Reducing the Case Acquisition and Maintenance Bottleneck with User-Feedback-Driven Case Base Maintenance

Markus Nick

Fraunhofer Institute for Experimental Software Engineering (IESE)
Fraunhoferplatz 1, 67663 Kaiserslautern
markus.nick@iese.fraunhofer.de

Abstract

Current case acquisition and case base maintenance techniques implement quality assurance for cases through reviews or by analyzing case properties before making a case available for retrieval. Since reviews of cases with much textual data, in particular, cannot be fully automated, this is done by a maintenance team. With limited resources, this maintenance team becomes a bottleneck. To reduce this bottleneck, our idea is to move parts of the case acquisition and maintenance tasks to the user. With this, the maintenance team needs to do case maintenance only for cases with an 'out of range' quality. Obviously, this idea requires changes to the CBR process and system design. This is described by our *experience feedback loop* design concept, which is presented in this paper. This design concept contains a user-feedback-driven case base maintenance technique as its core element. This design concept has been validated positively in real-world projects for intra-organizational CBR systems with tight integration into business processes and into the respective tools.

Introduction

Users expect a certain level of quality in the cases that they retrieve from a CBR system. The perceived usefulness of the retrieved cases from the users' view depends on the quality of the different knowledge containers. [Althoff et al. 2000] provided a cause-effect model for usefulness deficiencies. For optimal usefulness, the most relevant cases have to be retrieved, which depends on the similarity models, and the relevant cases have to be correct and understandable, which depends on the cases themselves, etc. Here, we focus on the quality of the cases, i.e., on case base maintenance problems. Particularly cases with much textual information –like lessons learned or mainly textually described problem-solution pairs– require editorial work in some way, i.e., they have to be entered into the system and reviewed regarding their quality [Basili et al. 1994, Weber et al. 2001].

CBR maintenance research has already developed techniques for supporting or automating the review of cases. Several techniques address the automatic detection of quality problems using technical properties of cases and case base (e.g., redundancy, coherence, consistency) [e.g., Reinartz et al. 2001, Racine and Yang 2001]. However, these can hardly detect mistakes in the contents, which requires reviews by humans. The collaborative maintenance technique of [Ferrario and Smyth 2001] automatically organizes such reviews of new cases by a group of reviewers.

Obviously, with evolving case knowledge, review process and maintenance team become a *case acquisition and maintenance bottleneck* - if the volume of the respective activities exceeds the available resources.

The goal of our approach is to reduce this bottleneck. The idea is to move parts of the case acquisition and maintenance tasks to the user in his business process. Instead of guaranteeing that all cases have a certain minimal quality level, we require of the user that (1) he is able to enter and update cases and (2) is able to better judge the usefulness of cases using the additional validity information, and (3) is willing to give feedback that can be used for automatically maintaining this case validity information. Thus, the maintenance team has to be triggered only if the quality/validity of a case is out of an acceptable range. This validity describes (1) how often an experience/case has or has not "worked" (i.e., its applicability) and (2) how well it performed (i.e., its effects).

This is realized with a user-feedback-driven case base maintenance strategy by our *experience feedback loop*, a design concept for CBR systems that consists of (1) a novel process model for CBR that enables the automatic maintenance of case validity information using feedback from the user during the CBR process and (2) a technique for automatically diagnosing the need for maintenance using this validity information.

In case studies in nine real-world projects, the feasibility of our approach has been evaluated and validated positively for tightly integrated intra-organizational CBR technology-based experience management systems.

This paper is structured as follows: First, the experience feedback loop design concept is presented. This includes process, validity function, impact on a CBR system's knowledge representation and validity-related operations on the case base, and an example. Second, the validity-based diagnosis for maintenance need is described. Third, the evaluation of the approach is presented. Finally, the paper is summarized and some conclusions are drawn.

