Computational Intelligence, Volume 17, Number 2, 2001

A CASE STUDY OF MAINTENANCE OF A COMMERCIALLY FIELDED CASE-BASED REASONING SYSTEM

IAN WATSON
Department of Computer Science, University of Auckland

This article is a case study of the maintenance required to the case base of a commercially fielded case-based reasoning (CBR) system that provides support for HVAC engineers enabling them to better specify HVAC installations. The article briefly describes the system and details how the case base grew rapidly, causing a problem of case redundancy. A simple algorithm to identify and remove redundant cases is described, along with the results of applying it to the case base. Case obsolescence also was encountered and partially remedied using DBMS techniques. The article analyzes the case-base maintenance (CBM) required by the system in terms of Richter's knowledge containers and Leake and Wilson's CBM framework and contrasts this case study with experience from NEC and DaimlerChrysler. The article observes that had maintenance of the case base been considered more explicitly during system design and implementation, some of the resulting maintenance would have been unnecessary. The article concludes by identifying lessons learned and highlighting the relationship between the sophistication of the case-representation and similarity metrics and the ease that CBM can be undertaken by nontechnical staff. This relationship does not always work in favor of the maintainer.

Key words: case-based reasoning, case-base maintenance, maintenance, software engineering.

1. INTRODUCTION

In some sense a case-based reasoning (CBR) system is never finished. The retain process of the CBR cycle (Aamodt and Plaza 1994) means that the case base is constantly growing. The concept of completeness in the knowledge-based systems literature only applies in domains where the domain theory or model is well understood. CBR systems most often operate in weak theory domains, and the case base could only be complete if all possible problems in the domain were covered. This is very rarely the situation.

Moreover, although the problem domain must be reasonably stable for similar problems to repeat and hence CBR to be useful (Kolodner 1996), the world does change. There are both explicit and implicit changes in the reasoning environment and problem focus that will influence the fit of the case base to the problem context. This will affect the quality and efficiency of the system's results. Thus it has been recognized for some time now within the CBR community that to keep a system's performance at acceptable levels, routine maintenance will be required (Watson 1997; Leake and Wilson 1998, Smyth 1998).

This article explores the case base maintenance (CBM) issues encountered after 2 years of operational use of a commercially fielded CBR system. These were case redundancy, caused by the acquisition of many functionally similar cases, and case obsolescence, caused primarily by equipment obsolescence. The article describes the issues encountered and the remedies applied and shows how initial design issues have affected the CBM required. The article then discusses the maintenance required by the system in terms of Richter's knowledge containers (1995) and Leake and Wilson's CBM framework (1998). The article concludes by noting that there may be a relationship between sophistication or complexity of the case representation, similarity metrics,

Correspondence should be addressed to the author at AI-CBR, Department of Computer Science, University of Auckland, Auckland 1, New Zealand; e-mail: *ian@ai-cbr.org*.

and retrieval algorithms and the complexity of CBM. This relationship has software engineering implications; effort put into design may reduce CBM but will increase the initial cost of the system.

2. BACKGROUND: THE COOL AIR SYSTEM

This article describes work carried out on a commercially fielded system within a small Australian engineering company (Western Air, Ltd.) that installs heating ventilation and air conditioning (HVAC) systems. In May 1998 the company rolled out a CBR system, called *Cool Air*, to support the sales engineering force in specifying and quoting for HVAC installations. The system is described in detail in Watson and Gardingen (1999) and Watson (2000); I was a consultant to the project. The system's architecture and functionality will be described briefly here to help understand the CBM issues subsequently encountered.

Cool Air is a distributed client server system operating via the Internet [or in Sengupta et al's terminology (1999), a Web-based CBR system]. A relational database resides on a server at the company head office. A sales engineer obtains customer requirements using a form that a Java client formats into XML and sends to a Java servlet on the server. The servlet queries the database using SQL and a technique called *query relaxation* that uses knowledge about the problem (e.g., a symbol hierarchy of equipment and components) to perform a similarity-based retrieval from the database. This returns a set of about 20 similar records, which are parsed into XML cases and sent back to the client-side applet.

The client-side applet then ranks the set of cases using a weighted nearest-neighbor algorithm (feature weights can be set by the sales engineers to reflect personal preferences). Cases contain a summary of the technical solution of an HVAC installation plus file links to detailed project information, technical drawings, full specification, and costs residing on an FTP server back at head office. This architecture is shown in Figure 1.

