NORTH-HOLLAND

A Comparison of Software Effort Estimation Techniques: Using Function Points with Neural Networks, Case-Based Reasoning and Regression Models

G. R. Finnie and G. E. Wittig
School of Information Technology, Bond University, Gold Coast, Queensland 4229, Australia

J-M. Desharnais
Software Engineering Laboratory in Applied Metrics, 7415 rue Beaubien Est, suite 509, Anjou, Quebec, Canada H1M 3R5

Estimating software development effort remains a complex problem attracting considerable research attention. Improving the estimation techniques available to project managers would facilitate more effective control of time and budgets in software development. This paper reviews a research study comparing three estimation techniques using function points as an estimate of system size. The models considered are based on regression analysis, artificial neural networks and case-based reasoning. Although regression models performed poorly on the data set of 299 projects, both artificial neural networks and case-based reasoning appeared to have value for software development effort estimation models. Case-based reasoning in particular is appealing because of its similarity to expert judgment approaches and for its potential as an expert assistant in support of human judgment.

INTRODUCTION

Software development involves a number of interrelated factors which affect development effort and productivity. Accurate forecasting has proved difficult since many of these relationships are not well understood. Improving the estimation techniques available to project managers would facilitate more effective control of time and budgets in software development.

Most estimation models in use or proposed in the literature are based on regression techniques (see for example Matson et al., 1994). A number of these models are discussed in the following section. Statistical regression models estimate software development effort as the dependent variable. Software size (in metrics like lines of code or function points) is used as an independent variable. In some models other parameters such as development programming language or operating system may be used as additional independent variables for a multiple regression model. Regression models have the advantage of a sound mathematical basis as well as measures of goodness of fit, i.e., how well the curve matches the given dataset (using for example R^2). New project estimates are obtained by substituting the new project parameters into the mathematical model. However regression models are very susceptible to the effect of outliers (i.e., one or two data items which may be completely out of line with the rest of the dataset). A good regression model also needs a relatively large dataset which can be a problem in the software estimation field. Matson et al. (1994) discuss some of the concerns of regression models and the dangers of ignoring the underlying assumptions of regression analysis.

Address correspondence to Dr. G. R. Finnie, School of Information Technology, Bond University, Gold Coast, QLD 4229 Australia.

J. SYSTEMS SOFTWARE 1997; 39:281–289

655 Avenue of the Americas, New York, NY 10010

0164-1212/97/$17.00
PII S0164-1212(97)00055-1

This paper examines the potential of two artificial intelligence approaches, i.e., artificial neural networks and case-based reasoning, for providing the basis for development effort estimation models in contrast to regression models. Artificial neural networks (ANNs) adopt a learning approach to deriving a predictive model. The network is designed for the specific set of inputs, e.g., function points, implementation language, etc. as well as the output(s), e.g., development effort (see later for a brief discussion). The network is presented with a set of known cases (the training set) which is used to "train" the network, i.e., establish the weights associated with each input in the network. Once the network is trained and stable, development effort for a new case can be predicted by substituting the relevant input values for the specific case. ANNs are recognized for their ability to provide good results when dealing with problems where there are complex relationships between inputs and outputs, and where the input data is distorted by high noise levels (Treigueiros and Berry, 1991). Although the potential for predictive accuracy is good, neural networks lack an explanation capability and do not provide an environment for direct user adaptation of results.

Case-based reasoning (CBR) (Watson and Marir, 1994) is a problem solving technique which solves new problems by adapting solutions that were used to solve old problems. CBR retrieves one or more cases similar to the current problem and attempts to modify these to fit the current problem parameters. In software development effort estimation, each case could be a previous software development while the current problem is one of extracting a suitable estimate for the current project. Adaptation is based on differences between the stored case and the current problem, e.g., the original case may relate to a system developed with an inexperienced programming team whereas the current system developers may be very familiar with the development environment. As a result, development should be faster and the time estimate adjusted accordingly. Case-based reasoners have an advantage in that they can justify decisions on the basis of the previous cases used in solving a problem. In addition the CBR approach is intuitively similar to the expert judgment adopted in many organizations that rely on use of experienced developers to estimate project effort. These individuals estimate by judgmental adaptation of past system developments.

