.** The environment, in which the software development takes place, is described using the project characteristics as input. The characteriza-

tion of the environment can be thought of as a set of models used in similar projects to the one to be planned. Models may describe possible project plans, the document structure of the software system, expected duration or effort distribution. In addition, suitable measurement goals are part of the characterization.

Figure 2:
The six steps of the Quality Improvement Paradigm (QIP)



- **Set goals.** Based on the characterization and the capabilities of strategic importance to the organization, a set of measurement goals is defined. Measurement goals may be project-specific or of general interest to the organization. A project-specific goal might be the adherence to the predicted effort while the reduction of development cost in the long run is of interest to the organization as a whole. Thus, measurement goals define the successful project and organization performance. Reasonable expectations with respect to the goals are derived from the baseline provided by the characterization step.
- **Choose models.** Depending on the goals set, the right set of models from the characterization step has to be chosen or created. Usually, the chosen models have to be tailored to the specific needs of the project. For instance, the effort distribution model might express the percentage of effort spent in each phase (requirements, design, code, test). If the total effort is known, the effort distribution model can be instantiated by replacing the percentiles with concrete effort numbers. Other models can be constructed using building blocks, e.g., a project plan can be constructed using building blocks for each phase.

The planning phase is followed by the actual performance of the project (step 4 of the QIP). During the performance the project is monitored to make sure that it uses its resources in the best possible way. For example, if the effort model predicts 400 hours for the design phase, and the actual design has taken 350 hours even though it is only to 50% complete, then obviously a problem has occurred for which a solution should be sought. At this point, reuse for performing the project comes in (see next subsection).

After the project has been completed, an evaluation takes place:

- Analyze. At the end of the project, the collected data and problems which occurred (and their solutions) are analyzed. The results of this analysis are lessons learned and improvement suggestions for future projects. For example, if a new technique for writing software requirements was applied in the project, one would like to know whether this requirements technique was helpful in the subsequent phases. Probably, some improvement suggestions regarding the technique will be provided by the project team.
- **Package.** The project feedback has to be consolidated into new, updated and refined models. For instance, if tutorial material exists describing the requirements technique, it can be updated using the lessons learned and the improvement suggestions from the analysis phase. This way, improvement suggestions find their way into future projects. This is resembled by the closed loop in Figure 2.

2.2 Reuse for Performing a Software Development Project

The project plan constructed in the planning phase can be used to guide the performance of the project. Each activity produces deliverables, usually some kind of document. Humans, however, start very seldom from scratch. Typically, something is reused. For example, instead of writing an informal application for a business trip (where we might forget lots of important information for the administration), we use an application form. The same is true for the development of software. Large deliverables can be assembled by reusing old pieces. To exploit reuse to the fullest extend possible it is necessary to provide support, e.g., in the form of an experience base where all reusable objects are stored. This is depicted in Figure 3.

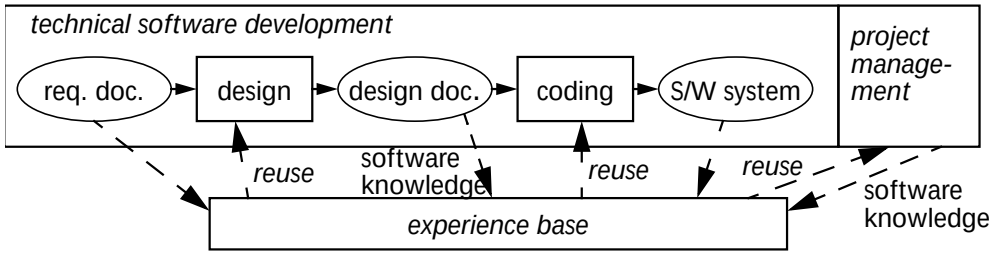


Figure 3: Reuse for software development must be supported by an experience base

Both technical activities and project management (e.g., for replanning) is supported by the experience base. Again, any kind of knowledge can be reused, not just code. For example, if a schedule slippage is detected, the experience base may be consulted for possible actions like reducing the number of reviews (which will, however, increase project risks).

Reuse for the performance of a software development project is typically described by a reuse model such as the one proposed by Basili and Rombach [BR91]. An enhanced version of this model is shown in Figure 4.