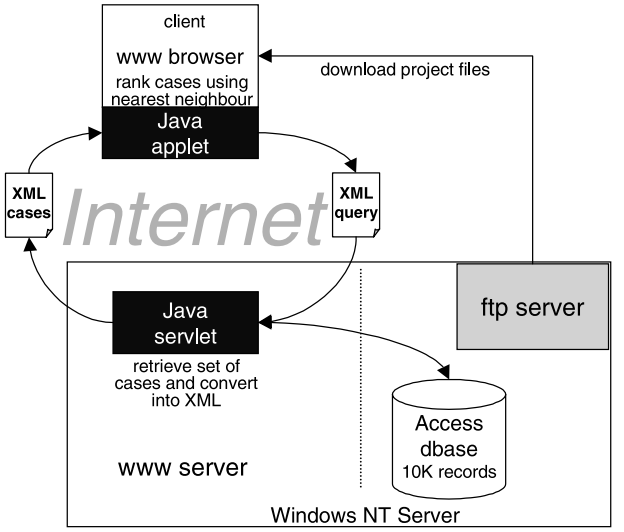

FIGURE 1. System architecture.

It was recognized during the initial design that there was a distinction between the corporate database of all HVAC installations that had to be complete for reporting, accounting, and legal reasons and the case base, which may contain a subset of these records for problem solving. Thus a field had been created in the database to contain a status flag that could be set to

- *Active.* The case is used by the case base for problem solving.
- *Archive.* Not used by the case base but is available for company records.
- *xxxx.* Used only by system maintainers as test cases.
- *Inspect.* A case requiring inspecting by a system maintainer (e.g., an installation came in significantly over budget or ended in litigation).

After some initial technical problems and some user acceptance problems, the system performed well, and a substantial return on investment was reported (Watson and Gardingen 1999).

3. CASE BASE USAGE AND GROWTH

The company realized from the outset that use of Cool Air could not be optional. To ensure consistency, all sales engineers had to use the system. This was ensured by making the system useful even to experienced engineers who did not believe they needed assistance in specifying projects. Thus from the initial design Cool Air had a form-filling role, work that had to done to record and process a quotation anyway for company records. In a sense, the case-based assistance was a bonus. Consequently, Cool Air was widely used from the start.

In May 0f 1998, the database contained approximately 10,000 records. These were all relatively recent HVAC installations dating back no more than 5 years. Projects were not consistently stored in a digital format until the mid-1990s.

The company employs approximately 100 sales engineers, each of whom deals with an average of five quotations a week (this average is a little misleading because project size and complexity vary greatly from simple residential systems to complex retail and commercial systems). Engineers work for 48 weeks in the year, and hence the company generates about 24,000 specifications and quotations a year. The company expects to win about 25 percent of the tenders (i.e., 6000 installations). Of these, from 10 to 20 percent will not proceed for various reasons. Thus the company expects to perform about 5000 HVAC installations per year. Actual figures are shown in Table 1.

All successfully completed installations are retained in the case base. Moreover, any installation problems are recorded and stored, enabling lessons learned from installations to be captured and used by engineers in future (Watson 2000).

TABLE 1. Number of HVAC Installations by Year

Year	No. of Installations
1998 (May–Dec)	2633
1999 (Jan–Dec)	5174
2000 (Jan–May)	1984
TOTAL	9791

The number of installations is therefore directly equivalent to the number of new cases retained by Cool Air. Thus the case base has almost doubled in 2 years (from 10,000 to 19,791 cases). This considerable growth raised concerns about the utility problem with respect to case retrieval (Smyth and Cunningham 1996) and suggested that a case deletion technique would be required to control case-base growth (Smyth and Keane 1995).

4. MAINTENANCE ISSUES

This section describes how two CBM issues were dealt with, namely, functionally redundant cases and obsolete cases. Several general system problems requiring maintenance other than to the case base also were found and are reported in Watson and Gardingen (1999).

4.1. Functionally Redundant Cases

Many HVAC installations are very similar, even identical to each other. For example, within a new housing development, there may be several identical house designs repeated throughout the development. Moreover, a developer frequently builds identical properties in different locations. Thus, within the case base, there are many functionally identical cases with different location and client details.

Cool Air has a two-stage retrieval process. In the first server-side process, a set of similar cases (approximately 20) is retrieved from the database and sent to the client-side applet. Clearly, there is no point is sending 20 identical cases where one would suffice.

