Software Development Effort Estimation using CBR: A Review

K.S Patnaik, S. Malhotra,
Department of Computer Science and Engineering
Birla Institute of Technology, Mesra
Ranchi-835214. (INDIA)

Bibhudatta Sahoo
Department of CSE
National Institute of Technology
Rourkela-769008 (INDIA)

Abstract

The software community acknowledges that timely, accurate estimates of development effort are important to the success of major software projects. Estimates are used when tendering bids, for evaluating risk, resource scheduling and progress monitoring. Inaccurate estimates have implications for all these activities, and so, ultimately, the success or failure of a project. To date most work has focused upon building algorithmic models of effort, for example COCOMO (Constructive Cost Model). These can be calibrated to local environment. This paper presents an alternative approach to estimation based upon the use of analogies (or CBR). The underlying principle is to characterize projects in terms of features (e.g. number of interfaces, the development method etc.). Completed projects are stored and then the problem becomes one of finding the most similar projects to the one for which prediction is required. Similarity is defined as Euclidean distance in n-dimensional space where n is the number of project features. Each dimension is standardized so all dimensions have equal weight. The known effort values of the nearest neighbors to the new project are then used as the basis for the prediction. Our analysis is based on IBM dataset of 24 projects. Our results show that estimation by analogy is a viable technique that, at the very least, can be used by the project mangers to complement current estimation techniques.

Keywords *– Software project,, case- based reasoning, similarity, effort prediction , cost estimation, analogy.*

Introduction

An important aspect of any software development project is to know how much it will cost. In most cases the major cost factor is labor. For this reason estimating development effort is central to the management and control of a software project. Over the past 20 years case-based reasoning (CBR) has been successfully applied to a wide range of problem domains. Our particular interest is in predicting effort (and related factors such as duration) for software projects .Of course, such predictions are required at the early stage of software life cycle. Effort estimation is problematic because it is difficult to derive accurate size and cost figures from the features of a project that are known early in the development process. This is important because software projects are difficult to justify or manage if it isn't possible to estimate how long they'll last and how much effort they'll consume. For this reason software development effort estimation modeling has been an active research topic for more than 30 years. Despite this activity, no one technique has been found to be consistently effective. Various research groups have explored the application of CBR methods to predicting effort, motivated in part by obvious similarities between project managers seeking to estimate based on recall of past similar projects, and the formal use of analogies in CBR[11][12][14][15].

Case-Based Reasoning

Case-based reasoning (CBR) was formalized in 1980s following from the work of Schank and others on memory. According to Riesbeck and Schank

" *A case-based reasoner solves new problems by adapting solutions that were used to solve old problems*"

It is based upon the fundamental premise that similar problems are best solved with similar solutions. According to Leak

"Case-based reasoning is reasoning by remembering".

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 casualties and relationships within the domain of interest. For poorly understood problem domains this is major benefit.

CBR is technique for managing and using knowledge that can be organized as discrete abstraction 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 characterized 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 repository of past (and therefore solved) cases. The repository is termed case-base.

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

- Avoids many problems associated with knowledge elicitation and codification.
- Only needs to address those problems that actually occur, whilst generative (i.e. algorithmic) systems must handle all possible problems.
- Handles failed cases, which enable users to identify potentially high-risk situations.
- Copes with poorly understood domains (for example, many aspects of software engineering) since solutions are based upon what has actually happened as opposed to hypothesized models.
- 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 USA and UK. 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 the 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.

As previously indicated, case-based reasoning has its heart the notion of utilizing the memory of past problems solved to tackle new problems[1]. Problems are organized as cases where each case comprises of two parts. These are the description part and a solution part. The description part is normally a vector of features that describes the solution for the specific problem and may vary in complexity from a single value for classification or prediction system to a set of rules or procedures to derive a solution that might include a range of multimedia objects such as video and sound files.

