Aspects of Case Based Reasoning for Cost and Effort Estimation - A Preliminary Study

Abstract

Making an estimate of the time and effort needed for a software project is a challenge, especially at the project start, nevertheless, an estimation is to be done, good or bad. Traditionally the project manager acts as an expert and makes a judgement about what he believes to be realistic needs. In the last years AI-techniques has begun to be used for estimation, and this paper describes the results of a study of applying one such technique, Case Based Reasoning. The results of the study are well in line with other research on CBR effort estimation and has lead to that the method has been chosen as one of the estimation methods that should be implemented in the tool Predictor.

Keywords: Effort estimation, CBR, Predictor

1. Introduction

Every project manager must somehow be able to deliver a product satisfying the customer's needs at the date scheduled and within budget. To do this he must face the problems of making schedules and budgets, tasks which demand that he estimates time, cost, and effort. In order to make realistic estimates, data from past projects is required. He can rely on his memory and adjust the estimates according to the differences between the current project and the past ones. This approach, usually called expert judgement, may be more or less reliable, depending on the human memory, the experience, the similarity of the projects, and the manager's ability to make accurate adjustments.

Existing research has provided several other techniques and software support for estimation, the main type being a number of *algorithmic models*, i.e. mathematical functions for the relationship between e.g. size and effort. However, the last years there have been a growing interest in alternative approaches, one of which is Case Based Reasoning. This paper presents some early results from using Case Based Reasoning for prediction support within the project "Prediction Assistance for Project Management"[1] (PPL). The project aim is to develop prediction models for project planning and monitoring based on measurement data, and to implement them in a project management tool set. Currently a prototype version of the tool has been developed. Later it is intended that the more promising parts shall be included in tools developed by the project's industrial partners.

The purpose of the study was to test the approach since we had no previous experience with it. Although CBR has been used for effort estimation with some success by others we needed to make sure that the approach was suitable to our needs, and was possible to realise within the boundaries set by the PPL project. Furthermore earlier research had showed some disagreement on important points that needed to be examined.

The outline of this paper is as follows. In Chapter 2 various strategies for estimation are discussed briefly followed in Chapter 3 by a more in depth motivation and implementation description of the approach used by us, Case Based Reasoning. Chapter 4 presents some empirical results of using the tool, and finally Chapter 5 presents some concluding remarks.

2. Cost Estimation

As was noted in the introduction, there are numerous techniques available for the estimation problem, as well as tools implementing them. In this chapter the two dominating techniques, *expert judgement*, and

[1] Stockholm University/Royal Institute of Technology, Sweden

algorithmic models are outlined followed in Section 2.3 by some of the AI-techniques that have been gaining interest in the last years.

2.1. Expert Judgement

Despite the availability of estimation tools, and although it has long been a truism that estimators commonly do not do any estimations at all [DeMarco, 1982], the most used method for estimating cost and effort is expert judgement, [Walkerden and Jeffery, 1996]. With this technique, an expert, or a group of experts, is called upon to produce an estimate using their own judgement. Commonly this involves looking at similar projects developed earlier, with the assumption that similar projects need approximately the same amount of resources to produce. The estimation can then be further altered to take into account differences between projects such as different size and development environments.

The method can actually work quite well, especially if the expert is thoroughly familiar with the type of application being developed as is seen in the study by [Vicinanza, et al., 1991] where the best estimator was capable of making estimates within 32% on average. Unfortunately, training someone to be a good estimator takes time, and can be quite costly, aspects which must not be overlooked in the Software Engineering field with its high turnover rates.

2.2. Algorithmic Models

The ad hoc nature of expert judgement as an estimation technique has led to a search for more accurate methods, preferably supported by computer tools. The most used method for cost estimation is to use some sort of algorithmic (or parametric) method. According to [NASA, 1995] such a model "is one that uses Cost Estimating Relationships (CERs) and associated mathematical algorithms (or logic) to establish cost estimates". Basically this means applying a mathematical model developed by regression to the cost estimation problem. Normally this model is a variant of

$$effort = a \cdot size^{ß}$$

where a indicates the relationship between size and effort, and ß indicates economies or diseconomies of scale depending on whether it is below or above one [Fenton and Pfleeger, 1997], [Walkerden and Jeffery, 1996]. To get a more realistic model it is common to adjust the estimated value by a number of factors, such as complexity, user commitment, application type etc.