Experience Feedback Loop

Process

The experience feedback loop's process describes a CBR process that operationalizes standard CBR cycles [Aamodt and Plaza 1994, Kolodner 1993, Reinartz et al. 2001] with a so-called "case evaluation/revision in the real world using feedback" – in the terms of Aamodt & Plaza or Kolodner. For this purpose, the CBR process is embedded into a business process and has feedback steps and validity monitoring. A separate organizational unit is responsible for the maintenance of the CBR system and the knowledge stored in it. We call this unit *experience factory* [Basili et al. 1994]. The combination of the experience factory approach with the CBR cycle of [Aamodt and Plaza 1994] by [Tautz and Althoff 1997] is the basis of our model. The experience feedback loop runs as follows (Figure 1):

Business process side: Using context information from the business process and further user input (e.g., problem specification) as a query specification, the case base is searched and the best-matching case is selected for reuse. If necessary, the user modifies this selected case before its application. If no case has been found, a new case is created (like a hypothesis), which is then treated as the selected or modified case. Then the user gives a priori feedback about his expectations regarding the effects of the case on his current task in the business process. Then the user applies the case in the business process. If the case does not work for the task of the user, the user either modifies the case, which creates a new case (subsequent modifications do not create further new cases), or he restarts with «search and select», or he gives up and ends the application with negative a posteriori feedback. If the user could apply the case successfully, appropriate feedback is given and actual effects are stated.

Experience factory side: The validity monitoring component collects the feedback and automatically calculates the validity using the validity function explained below. The validity information is used for enhancing search result presentations and for automatically triggering maintenance for cases with insufficient validity.

User-Feedback-based Case Validity Function

Using the collected feedback for every reuse attempt, the validity is calculated for each case. This case validity is defined as

validity(case C) := < applicability score of C,
effects statistics of C >

The *applicability* describes if a case "worked" in a reuse attempt. For each reuse attempt, the user specifies whether the reused case was «applicable» or «not applicable». If a new case has been created, the user specifies whether the trial of the case "worked" (i.e., «applicable») or not (i.e., «not applicable»).

For each case, the applicability information from all related reuse attempts is aggregated:

applicability score(case C) :=
[#reuse attempts where C was applicable,
#reuse attempts where C was not applicable]

Obviously, the value range for the applicability and the applicability score function are application-independent.

The *effects* describe how well a case "worked" in a reuse attempt. For each reuse attempt, the *expected effects* can be collected before the application and the *actual effects* can be collected after the application. The interpretation of the effects obviously also depends on the applicability result for the reuse attempt because the effects are positive for "applicable" and negative for "not applicable". Furthermore, expected and actual effects can differ. There are two reasons: (1) The context changed during the reuse or (2) the case appears in a way that leads to misunderstanding and wrong expectations. The latter case impacts validity negatively because expectations that are too low or too high can lead to the selection of cases other than the one that would be optimal for the current work context.

The result of the aggregation of the effects for a case are the so-called *effects statistics*. Obviously, their definition is application-specific – like the definition of the effects themselves.

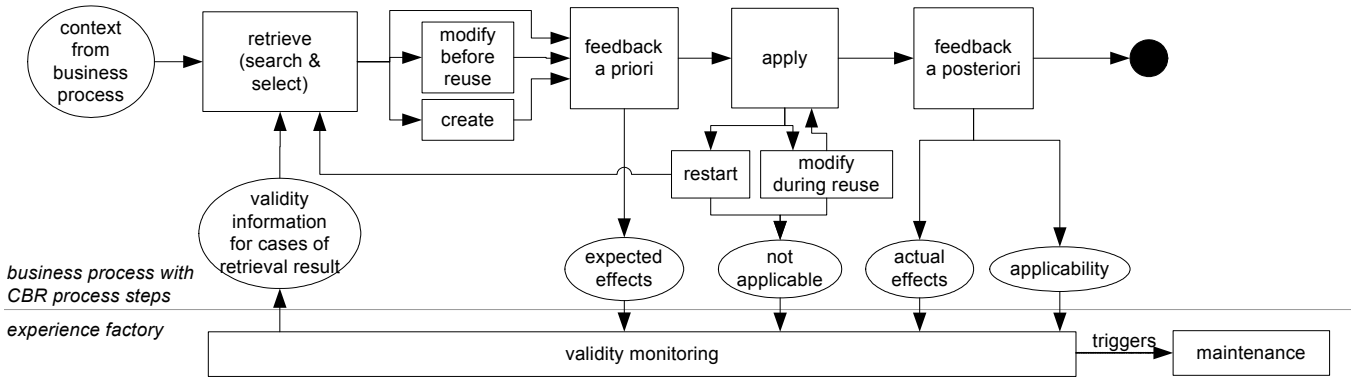
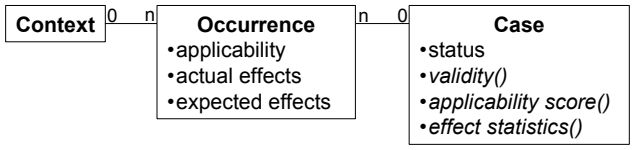


Figure 1: The experience feedback loop's process is a CBR process with feedback steps and validity monitoring, where cases are mainly entered and updated by the users during their work in their business processes.