The Basic CBR cycle

Case-based reasoning(Klodner 1992; Watson & Marir 1994; Slade 1991) is a relatively simple concept- it involves matching the current problem against ones that have already been encountered in the past and reworking the solutions of the past problems in the current context. It can be represented as a cyclical process that is divided into four following sub processes depicted in Figure 1(Aamodt & Plaza 1994):

- Retrieve the most similar cases or case from the case base.
- Reuse the case to solve the problem
- Revise the proposed solution, if necessary.
- Retain the solution for future problem solving.

[1] Strictly speaking some authors differentiate between interpretive and problem solving CBR. Interpretive CBR focuses upon classification rather than direst problem solving, although it could always be argued that classification can be viewed as a sub goal to solving another problem. Whatever, this is not a distinction that is pursued in this chapter.

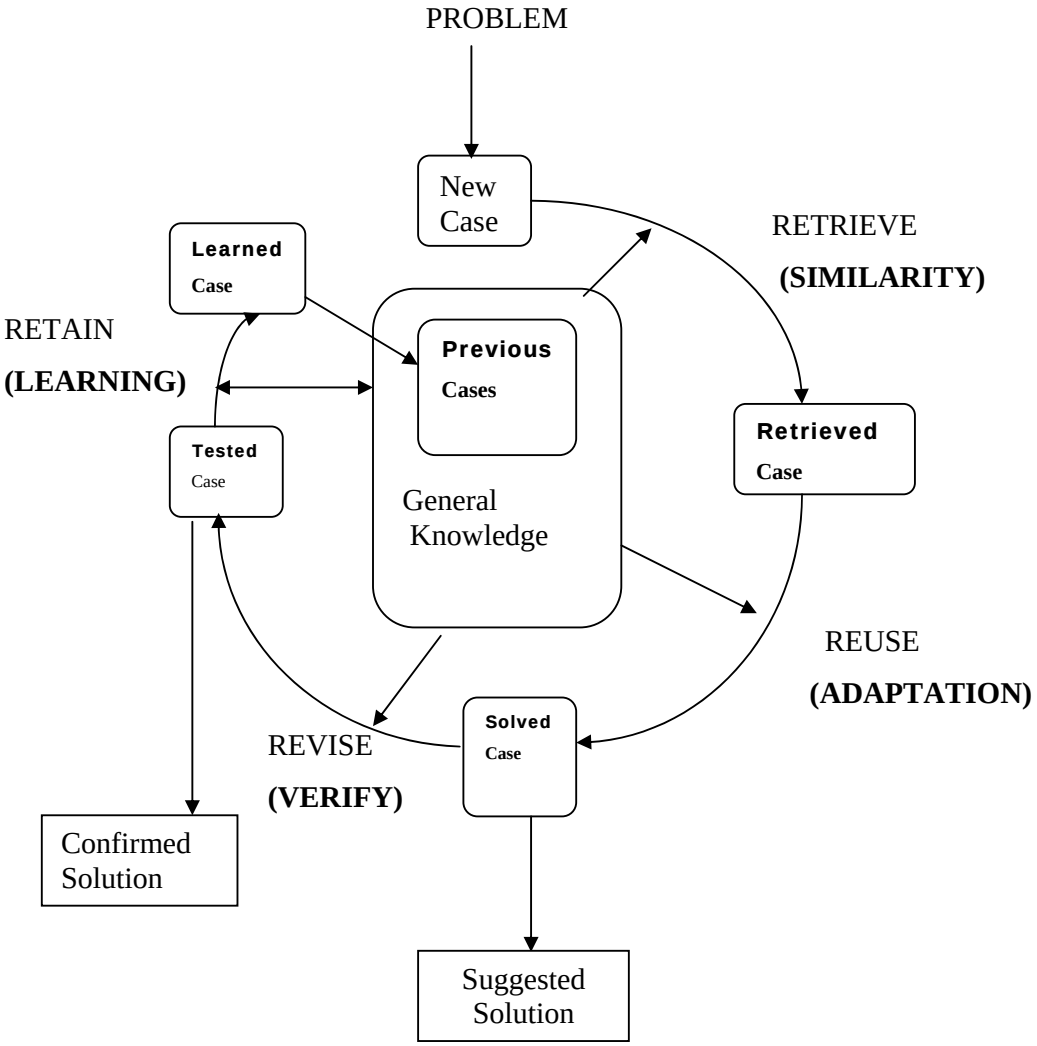

Fig 1: the CBR cycle

A new problem, described as a case, is compared to the existing cases in the case base and the most similar case or cases are retrieved. These cases are combined and reused (i.e. adapted) to suggest a solution for the new problem. The solution proposed may need to be revised (i.e. evaluated and corrected) somewhat if it is not a valid solution. This verified solution is retained by adding it a new case to the case base or as amendments to existing cases in the case base for use in future problem solving.

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 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 that are being explored. 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 target and utilize the known solution of the nearest *k* cases. 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 neighbor (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.

Proposed CBR algorithm for cost estimation

CBR offers a number of advantages over the other cost estimation techniques. The developers of the various algorithmic models have attempted to derive models that quantify the casual dependencies within the domain. As these models do not effectively solve the problem, this suggests that domain is difficult to model without clear rules or a clear understanding of all the different elements that contribute to cost estimation. The main advantage of CBR over the use of algorithmic models is that the use of CBR avoids the need to model the domain.

One of the difficulties with any of the existing techniques is the lack of project history data within organizations. This data is used to calibrate algorithmic models and for comparison purposes in estimation by analogy. A lack of data also means that problem space is not uniformly covered. There may be a greater intensity of sample projects in some areas than in others. There may be other areas of the problem space where there is no past project data available. For techniques such as algorithmic techniques and estimation by analogy this makes deriving general models from data unworkable. CBR however will use the data that is relevant and available to make prediction. Furthermore, a case base will also provide an effective means of storing data on historical projects.

