9. Case-based Reasoning and Software Engineering

Martin Shepperd

Empirical Software Engineering Research Group, Bournemouth University,
Bournemouth, BH1 3LT, UK
email: mshepper@bmth.ac.uk

9.1 Introduction

Case-based reasoning (CBR) was first formalised in the 1980s following from the work of Schank and others on memory [1], and is based upon the fundamental premise that similar problems are best solved with similar solutions [2]. The idea is to learn from experience. However, a crucial aspect of CBR lies in the term "similar". The technique does not require an identical problem to have been previously solved. Also CBR differs from many other artificial intelligence techniques in that it is not model based. This means, unlike knowledge based approaches that use rules, the developer does not have to explicitly define causalities and relationships within the domain of interest. For poorly understood problem domains this is a major benefit.

CBR is a technique for managing and using knowledge that can be organised as discrete abstractions of events or entities that are limited in time and space. Each such abstraction is termed a case. Software engineering examples could be projects, design patterns or software components. Cases are characterised by vectors of features such as file size, number of interfaces or development method. CBR systems typically function by solving the new problem, often termed the target case, through retrieving and then adapting similar cases from a repository of past (and therefore solved) cases. The repository is termed the case-base.

CBR is argued to offer a number of advantages over many other knowledge management techniques, in that it:

1. avoids many of problems associated with knowledge elicitation and codification.
2. only needs to address those problems that actually occur, whilst generative (i.e. algorithmic) systems must handle all possible problems.
3. handles failed cases, which enable users to identify potentially high risk situations.
4. copes with poorly understood domains (for example, many aspects of software engineering) since solutions are based upon what has actually happened as opposed to hypothesised models.
5. supports better collaboration with users who are often more willing to accept solutions from analogy based systems since these are derived from a form of reasoning akin to human problem solving. This final advantage is particularly

important if systems are not only to be deployed, but also to have trust placed in them.

Since the 1980s CBR has generated significant research interest and has been successfully applied to a wide range of problem domains. Typical applications are diagnostic systems, for instance, CASCADE addressed solving problems with the operating system VMS. More recently, Alstom have deployed CBR technology, in conjunction with data mining of past fault data, to support diagnosis of system error messages from the on-board computers which control all the train electronics. Another application area has been legal systems, unsurprisingly, since the concept of precedent and case law lie at the heart of many judicial systems such as those of the UK and USA. Design and planning are other problem domains that have also been tackled. For instance CADET was developed as an assistant for mechanical designers and ARCHIE provides support for architects. Decision support, classification (e.g. PROTOS was developed to classify hearing disorders) and e-commerce (e.g. a last minute web-based travel booking system that uses a CBR engine in order to overcome the problem of not always being able to exactly match client requirements) are other problem domains that have been successfully tackled using CBR. Although a little dated, Watson and Marir [3] provide detailed descriptions of a wide range of CBR applications. Lists of more recent examples of applications may be found at [4, 5].

The remainder of this chapter provides more background on CBR technology (principally from a machine learning viewpoint), reviews some specifically software engineering applications of CBR, namely project effort prediction, defect prediction, retrieval from component repositories and the reuse of successful past experience. It then goes on to consider some of the outstanding challenges (e.g. similarity measures, feature and case subset selection, dimension rescaling and learning adaptation rules) and point to potentially fruitful areas of future work.

9.2 An Overview of Case-Based Reasoning Technology

As previously indicated, case-based reasoning has at its heart the notion of utilising the memory of past problems solved to tackle new problems[1]. Problems are organised as cases where each case comprises two parts. These are the description part and a solution part. The description part is normally a vector of features that describe the case state at the point at which the problem is solved. The solution part describes the solution for the specific problem and may vary in complexity from a single value for a classification or prediction system to a set of rules or

[1] Strictly speaking some authors such as [6] differentiate between interpretetive and problem solving CBR. Interpretetive CBR focuses upon classification rather than direct problem solving, although it could always be argued that classification can be viewed as a subgoal to solving another problem. Whatever, this is not a distinction that is pursued in this chapter.

procedures to derive a solution that might include a range of multimedia objects such as video and sound files.

9.2.1 The Basic CBR Cycle

Aamodt and Plaza [7] helpfully identify four stages of CBR — sometimes referred to as the R^4 model — that combine to make a cyclical process:

1. Retrieve similar cases to the target problem
2. Reuse past solutions
3. Revise or adapt the suggested solutions to better fit the target problem
4. Retain the target and solution in the case-base