The most common problem with the algorithmic models is that they are normally only relevant for a specific environment, i.e. the one they were developed in. In other domains they are rarely accurate as is showed in e.g. [Kemerer, 1987]. Partly this problem can be handled by calibrating the model to the local environment, but great care must be taken to choose a suitable model. To meet this problem there has been some interesting attempts of developing small local models, [Kok, et al., 1990]. In this case there is no problem with models that are unsuitable for the environment, but this is offset by the fact that the organisation has to have enough data and knowledge to be able to build and validate their own model. In particular the amount of data required prevents the more widespread usage of local algorithmic models.

The most well known algorithmic model is probably COCOMO, [Boehm, 1981], now COCOMO II, [Boehm, 1998], but there are many others, both non-proprietary and commercial. For a recent survey of algorithmic techniques (as well as other cost estimation processes) see [Walkerden and Jeffery, 1996].

2.3. AI Techniques

The problems with the algorithmic models have led to a growing interest in other techniques for estimation. In particular two AI-techniques, Artificial Neural Networks, and Case Based Reasoning, which will be briefly discussed below, has been used. For more elaborate summaries of work on non-

algorithmic approaches to cost estimation see [MacDonell and Gray, 1996], and [Schofield, 1998]. Apart from the two approaches discussed here these studies also include such approaches as Fuzzy Logic, Regression/Classification Trees, and Rule Based systems.

2.3.1. Neural Networks

When Neural Networks (NN) are used for cost estimation, the estimation function is represented by a network of units, each of which represent a simple function. With each connection between two units, a weight is associated, and it is this weight that is normally used to train the network. For estimation purpose the inputs of the network represent the information available about a project, and the output is the prediction.

The most common form of NN is feed-forward networks, meaning networks where the connections are in one direction only and there are no loops. A type which seems to dominate effort estimation applications too. Training in such networks is usually done by back-propagation, an algorithm for dividing the blame for the error between the actual and expected values between the various weights. For more information on NN and back-propagation see e.g. [Russel and Norvig, 1995].

At an early stage of the project an NN approach was considered a strong alternative to CBR. Apart from implementation issues, the main reasons why CBR was preferred was the black-box nature of the NN approach and that it demands a large number of training cases to perform well.

NN applications to cost an effort estimation include the work by [Bode, et al., 1995], [Samson, et al., 1997], [Srinivasan and Fisher, 1995], and [Venkatachalam, 1993].

2.3.2. Case Based Reasoning

The other AI-technique that has become popular is Case Based Reasoning (CBR) or estimation by analogy, as it is sometimes known[2]. It can be seen as an attempt to formalise one of the methods commonly used by humans for solving this type of problem. It is commonly assumed that this means that the method is more acceptable than many other methods since it is possible to understand how it works and examine the reasons for the estimation given. The method is based on the assumption that there are data from past projects, describing the projects in terms of a number of attributes, e.g. time, costs, application type, development environment, and product size. These data are then used by applying the following three steps:

- The first step is to find suitable analogies. In this case these are the projects in the case base that are similar enough in some respect to the one we want to estimate. A vital question here is how to determine the similarity of projects. The most common technique is nearest neighbour algorithms, but there are many other methods used in other CBR applications such as template retrieval and fuzzy similarity measurements [Kolodner, 1993].
- The next step is to adjust the retrieved cases. In some applications of CBR such as reasoning about law cases this might be the most important step. For effort estimation however it is normally considered enough if the difference in size is taken into account.
- The last step is to make an actual estimation. In this domain it is usually done by taking the average of the effort of the retrieved cases.

Examples of usage of CBR for estimation can be found in e.g. [Vicinanza, et al., 1990], [Finnie, et al., 1997], and [Shepperd, et al., 1996]. All of these studies report estimation accuracy's that are in line with or better than most other estimation methods.

[2] The exact meaning of these two terms is not totally defined. In this paper they are used interchangeably though.