CBR also has the advantage of possessing the capability to explain its reasoning. It is possible to view the cases, which are retrieved as similar to the target case, and to view the cases, which are retrieved as similar to the target case, and to view the adaptation strategies that operate on the retrieved cases, which result in prediction. It also allows manual adaptation so an expert can extrapolate from the similar retrieved cases and adjust the recommended solution if they feel it necessary.

Lastly, as CBR is a machine learning technique, a CBR system will augment its case base with new project scenarios over time. This is important, as the software development process is a constantly changing process with new technologies, new methods, and new techniques continually being introduced and adopted. A good cost estimation technique needs to be able to handle this natural evolution of software development.

Case based reasoning has four distinct aspects:

- Characterization of cases
- Storage of past cases

- Retrieval of similar cases to use as analogies
- Utilizing the retrieved case to solve the target case problem, sometimes known as case adaptation.

First and foremost important in CBR is Case representation. We have *n* projects or cases, each of which needs to be characterized in terms of a set of *p* features. In addition, we must also know the feature that is to be predicted. Features can either be continuous (e.g. experience of the project manager), discrete (e.g. the number of interfaces) or categorical (e.g. development environment). In practice, many approaches treat discrete features as if they were continuous. Historical project data is collected and added to the case base. When a prediction is required for a new project this project is referred to as target case. The target case is also characterized in terms of the *p* features. Incidentally this imposes a constraint on the feature set in that it should only contain features for which the values will be known at prediction time.

FEATURES
IN
OUT
INQ
FILE
ADJ
Effort

Fig 2:Case Representation for IBM Dataset

These are the features used for Function Point Analysis [1][4][5][6], which are known at prediction time (i.e. time at which requirement document is being written).

IN is the number of inputs. Each data input is counted. Data inputs are distinguished from user inquiries that are counted separately.

OUT is the number of outputs. Output refers to reports, screen and error messages etc. Individual data items within a report are counted separately.

INQ is the number of inquiries, which refers to number of distinct interactive queries made by the user, which require specific action by the system.

FILE is the number of files. Each logical file, e.g. groups of logically related data, is counted as a file.