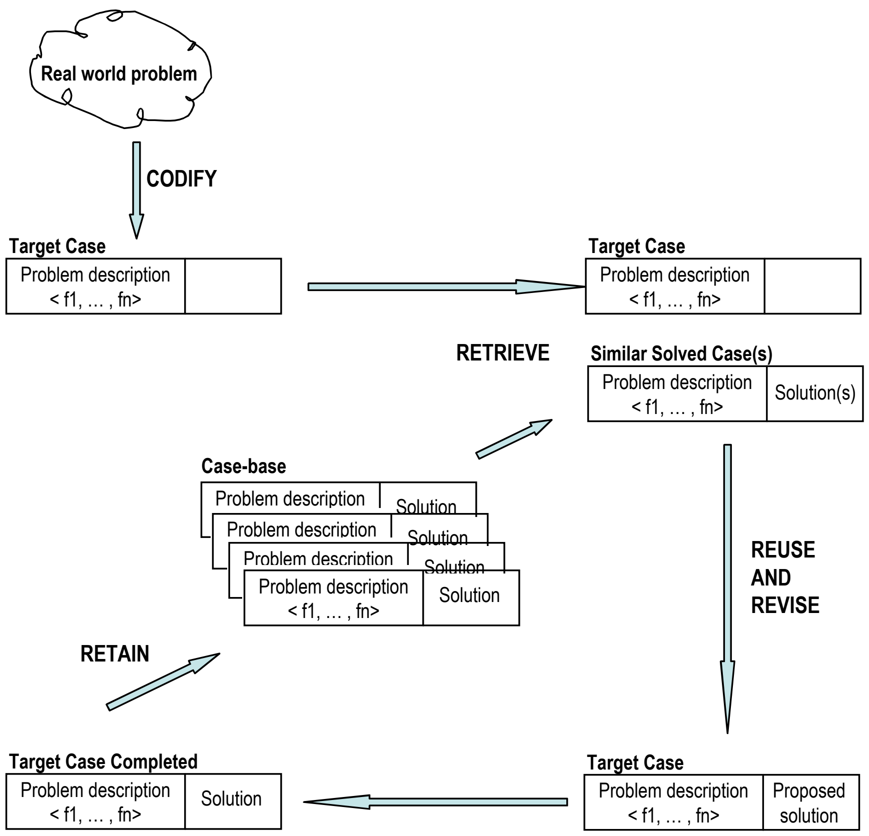

Fig. 9.1 The CBR Process (Adapted from Aamodt and Plaza [7])

Figure 9.1 illustrates this cycle diagrammatically. Central is the case-base, which is a repository of completed cases, in other words the memory. When a new problem arises it must be codified in terms of the feature vector (or problem description) that is then the basis for retrieving similar cases from the case-base.

Clearly, the greater the degree of overlap of features, the more effective the similarity measures and case retrieval. Ideally the feature vectors should be identical since CBR does not deal easily with missing values, although of course there are many data imputation techniques that might be explored [8]. Measuring similarity lies at the heart of CBR and many different measures have been proposed. Irrespective of the measure, the objective is to rank cases in decreasing order of similarity to the target and utilise the known solutions of the nearest k cases. Choosing a value for k is a matter of some debate but for a systematic exploration see [9]. Solutions derived from the retrieved cases can then be adapted to better fit the target case either by rules, a human expert or by a simple statistical procedure such as a weighted mean. In the latter case the system is often referred to as k-nearest neighbour (k-NN) technique. Once the target case has been completed and the true solution known it can be retained in the case-base. In this way the case-base grows over time and new knowledge is added. Of course it is important not to neglect the maintenance of the case-base over time so as to prevent degradation in relevance and consistency.

This CBR process is best illustrated by an example. Consider the problem of a project manager predicting how many resources to allocate for the development of different software components. Knowledge or memory of the past is the basis for predicting future effort. Here the case is a software component. Each case will comprise a vector of features to describe each component. Examples of features might include the programming language (categorical), the number of interfaces (discrete) and the time available to develop, since severe schedule compression may adversely affect the development effort (continuous). Notice how the vector can comprise features of different types. This adds some complexity to the way in which distance is measured. The choice of features is arbitrary and may be driven by both pragmatic considerations — what is easily available — and domain considerations — which features best characterise the problem. One constraint is that the values for the features must be knowable at the time the prediction is required which will usually militate against the use of features such as code length. For effort prediction the solution part of the case is trivial, merely a single value denoting the actual effort consumed.

For our effort prediction problem, the case-base will grow as components are completed and the solution, i.e. the actual required amount of effort in person hours, or whatever, becomes known. When a new prediction problem arises, the new component must be described in terms of the feature vector so that it can be viewed as the target case. The problem then becomes one of retrieving similar cases from the case base and using the known effort values as a basis of the prediction for the target case. The prediction may be modified by the application of rules, typically obtained from a domain expert such as an experienced project manager, or by a simple procedure such as finding the mean. Once the component has been completed and the true effort value is known the case can be added to the case-base. In this way the case-base is enlarged over time and can also follow trends or changes in the underlying problem domain, such as the introduction of new technologies and programming languages. For this reason some similarity measures explicitly include a notion of recency so that newer cases are preferred.

9.2.2 Similarity Measures

As mentioned, measuring similarity has generated a range of different ideas. These include:

- nearest neighbour algorithms are the most popular and are based upon straightforward distance measures for each feature. Each feature must be first standardised, so that the choice of unit has no influence. Some variants of this algorithm will enable the relative importance of features to be specified, although for poorly understood problem domains this may be very problematic. A common algorithm is given by Aha [10]

$$SIM(C_1,C_2,P) = \frac{1}{\sqrt{\sum_{j \in P} Feature\_dissimilarity(C_{1j},C_{2j})}}$$

  where P is the set of *n* features, C_1 and C_2 are cases and

$$Feature\_dissimilarity(C_{1j},C_{2j}) = \begin{cases} (C_{1j} - C_{2j})^2 \\ 0 \\ 1 \end{cases}$$

  where (i) the features are numeric, (ii) if the features are categorical and $C_{1j}=C_{2j}$ or (iii) where the features are categorical and $C_{1j}\neq C_{2j}$ respectively.
- manually guided induction: here an expert manually identifies key features, although this reduces some of the advantages of using a CBR system in that an expert is required.
- template retrieval: this is similar to query by example database retrieval in that the user supplies values or ranges for a subset of the problem description vector, and all the cases that match are retrieved.
- specificity preference: here cases are preferred that match features exactly over those that match generally.
- frequency preference: here preference is given to those cases that have been most frequently retrieved in the past.
- recency preference: this type of algorithm favours more recently matched cases over those that have not been matched for some period of time.
- object-oriented similarity: for complex problem domains it may be necessary to make similarity comparisons between differently structured cases. In the object-oriented approach cases are represented as collections of objects (each object has a set of feature-value pairs) organised in a hierarchy of part-of relationships [11].
- fuzzy similarity: this approach uses concepts such as at-least-as-similar and just-noticeable-difference [12] as opposed to crisp values.