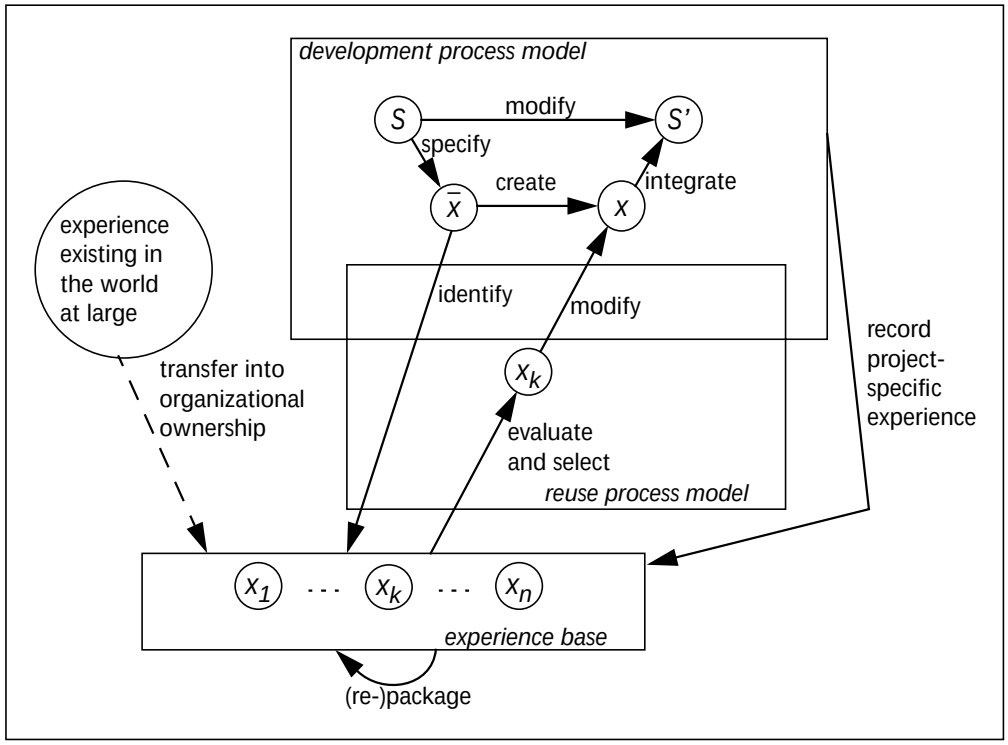


Figure 4: Reuse-oriented software development model based on [BR91]

Given a system *S* where a new object is to be integrated, a specification $\bar{x}$ of an object *x* is defined. The next step is to identify a set of reuse candidates $x_1, \ldots, x_n$. These candidates are evaluated. Eventually the best suited candidate x_k is selected. Depending on how close x_k is to *x*, *x* is created from scratch, or x_k is modified in a suitable way. Then, the new object is integrated into *S* resulting in a new system *S'*. An alternative to modifying x_k in such a way that it satisfies $\bar{x}$ is to modify *S* (and consequently $\bar{x}$) so that *x* can be integrated more easily. The last step of the process is to record the project-specific experience (e.g., *x* along with an evaluation whether *x* was successfully integrated into *S*) in order to improve the reuse support. The integration of the new experience into the experience base is referred to as "packaging". Integrating new experience may require to restructure parts of the experience base. This is referred to as "repackaging".

The experience base may not only be populated by experience gained by the software development organization itself, but also through transferring existing experience from outside the organization. Such experience can be found, e.g., in literature.

3 An Organizational Structure for Reusing Software Knowledge

For large-scale reuse to work, a distinct support organization (separate from the software development project organizations) must be established [Gri94]. Such a support organization must carefully package, document and certify (where applicable) software artifacts. Klotz puts it this way: "It is not enough that each individual is creative and capable of learning – the organization as a whole must also be capable of learning" [Klo94]. The reason is that development project organizations have different aims than the organization as a whole.

While the project organization wants to make sure its deliverables fulfill their requirements under given constraints (such as schedule and cost constraints), the organization as a whole wants to avoid "reinventing wheels" and making a mistake twice. Due to these different aims, a project organization is not interested in providing knowledge, because this involves extra effort on its side. Consequently, the organization as a whole must be represented by a distinct organization. Basili et al. [BCR94] call this organization the *Experience Factory* (EF). The term includes the experience base already introduced as well as people retrieving, storing and maintaining software knowledge (see Figure 5).