3. CBR in Predictor

The relatively good results given by earlier studies using CBR for the estimation task in combination with that it is an intuitively acceptable way of doing an estimation, lead to that it was chosen as one of the estimation method to be included in Predictor, a prediction support tool which is one of the main deliverables of the PPL project. At the time of writing two components have been implemented in Inprises Delphi. The first is called *TAnalogy* and implements the first step of the CBR approach described above, the other is *TPredictor* which implements the third one. There is currently no adjustment of the retrieved cases.

The similarity measure used by TAnalogy is a nearest neighbour algorithm called Weighted Euclidean Distance (WED, Figure 1) which is a generalisation of Pythagora's Theorem to n dimensions. WED measures the similarity between two projects (p and q in Figure 1). In Figure 1, n is the number of attributes, p_i and q_i are the values for attribute i for project p and q respectively, D is a distance function, and W_i is the weight of attribute i. To find the projects in the case base which are most similar to the current one the WED-distance is calculated between it and all other projects in the case base. The projects which have the least distance to the given one are considered the most similar.

$$\text{WED}=\sqrt{\sum_{i=1}^{n}\left(D(p_i,q_i)\cdot W_i\right)^2}$$

Figure 1: Weighted Euclidean Distance

The motivation for the choice of WED was that:

1. It is easy to calculate, and thus fast. Although it might be argued that the user is more interested in the results than the time taken to get them, this is still important for the approach to be accepted.
2. It is also a measurement of similarity that it is possible to understand for the user. The hope is that this will make it easier for the user to trust the program.
3. Earlier studies using nearest neighbour algorithms, e.g. [Finnie, et al., 1997], [Shepperd and Schofield, 1997], have shown good results.

4. Empirical Results

4.1. How to measure the quality of the predictions

Around TAnalogy a test environment has been built that allows automatic experiment using the ”leave one out”, or “jack knifing” technique. With this technique a prediction is made for each project in the case base using the rest of the projects. To measure the quality of the predictions made by the system we are using two metrics:

- Pred(25) measures the percentage of the predictions that falls within 25% of the actual values
- The Magnitude of the Relative Error (MRE) measures the average absolute percentage of error. It is calculated as:

  $$\text{MRE}=\frac{1}{n}\sum_{i=1}^{n}\frac{|A_i-E_i|}{A_i}$$

  where A_i is the actual value, and E_i is the estimated value. To facilitate reading it is common to multiply the MRE by 100 to get a percentage rather than a ratio.

[Conte, et al., 1986] suggests that an estimation is acceptable if Pred(25) is at least 75%, and MRE is ≤25%. Tate and Verner suggest a value of Pred(30)=70% which according to [MacDonell and Gray, 1996] is more realistic.

4.2. Data Analysis

Experiments have been conducted in the test environment on a four publicly available datasets described in Table 1. Apart from the general question of the quality of the predictions made by the system, we wanted to examine the following things:

- How many analogies give the best result?
- Does arithmetic or geometric (Figure 2) mean give the best predictions?

$$\sqrt[n]{\prod_{i=1}^{n} v_i}$$

Figure 2: Geometrical mean

Name	No Projects	No Attributes	Source
Albrecht	24	7	[Albrecht and Gaffney, 1983]
DACS Productivity Dataset[3]	487	14	[DACS, 1992]
Kemerer	15	6	[Kemerer, 1987]
Kitchenham and Taylor	33	10	[Kitchenham and Taylor, 1985]

Table 1 Datasets used

The results of the experiments are summarised in Table 2 below. The quality of the predictions turned out to be in line with the ones reported in other studies, but there are inconsistencies in the results. Since [Shepperd and Schofield, 1997] also used two of these datasets, the Albrecht, and the Kemerer dataset, a comparison with their tool Angel is close at hand, and is given in Table 3. Here we see that Angel performed better on the Albrecht dataset, and Predictor on Kemerers.