This research compares the effectiveness of back-propagation artificial neural networks in estimating software development effort with the effectiveness of a CBR model and a multiple linear regression model. All three approaches were tested on a relatively large (299 cases) software development project dataset.

SOFTWARE EFFORT ESTIMATION MODELS

Numerous models have been proposed for software development effort estimation, e.g., COCOMO, SLIM, Estimacs, Function Point Analysis (FPA) (Albrecht and Gaffney, 1983; Function Point Counting Practices Manual, 1994; Symons, 1991), SPANS, Checkpoint and COSTAR. However no model has proved to be outstandingly successful at effectively and consistently predicting software development effort. The models are based on regression analysis of some set of past cases. Independent variables include an estimate of system size (in lines of code or function points). Matson et al. (1994) review some of the problems associated with regression models, specifically as applied to FPA. Miyazaki et al. (1994) propose a more robust form of least squares calculation for determining parameter values which they consider to be better able to deal with the outliers prevalent in software effort estimation data. All the models presume the ability to estimate system size early in the life cycle which is a major weakness of the lines of code models. FPA estimates can be generated reasonably accurately at an early stage of the development.

Kemerer (1987) performed an empirical validation of four algorithmic models (SLIM, COCOMO, Estimacs and FPA) using data from projects outside the original model development environments without re-calibrating the models. The results indicate to what extent these model are generalisable to different environments. Most models showed a strong over-estimation bias and large estimation errors with the mean absolute relative error (MARE) ranging from an average of 57 percent to almost 800 percent. Ferens and Gurner (1992) evaluated three development effort prediction models (SPANS, Checkpoint and COSTAR) using 22 projects from Albrecht's database, and 14 from Kemerer's dataset. The prediction error is large, with the MARE ranging from 46 percent for the Checkpoint model to 105 percent for the COSTAR model.

Jeffery and Low (1990) conducted a study to investigate the need for model calibration at both the industry as well as the organization level. Again the MARE was high, ranging from 43 to 105 percent for the three companies which were used in the study. Jeffery et al. (1993) compared the SPQR/20 model

to FPA using data from 64 projects within one organization. The models were re-calibrated to the local environment to remove over- or under-estimation biases. The estimation errors are considerably less than those of previous studies with MAREs of approximately 12 percent which reflects the benefits of model calibration.

Artificial intelligence techniques do not appear to have been widely used in development effort estimation although they have been proposed and used in other areas of software engineering (Ramsey and Basili, 1989). Matson and Mellichamp (1993) developed a knowledge-based system to assist analysts in estimating a system size in function points. Mukhopadhyay et al. (1992) developed a model (Estor) for development effort estimation, based on case-based reasoning which was evaluated against expert judgment, COCOMO, and Function Point Analysis (FPA). In this research none of the models were able to improve on the performance of the expert. The estimation error of the expert and of Estor were considerably less than FPA and COCOMO. Despite the improvement the errors are still large with Estor's MARE greater than 50 percent. In the Estor CBR model the system size is based on elapsed time (for previous cases) and any adaptation of this time is done on the basis of rules extracted from a protocol analysis of an expert estimator's performance on an actual estimation task.

Srinivasan and Fisher (1995) used two machine learning approaches to estimating software development effort which were a decision tree-based approach and the use of ANNs. The two approaches were trained on two fairly small datasets, i.e., the 63 cases from Boehm's COCOMO study (1981) and the 15 from Kemerer's study (1987). The techniques appeared to be competitive with regression approaches in terms of the accuracy of the effort estimate, although the authors warn against incautious use of historical data.

The present research extends preliminary work done using the ANN and CBR techniques to analyse data in the ASMA (Australian Software Metrics Association) database (Finnie and Wittig, 1996).

RESEARCH DATA

This database is the result of a compilation of 299 projects from 17 different organizations (Desharnais et al., 1990). The standard deviation and the skewness of the data suggests the possible presence of outliers, but none of these were excluded from the analysis. Table 1 gives the mean for each statistic

Table 1. Summary of Project Data

	Description	Mean	Min.	Max.	Skew.
Effort	Effort in hours	7086	247	86478	4.75
UFP	Unadjusted function points	298	48	1257	1.81
FP	Adjusted function points	267	40	1182	1.86
UFP/ Hour	Productivity	0.071	0.008	0.696	3.99

(e.g., the project development effort), the maximum and minimum values as well as the skewness of the data. The table refers to the actual project data recorded.