ADJ is the adjusted value. It depends upon the environment in which the product is to be made.

Effort is the actual effort obtained.

The next step is to measure the similarity between the target case and other cases in *p*-dimensional feature space. There are variety of approaches including a number of preference heuristics proposed to measure similarity.

- *Nearest Neighbor Algorithms*: these are the most popular and are either based upon straightforward distance measures or the sum of squares of differences for each variable. In either case each variable must be first standardized (so that it has an equal influence) and then weighted according to the degree of importance attached to the feature. Standardization is done so that choice of unit has no influence. A common algorithm is given by Aha

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

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 1) the features are numeric, 2) if the features are categorical and $C_{1j} = C_{2j}$ and 3) where the features are categorical and $C_{1j} \neq C_{2j}$.

- *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 function in a similar fashion to query by example database interfaces, that is the user supplies the values for ranges, and all cases that match retrieved.
- *Goal directed preference*: Select cases that have the same goal as the current case.
- *Specificity preference*: Select cases that match features exactly over those that matches generally.
- *Frequency preference*: select cases that are most frequently retrieved.
- *Regency preference*: choose recently matched cases over those that have not been matched for a period of time.
- *Fuzzy similarity*: where concepts such as at-least-as-similar and just noticeable-difference are employed.
- *Object-oriented similarity:* for complex problem domains it may be necessary to make similarity comparisons between differently structured cases. In object-oriented approach cases are represented as collections of objects (each object has a set of feature-value pairs) organized in a hierarchy of part-of relationships.

The approach we adopted here is to allow estimators the freedom to utilize those features that they believe best characterize their projects and are most appropriate to their environments. Consequently, we used Euclidean distance in *p*-dimensional feature space (Nearest Neighbor Algorithm) as a means of measuring similarity between the cases.

The most similar cases or project are then used, possibly with adaptation to generate a prediction for the target case. We adopted a simple analogy adaptation strategy of the mean of the *k*-nearest neighbors, where *k is an integer such that 0<k< n.* when k=1 then the technique is a simple nearest neighbor method. As *k* tends toward *n* so the prediction approach tends towards merely using the sample mean. The estimator determines the choice of k.

Once the target case has been completed it can be added to the case base for future use in estimation. In this way the cost is estimated using CBR.

Limitations of CBR

- Feature and case subset selection- another difficulty for CBR, that is common to all machine learning approaches, is that 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 standardization used by some similarity measures can be more important than others, however, the standardization will assign each feature equal influence
- Similarity Measures- Similarity measures suffers 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 tends to be fairly crude in that they tend to adopt a Boolean approach: features match or fail to match with no middle ground. A second criticism of many of these similarity measures is that they fail to take into account information which can be derived from structures of data, thus, they are weak for higher order feature relationships such as one might expect see exhibited in legal systems.
- There are some problem domains that are not so well suited to CBR. One or more of the following can characterize these as follows:
    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 its constituents parts.

2. Few cases available due to lack of systematically organized data, typically due to information not being recorded or being primarily in a natural language format. CBR does not deal with large quantities of unstructured text.
3. The problem domain can be easily modeled and is well understood, for example when regression techniques can find simple structural equations that have high explanatory power. In such situation it would seem wiser to use the model based technique.

Results

A case representation capturing the available predictive feature is identified and presented from IBM dataset [6] having the features as shown in Fig 2. A jack- knifing procedure was adopted for the analogies based prediction in earlier research papers, but in our analysis 12 cases were taken as past project data and full CBR cycle was applied to the new cases (remaining 12 cases).