Three solutions to this problem were considered:

1. *Just send one case to the client when all cases in the retrieved set are identical.* This was rejected because the servlet does not know that the cases have the same similarity measure. The SQL query retrieves a set that matches within defined limits, and the production of a numeric similarity metric is done by the client-side applet. Moreover, even if this were possible, it is undesirable because the sales engineers want to be presented with a set of alternatives from which they choose and create a solution. They do not want to be given a single solution.
2. *Change the retrieval algorithm on the server side so that it could measure similarity, reject identical redundant cases, and construct a useful set of alternatives to send to the client applet.* This was rejected because it would have meant completely changing the server-side algorithm, which was felt to be working fine. Moreover, it did not confront the problem of the presence of functionally redundant cases in the case base.
3. *Examine the case base and identify and remove functionally redundant cases.*

Solution 3 was chosen as being the sensible solution for which there were three alternative approaches:

1. *Manual.* An experienced engineer would examine the case base periodically and identify and remove redundant records.
2. *Semiautomatic.* An algorithm would analyze the case base and automatically identify sets or clusters of similar cases and flag these, and a person would select one case from the set to represent it; the others would be archived.

```
begin
        for i = 1 to n
                for j = 1 to n
                        if sim(case_i,case_j) >= t
                                then append(set_i ,case_j)      // add the case to a list
                        j++
                end
                if length(set_i) not(> m)
                        then delete(set_i)                      // deletes sets with fewer than m
members
                i++
        end
        for i 1 to s
                for j = i+1 to s
                        if set_i == set_j
                                then delete(set_j)              // deletes the first identical set of
the pair
                        j++
                end
                i++
        end
end

where:  n = number of cases
        t = similarity threshold (default = 1.0)
        m = membership threshold  (default = 5)
        s = number or sets created
```

FIGURE 2. Redundant Set Identification algorithm.

3. *Automatic.* An algorithm would be designed to analyze the case base and automatically identify and remove redundant cases. This algorithm could be run periodically (perhaps weekly) to remove redundancy.

Solution 1 was rejected because the task would be difficult and tedious to perform manually. Solution 2 was chosen, at least initially, because its success or failure would help determine if a fully automated approach was achievable.

Redundancy Algorithm Design. Each record in the database contains a field to reference installations that were part of a larger development, such as a housing, apartment, or retail development. These units within a large development were likely to be similar or even identical. However, this could not be guaranteed because a proportion of multiunit developments are made up of unique units (this is often used as a selling feature). Moreover, this reference does not identify commonly repeating standard designs used by many developers in many locations. Consequently, using an SQL query to simply identify all units within multiunit developments would not solve the problem.

An algorithm had to be developed to inspect the case base and identify all identical cases. The algorithm (shown in Figure 2) takes each case in turn and compares it with every case in the case base. Identical cases (or those which exceed the similarity threshold t) are added to the case's similarity set. However, if $case_1$ is identical to $case_3$, $case_7$, and $case_9$, this results in four identical similarity sets being created, each containing cases 1, 3, 7 and 9. Consequently, it is necessary to compare all the similarity sets with each other and delete identical sets.

It was recognized that comparing each case with every other case is not a computationally efficient solution. However, since the algorithm need only be run periodically and can be run off-line overnight or on the weekend, this is unlikely to cause problems in the future. Processing time is much cheaper to the company than consultancy time.

The Redundant Set Identification (RSI) algorithm shown in Figure 2 outputs a list of similarity sets each containing six or more cases that are identical or whose similarity exceeds the a predefined similarity threshold. No two sets have the same membership.

Initially, the similarity threshold was set to 1.0 (i.e., identical), but the set membership threshold was set to 5. It was felt by engineers that being shown that there were

TABLE 2. Sets Identified with Similarity of 1.0

Set Membership	<10	<25	<50	<75	<100	<125	<150	≥= 150	Totals
No. of sets	11	16	21	17	5	2	5	0	77
No. of redundant cases	86	288	777	1122	438	222	655	0	3587

several identical solutions provided a measure of confidence in the solution suggested, and there would be 15 or so alternatives available if needed.

Once the sets were identified, the system maintainer could examine each set in turn, choose a single case to represent the set, and set the status flag of the other members to *archive*.