These similarity measures suffer from a number of disadvantages.

First, symbolic or categorical features are problematic. Although there are several algorithms that have been proposed to accommodate categorical features, these tend to be fairly crude in that they tend to adopt a Boolean approach: features match or fail to match with no middle ground. Note though that the fuzzy similarity can be an exception since the linguistic concepts of, say, “quite similar” might be applied to some categorical features, for example, comparing a feature programming language containing the values C and C++.

A second criticism of many of these similarity measures is that they fail to take into account information which can be derived from the structure of the data, thus, they are weak for higher order feature relationships such as one might expect to see exhibited in legal systems. By contrast, the object-oriented similarity measures can still be applied to complex problem domains where it may be necessary to assess similarity between differently structured cases. Here, in order to consider similarity it is necessary to take into account both intra- and inter-object similarity. Intra-object similarity is based on common properties, however, the difference between two cases may reside in their differing class structures rather than in their shared features, hence the need for a measure to take into account inter-object similarity. An example might be comparing software projects that are differently comprised of staff and staff roles. For instance Project (case) A may comprise management, clerical and technical teams, each characterised by their own set of features, whilst Project (case) B might comprise technical and sales teams. A traditional similarity metric can only compare features in common but cannot compare the differing structures of these two projects or cases. Bergmann and Stahl [11] describe a sophisticated similarity metric based on the product intra- and inter-object similarity. The main difficulties for such metrics are validation and encouraging collaboration between the human user and the CBR system since this approach is somewhat less intuitive than a simple Euclidean distance measure.

9.2.3 Feature and Case Subset Selection

Another difficulty for CBR, that is common to all machine learning approaches, is that the similarity measures retrieve more useful cases when extraneous and misleading features are removed. Knowing which features are useful is not always obvious for at least three reasons. First, the features contained in the feature vector are often determined by no more a systematic reason than availability. Second, the application domain may not be well understood: there is no deep theory to guide. Third, the feature standardisation used by some similarity measures can substantially complicate any analysis. This is because some features may actually be more important than others, however, the standardisation will assign each feature equal influence. In such circumstances co-linearity can be usefully exploited. Effectively by using several closely related features, one underlying dimension can be made more important in the search for similar cases. Deciding which features to remove is known as the feature subset selection problem. There is an equivalent problem relating to case removal, known rather unsurprisingly, as the case subset selection problem. Here the situation is one of eliminating unhelpful solutions from the case-base. Unfortunately both are computationally intractable since they are NP-hard search problems. It is interesting to note though that in general, the pattern is for smaller, more relevant case-bases to substantially outperform larger less focused ones.

Approaches to searching for subsets fall into two categories: filters and wrappers [13]. Filters operate independently of the CBR algorithm reducing the number of features prior to training. By contrast, wrappers use the CBR algorithm itself on some sample of the data set in order to determine the fitness of the subset. This tends to be computationally far more intensive, but generally can find better subsets than the filter methods. Various wrapper methods have been investigated by a number of researchers. Early versions of ANGEL [14] addressed the problem of searching for the optimal feature subset by an exhaustive search using a jack knife[2] on the case base in order to determine fitness. However, as previously stated, the search is NP-hard so once the number of features exceeds 15-20 this becomes computationally intractable. Other approaches have included different variants of hill climbing algorithms [16], sequential feature selection algorithms, both forward and backward [17] and genetic algorithms [18]. These have generally been reported to lead to good improvements in solution quality without the prohibitive computational cost of an exhaustive search.

Essentially all these methods have a search component to generate candidate subsets from the space of all possible subsets and a fitness function which is a measure of the error derived from the solution proposed by the CBR system using the subset, trained on a sample from the data set and validated on a holdout sample. Typical sampling techniques are the jack knife and *n*-fold[3] validation. The fitness function is generally a measure of deviation between the proposed and desired solution, and as such is a cost that should be minimised. The exact nature of the measure will depend upon the nature of what is being predicted but is usually either based on the cost of misclassifications or the sum of absolute residuals.

9.2.4 Adaptation

Another important aspect of CBR is adaptation of the solution, particularly when even the most similar cases differ substantially from the target case. This might occur if the case-base is small or heterogeneous. The simplest approach, that of *k*-NN systems, is to use the solution of the nearest neighbour, or mean (possibly distance weighted so that the nearest solutions are most influential) of several neighbours. Hanney and Keane [19] describe an interesting alternative, which

[2] A jack knife is a validation strategy that works by successively holding out each case, one at a time, and using the remainder of cases to generate the prediction for the hold-out case [15]

[3] *n*-fold validation is another common validation procedure within the machine learning community whereby the data set is divided into *n* approximately equal subsets. Each subset is successively held-out and then returned to the training set. This process is repeated *n* times so that each case forms part of the hold-out set exactly once. This is a generalization of the jack knife where *n* is the total number of cases in the case-base.

learns how to adapt by comparing feature differences and solution differences. Unfortunately, this structural approach is limited to linear, or near linear problems. Another widely used adaptation strategy is the use of rules to modify proposed solutions. The difficulty here is that the motivation for using CBR in the first place is often the challenge of performing knowledge elicitation, so where do the rules come from [6]? Whilst Watson and Marir [3] identify a number of additional adaptation strategies, *k*-NN and rule based approaches are the most popular.