Since there are only two small datasets to build the comparison on, it is hard to say what the results depend on. Worth to note is that the Kemerer dataset contained no missing values, i.e. for all projects in the case base all attributes were recorded, which where not the case with the Albrecht dataset. It is thus quite possible that the worse values for the Albrecht dataset (Table 3) were caused by our way of handling missing values in the case base. This was an incremental approach where only the attributes that all projects had in common where used and projects were successively removed from the case base in the hope that the remaining project should have more attributes in common. To see whether or not this held, we removed all cases with missing values from the case base and ran the experiments again, getting the values in Table 4. As we can see, the results for the Albrecht dataset is now in line with Angel, which indicates that a simple removal strategy might work better for these small datasets than more advanced algorithms. Similar experiments with the DACS dataset increased the quality of the predictions, but nowhere near as much as for the Albrecht dataset.

[3] Note that several projects were removed from the DACS dataset since they did not include the attribute we were trying to predict, the number of man-months needed for the project.

	No of predictors:	1		3		5	
Dataset	**Mean \ Metric**	**MRE**	**Pred(25)**	**MRE**	**Pred(25)**	**MRE**	**Pred(25)**
Albrecht	**Arithmetic**	92	39	75	30	82	39
	Geometric	92	39	70	34	73	39
DACS	**Arithmetic**	271	14	241	13	334	13
	Geometric	271	14	118	15	128	14
Kemerer	**Arithmetic**	54	33	75	26	74	26
	Geometric	54	33	72	33	63	46
Kitchenham and Taylor	**Arithmetic**	104	18	94	30	83	36
	Geometric	104	18	91	24	78	27

Table 2 Empirical results

	Angel	Predictor
Dataset \ Quality	**MRE**	**MRE**
Albrecht	62	92
Kemerer	62	54

Table 3 Comparison with Angel using one project as predictor

	Angel	Predictor
Dataset \ Quality	**MRE**	**MRE**
Albrecht	62	66
Kemerer	62	54

Table 4 Comparison with Angel using one project as predictor and no missing values

Since there is great variance in the quality of the predictions between the datasets, we compared the attributes used in the different datasets with each other. The datasets that performed well were the ones that had recorded several attributes related to the size of the projects, e.g. both Lines of Code, and Function Points in the Kemerer dataset. This confirmed the rather obvious idea that it is important to record the "right" attributes for the projects, i.e. attributes with a high correlation to effort. In the DACS dataset, which is large, and contains many attributes, but still scored badly, most of the attributes dealt with which programming language was used.

On the questions on how many projects to use as predictors, and whether to use arithmetic or geometric mean to calculate the estimation, [Shepperd and Schofield, 1997] and [Finnie, et al., 1997] have different opinions. According to [Shepperd and Schofield, 1997] the best predictions were made by taking the value of the attribute directly from the single most similar project gave the best result. [Finnie, et al., 1997] on the other hand claims that the best predictions were made by taking the geometrical mean of the three most similar projects. Our experiments support the usage of geometrical mean, but is not able to give a

conclusive answer to how many projects to use as predictors. There is an indication though that the best number of predictors varies with the size of the case base which seams reasonable. After all, if the case base is small, we are lucky if even a single project is similar enough to the one we are interested in, but with a large case base, many projects might be interesting. To try this hypothesis we ran an experiment with the largest dataset to our disposal, the one from DACS. What we did was that we let all projects that were given a similarity of at least 90%[4] contribute to the prediction. As we can see from the results in Table 5, the results are better than before, although there is still room for improvement before they are good enough to be used in practice.

	Arithmetic mean		Geometric Mean	
Dataset \ Quality	**MRE**	**Pred(25)**	**MRE**	**Pred(25)**
DACS	222	15	91	24

Table 5 Results using projects with more than 90% similarity to make predictions for the DACS dataset

5. Concluding Remarks

Predicting the cost and effort needed for a software development project is a difficult problem where much work have gone into the development of models, methods, and tools to help the task. This paper has reported on a preliminary study of a CBR effort estimation model which is being implemented in the prediction support tool Predictor.

The results showed a reasonable accuracy of the method with results well in line with other estimation models although there is still room for much improvement, and we intend to continue on this path. Currently our main effort is directed at implementing Predictor which apart from the CBR estimation technique discussed here also will include support for the development of local algorithmic models.