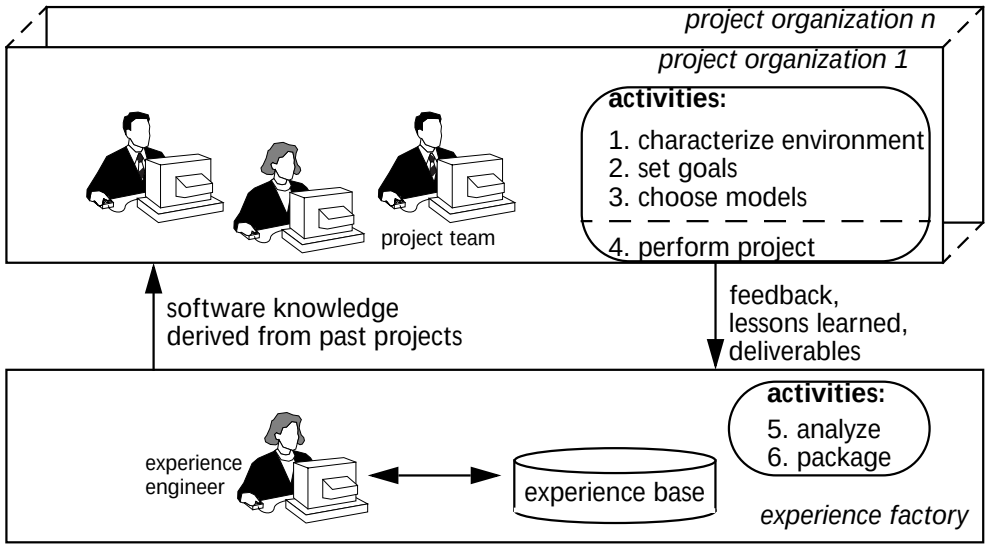


Figure 5: There needs to be a distinct organization for managing software experience: the experience factory

In [BC95] Basili and Caldiera associate with each organizational unit the steps of the QIP. Both organizational units participate in all steps,;however, for each step one unit takes over the responsibility: steps 1-4 (planning and performing a project) are the responsibility of the project organization while steps 5 and 6 (analyzing and packaging) are the responsibility of the EF.

In the next section, we will see that this organizational concept can be used together with the CBR approach for reusing software knowledge.

4 Comparison of the CBR Approach and the Reuse Approaches

For comparing both the QIP and the reuse-oriented software development model with the CBR approach, we use the task-method decomposition model proposed by Aamodt and Plaza [AP94a] for four main reasons. First the task-method decomposition model bases on Aamodt and Plaza's CBR cycle which does not differ from other CBR cycles described in the literature with respect to its basic contents [Kol93]. Second the combination of CBR and knowledge level analysis [AA96, Aam95, AP94b] is very helpful for our problem at hand (namely to find a technological basis for an EF). Third the task-method decomposition model has been successfully used within the evaluation of the Inreca CBR system as a means of analysis and comparison [Alt96, AA96]. Fourth the CBR cycle of Aamodt and Plaza appears to be widely accepted in the literature [Wes95, Alt96, Ehr96, BW96, AABM95, VMB96, LBP+96, etc.].

We now shortly introduce the four top-level (sub)tasks "retrieve", "reuse", "revise", and "retain" of the CBR task-method decomposition model.

- **Retrieve** is decomposed into "identify features" (identifies relevant set of problem descriptors), "search" (returns a set of cases which might be similar to the new case), "initially match" (chooses a set of cases that are similar to the new case), and "select" (chooses the most suitable case from the set). "Search" and "initially match" can be interpreted as a combined task, because the identification of similar cases involves a goal-oriented search, i.e., the search for similar cases. Combining "search" and "initially match" allows for an efficient implementation, since the task "search" does not need to return all possibly similar cases.
- **Reuse** is decomposed into "copy" (takes the selected case as the basis) and "adapt" (transforms the solution of the selected case to a solution for the new case).
- **Revise** is decomposed into "evaluate solution" (evaluates the success of the solution constructed in the reuse task) and "repair fault" (detects defects in the current solution and generates or retrieves explanations for them).
- **Retain** is decomposed into "extract" (identifies the information which must be stored), "index" (identifies the types of indexes needed for future retrieval

as well as the structure of the search space), and "integrate" (updates the knowledge base with the parts of the actual experience likely to be useful in future problem solving).