9.2.5 Unsuited Problem Domains

So far this section has focused on the successful application of CBR technology. It is, however, also important to stress that there are problem domains that are not so well suited to CBR. These can be characterised by one or more of the following:

1. lack of relevant cases, for example when dealing with an entirely new domain. In truth, such situations will be extremely resistant to solution by any technique, though one possibility is a divide and conquer strategy so whilst the problem may be novel in its entirety, it may be that useful analogies may be sought for some, or all, of its constituent parts.
2. few cases available due to a lack of systematically organised data, typically due to information not being recorded or being primarily in a natural language format. CBR does not deal well with large quantities of unstructured text[4].
3. the problem domain can be easily modelled and is well understood, for example when regression techniques can find simple structural equations that have high explanatory power. In such circumstances it would seem wiser to use the model based technique.

This overview has been necessarily brief. For more detail, the reader is referred to the classic book by Kolodner [22], more recent works such as Althoff [23], Bergmann [24] and for a comparison of different approaches, to the paper by Finnie and Sun [25].

[4] This not to say there has been no research into textual CBR. Much work has focused on the extraction of pre-determined features. Where the set of features required for describing each case varies greatly, then an interactive CBR method (see for example, Aha *et al.*, [20, 21]) may be useful for guiding the author through the elicitation process (i.e. through a series of prompted questions whose answers assign values to relevant attributes). One advantage of this method is that it can help avoid some standard problems with information retrieval systems (e.g. how to interpret text expressions that have multiple potential meanings) by clarifying the lesson writer's inputs during elicitation. However, in general natural language processing (NLP) remains an extremely intractable problem.

9.3 Software Engineering Applications of CBR

Having considered case-based reasoning in general we now turn to its application to problems drawn from the domain of software engineering. Broadly speaking, this work falls into two categories: prediction and reuse type applications. We discuss each in turn.

9.3.1 Prediction in Software Engineering

It has long been recognised that a major contribution to successful software engineering is the ability to be able to make effective predictions particularly in the realms of costs and quality. Consequently there has been significant research activity in this area, much of which has focused on effort and defect prediction. Both these problems are characterised by an absence of theory, inconsistency and uncertainty that make them well suited to CBR approaches.

It was suggested in the early 1980s that analogy might form a good basis for software project effort prediction [26]. However, the earliest work to formalise this process was by Vicinanza and co-workers [27]. They developed a CBR system with rule-based adaptation named Estor. This involved knowledge elicitation from a domain expert — an experienced software project manager — to derive adaptation rules. They reported encouraging results based, upon a small industrial dataset of 15 projects [28]. Estor was comparable to the expert and significantly more accurate than COCOMO model [26] or function points [29]. However, their approach requires access to an expert in order to derive estimation rules and create a case-base. Also the rules are couched in terms of the particular set of features in Kemerer's data set that severely limits their applicability as there are wide discrepancies in the range and types of features collected by different software organisations.

Another early project [30] entitled FACE (Finding Analogies for Cost Estimation) also used CBR technology and reported results based upon another publicly available data set, COCOMO [26]. The authors reported accuracy levels of MMRE[5] = 40-50%, however, the system was only able to make predictions for 46 out of a total of 63 projects. By contrast, Finnie *et al.* [31] reported good results using CBR with adaptation rules for a large industrial data set of 299 projects, split into a training set of 249 projects and a validation set of 50 projects. Their CBR approach proved to be significantly more accurate than a regression-based approach and comparable with an artificial neural net (ANN), with the added advantage of better explanatory value than the ANN. As with the Vicinanza work, the disadvantage of this approach is that new adaptation rules must be derived for new data sets.

[5] MMRE or mean magnitude of relative error is a widely used accuracy indicator by software project cost researchers. It is defined as $1/n \sum \text{abs}((\text{act}_i\text{-pred}_i)/\text{act}_i)$ where i is the ith prediction and there are a total of n cases. One disadvantage of MMRE is that it is asymmetric, nevertheless it is widely quoted.

At the same time a simpler approach was being pursued by Shepperd and others [32, 14] based on the idea of a *k*-NN system named ANGEL. The work was guided by the twin aims of expediency and simplicity so as to make the approach as widely applicable as possible whilst at the same time providing transparency in order to increase trust by project managers. Similarity was defined in terms of Euclidean distance between arbitrary sets of project features, such as number of interfaces, development method, application domain and so forth. The number and type of features chosen could depend upon what data is available to characterise projects. The authors reported having analysed datasets with as few as one feature and as many as 29 features. Features could be either categorical or continuous and are standardised so that each feature has equal influence. The other distinctive characteristic of the ANGEL approach is the implementation of an automated feature subset selection search.

As per Finnie *et al.*, Shepperd and co-workers used stepwise regression analysis as a benchmark for evaluating the predictive performance of ANGEL.