Size in function points and project effort was determined for each project on completion. This has the advantage in an *ex post facto* analysis of removing any difficulties caused by changes in project specifications. The primary motivation for the introduction of FPA was however the need to determine an estimate of system size and hence development effort early in the development life cycle. Betteridge (1992) argues that there appears to be reasonable consistency between function point estimates early in the life cycle and the function point count on completion.

PERFORMANCE EVALUATION

Different error measurements have been used by various metrics researchers, but for this project the main measure of model performance is the Mean Absolute Relative Error (MARE) (or average *Magnitude of Relative Error* (Srinivasan and Fisher, 1994)). MARE is the preferred error measure of software measurement researchers and is calculated as follows:

$$\text{MARE} = \left(\sum_{i=1}^{n} \left| \frac{\textit{estimate} - \textit{actual}}{\textit{actual}} \right| \right) \div n$$

where:

estimate is the model estimate for each observation

n is the number of observations

actual is the observed development effort value.

FUNCTION POINT ANALYSIS

Function Point Analysis is a two stage process (Function Point Counting Practices Manual, 1994). In the first phase unadjusted function points are computed on the basis of an estimate of the number of inputs, outputs, inquiries, internal files and external files in the system. Each of these in turn is

assessed as simple, average or complex and assigned a weight accordingly (see Table 2). For example a system with 2 simple inputs, 4 average outputs, 3 complex inquiries, an average and a complex external interface file and 3 simple internal files would have a total unadjusted function point count (UFP) of

$$2 \times 3 + 4 \times 5 + 3 \times 6 + 1 \times 5 + 1 \times 10 + 3 \times 7$$
$$= 70 \text{ UFP}.$$

The total UFP count is the sum of all these weighted components. In the second phase the unadjusted function point count is adapted by the use of 14 general system characteristics (gsc) which attempt to compensate for differences in technical complexity between system developments, e.g., in terms of data communications or on-line data entry requirements.

General system characteristics (gsc) are rated on a score of 0 (no influence) to 5 (essential). The total gsc score (called the total Degrees of Influence DI) can therefore range from 0 to 70. The UFP count is adjusted as

$$\text{FP} = \text{UFP} \times (0.65 + 0.01 \times \text{DI}).$$

Initial function point estimates can be determined with reasonable accuracy from the project specification fairly early in the systems development life cycle.

REGRESSION MODELS

The sample of 299 was divided randomly into 50 test cases and 249 training cases. No attempt was made to remove outliers or stratify the selection. Random selection was determined by a random number generator. Regression was performed on the 249 training cases with a variety of models to estimate development effort. The six models used were based on:

unadjusted function points (UFP)

function points (FP, i.e., UFP adjusted by the general system characteristics)

log-linear transformation of UFP and development effort

log-linear transformation of FP and development effort

multiple regression with UFP and the 14 general system characteristics

multiple regression with log-linear UFP and the 14 gscs.

Since the general system characteristics are used to compute function points from unadjusted function points there is no need to perform multiple regression with FP and the gsc.

These models were then used to estimate development effort in the other 50 cases. The quality of the estimation is based on two measures, the Mean Absolute Relative Error (MARE) and the proportion of the estimate within 25% and within 50% of the actual development effort.

Table 2. Function Point Allocation

Description	Simple	Average	Complex
External Input	3	4	6
External Output	4	5	7
External Inquiry	3	4	6
External Interface File	5	7	10
Internal File	7	10	15

ARTIFICIAL NEURAL NETWORK MODELS

ANNs have the ability to model complex non-linear relationships and are capable of approximating any measurable function. ANNs have several features which make them attractive prospects for solving pattern recognition tasks without having to build an explicit model of the system.

In a broad sense the network itself is a model because the topology and transfer functions of the nodes are usually formulated to match the current problem. Many network architectures have been developed for various applications. Back-propagation networks are used for estimating software development effort in this project and a typical network architecture is shown in Figure 1. The network inputs include the system size, several gsc and the programming environment. The only output is the estimated development effort.