4.1 Comparison of the CBR Approach and the QIP

The steps of the QIP compare to the CBR approach as follows:

- **QIP step 1 (characterize).** In this step project characteristics are used to retrieve a set of relevant models. This corresponds to "identify features", "search" and "initially match" of the task "retrieve". Since no best candidate is selected at this point, the "select" task is not part of QIP step 1.
- **QIP step 2 (set goals).** In this step measurement goals for the software development projects are chosen. This can be interpreted as "selecting goal cases"[1], i.e., the responsible manager looks for strategic improvement goals and/or combinations of them (e.g., "reduce the software development effort by 30%"). Therefore, QIP step 2 corresponds to the task "select" with respect to measurement goals (a subset of all relevant models returned in QIP step 1).
- **QIP step 3 (choose models).** Here, the rest of the relevant models (describing products, processes, expectations for the goals) is selected in accordance with the goals selected in QIP step 2. Therefore, QIP step 3 corresponds to the task "select" with respect to everything but measurement goals. In addition, a project plan is assembled, i.e., the relevant models are integrated. This typically requires modification of the retrieved models. Hence, QIP step 3 corresponds also to the "reuse" task.
- **QIP step 4 (perform).** During this step the project is performed. Even though the CBR process implies, that the solution is applied between the tasks "reuse" and "revise", it does not provide an explicit task for this. Therefore, there is no correspondence to QIP step 4. One of the reasons is that the project is usually not executed by the people responsible for running the CBR system. In terms of the EF concept, CBR specialists work in the EF, while the project is performed by the people in the project organization. Nevertheless, a model describing the CBR approach should also consider organizational issues, meaning that it should include the case application explicitly.
- **QIP step 5 (analyze).** In this step the project performance is analyzed. Lessons learned and improvement suggestions with respect to the knowledge

1 Another correspondence here is between defining similarity (as a special kind of general knowledge usually done by a domain expert) in CBR and defining goals (as concrete improvement goals derived from strategic business goals usually determined by the responsible manager) in QIP [AW97]. In both cases the resulting definition guides the later selection process and, thus, is of crucial importance for the whole procedure.

applied are written. Hence, QIP step 5 corresponds to the task “revise”, but also to “extract” of the task “retain”, because the output of QIP step 5 is exactly what has to be packaged in the next step.

- **QIP step 6 (package).** Here, the lessons learned and improvement suggestions are integrated into the experience base. This includes formalizing the experience as well as restructuring the experience base. Therefore, QIP step 6 corresponds to “index” and “integrate” of the task “retain”.

Table 1 summarizes the correspondences.

Table 1: Summary of the comparison between the CBR cycle and the QIP

CBR		QIP 1	QIP 2	QIP 3	QIP 4	QIP 5	QIP 6
retrieve	identify features	X					
	search	X					
	initially match	X					
	select		X (goals)	X (other)			
reuse	copy			X			
	adapt			X			
revise	evaluate solution					X	
	repair fault					X	
retain	extract					X	
	index						X
	integrate						X

4.2 Comparison of the CBR Approach and the Reuse-Oriented Software Development Model

Reuse within a software development project basically consists of seven steps which are related to the CBR approach as follows (see Figure 4 on page 6):

- **Specify.** In this first step the need for a new object is recognized, and the needed object is specified. This step corresponds to “identify features” of the task “retrieve”.
- **Identify.** An initial set of possible candidates is identified. This corresponds to “search” and “initially match” of the task “retrieve”.
- **Evaluate and select.** The most suitable candidate is selected. This step corresponds to “select” of the task “retrieve”.
- **Modify or create.** Either the most suitable candidate is modified to fulfill the initial specification or a new object is built from scratch. Both correspond to

the task “reuse”. However, the creation of a completely new case is only indirectly covered by the CBR cycle[1].

- **Integrate.** The new object is integrated into its surrounding system. Again, as the CBR cycle does not include the application of cases explicitly, this step has no correspondence. In terms of the EF concept, integration work is done in the project organization. Usually, the EF people are not involved in this step.
- **Record experience.** The success of using the new object is evaluated. This corresponds to the task “revise” as well as to “extract” of the task “retain”.
- **(Re-)package.** This involves the integration of the experience into the experience base. Hence, this step corresponds to “index” and “integrate” of the task “retain”.