Impact on Knowledge Representation Model and Knowledge Processing

Collecting such feedback has an impact on the design of the CBR system regarding the knowledge representation model and case base operations.

Knowledge representation model: To allow computation of the validity function by the system, the applicability and effects information has to be stored in the case base. For this purpose, a so-called occurrence entity is added for each case entity. Occurrences reflect actual reuse attempts that succeeded or failed – in a specific context. In an occurrence, the applicability and effects information for a reuse attempt is stored. Thus, –if a context is stored as well– the occurrence qualifies the relation between case and context. The respective knowledge model pattern is as follows:



There are two mandatory attributes: The occurrence's «applicability» as specified above, and a «status» attribute for the case. Before being applied, the status is «unconfirmed». After application, the status becomes «confirmed». When the case becomes obsolete, the status is changed to «obsolete». The retrieval searches only in «confirmed» cases and uses the validity-related functions to enhance the search result presentation.

Case base operations for validity monitoring: The applicability score *[#applicable, #not applicable]* is calculated as the cardinality of the case-occurrence relation with respect to «applicable» and «not applicable». For the effects statistics, the calculation is application-specific – as stated above. For this automatic validity calculation purpose, the validity monitoring component must store all feedback for the current reuse attempt in its occurrence instance.

For each reuse attempt, an occurrence instance is created immediately after the selection of the case. If no case is selected for reuse, a new case is created and the new occurrence is linked to it. This is necessary for storing a priori feedback. Furthermore, this allows collecting and storing feedback for all CBR process steps after the retrieval step.

The modification of cases is divided into semantic and syntactic changes. Only semantic changes change the meaning of a case. Thus, syntactic changes can be made at any time without impacting the validity values.

Whenever a confirmed case is to be modified semantically, a new «unconfirmed» case is created and the link from the occurrence for the current reuse attempt is moved to the new case. If the experience factory itself triggers a semantic change (i.e., without relation to reuse attempts), the new case's status must be set to «confirmed» to include it in future retrievals.

Example from Project indiGo

The example from the Project indiGo (www.indigo.fraunhofer.de) has the following setting: A company has an experience management system for project management experiences focusing on risks, observations, and problem-solution pairs about customer interaction. For example, experiences on how timely a partner delivers the required input are used for optimizing project plans. These experiences are stored as cases in a CBR system, which allows similarity-based retrieval of experiences considering situation and project context characteristics. The experience base is queried before project acquisition talks, during project planning, and before project meetings. The experiences give insights about the specific issues to consider with a specific customer or a specific domain. Experiences are stored after the respective events (i.e., acquisition talks and meetings) or at the end of the respective phases.

With an experience feedback loop, the usage of an experience is tracked and feedback is collected for every application of an experience. For each reuse of a risk experience, an occurrence links the risk to the project where it was applied. This (1) describes the context of the risk and (2) allows determining the applicability score as number of project contexts where the risk has been applicable and where it failed (i.e., not applicable).

For an example risk "needed data not available on time", this is seen by the user over time as shown in Figure 2. The risk is first encountered in Project X1. After Project X1, its applicability score is [1:0]; after Project X2, it is [2:0]; and after Project X3, it is [3:0]. Project Y1 is conducted after the change to the business processes. Thus, the risk does not apply and the applicability score becomes [3:1]. For Project Z, the project planner sees that he has to take care because the risk was not applicable all the time. He checks the reuse history of the risk and notices that it was not applicable the last time, which might mean that company A has improved. However, he decides to still add some slack time for data from A in order to be on the safe side.

The example shows that the user is better informed in his decision making and can use the validity information to judge how much he can trust an experience/case.