Redundancy Algorithm Results. The RSI algorithm was run over the case base of 19,791 cases. The similarity threshold was set to 1.0 (identical), with the results shown in Table 2. A total of 3587 redundant cases were identified in 77 sets, or 18.1 percent of the cases were identified as being redundant. This significant percentage was not surprising because if redundant cases were not sufficiently common to be a problem, they would not have been noticed by users.

Since cases could be very similar, though not identical, and still be functionally redundant (i.e., there are no significant difference in the HVAC specifications), the similarity threshold was reduced to 0.95 (i.e., 95 percent similarity). The results are shown in Table 3. The number or redundant cases identified had increased to 5427 (27.4 percent of the case base), the number of similarity sets only increased by 11, while the number of cases increased by 1840. This is so because with the weaker similarity threshold more cases are being added to existing similarity sets. If the similarity threshold were set to zero, all cases would belong to a single similarity set.

Selecting a Set Representative. Once the similarity sets were identified, the next task was to examine each set and select a single case to represent it. The remaining cases in the set would have their status flag set to *archive* and thus be ignored in future case base retrievals. Three strategies were considered:

1. Manually select the representative.
2. Randomly select the representative.
3. Select the median case, i.e., the case with the greatest similarity to all cases in the set.

Solution 1 was rejected because the engineer selected to perform the task said that he found it difficult to decide and admitted to randomly selecting a "likely looking

TABLE 3. Sets Identified with Similarity of 0.95

Set Membership	<10	<25	<50	<75	<100	<125	<150	≥=150	Totals
No. of clusters	15	19	25	21	7	4	6	1	98
No. of redundant cases	135	437	1153	1512	665	487	882	156	5427

```
begin
   for i = 0 to s
         n = length(set_i)
         for j = 1 to n
               sim_j = sim(case_i, element(set_i , 1-n)) - 1          // calc similarity to
other cases
               append(simList , sim_j)                                // add that total to the list
               j++
         end
         sort(simList)                                                // sort the list by
descending order
         append(caseList, element(simList, 1))        // add the first element to the
case list
         i++
   end
end

where:   s = number of similarity sets
```

FIGURE 3. Median Case Identification algorithm.

candidate." In effect, little difference from solution 2. A simple algorithm was written to select the median cases, as shown in Figure 3.

This algorithm creates a list containing the representative case from each similarity set (i.e., the case with the highest total similarity to other cases in its set, in the event of several cases having an equally high total similarity, the first case was selected). These cases are retained, whereas all other cases in the similarity sets have their status flags set to *archive*.

Application of this algorithm reduced the case base by 5329 cases (i.e., 5427 cases less one representative from 98 similarity sets). The new case base contained 14,462 cases, which still represents a significant increase in case base size from its original size.

4.2. Functionally Obsolete Cases

The second CBM issue related to case obsolescence. Over time, HVAC equipment is withdrawn and replaced, and working practices change. Cases referring to installations using obsolete products or techniques need to be deleted from the case base to prevent inexperienced engineers including them in new specifications and quotes. The company releases weekly technical memoranda by email and specific working practice guidelines, which are updated quarterly. Moreover, sales engineers receive twice-annual training to ensure that they are up to date with current products and practice.

Some CBR systems retain details of obsolete cases because these may provide useful analogies for problem solving in future. It is common for trouble-shooting or diagnostic case bases to retain cases referring to problems with obsolete equipment because similar problems may occur in the future with new equipment (Klahr 1996). However, it was decided by management that installations using obsolete equipment need not be retained for problem solving in Cool Air.

It was a relatively easy administrative job to query the database to identify and archive cases that refer to obsolete equipment so that they are not included in the case base retrieval process. This is done each time there is a significant product change. However, changes also need to be made to the symbol hierarchies used by the SQL query relaxation technique [see Kitano and Shimazu (1996) and Watson and Gardingen (1999) for a description of this technique]. This was not anticipated during the design of the system. Editing the symbol hierarchies (a sample is shown in Figure 4) to remove obsolete items of equipment or entire classes of equipment is not simple. They are stored as tables within the database, and a good knowledge of the table structure and relations between them is required to ensure that the hierarchy is not corrupted.