6. References

A. J. Albrecht and J. E. Gaffney, jr, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," *IEEE Transactions on Software Engineering*, vol. 9(6), November 1983.

J. Bode, S. Ren, and Z. Shi, " Application of 3-Layer Perceptrons to Cost Estimation," presented at IEEE, 1995.

B. W. Boehm, *Software Engineering Economics*: Prentice Hall, 1981.

B. W. Boehm, et al, *COCOMO II Model Definition Manual*: South Carolina University, 1998.

S. D. Conte, H. E. Dunsmore, and V. Y. Shen, *Software Engineering Metrics & Models*: Benjamin/Cummings Publishing Company, 1986.

DACS, "The DACS Data Compendium," Kaman Sciences Corporation, Technical Report 30602-89-C-0082, 1992.

[4] If WED is standardised to a value between 0 and 1, this means that the similarity score should be ≤0.1 for the project to be used.

T. DeMarco, *Controlling Software Projects: Management, Measurement, and Estimation*: Prentice Hall/Yourdon Press, 1982.

N. E. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous & Practical Approach*: Thomson Computer Press, 1997.

G. R. Finnie, G. E. Wittig, and J.-M. Desharnais, "Estimating Software development Effort with Case-Based Reasoning," in *Proceedings of the second International Conference on Case-Based Reasoning ICCBR-97*, D. B. Leake and E. Plaza, Eds., 1997.

C. F. Kemerer, "An Empirical Validation of Software Cost Estimation Models," *Communications of the ACM*, vol. 30(5), May 1987.

B. A. Kitchenham and N. R. Taylor, "Software Project Development Cost Estimation," *Journal of Systems and Software*, vol. 5, pp. 267-278, 1985.

P. A. M. Kok, B. A. Kitchenham, and J. Kirakowski, "The MERMAID Approach to software cost estimation," in *Proceedings Esprit Technical Week*, 1990.

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

S. G. MacDonell and A. R. Gray, "Alternatives to Regression Models for Estimating Software Projects," in *Proceedings of the IFPUG Fall Conference*. Dallas TX: IFPUG, 1996, pp. 279.1-279.15.

NASA, *Parametric Cost Estimating Handbook*, http://www.jsc.nasa.gov/bu2/PCEHHTML/pceh.htm , 1995.

S. Russel and P. Norvig, *Artificial Intelligence A Modern Approach*: Prentice Hall, 1995.

B. Samson, D. Ellison, and P. Dugard, "Software Cost Estimation using an Albus perceptron (CMAC)," *Information and Software Technology*(39), pp. 55-60, 1997.

C. Schofield, "Non-Algorithmic Effort Estimation Techniques," Empirical Software Engineering Research Group, Bournemouth University, Technical Report TR98-001, 1998.

M. Shepperd and C. Schofield, "Estimating Software Project Effort Using Analogies," *IEEE Transactions on Software Engineering*, vol. 23(12), November 1997.

M. Shepperd, C. Schofield, and B. Kitchenham, "Effort Estimation Using Analogy," in *Proceedings of ICSE-18*, 1996, pp. 170-178.

K. Srinivasan and D. Fisher, "Machine Learning Approaches to Estimating Software Development Effort," *IEEE Transactions on Software Engineering*, vol. 21(2), February 1995.

A. R. Venkatachalam, "Software Cost Estimation Using Artificial Neural Networks," in *Proceedings of the International Joint Conference on Neural Networks*. Nagoya, 1993.

S. Vicinanza, M. J. Prietula, and T. Mukhopadhyay, "Case-based Reasoning in Software Effort Estimation," in *Proceedings of the 11th International Conference on Information Systems*, 1990.

S. S. Vicinanza, T. Mukhopadhyay, and M. J. Prietula, "Software-Effort Estimation: An Exploratory Study of Expert Performance," *Information Systems Research*, vol. 2(4), pp. 243-262, December 1991.

F. Walkerden and R. Jeffery, "Software Cost Estimation: A Review of Models, Processes and Practice," Centre for Advanced Empirical Software Research, School of Information Systems, University of New South Wales CAESAR Technical Report 96/01, 1996.