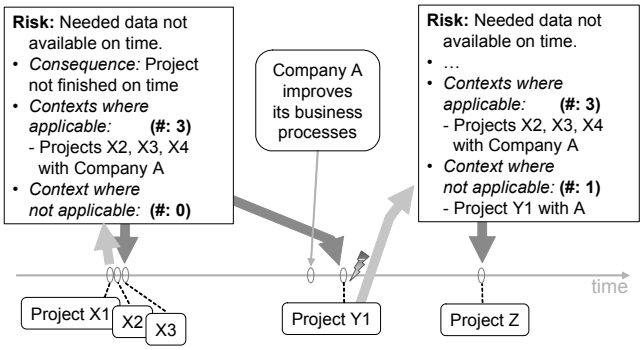


Figure 2: Experience reuse and maintenance with an experience feedback loop by the example of the Project indiGo

Using Case Validity To Diagnose Maintenance Need

Using the validity function, the need for maintenance can be diagnosed for the cases in the case base.

The first diagnosis step can be fully automated as follows: A tuple <case, context, applicability, actual effects> is compared pair-wise for case-context pairs. For each element of the tuple, the result is «same» or «different». «same» refers to identical case attribute values. For textual attributes, this would mean that the text has the same meaning (how to do this technically for text is beyond the scope of this paper). For this comparison of tuple pairs, a complete table has been compiled (Table 1). For each possible comparison result, there is one row in the table that specifies the top-level diagnosis and refers to the respective maintenance policy that has to be applied.

There are four validity anomalies that can be diagnosed:

- A *context applicability inconsistency* is defined as the same case having been applied in the same context and situation with different results regarding applicability.
- An *effect inconsistency* is defined as the same case having been applied in the same context and situation with the same applicability and with different effects.
- When variations in the case (e.g., slightly different reactions) lead to a slightly different effect, this is referred to as *effect variation*. An effect variation also means that case and context do not capture in sufficient detail what led to the effect variation.
- An *alternative in a specific context* is a case that is different regarding its contents, but has the same applicability and same effects for a specific context. This can be relaxed to having "almost the same effects" (however, this would introduce an overlap with the definition of "effect variation").

The maintenance policies required for Table 1 are shown in Figure 3. Each has a condition that activates the policy and

Table 1: Combinations of applicability and effects for groups of case-context pairs, respective diagnosis and maintenance policies.

observation/ indication[a] (regarding >=2 case-context pairs)				***diagnosis***	***OK?***	**maintenance policy**
cases	**contexts**	**applicability**	**actual effects**			
s	s	d	d	***context applicability inconsistency***	*no*	**MP-1** "Detect context applicability inconsistencies."
s	s	d	s	***context applicability inconsistency*** *or typo[b]*	*no*	**MP-1** "Detect context applicability inconsistencies."
s	d	s[c]	*	*more generality (if all applicable)*	*ok*	
				more negative cases (if all not applicable)[c]	*no*	**MP-3** "Too many negative contexts for a specific case."
s	d	d	*	*more "experiences" about applicability*	If #pairs >=3 then a subset can lead to one of the diagnoses in the three above rows.	
s	s	s	d	***effect inconsistency***	*no*	**MP-2** "Detect effect inconsistencies."
				effect variation	*ok*	Optimization regarding effect is possible
d	s	s	s	***context-specific alternatives***	*ok*	
d	s	d	*	*more different cases*	*ok*	
d	d	*	*			

a. 's' = same, 'd' = different. '*' = 'same/different'. This refers to semantically same/different items (see explanation in text).

b. Having the same effects is actually nonsense, which favors the typo as diagnosis.

c. This is the only case where the applicability value itself makes a difference for the diagnosis because the diagnosis itself refers to a applicability value.

Figure 3: Maintenance policies for Table 1

- ***MP-1*** *"Detect context applicability inconsistencies."*
  - *Condition:*[1]
    For the occurrences of a case C:
    A new occurrence O1 and an old occurrence O2 exist with
    O1.context = O2.context and
    O1.applicability != O2.applicability.
  - Diagnosis 1: user has selected wrong applicability
  - Diagnosis 2: identify further attribute in case or context that allows to differentiate for the occurrences
    - extend knowledge model with new attribute
    - if attribute is in context:
      - copy context and move link to copied context
      - set value for new attribute for old and new context
    - if attribute is in case:
      - copy case and move link with context to copied case
      - set value for new attribute for old and new case
- ***MP-2*** *"Detect effect inconsistencies."*
  - [as for the context applicability inconsistency]
- ***MP-3*** *"Too many negative contexts for a specific case*." [2]
  - *Condition:* for a case,
    ratio #'not applicable' / #finished_occurrences > X%
    AND #finished_occurrences > N
  - Diagnosis 1: errors in the case itself
    -> modify case; set status to 'obsolete' if further usage of the old case should be avoided
  - Diagnosis 2: the case was proposed due to bugs in the retrieval

1 For offline or cumulative maintenance, the checked occurrences have to be marked.

2 #finished_occurrences := (#occurrences where applicability != undefined)

further diagnosis and maintenance actions. The different further diagnoses for some maintenance policies show that human interaction is required to make the respective decisions. Furthermore, the maintenance actions themselves include human decision making, e.g., determining which attribute is missing due to a context applicability inconsistency and determining if adding the attribute is reasonable in an economical sense.