Table 2 summarizes the results of this comparison.

Table 2: Summary of the comparison between the CBR cycle and the reuse-oriented software development model

CBR		specify	identify	evaluate and select	modify or create	integrate	record experience	(re-) package
retrieve	identify features	X						
	search		X					
	initially match		X					
	select			X				
reuse	copy				X			
	adapt				X			
revise	evaluate solution						X	
	repair fault						X	
retain	extract						X	
	index							X
	integrate							X

The result of the two comparisons carried out in this section is twofold. First we have shown that there are many commonalities between the CBR cycle and the two software knowledge reuse models. This can be viewed as a successfully passed plausibility test for CBR as a candidate technology for realizing software knowledge reuse. Second in both comparisons there is one basic correspon-

1 For instance, selecting some kind of default case that contains an empty solution, i.e., no solution, such that the solution of the new case must be created completely by hand. This would be the extreme case of a 100% adaptation.

dence missing. For the same reason already seen with the perform step of the QIP, the CBR cycle does not include any explicit correspondence for the "integrate" step of the reuse model. The consideration of the underlying reasons leads to the organizational issues of using CBR systems.

5 Organizational "Implementation" of Case-Based Reasoning

Since we want to use CBR technology for realizing software knowledge reuse in industrial environments we must cope with organizational aspects. What we now suggest is to extend the interpretation of the CBR cycle to *explicitly* include human activities as, e.g., it is done by an EF. There exists quite a natural extension because CBR was originally developed for being both a methodology for building intelligent systems and a model of people [Kol93, Sch82, Str92]. For instance, Janet Kolodner states it as follows: "CBR is both, and explorations in CBR have been of both the ways people use cases to solve problems and the ways we can make machines use them" [Kol93, p. 27]. Thus, CBR is a rather general framework for describing problem solving and learning based on experiences. We would like to call such an extended CBR cycle, which also explicitly includes human activities, a "CBR Process Model". Based on our analysis above, it is obvious that the EF concept and the CBR process model have many correspondences and analogies. An EF is an infrastructure for the systematic reuse of all kinds of experiences during software development. Compared to this the CBR process model is more general because it models experience based reasoning and acting for any kind of problem (e.g., business processes). However, we believe that the organizational structure introduced in this paper can also be used for application areas of CBR[1] other than software knowledge reuse and, thus, is of further relevance for the CBR community.

We shortly want to motivate this with the following example where QIP is used for the build-up of (any kind of) knowledge[2] according to the following general procedure:

1 It will be an interesting research task to analyze the commonalities and differences with other approaches to the organizational implementation of CBR like [Bar97] or [BWA+97].
2 This means processing of knowledge from the perspective of an organizational unit, i.e., the built-up knowledge can be processed by humans, computers, or both.

- **QIP step 1 (characterize).** Analyze the status-quo: What do you already know? Which knowledge areas are weak, which are strong?
- **QIP step 2 (set goals).** Decide in which areas you want to build up your knowledge and to what extent.
- **QIP step 3 (choose models).** Plan how to reach your goals, i.e., choose a process which describes how to proceed.
- **QIP step 4 (perform).** Proceed according to your plan.
- **QIP step 5 (analyze).** Check if you have reached your goals. If not, why not?
- **QIP step 6 (package).** Integrate the new knowledge into the CBR system and start the next cycle.

In [Bas95] Basili compares the QIP to several other quality improvement approaches. For some of them CBR has been applied at least partly [AILM97].

As a consequence, we would like to suggest to join research efforts on the practical realization of the CBR process model and the EF within industrial and other business environments, because we believe that there are many helpful commonalities and supplementations.

6 Current Status

Our current work bases on the general framework described in [AA96, ABS96], and [Alt96]. [BAM97] describes the analysis of a collection of experiences with respect to the application of CBR and the development of CBR systems, gathered by several questionnaires[1]. These experiences have been made reusable by means of CBR technology[2], i.e., each questionnaire has been represented as a structured case. For instance, for a concrete application problem it can be searched for a similar problem or a similar CBR tool (i.e., a CBR tool that in principle can deal with the problem).

1 50 questionnaires have been collected from 14 countries.