Classical back-propagation (Rumelhart et al., 1986) is a gradient method of optimisation executed iteratively, with implicit bounds on the distance moved in the search direction in the weight space. This is achieved by incorporating a learning rate (the gain) and the momentum term (the damping factor) in the model.

Each input into the neuron is modified by multiplying each input by a weight for that connection. In Figure 2 these are marked as w_{1j} through to w_{ij} for all the inputs. These weighted inputs are then summed by the neuron, and with reference to a threshold value, will determine the neuron output.

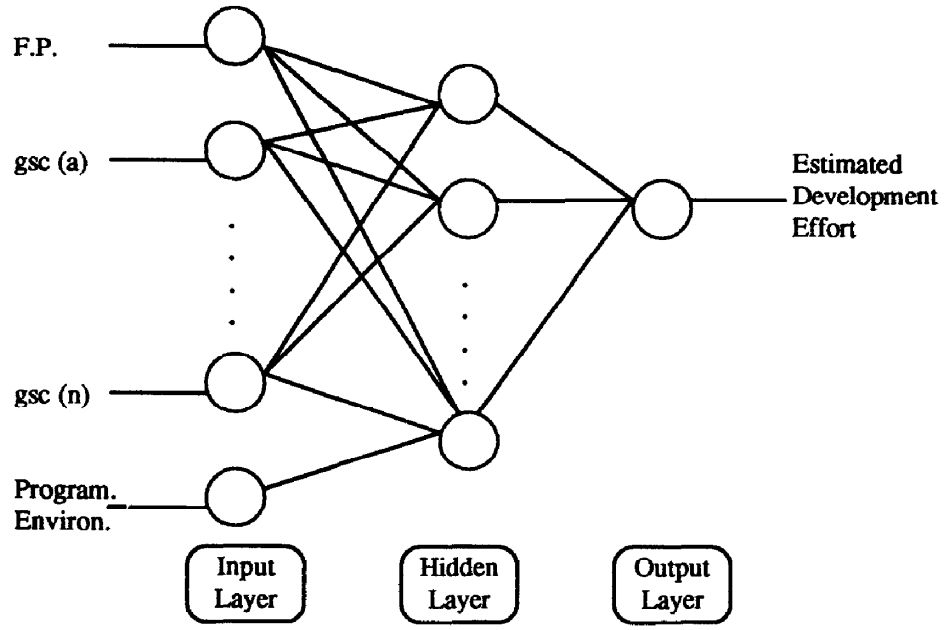

Figure 1. Network Architecture for Software Development Effort Estimation.

The output is described by two sets of equations. The first one, which is the combination operation of the neuron yielding U_j for the jth neuron, is

$$U_j = \Sigma(x_i \times w_{ij}) - t_j.$$

U_j is biased adjusted by a previously established threshold value, t_j, and is then subjected to the activation or threshold function. Where the activation function is sigmoidal, the equation is as follows:

$$Y_j = (1 + e^{-U_j})^{-1}.$$

This equation implements the firing of the neuron. Numerous activation functions have been used, such as gaussian, linear, step-wise, or the most commonly