Since Table 1 is complete regarding the comparison results, the first diagnosis step can be automated, e.g., using our EMSIG framework [Nick et al. 2001]. Furthermore, thresholds regarding the number of allowed case-context pairs for each validity anomaly allow specifying quality requirements from new and updated cases in a well-defined and controllable manner.

Evaluation

To evaluate the feasibility of the experience feedback loop concept, its crucial points, which enable its proper function, have to be analyzed. Since its proper function is based on the correct interaction of the users with the CBR system, the respective capabilities and the willingness of the users have to be evaluated and impacting factors have to be identified. This leads to the following three questions, which address the respective items from the introduction section:

1. Do users understand validity information?
2. Do users accept such systems where they have to enter cases and give feedback?
3. Do users give a sufficient amount of feedback?

For all questions, it has to be analyzed under which circumstances this is feasible.

Research method: For obtaining evident evaluation results, we chose case studies in real-world projects to get realistic results. The experience feedback loop concept and its refinements have been iteratively developed and tried in nine real-world projects (i.e., research and industrial projects) since 1998. Details can be found in [Nick 2005]. For this paper, we highlight two projects:

CBR-PEB is a web-based information system that provides a search on CBR products and applications. It is publicly accessible (http://demolab.iese.fhg.de:8080).

ITFMS is an experience-based IT failure management system, which has been used since the 4th quarter of 2004 by the IT service of the city treasury of Cologne. The system provides experience-based suggestions for reactions to IT failures. This is integrated into the failure handling process and system. The system has its roots in the SKe project [Nick et al. 2003]. Its concepts have also been transferred and extended in an industrial project.

Understandability of validity information: In 2 projects, we surveyed intended users and other stakeholders in workshops during the development of the CBR systems. In both cases, the applicability score, i.e., *[#applicable, #not applicable]*, was perceived as easily understandable.

Overall acceptance: Sustained usage is a long-term indicator for the acceptance of the system. The usage of the system is monitored over time. For this purpose, a system is regarded as accepted if it has been used more than one year in accordance with expected trends.

In general, the systems developed in seven of the projects show sustained usage for more than one year. Systems from two projects are too new to be evaluated regarding sustained usage. Thus, acceptance is given.

In general, we identified a lower usage level in the cross-organizational systems (3 projects – including CBR-PEB) in comparison to the intra-organizational systems. Particularly the acquisition and recording of new cases has been more difficult than in the intra-organizational setting. Thus, the factor intra-/cross-organizational setting is regarded as a major factor for the acceptance of the experience feedback loop.

Rate of feedback: Another crucial issue is the feedback rate, because the validity function can only be sufficiently correct if there is a representative amount of feedback. For CBR-PEB, the rate over five years was approx. 9%. For the ITFMS, the rate was almost 100%. The major differences between the two systems are (1) an integration of the ITFMS into day-to-day work (which is not at all the case for CBR-PEB), (2) a relatively short-term reuse of the information entered into the system by a closed user group ("relatively short-term" means less than one year), and (3) intra-organization vs. cross-organization use.

Conclusions and Future Work

For CBR systems in domains with evolving case knowledge, the presented experience feedback loop design concept reduces the case acquisition and maintenance bottleneck. This is achieved by (1) the creation and update of cases by the user in his business processes and (2) automated detection of maintenance need using a user-feedback-based validity function. Case studies in real-world projects demonstrate the feasibility regarding the critical points of this approach. With this, it is feasible to limit the maintenance team's tasks to the maintenance of cases with automatically identified validity problems that have occurred in reuse attempts.

In contrast to current approaches that implement a 'review of case quality before application' strategy [e.g., Ferrario and Smyth 2001], the experience feedback loop implements a kind of 'review by application' strategy. This becomes acceptable for the user by making the validity of cases transparent to him, which requires the validity to be sufficiently correct and up-to-date.