2 We used the CBR-Works system from tecInno (Kaiserslautern, Germany). The cases have been represented (and are available) in the Casuel common case and knowledge representation format [MBC+94]. It is planned to make this system accessible via WWW based on the CBR-Works web server.

There are a number of ongoing industrial, research and in-house projects at our institute in the context of EF development. Currently the main effort here has been on identifying and capturing experiences (data, information, and knowledge) for potential reuse. Up to now CBR has not been used for their realization.

7 Summary and Outlook

We showed that the CBR approach as described by Aamodt and Plaza [AP94a] is similar to the models used in the area of software engineering for reusing knowledge both on the project planning level and the project performance level. In the second part, we introduced an organizational structure for reusing software knowledge.
We plan to use a CBR system for software knowledge reuse and evaluate its benefits. For this purpose a common model incorporating the CBR process model and the reuse process models is being developed. Such a model will describe CBR-based reuse of software knowledge in sufficient detail. The organizational model introduced in this paper can be used as a basis for further evaluation with respect to useful support by using CBR systems.

8 Acknowledgements

We would like to thank Günther Ruhe and Frank Bomarius for the fruitful discussions. Jürgen Münch and Martin Verlage raised many issues we had to think about. Thanks also to Christiane Gresse and the anonymous referees for reviewing the submitted paper. They all have contributed to this final version in a constructive way.

9 References

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

[AABM95] K.-D. Althoff, E. Auriol, R. Barletta, and M. Manago. *A Review of Industrial Case-Based Reasoning Tools*. AI Intelligence. Oxford (UK), 1995.

[Aam95] A. Aamodt. Integration of knowledge acquisition and machine learning. In Y. Kodratoff and G. Tecuci, editors, *Knowledge Acquisition and Learning by Experience: The Role of Case-Specific Knowledge*, pages 197–245. Kluwer Academic Publishers, 1995.

[ABS96] Klaus-Dieter Althoff and Brigitte Bartsch-Spörl. Decision support for case-based applications. *Wirtschaftsinformatik*, 38(1):8–16, February 1996.

[AILM97] D. A. Adams, C. Irgens, B. Lees, and E. MacArthur. Using case outcome to integrate customer feedback into the quality function deployment process. In R. Bergmann and W. Wilke, editors, *Proceedings of the Fifth German Workshop on Case-Based Reasoning*, LSA-97-01E, pages 1–9. Centre for Learning Systems and Applications, University of Kaiserslautern, March 1997.

[Alt96] Klaus-Dieter Althoff. Evaluating case-based reasoning systems: The Inreca case study. Postdoctoral thesis, University of Kaiserslautern, July 1996. (submitted).

[AP94a] Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AICOM*, 7(1):39–59, March 1994.

[AP94b] E. Armengol and E. Plaza. A knowledge level model of case-based reasoning. In S. Wess, K.-D. Althoff, and M. M. Richter, editors, *Topics in Case-Based Reasoning*, pages 53–64. Springer-Verlag, 1994.

[AW97] Klaus-Dieter Althoff and Wolfgang Wilke. Potential uses of case-based reasoning in experienced based construction of software systems and business process support. In R. Bergmann and W. Wilke, editors, *Proceedings of the Fifth German Workshop on Case-Based*

Reasoning, LSA-97-01E, pages 31–38. Centre for Learning Systems and Applications, University of Kaiserslautern, March 1997.

[Bas95] Victor R. Basili. The Experience Factory and its relationship to other quality approaches. In Marvin V. Zelkowitz, editor, *Advances in Computers, vol. 41*, pages 65–82. Academic Press, 1995.

[BC95] Victor R. Basili and Gianluigi Caldiera. Software quality management: Strategy and practice. *Sloan Management Review*, pages 55–64, Fall 1995.

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

[BE96] R. Bergmann and U. Eisenecker. Case-based reasoning for supporting reuse of object-oriented software: A case study (in German). In M. M. Richter and F. Maurer, editors, *Expert Systems 95*, pages 152–169. infix Verlag, 1996.

[BR91] Victor R. Basili and H. Dieter Rombach. Support for comprehensive reuse. *IEEE Software Engineering Journal*, 6(5):303–316, September 1991.