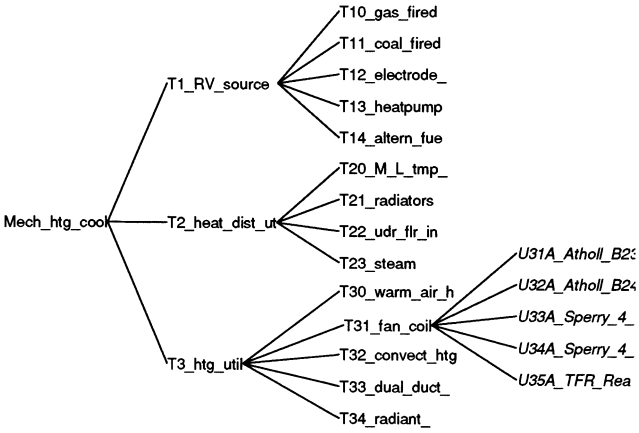

FIGURE 4. A portion of a symbol hierarchy for mechanical heating and cooling systems.

It is not clear that this can be done automatically or even semiautomatically. A graphic hierarchy editor would greatly help the editing task and make it more feasible for a domain expert to do the maintenance rather than a programmer. This has been suggested to the company but is currently beyond its budget.

Finally, it remains unclear how to identify records where obsolete working practices were used because these are not explicitly referred to in the record structure but remain hidden in the supporting files on the FTP server (see Figure 1) or are not even recorded at all. Working practices were not considered as important during the design of either the database or the CBR system, and this is an ongoing issue that has not been resolved.

It is worth comparing this with the experience of Kitano and Shimazu (1996) at NEC with SQUAD. Cool Air's query-relaxation SQL technique is derived from that of the SQUAD system, but beyond this, the systems differ significantly. The SQUAD system was design as a corporate memory of software quality-control problems. While it contains over 20,000 cases, it is not clear to what degree redundancy or obsolescence of cases is or was a problem. However, Kitano and Shimazu (1996) do state that using a DBMS to manage a large case base is extremely useful.

5. DISCUSSION

The 2 years of use of Cool Air have provided an interesting case study of a commercially fielded CBR system. CBM was recognized as an issue during the initial design, and management was aware that the system would require regular maintenance if it were to remain useful.

It is useful to analyze the maintenance the system has required in terms of Richter's knowledge containers terminology (Richter 1995), since, as Richter noted, in theory, each of the knowledge containers of a CBR system may require maintenance as the domain knowledge changes subtly and the case base grows.

1. *Vocabulary.* Cool Air has the same case features now as it did 2 years ago; no vocabulary maintenance has been done. However, use of the system has shown that we failed to capture knowledge about working practice or methods within the case vocabulary, and this should be addressed in the future.

2. *Similarity measure.* The concept hierarchy is an essential component of the similarity-based retrieval from the database. This has required maintenance. However, other components within the system's similarity measure have not required maintenance. It would be unwise to generalize from this because Cool Air is a very relaxed system in that it need only retrieve good or likely candidates. A CBR system that has to retrieve with greater precision may well require maintenance in this area.
3. *Case base.* This has required extensive maintenance that is repeated periodically. Two forms of CBM were carried out: (1) redundant cases were identified and removed using an introspective reasoning technique approximately on a monthly cycle, and (2) obsolete cases were removed using standard DBMS tools whenever a significant obsolescence was identified. It is worth remembering here that DBMS techniques were not sufficient to identify redundant cases because redundancy is dependent on similarity.
4. *Solution transformation.* Cool Air does not perform adaptation; this is left to the engineers, so no maintenance was required here.

It can be argued that the redundancy problems we encountered were due to (or exacerbated by) the design of Cool Air and the two-stage retrieval process in particular. Although we have not observed the utility problem (i.e., retrieval performance has not suffered), with a case base doubling in size over 2 years, it would be unwise to ignore the utility problem in the long term.

In comparison with the CBM processes described by Goker and Roth-Berghofer (1999) for the HOMER system, it must be recognized that Western Air is a very small company in comparison with DaimlerChrysler and that management processes throughout are much more ad hoc and flexible. Yet it is clear that like HOMER, Cool Air does follow a distinct "maintenance cycle," although the "retain" task is done automatically without intervention from a case-base administrator. However, the maintenance of the knowledge containers is undertaken periodically in a "refine" task and is largely done by introspection using the algorithms described above as recommended for HOMER.