The case validity function is guaranteed to be correct and up-to-date only if the process steps of the experience feedback loop are executed correctly and if the feedback is stored correctly and completely. Theoretically, this can only be guaranteed if (1) the feedback loop process steps are tightly integrated into the surrounding business process (e.g., through process integration for stable business processes) and (2) the maintenance operations are guaranteed to be executed (e.g., through tool integration). The case studies in real-world projects demonstrate that this tight integration is essential for a validity function with proper results. This requirement is acceptable in practice because recent studies underline the necessity of such a tight integration for the success of knowledge management systems in general [e.g., Mertins et al. 2003, Weber et al. 2001].

The maintenance policies for diagnosing the need for maintenance are complete in the sense that they address all possible validity anomalies. Thresholds for the anomalies limit the triggering of maintenance to cases where the anomalies are considered to be severe enough and, therefore, reduce the effort on the part of the maintenance team. Currently, these thresholds are being determined in workshops with the stakeholders in order to ensure that the thresholds adhere to the organization's existing policies.

The experience feedback loop design concept has been systematically refined into a set of 26 models (see my work described in [Nick 2005]). This model set enables a checklist-based requirements analysis and the design of experience management systems. The model set addresses further issues such as reuse of several items (as a configuration). The usage of the model set for requirements analysis and design showed an efficiency improvement of a factor >3 for experienced and >11 for inexperienced developers of experience management systems. The measured efficiency improvements make the design method applicable to small/medium-scale systems and rapid prototyping.

Currently, we are working on extending and tailoring the concept to enable systems to be self-aware of how much they can trust their knowledge and use this in their actions. This requires, e.g., to learn and adapt thresholds automatically over time and to cope with incomplete feedback and noise. Such techniques are relevant for the field of Ambient Intelligence where nodes/devices of any size have to run, make decisions, and learn from effects on the environment with as little human intervention as possible to achieve, on the human side, a "feeling" of being in an intelligent adaptive environment –as a kind of friend, *Fr.* bel ami– that reliably acts and learns on its own.[1]

1 Project BelAmI (Bilateral German-Hungarian Research Collaboration on Ambient Intelligence Systems) www.belami-project.org

References

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

Althoff, K.-D., Nick, M., and Tautz, C. 2000. Systematically diagnosing and improving the perceived usefulness of organizational memories. In Ruhe, G. and Bomarius, F., editors, *Learning Software Organizations - Methodology and Applications*, Springer Verlag, Heidelberg, Germany. Number 1756 in Lecture Notes in Computer Science, 72–86.

Basili, V. R., Caldiera, G., and Rombach, H. D. 1994. Experience Factory. In Marciniak, J. J., editor, *Encyclopedia of Software Engineering*, John Wiley & Sons. volume 1.

Ferrario, M. A. and Smyth, B. 2001. Distributing case-base maintenance: The collaborative maintenance approach. *Computational Intelligence*, 17(2):315–330.

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

Mertins, K., Heisig, P., and Vorbeck, J., editors 2003. *Delphi study on the future of knowledge management - Overview of the results*, 179–190. Springer, Berlin.

Nick, M. 2005. *Experience Maintenance through Closed-Loop Feedback*. PhD thesis, University of Kaiserslautern. Published by Fraunhofer IRB Verlag.

Nick, M., Althoff, K.-D., and Tautz, C. 2001. Systematic maintenance of corporate experience repositories. *Computational Intelligence*, 17(2):364–386.

Nick, M., Groß, S., and Snoek, B. 2003. How knowledge management can support the it security of egoverment services. In Wimmer, M., editor, *Proceedings of the Fourth Conference on Knowledge Management in Electronic Government (KMGov)*, Lecture Notes in Computer Science.

Racine, K. and Yang, Q. 2001. Redundancy detection in semistructured case bases. *IEEE Transactions on Knowledge and Data Engineering*, 13(3):513–518.

Reinartz, T., Iglezakis, I., and Roth-Berghofer, T. 2001. Review and restore for case base maintenance. *Computational Intelligence*, 17(2):214–234.

Tautz, C. and Althoff, K.-D. 1997. Using case-based reasoning for reusing software knowledge. In Leake, D. and Plaza, E., editors, *Proceedings of the Second International Conference on Case-Based Reasoning*, Springer Verlag.

Weber, R., Aha, D. W., and Becerra-Fernandez, I. 2001. Intelligent lessons learned systems. *International Journal of Expert Systems - Research & Applications*, 20(1).