Table 9.1 Comparison of CBR and Regression Effort Prediction Accuracy (Adapted from Shepperd and Schofield [32])

Data Set	Source	No. of Cases	No. of Features	ANGEL (MMRE)	Stepwise Regression (MMRE)
Albrecht	[29]	24	5	62%	90%
Atkinson	[33]	21	12	39%	45%
Desharnais	[34]	77	9	64%	66%
Finnish	Finnish Dataset: dataset made available to the ESPRIT Mermaid Project by the TIEKE organisation	38	29	41%	101%
Kemerer	[28]	15	2	62%	107%
Mermaid	MM2 Dataset: Dataset made available to the ESPRIT Mermaid Project anonymously	28	17	78%	252%
Real-time 1	Not in the public domain	21	3	74%	N/A.
Telecom 1	[32]	18	1	39%	86%
Telecom 2	Not in the public domain	33	13	37%	142%

Table 9.1 summarises the results from an empirical evaluation of ANGEL based upon 9 different data sets. It can be seen that for these data sets the *k*-NN approach consistently outperformed regression-based models. Subsequent studies have reported more mixed experiences. A study of software maintenance effort [35] found similar results. However, other researchers, most notably [36, 37] obtained conflicting results where the regression model generated significantly better results than the ANGEL based approach. While there are some differences in implementation, in particular [36] used a different procedure to select the best feature

subset based on a filter, this does not fully explain differences in the results. Doubtless, the underlying characteristics of the problem data set are likely to exert a strong influence upon the relative effectiveness of different prediction systems. For example, the two datasets [36] used, both appear to contain well-defined hyperplanes such that the regression procedures are able to generate models with good explanatory power as evidenced by the high R squared values. One would not expect cased-based reasoning to perform well since instead of interpolating or extrapolating it endeavours to draw data points to the nearest cluster. Clearly this is not an effective strategy if the data falls upon, or close to, a hyperplane. In other words, a linear function exists that "explains" the relationship between the dependent variable and the independent variables.

Recent work has shown that the difficulties with feature and case subset selection for large data sets can be overcome using search metaheuristics, for example random mutation hill climbing and forward and backward selection search, drawn from the artificial intelligence community [38]. These techniques resulted in substantial improvements in the performance of ANGEL, typically from an MMRE of in excess of 50% down to 15%.

Despite these advances, CBR prediction of effort is still an uncertain process with quite variable levels of accuracy. This should not be too surprising as the pursuit of a "best" or universal prediction technique is unlikely to be a fruitful quest. Probably what is most encouraging is the results of an experiment on professional project managers that found that *k*-NN (ANGEL) augmented by expert judgement led to the most accurate effort prediction [37].

Another prediction problem that has been tackled with CBR technology is classifying software components into low and high levels of defects [12]. The authors report a success rate in excess of 85% when studying a military command, control, and communications system. One interesting aspect of this work is their use of fuzzy rather than crisp values to describe case features coupled with fuzzy logic to assess similarity. Fuzzy logic is a form of logic used in some systems in which feature set membership can be described in terms of degrees of truthfulness or falsehood represented by a range of values between 1 (true) and 0 (false). For example, a software component might be described as belonging to the set of large components to a degree 0.8, in other words it is believed to be quite large. Note this is quite different from making a probabilistic statement where p=0.8 that the component is large. Set membership may also overlap so we might also have the same component with a membership of the set of medium components to the degree 0.3. Since we are not dealing with probabilities there is no requirement for the degrees of set membership to sum to unity.

9.3.2 Reuse in Software Engineering

The concept of reuse, within software engineering, has long been acknowledged as an important potential source of productivity gain. Moreover, reuse has been seen in a much broader sense than just software or code artefacts to include designs, patterns, specifications, processes and software project experience in general. Reuse is perceived as a natural application for CBR since exact matching is generally very difficult to achieve because it is precisely the *difference* between software projects that makes software engineering a challenging discipline. Instead the problem is to retrieve *similar* components.

An early contribution was by Maiden and Sutcliffe [39, 40] who suggested that analogical reasoning techniques might be employed to support the reuse of software specifications. This was achieved by mapping both the target and source (case-base) requirements specification descriptions into more abstract representations to facilitate the measurement of similarity. In this system a domain model of requirements is based on object structural knowledge, actions, object types, pre and post condition constraints on state transitions, transformations that lead to state transitions and events that trigger transformations. To determine if two requirements are similar, Maiden and Sutcliffe compare the domains using four different dimensions (semantic, structural, pragmatic and abstract) utilizing a structural coherence algorithm. The target requirement is compared to the requirements in the abstract domain hierarchy to form a set of possible matches. Next a heuristic-based abstraction selector is used to select the best abstract domain from the candidate set. Two domains are considered similar only if they share the same abstract domain class.

Another early application of CBR technology was to support the reuse of software packages within Ada and C program libraries [41]. They used a distance measure based on a combination of semantic networks (providing conceptual connectivity) and the faceted index approach (that allows the user a view from different perspectives) [42] and demonstrated their ideas with a prototype system and some examples. Interestingly, the authors also noted another potential application in the form of Basili's Goal Question Metric framework [43] together with process reuse.