[Bar97] Brigitte Bartsch-Spörl. How to introduce cbr applications in customer support. In R. Bergmann and W. Wilke, editors, *Proceedings of the Fifth German Workshop on Case-Based Reasoning*, LSA-97-01E, pages 39–48. Centre for Learning Systems and Applications, University of Kaiserslautern, March 1997.

[BAM97] Brigitte Bartsch-Spörl, Klaus-Dieter Althoff, and Alexandre Meissonnier. Learning from and reasoning about case-based reasoning systems. In *Proceedings of the Fourth German Conference on Knowledge-Based Systems (XPS97)*, March 1997.

[BW96] Brigitte Bartsch-Spörl and Stefan Wess (eds.). Special issue on case-based reasoning (in German). *Künstliche Intelligenz*, 1, 1996.

[BWA+97] R. Bergmann, W. Wilke, K.-D. Althoff, R. Johnston, and S. Breen. Ingredients for developing a case-based reasoning methodology. In R. Bergmann and W. Wilke, editors, *Proceedings of the Fifth German Workshop on Case-Based Reasoning*, LSA-97-01E, pages 49–58. Centre for Learning Systems and Applications, University of Kaiserslautern, March 1997.

[Ehr96] Dieter Ehrenberg (ed.). Special issue on case-based decision support

(in German). *Wirtschaftsinformatik*, 38(1), 1996.

[FGGH96] C. Fernández-Chamiso, P. A. Gozales-Cálero, M. Gómez-Albarrán, and L. Hernández-Yanez. Supporting object reuse through case-based reasoning. In I. Smith and B. Faltings, editors, *Advances in Case-Based Reasoning*, pages 135–149. Springer-Verlag, 1996.

[Gri94] Martin L. Griss. Software reuse experience at Hewlett-Packard. In *Proceedings of the Sixteenth International Conference on Software Engineering*, page 270. IEEE Computer Society Press, May 1994.

[Klo94] Ulrich Klotz. Out of the economic crisis with organizations capable of learning (in German). *Ergonomie und Informatik*, pages 3–12, July 1994.

[Kol93] Janet Kolodner. *Case-Based Reasoning*. Morgan Kaufmann, 1993.

[LBP+96] Mario Lenz, Hans-Dieter Burkhard, Petra Pirk, Eric Auriol, and Michel Manago. CBR for diagnosis and decision support. *AI Communications*, 9(3):138–146, 1996.

[MBC+94] M. Manago, R. Bergmann, N. Conruyt, R. Traphöner, J. Pasley, J. LeRenard, F. Maurer, S. Wess, K.-D. Althoff, and S. Dumont. Casuel: A common case representation language. Technical Report Deliverable D1, Version 2.01, Esprit Project Inreca (P6322), 1994.

[OHPB92] Eduardo Ostertag, James Hendler, Rubén Prieto-Díaz, and Christine Braun. Computing similarity in a reuse library system: An AI-based approach. *ACM Transactions on Software Engineering and Methodology*, 1(3):205–228, July 1992.

[PF87] Rubén Prieto-Díaz and Peter Freeman. Classifying software for reusability. *IEEE Software*, 4(1):6–16, January 1987.

[Rom96] H. Dieter Rombach. New institute for applied software engineering research. *Software Process Newsletter*, pages 12–14, Fall 1996. No. 7.

[Sch82] Roger C. Schank. *Dynamic Memory: A Theory of Learning in Computers and People*. Cambridge University Press, 1982.

[SPM94] Wilhelm Schäfer, Rubén Prieto-Díaz, and Masao Matsumoto. *Software Reusability*. Ellis Horwood, 1994.

[Str92] Gerhard Strube. The role of cognitive science in knowledge engineering. In F. Schmalhofer, G. Strube, and Th. Wetter, editors, *Con-*

temporary Knowledge Engineering and Cognition, pages 161–175. Springer-Verlag, 1992.

[VMB96] Manuela Veloso, Hector Muñoz-Avila, and Ralph Bergmann. Case-based planning: Selected methods and systems. *AI Communications*, 9(3):128–137, 1996.

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

[ZS95] Mansour Zand and Mansur Samadzadeh. Software reuse: Current status and trends. *Journal of Systems and Software*, 30(3):167–170, September 1995.

Document Information

Title:	Using Case-Based Reasoning for Reusing Software Knowledge
Date:	August 22, 1997
Report:	IESE-004.97/E
Status:	Final
Distribution:	Public