Accuracy is usually defined in the terms of mean magnitude of relative error (MMRE)[6], which is mean of absolute percentage errors:

$$\sum_{i=1}^{i=n}\left(\frac{|E-P|}{E}\right)_i \frac{100}{n}$$

or accuracy can defined as measure of mean absolute residual(MAR)[6], which is mean of absolute error.

$$\sum_{i=1}^{i=n}\left(|E-P|\right)\frac{1}{n}$$

where there are n projects, E is actual effort and P is the predicted effort. Yet another approach is to use Pred(25)[14]] which is the percentage of predictions that fall within 25 percent of the actual value. Clearly the choice of accuracy to a large extent depends upon the objectives of those using the prediction system. For example MMRE is fairly conservative with a bias against overestimates while Pred(25) will identify those prediction systems that are generally accurate but occasionally widely inaccurate. In this paper we have decided to adopt MMRE and MAR as prediction performance indicators since they are widely used, thereby rendering our results more comparable with those of other workers.

In this section we organize the results as

- Use of dissimilarity or distance information to form an effective adaptation strategy.
- Comparison of software project effort prediction using analogy with an algorithmic approach.

a*) Use of dissimilarity or distance information to form an effective adaptation strategy.*

This part of our investigation is concerned with relationship between nearest neighbor (k) and the MMRE. *k* can be dynamically determined as the number of cases that fall within distance *d*, of the target case. Table 1 reveals the accuracy (MMRE) with increasing neighbor (*k*).

Neighbor(*k*)	MMRE (%)
1	32.41
2	72.001
3	129.45
4	885.4

Table 1. *k vs.* MMRE

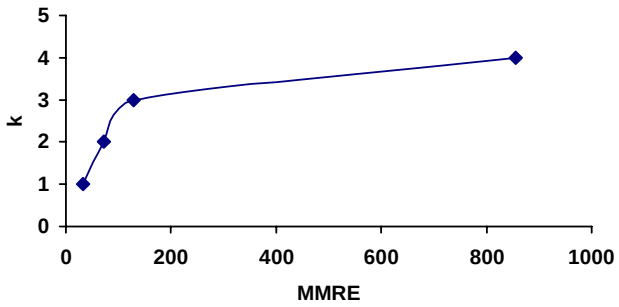

Fig 3: Summarizes graphically the data in Table 1.

Table 2 shows the variation of increasing the neighbor distance (*k*) with MAR (mean absolute residual).

Neighbor(*k*)	MAR
1	2.3877
2	6.308
3	12.8257
4	69.07

Table 2. *k vs* MAR

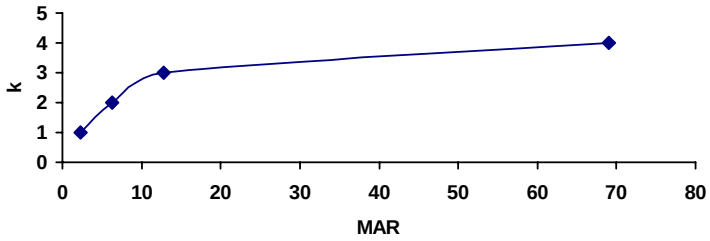

Fig 4: Summarizes graphically the data in Table 2.

b*) Comparison of software project effort prediction using analogy with an algorithmic approach.*

Table 3 shows the accuracy of respective methods using the MMRE when applied on the IBM dataset. For CBR, full CBR cycle was applied while taking 12 cases as past project data, and for regression model entire dataset was used. Regression technique applied here is stepwise regression.

Dataset	CBR (MMRE) (%)	Regression (MMRE) (%)
IBM Dataset	32.41	90

Table 3. Comparison of CBR with regression technique

Conclusion

Now a day's software engineering community is experiencing a significant challenge, as the accurate estimation of software project must be made at an earlier stage in the development process. We conclude that a comprehensive case representation is not available early in the project and suggest instead that the objective should be risk assessment rather than cost estimation. The lack of features to predict size early in the development life cycle indicates a limitation of conventional CBR model. In this research, we have found that the MMRE using CBR is found to be approximately 32% which is quite acceptable compared to the MMRE using regression analysis (approx. 90%). Further estimation by analogy helps in circumstances where it is not possible to generate an algorithmic model, such as Function points. We believe this type of situation is quite common at initial stage of project, for example in response to an invitation to a tender this makes analogy very alluring for producing very early estimates. This analogy offers an added advantage, as it is very intuitive method. Our approach is user friendly as it allows them to assess the reasoning process behind a prediction by identifying the most analogous projects thereby increasing, or reducing, their confidence in the prediction.