The most ambitious form of CBR-supported reuse is that of experience reuse, in other words to explicitly learn from past software projects and to make the lessons widely available through sophisticated retrieval mechanisms using similarity metrics. Such metrics are important due to the difficulty of finding exact project matches within the domain of software engineering. Of course the idea of experience reuse, or what is often termed a lessons learned (LL) system is not unique to software engineering. For an interesting review of LL systems in commercial, government and military applications see Weber *et al.* [44].

Much of the motivation for experience reuse within the domain of software engineering stems from Basili's ideas of an Experience Factory [45] although other researchers have reached similar conclusions, for example Grupe *et al.* [46]. An Experience Factory (EF) is based upon a number of premises:

1. a feedback process is required to best support learning and improvement.
2. experience must be viewed as a resource for an organisation and therefore stored appropriately in an experience-base.
3. experience must be appropriately packaged in order to support appropriate reuse, for example it might be unwise to reuse the successful experiences of writing game software when developing a protection system for a nuclear reactor.

4. mechanisms must be provided to support the retrieval of experience packages.

These ideas are closely aligned with CBR technology so that it is no surprise that many researchers have seen organisational learning as a natural application, see for example Tautz and Althoff [47]; von Wangheim *et al.* [48]. The Quality Improvement Paradigm (QIP)/EF provides a framework for continuous learning about software engineering practices and techniques. In other words it provides "an organisational infrastructure necessary for operationalising CBR systems in industrial environments" [49].

In order to make a reuse decision it is necessary to characterise (1) the technology, (2) the goal and (3) the context or domain in which the technology will be applied, e.g. developer experience. The context is particularly emphasised because the diversity of software engineering activities and problem domains might otherwise result in appropriate reuse. The context often is assessed subjectively e.g. on a five point scale. Typically a project is viewed as a case. This implies the following process:

1. Decide upon the task and goal. This will determine the relevant context features.
2. Characterise the new project (case) in terms of relevant features.
3. Perform a similarity based retrieval of other projects. The retrieval may be in two stages, first use a clustering or filter approach to find broadly similar projects and then second use a distance metric.
4. Adaptation of the most relevant retrieved case(s) since it may not be possible to use the retrieved experience directly.
5. Perform the project
6. Evaluate the project based on empirical evidence collected during the running of the project. Empirical evidence is encouraged in order to promote objectivity.
7. Identify lessons learned that can be added to the experience or case-base

Two features distinguish the EF from many more general LL systems. First, there is the explicit notion of context. Second, there is the use of empirical evidence in order to evaluate potential new cases. These address some of the reported problems of poor usage rates for deployed LL systems by Weber *et al.* [44] such as difficulties in retrieving *relevant* cases and validation of experience prior to storing within the LL system.

Maintenance of the EF is another challenge in order to avoid obsolete, inconsistent, unvalidated or subjective, irrelevant and redundant cases. Weber *et al.* report on a number of LL systems that contain in excess of 30000 cases or lessons. Interestingly, in an example of case-base maintenance they describe how it was possible to reduce from 13000 cases to 2000 cases.

For further information on the topic of EFs see chapter 13 entitled "Making Software Engineering Competence Development Sustained through Systematic Experience Management" within this book.

9. 4 Summary and Future Work

In this chapter we have seen how case-based reasoning is a relatively recent technology that has emerged from the artificial intelligence and cognitive science communities. It is based on the idea of memory rather than explicit models. It would also seem to fit closely with how humans often solve problems, that is by means of analogy [50]. This is important as it can help users to trust CBR systems and, potentially, to better interact with them. We have also seen that CBR approaches do not require a deep understanding of the problem domain, which suggests they are well suited to many software engineering problems. This is because we are dealing with creative processes, complexity, change and uncertainty. There is also a strong sense within software engineering circles that reuse is important. Again CBR is appropriate since it provides a mechanism of organising, storing and reusing an organisation's memory or experiences. Thus it is unsurprising that a major application area is that of implementing experience-bases. The other principal area is that of prediction. Here CBR is more seen as another machine learning, or inductive technique, but one that has good explanatory value and with which the user can interact.

Whilst there are undoubtedly exciting opportunities for the deployment of CBR methods there remain many challenges.

First, is the challenge of adaptation. As has been seen from the examples discussed in this chapter, there are two main approaches for adaptation. One is rule based, which can embody substantial domain knowledge, but suffers from specificity to a particular case-base plus there are the difficulties of elicitation. Rule induction techniques may help overcome the latter problem. The other approach is to use simple arithmetic techniques and rely more on feature and case subset selection. This approach can be particularly vulnerable to novel problems.

Second, is the challenge of constructing cases from richer sources of data. Many of the software engineering applications described above are restricted to simple numeric information. Even categorical features can be troublesome. There has been a range of work looking at textural CBR. Some researchers, for example Grupe *et al.* [46] have looked at using textural information by means of trigrams. Others have deployed a range of other information retrieval techniques. Nevertheless in a recent survey, Weber *et al.* [44] comment "our survey reinforced that the two most evident problems contributing to the ineffectiveness of LL systems concern text representations for lessons and their standalone design. Text formats are troublesome for computational treatment, and attempts to create structure in records have rarely addressed core issues, such as highlighting the reuse component of a [case]." (p32). Perhaps markup languages such as XML may also be a means of dealing with semi-structured data. Aha and Wettscherek [51] argue that CBR should move beyond simple vector based approaches and consider a range of richer forms of case representation such as directed graphs, preference pairs and Horn clauses. Whatever approach, making use of richer sources of information is likely to be extremely fruitful when considering the range of data that is typically available in software engineering projects and a growing research topic.