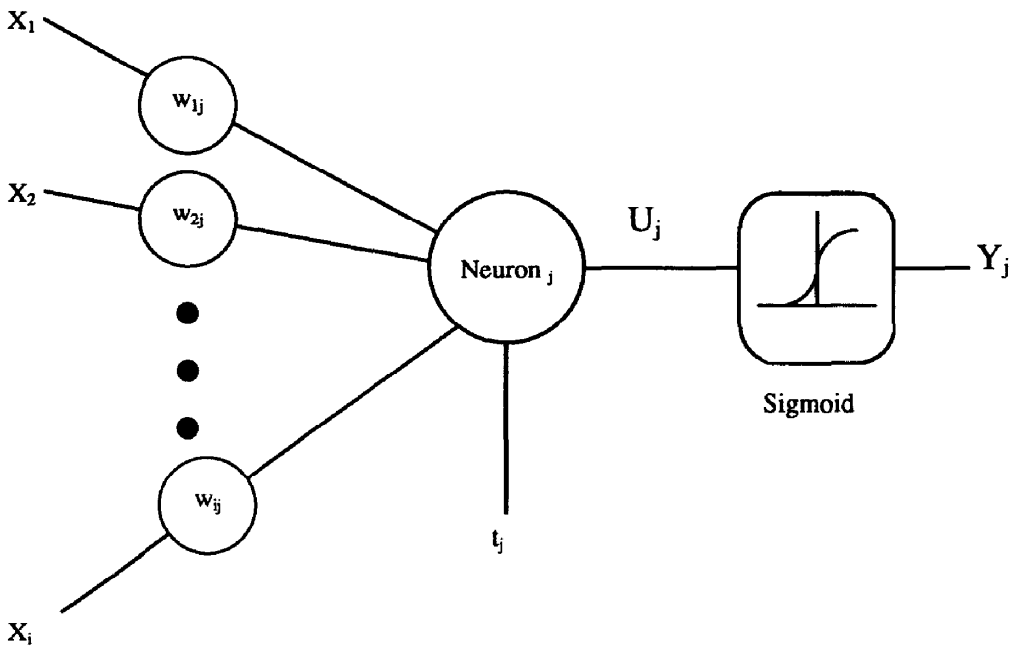

Figure 2. Model Neuron.

used one which is the sigmoid function. This output, Y_j, is the input to the next layer, or the response of the network if this neuron is in the last layer.

The difference between the actual and desired output is determined continuously and repetitively for each set of inputs and corresponding set of outputs produced in response to the inputs. A portion of this error is propagated backward through the network. This is shown graphically in Figure 3. At each neuron the error is used to adjust the weights of the connections so that for the next epoch the error in the network response will be less for the same inputs. This process continues until the network performance on the test set is optimised.

The performance of neural networks depends on the architecture of the network and their parameter settings. Determining the architecture of a network (size, structure, connectivity) affects the performance criteria, such as the learning speed, accuracy of learning, noise resistance and generalisation ability. There is no clearly defined theory which allows for the calculation of the ideal parameter settings and as a rule even slight parameter changes can cause major variations in the behaviour of almost all networks.

A CASE-BASED REASONING MODEL

Case-based reasoning is a technique attracting considerable attention in areas such as production planning, law and medicine (Marir and Watson, 1994). It has the capability to model the way experts in many fields approach problem solving, i.e., by adapting past cases which appear similar to the present problem. Adaptation is done on the basis of similarities and differences between the current problems and one or more prior cases or parts of cases. Experts in fields like medical diagnosis will more often work from the basis of past experiences and medical cases than by attempting to construct a diagnosis from first principles. The case-based reasoning paradigm has an intuitive appeal for use in software effort estimation as it has the capability to model the way expert estimation is performed as well as explaining the reasoning applied to adapt past cases.

To develop a suitable CBR system for software estimation it was necessary to establish those factors which could be used for adaptation, i.e., if the current problem differs in some way from a stored case how can one adapt the effort estimate. The test data available was divided into 249 "training cases" to be held in the case base and 50 test cases to evaluate