Future work

The major thrust is finding the better ways to support collaboration between the human expert and the CBR system. For many applications particularly when dealing with infrequent but high value problems such as experience factory supported decision-making and project prediction may be inappropriate and therefore should be explicitly addresses the problem of how to bring about the most effective forms of interaction between the human and the CBR system.

References

[1] Allan J Albrecht and John E Gaffney, "Software Function, Source Lines Of Code and Development Effort : A Software Science Validation." IEEE Transactions on Software Engineering Vol.SE-9 No-6 pp 639-648 Nov 1983.

[2] Lawerence H Putnam, "General Empirical Solution To The Macro Software Sizing And Estimating Problem."IEEE Transactions on Software Engineering Vol.SE-4 No-4 pp 345-361 July 1987.

[3] Barry W Boehm and Philip N Papaccio, "Understanding And Controlling Software Costs."IEEE Transactions on Software Engineering Vol.Se-14 No-10 pp 1462-1477 Oct 1988.

[4] Graham C Low and D.Ross Jeffery, "Function Points In The Estimation And Evaluation Of The Software Process,"IEEE Transactions on Software Engineering Vol-16 No-1 pp 64-71 Jan 1990.

[5] Charles R Symons, "Function Point Analysis: Difficulties And Improvements," IEEE Transactions on Software Engineering Vol-14 No-1 pp 2-11 Jan1988.

[6] Jack E Matson,Bruce E Barrett and Joseph M Mellichamp, "Software Development Costs Estimation Using Function Points,"IEEE Transactions on Software Engineering Vol-20 No-4 pp 275-286 April 1994.

[7] Barry W Boehm,"Software Engineering Economics, "IEEE Transactions on Software Engineering Vol.SE-10 No-1 Jan 1984.

[8] Chris Schofield, "Non Algorithmic Effort Estimation Techniques," Empirical Software Engineering Research Group,ESERG-TR98-01.

[9] K.Srinivasan and Douglas Fisher, "Machine Learning Approaches To Estimating Software Development Effort,"IEEE Transactions on Software Engineering Vol-21 No-2 pp 126-136 Feb 1995.

[10] Pankaj Jalote, "An Integrated Approach To Software Engineering," Narosa Publishing Home," 6th Edition 1995.

[11] Sarah Jane Delany and Padraig Cunningham, "The Application of Case-Based Reasoning to Early Software Project Cost Estimation and Risk Assesment", TCD-CS-200-10.

[12] Gada Kaoda, Michelle Cartwright, Linguang Chen, and Martin Shepperd, " Experiences Using Case-Based Reasoning to predict Software Project Effort", ESERG, January 27,2000.

[13] Colin Kirsopp, Emilia Mendes, Rahul Premraj and Martin Shepperd, " An Empirical Analysis of Linear Adaptation Techniques for Case-Based Prediction", Bournemouth University, U.K.

[14] Martin Shepperd and Chris Schofied, "Estimating Software Project Effort Using Analogies", IEEE Transactions on Software Engineering, Vol. 23, No. 12, November 1997.

[15] Gada Kadoda, Michelle Cartwright and Martin Shepperd, " On configuring a Case-Based Reasoning Software Project Prediction System", ESERG, September 2000.

[16] Sarah Jane Delany, Padraig Cunningham and Wolfgang Wilke, "The Limits of CBR in Software Project Estimation", http://www.citeseer.nj.nec.com.

[17] Ivo Vollrath, Wolfgang Wilke and Ralph Bergmann, " Case-Based Reasoning Support for Online Catalog Sales", IEEE Internet Computing, July-August 1998.

[18] Hisato Adachi, Yoshizumi Kobayashi and Tadashi Ohta, "Software Design Support Using Case-Based Reasoning", IEEE, Vol. 4, pp 85-90, 1994.