The third challenge is that of finding better ways to support collaboration between the human expert and the CBR system. In the past, in some quarters, there has been a tendency to view many of these systems as replacements for the human. For many applications, particularly when dealing with infrequent but high value problems such as experience factory supported decision-making and project prediction, this view may be inappropriate and therefore we should explicitly address the problem of how to bring about the most effective forms of interaction between the human and the CBR system. Given the findings of Weber *et al.* [44] of the limited impact of deployed lessons learned systems this final challenge is of great significance to the practical benefits of CBR systems.

References

1. Schank, R., *Dynamic Memory: A theory of reminding and learning in computers and people*, Cambridge, UK: Cambridge University Press, 1982.
2. Leake, D., *Case-based reasoning: experiences, lessons, and future directions*, Menlo Park, CA: AAAI Press, 1996.
3. Watson, I. and F. Marir, Case-Based Reasoning: A Review. *The Knowledge Engineering Review* 9(4): pp327-354, 1994.
4. Case-Based Reasoning Homepage, University of Kaiserslautern. Available from: www.cbr-web.org, [Accessed December, 4 2002].
5. Success stories, INRECA Center, University of Kaiserslautern. Available from: www.inreca.org/data/cbr/success.html, [Accessed December, 4 2002].
6. Leake, D., *CBR in context: the present and the future*, in *Case based reasoning: experiences, lessons and future directions*, Leake, D., Editor, AAAI Press: Menlo Park. pp1-35, 1996.
7. Aamodt, A. and E. Plaza, Case-based reasoning: foundational issues, methodical variations and system approaches. *AI Communications* 7(1), 1994.
8. Little, R.J.A. and D.B. Rubin, *Statistical Analysis with Missing Data.* 2nd ed, New York: John Wiley & Sons, 2002.
9. Kadoda, G., M. Cartwright and M.J. Shepperd. Issues on the effective use of CBR technology for software project prediction. *4th Intl. Conf. on Case based Reasoning*, Vancouver, 2001.
10. Aha, D.W. Case-based learning algorithms. *1991 DARPA Case-Based Reasoning Workshop*: Morgan Kaufmann, 1991.
11. Bergmann, R. and S. Stahl. Similarity Measures for Object-Oriented Case Representations. *8th European Workshop on Case-Based Reasoning (EWCBR'98)*, 1998.
12. Schenker, D.F. and T.M. Khoshgoftaar. The application of fuzzy enhanced case-based reasoning for identifying fault-prone modules. *3rd IEEE International High-Assurance Systems Engineering Symposium.* pp90-97, Washington, D.C.: IEEE Computer Society, 1998.
13. Kohavi, R. and G.H. John, Wrappers for feature selection for machine learning. *Artificial Intelligence* 97: pp273-324, 1997.

14. Shepperd, M.J., C. Schofield and B.A. Kitchenham. Effort estimation using analogy. *18th Intl. Conf. on Softw. Eng.* pp170-179, Berlin: IEEE Computer Press, 1996.
15. Efron, B. and G. Gong, A leisurely look at the bootstrap, the jackknife and cross-validation. *The American Statistician* 37(1): pp36-48, 1983.
16. Skalak, D.B. Prototype and feature selection by sampling and random mutation hill climbing algorithms. *11th Intl. Machine Learning Conf. (ICML-94).* pp293-301: Morgan Kauffmann, 1994.
17. Aha, D.W. and R.L. Bankert, *A comparative evaluation of sequential feature selection algorithms*, in *Artificial Intelligence and Statistics V.*, Fisher, D. and Lenz, J.-H., Editors, Springer-Verlag: New York, 1996.
18. Whitley, D., J.R. Beveridge, C. Guerra-Salcedo and C. Graves. Messy genetic algorithms for subset feature selection. *International Conference on Genetic Algorithms, ICGA-97.*, 1997.
19. Hanney, K. and M.T. Keane. The adaptation knowledge bottleneck: how to ease it by learning from cases. *2nd Intl. CBR Conf.* pp359-370, 1997.
20. Aha, D.W. and L.A. Breslow, *Refining conversational case libraries*, in *Case-Based Reasoning Research and Development*, Leake, D. and Plaza, E., Editors, Springer-Verlag: Berlin. pp267-278, 1997.
21. Aha, D.W., T. Maney and L.A. Breslow, *Supporting dialogue inferencing in conversational case-based reasoning*, in *Advances in Case-Based Reasoning*, Smyth, B. and Cunningham, P., Editors, Springer-Verlag: Berlin. pp262-273, 1998.
22. Kolodner, J.L., *Case-Based Reasoning*: Morgan-Kaufmann, 1993.
23. Althoff, K.-D., *Case-Based Reasoning*, in *Handbook on Software Engineering and Knowledge Engineering. Vol. 1 "Fundamentals"*, Chang, S.K., Editor, World Scientific. pp549-588, 2001.
24. Bergmann, R., *Experience Management - Foundations, Development Methodology, and Internet-Based Applications.* Lecture Notes in Artificial Intelligence. Vol. LNAI 2432, Berlin: Springer Verlag, 2002.
25. Finnie, G.R. and Z. Sun, R^5 model for case-based reasoning. *Knowledge-Based Systems* 16(1): pp59-65, 2002.
26. Boehm, B.W., *Software Engineering Economics*, Englewood Cliffs, N.J.: Prentice-Hall, 1981.
27. Mukhopadhyay, T., S.S. Vicinanza and M.J. Prietula, Examining the feasibility of a case-based reasoning model for software effort estimation. *MIS Quarterly* 16(June): pp155-171, 1992.
28. Kemerer, C.F., An empirical validation of software cost estimation models. *Communications of the ACM* 30(5): pp416-429, 1987.
29. Albrecht, A.J. and J.R. Gaffney, Software function, source lines of code, and development effort prediction: a software science validation. *IEEE Transactions on Software Engineering* 9(6): pp639-648, 1983.
30. Bisio, R. and F. Malabocchia. Cost estimation of software projects through case base reasoning. *1st Intl. Conf. on Case-Based Reasoning Research & Development.* pp11-22: Springer-Verlag, 1995.