It is also worth analyzing how the maintenance of Cool Air is performed in terms of the case base maintenance (CBM) framework created by Leake and Wilson (1998). Cool Air's maintenance policy is as follows:

- *Type of data. None*. Statistics are gathered on global usage of the case base, usage of individual cases, and the retention of new cases, but this information is not used to inform either the retention of new cases, when to perform maintenance or what maintenance, to perform.
- *Timing. Ad hoc and conditional*. The removal of redundant cases is currently done at irregular intervals. The removal of obsolete cases is conditional on the identification of product obsolescence.
- *Integration. Off-line*. Data collection is performed off-line when the system is not being used.
- *Triggering. Result-based*. Maintenance will be triggered if a user reports a problem with system
- *Revision level. Knowledge-level*. Maintenance focuses on deleting cases.
- *Execution. None*. The system executes no maintenance changes itself.
- *Scope of maintenance. Broad*. Maintenance operations typically will affect many cases in the case base. However, if maintenance were scheduled regularly, perhaps after each case retention, then the scope of maintenance would become *narrow*.

With hindsight, the design of Cool Air should be changed to examine each new case before it is added to the system and only retain it if it is significantly different from other cases already in the case base (i.e., the case is useful). This not only would be a simpler algorithm to apply than the maintenance algorithms shown in Figures 2 and 3, but it also would remove redundancy completely in future. Making this small implementational change would change Cool Air's maintenance policy dramatically to an introspective, continuous, on-line, narrow policy.

6. LESSONS LEARNED

The development of the Cool Air system, like many programs within small companies, came about partially because of chance and the vision of one or two people rather than as the product of a carefully managed process. The initial goal of the modest development project, as reported in Watson and Gardingen (1999), was to develop a system as quickly and cost-effectively as possible. The need to maintain the case base was made explicit to management from the outset. Yet CBM become more complex than envisioned. The following lessons can be learned from our experience:

- case base developers cannot be certain that they have elicited all correct case features. While not a CBM issue per se, developers should plan to revisit the case features some time after the system is rolled out. (*Note*: This is a different issue from the emergence of new case features with time, which is a CBM issue.)
- Storing cases in a DBMS is essential if the case base is of a reasonable size. Most commercial CBR tools now provide this functionality and no longer keep their cases in proprietary formats. A DBMS greatly helps with reporting on the case base, removing or archiving obsolescent cases, and general case-base management.
- A more complex case representation will be more complex to maintain. However, maintenance may be required less often.
- If developers inherit a case base from a legacy database, they should consider using an algorithm to analyze the coverage of the case base, such as the one proposed by Smith and McKenna (1998). CBM was greatly increased by the presence in the original case base of many redundant cases. Highlighting this would alert developers to the need to explicitly deal with redundancy at an early stage before even more redundant cases are acquired through new case retention.
- Developers should look at each of Richter's knowledge containers (1995) and inquire in advance what the maintenance needs of each container might be. This would have brought to our attention the need to maintain the symbol hierarchy as equipment changed.

7. CONCLUSIONS

Cool Air provides a snapshot of how a CBR system has been kept in regular, profitable use for 2 years. Although some of the issues dealt with here are specific to this application, the domain, and the company, it is possible to generalize lessons learned from the case study. Although the designers of Cool Air explicitly considered maintenance from the outset, in retrospect this was exclusively from a managerial viewpoint to ensure management commitment to expenditure on regular maintenance. Had we considered CBM from an implementational viewpoint during design, with the exception of obsolescence, we could have better designed maintenance into Cool Air. This

would have been both more elegant and more cost-effective. In our defence, I would say that probably in common with most CBR system implementers we were so focused on "getting retrieval to work" that maintenance became a "we'll deal with that later" issue. Designers of CBR systems therefore should remember that they need to design a system from the outset that deals with all the processes of the CBR cycle and not just the retrieval, reuse, and revision processes.

There is, however, an important software engineering issue here. Developers of CBR systems will always be under pressure to deliver a functional CBR system quickly. One of the selling points of CBR systems has been that that they can be implemented quickly (Harmon 1992). We have recognized that more complex case representations, similarity metrics, and retrieval algorithms can improve the precision and/or efficiency of retrieval. However, this complexity increases the cost of developing systems. This tradeoff is also true for CBM. A richer case representation would have partially solved the redundancy problem identified in Section 4.1.1. More extensive knowledge engineering might have captured that working practices should have been a case feature, as noted in Section 4.2. Both of these would have increased the development cost, would have reduced the return on investment, and may have resulted in the system never getting the management backing required to deliver it. A great strength of CBR is its simplicity—the symbol hierarchy used by the system to assess similarity (see Figure 4) is already an example where the complexity of the system has exceeded the ability of its end users to maintain it without professional programming assistance.