31. Finnie, G.R., G.E. Wittig and J.-M. Desharnais. Estimating software development effort with case-based reasoning. *2nd Intl. Conf. on Case-Based Reasoning*. pp13-22, 1997.
32. Shepperd, M.J. and C. Schofield, Estimating software project effort using analogies. *IEEE Transactions on Software Engineering* 23(11): pp736-743, 1997.
33. Atkinson, K. and M.J. Shepperd. The use of function points to find cost analogies. *5th European Software Cost Modelling Meeting*, Ivrea, Italy, 1994.
34. Desharnais, J.M., *Analyse statistique de la productivitie des projets informatique a partie de la technique des point des fonction*, University of Montreal, 1989.
35. Niessink, F. and H. van Vliet. Predicting maintenance effort with function points. *Intl. Conf. on Softw. Maint.* pp32-39, Bari, Italy: IEEE Computer Society, 1997.
36. Briand, L., T. Langley and I. Wieczorek. Using the European Space Agency data set: a replicated assessment and comparison of common software cost modeling techniques. *22nd IEEE Intl. Conf. on Softw. Eng*. pp377-386, Limerick, Ireland: Computer Society Press, 2000.
37. Myrtveit, I. and E. Stensrud, A controlled experiment to assess the benefits of estimating with analogy and regression models. *IEEE Transactions on Software Engineering* 25(4): pp510-525, 1999.
38. Kirsopp, C., M.J. Shepperd and J. Hart. Search heuristics, case-based reasoning and software project effort prediction. *GECCO 2002: Genetic and Evolutionary Computation Conf.*, New York: AAAI, 2002.
39. Maiden, N.A., Analogy as a paradigm for specification reuse. *Software Engineering Journal* 6(1): pp3-15, 1991.
40. Maiden, N.A. and A.G. Sutcliffe, Exploiting reusable specifications through analogy. *Communications of the ACM* 35(4): pp55-64, 1992.
41. Ostertag, E., J. Hendler, R. Prieto-Díaz and C. Braun, Computing similarity in a reuse library system: an AI-based approach. *ACM Transactions on Software Engineering Methodology* 1(3): pp205-228, 1992.
42. Priéto-Diaz, R. and P. Freeman, Classifying software for reusability. *IEEE Software* 4(1): pp6-16, 1987.
43. Basili, V.R. and H.D. Rombach, The TAME project: Towards Improvement-oriented software environments. *IEEE Transactions on Software Engineering* 14(6): pp758-771, 1988.
44. Weber, R., D.W. Aha and I. Becerra-Fernandez, Intelligent lessons learned systems. *Expert Systems with Applications* 20(1): pp17-34, 2001.
45. Basili, V.R., G. Caldiera and H.D. Rombach, *Experience factory*, in *Encyclopaedia of Software Engineering*, Marciniak, J.J., Editor, John Wiley & Sons: New York. pp469-476, 1994.
46. Grupe, F.H., R. Urweiler, N.K. Ramarapu and M. Owrang, The application of case-based reasoning to the software development process. *Information & Software Technology* 40(9): pp493-500, 1998.

47. Tautz, C. and K.-D. Althoff. Using case-based reasoning for reusing software knowledge. *2nd Intl. Conf. on Case-Based Reasoning*, Providence, RI: Springer-Verlag, 1997.
48. von Wangenheim, C.G., K.D. Althoff and R.M. Barcia, *Goal-oriented and similarity-based retrieval of software engineering experienceware*, in *Learning Software Organizations: Methodology and Applications (LNCS, 1756)*, Ruhe, G. and Bomarius, F., Editors, Springer-Verlag: Berlin. pp118-141, 2000.
49. Althoff, K.-D., A. Birk, C.G. von Wangenheim and C. Tautz, *Case-Based Reasoning for Experimental Software Engineering*, in *Case-Based Reasoning Technology – From Foundations to Applications*, Lenz, M., et al., Editors, Springer Verlag: Berlin. pp235-254, 1998.
50. Klein, G., *Sources of Power: How People Make Decisions*, Cambridge, Ma: MIT Press, 1998.
51. Aha, D.W. and D. Wettscherek. Case-based learning: beyond classification of feature vectors. *European Conf. on Machine Learning (ECML-97)*, 1997.

Biographical details:

Martin Shepperd has a chair of software engineering at Bournemouth University, UK. He received his PhD in computer science in 1991 from the Open University, UK. His main research interests are empirical aspects of software engineering and machine learning. He has published more than 75 papers and 3 books. Presently he is co-editor of the journal *Information & Software Technology* and associate editor of *IEEE Transactions on Software Engineering.*