Developers of commercial CBR systems are advised to keep their system simple but to recognize that there is a relationship between sophistication or complexity of the CBR system and the complexity of CBM. This relationship has software engineering implications; effort put into design, representation, and architecture may reduce CBM but may increase the complexity and initial cost of the system. Moreover, although a more complex system might require less CBM (perhaps because more routine tasks are automated), that which is required may need the skills of a specialist knowledge engineer or programmer.

ACKNOWLEDGMENTS

I would like to acknowledge the access provided to the system by Western Air, Ltd., and to the cooperation of all staff there. I also would like to acknowledge the contributions of the reviewers and in particular the editors of this special issue for their assistance in improving this article.

REFERENCES

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

GOKER, H., and T. ROTH-BERGHOFFER. 1999. Development and utilization of a case-based help-desk support system in a corporate environment. *In* Case-Based Reasoning Research and Development. *Edited by* Althoff et al. Springer-Verlag, Berlin, pp. 132–146.

HARMON, P. 1992. Case-based reasoning III. Intelligent Software Strategies, **8**(1).

KITANO, H., and H. SHIMAZU. 1996. The experience-sharing architecture: A case study on corporate-wide case-based software quality control. *In* Case-Based Reasoning: Experience, Lessons and Future Directions. *Edited by* D. B. Leake. AAAI Press/The MIT Press, Menlo Park, CA, pp. 235–268.

KLAHR, P. 1996. Global case-based development and deployment. *In* Advances in Case-Based Reasoning, *Edited by* I. Smith and B. Faltings. Springer-Verlag, Berlin, pp. 392–399.

KOLODNER, J. L. 1996. Making the implicit explicit: Clarifying the principles of case-based reasoning. *In* Case-Based Reasoning: Experiences, Lessons, and Future Directions. *Edited by* D. B. Leake. AAAI Press/The MIT Press, Menlo Park, CA.

LEAKE, D. B., and D. C. WILSON. 1998. Categorizing case base maintenance: Dimensions and directions. *In* Advances in Case-Based Reasoning. *Edited by* B. Smyth, and P. Cunningham. Springer-Verlag, Berlin, pp. 196–207.

RICHTER, M. 1995. The Knowledge Contained in Similarity Measures. Invited talk at ICCBR'95. *Available at* www.cbr-web.org/documents/Richtericcbr95remarks.html.

SENGUPTA, A., D. C. WILSON, and D. B. LEAKE. 1999. On constructing the right sort of CBR implementation. *In* Proceedings of the IJCAI-99 Workshop on Automating the Construction of Case Based Reasoners, Stockholm, Sweden.

SMYTH, B. 1998. case base maintenance. *In* Proceedings of the 11th International Conference on Industrial and Engineering Applications of AI and Expert Systems.

SMYTH, B., and P. CUNNINGHAM. 1996. The utility problem analyzed: A case-based reasoning perspective. *In* Advances in Case-Based Reasoning. *Edited by* I. Smith and B. Faltings. Springer-Verlag, Berlin, pp. 392–399.

SMYTH, B. and M. KEANE. 1995. Remembering to forget: A competence-preserving case deletion policy for case-based reasoning systems. *In* Proceedings of the 13th International Joint Conference on Artificial Intelligence. pp. 377–382.

SMYTH, B. and E. MCKENNA. 1998. Modelling the competence of case-bases. *In* Advances in Case-Based Reasoning. *Edited by* B. Smyth and P. Cunningham. Springer-Verlag, Berlin, pp. 208–220.

WATSON, I. 1997. Applying Case-Based Reasoning: Techniques for Enterprise Systems. Morgan Kaufmann, San Francisco.

WATSON, I. 2000. A case-based reasoning application for engineering sales support using introspective reasoning. *In* Proceedings of the 17th. National Conference on Artificial Intelligence (AAAI-2000) and the 12th Innovative Applications of Artificial Intelligence Conference (IAAI-2000), July 30–August 3, Austin Texas. AAAI Press, pp. 1054–1059.

WATSON, I. and D. GARDINGEN. 1999. A distributed case-based reasoning application for engineering sales support. *In* Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99), Vol. 1. Morgan Kaufmann, San Francisco, pp. 600–605.