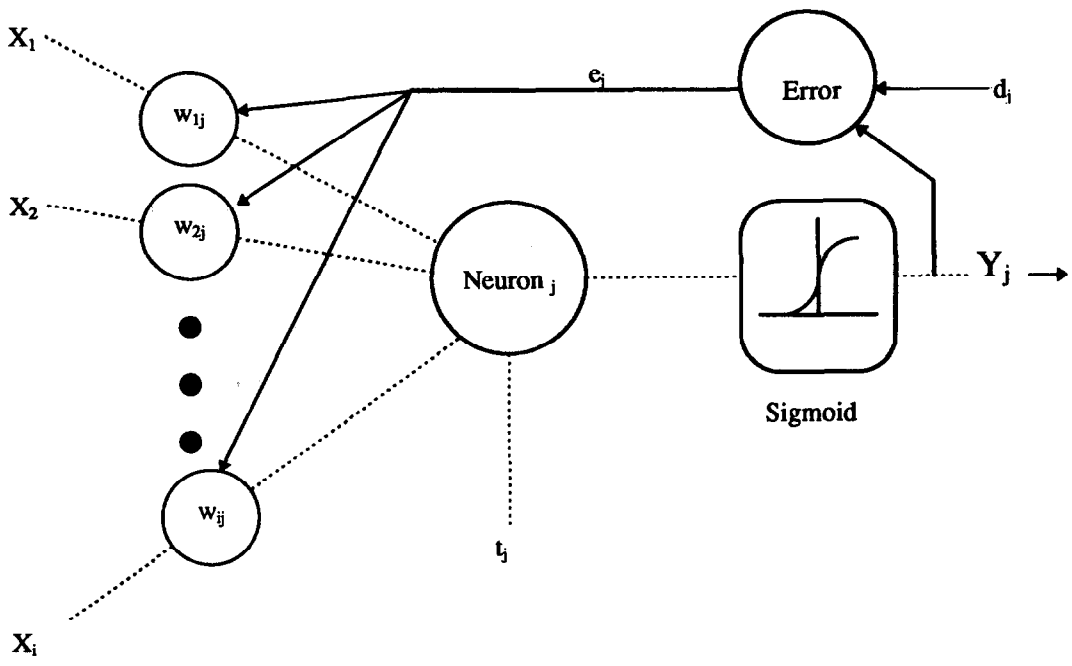

Figure 3. Error Back-Propagation.

the quality of the model. The key concept used to determine case adaptation was to identify factors which contributed to significant differences in productivity between cases.

The gsc were analyzed in the sample cases to determine whether there were any significant differences in productivity (using a simple t-test) between those with a low rating on the gsc (a score of 0-2) and those with a high rating on the gsc (3-5). Only 6 of the gsc in fact showed any significant difference (at $p < 0.05$) which raises some concern about the validity of these factors to identify let alone compensate for differences in system development effort. The six factors were

Performance ($t = 3.59$, $p = 0.0002$)

On-line data entry ($t = -2.16$, $p = 0.02$)

End-user efficiency ($t = 1.88$, $p = 0.03$)

Complex processing ($t = 3.50$, $p = 0.0003$)

Multiple sites ($t = 4.42$, $p = .00001$)

Facilitate change ($t = 4.27$, $p = 0.00001$)

A CBR system was developed based on Remind 1.3. Remind is a commercially available case-based reasoning tool produced by Cognitive Systems, Inc. Nearest neighbor retrieval was used to identify the cases most similar to a given problem case, based on the significant gsc, system size (in unadjusted function points) and the ratios of inputs, outputs, inquiries, internal files and external files to the total UFP count. The latter were included when it was noted that the relative proportions of factors such as inquiries in the total UFP count appeared to significantly influence productivity. Ten cases were initially kept out of the sample of 249 in order to provide a test set to allow experimentation with different retrieval and adaptation combinations. Adaptation rules were developed based on the differences in productivity for each significant gsc. This was done by developing a simple linear regression model for each significant gsc with the productivity. Adaptation rules then adjusted the software development effort estimate by a multiplier based on the differences in score for any significant gsc.

As an example, suppose that project A stored in the case base has a complex processing rating of 0, a function point count of 200 and a development effort of 1000 hours. Project B for which the effort is to be estimated has a function point count of 100 and a complex functions rating of 5. If the multiplier has determined a 5 percent difference in productivity for each point on the gsc scale, there will be a 25 percent difference in productivity between the cases, and project B will be estimated to take

$$(100 \div 200) \times 1000 \times 1.25 = 625 \text{ hours.}$$

All adjustments are combined before the final estimate is determined. In an attempt to smooth out the effect of single incorrect estimates, an average of the estimates based on the three nearest neighbors was computed. After a number of experiments to establish a reasonably good model on the ten test cases, the model was run against the 50 cases randomly selected from the original sample. The results are discussed below in comparison with those obtained from the other methods.

OVERALL RESULTS

The results from the 50 test cases using six different regression models are summarized in Table 3 below. The table gives the Mean Absolute Relative Error for each sample, the proportion within 25 percent of the actual development effort, the proportion within 50 percent of the actual effort and the proportion outside this range.

The predictive accuracy of FPA with regression models clearly shows considerable error and even the best case only provides 40 percent (20 out of 50 cases) of the test cases with an effort prediction within 25 percent of the actual.

Several artificial neural networks were developed to compare the estimation error using as inputs

unadjusted function points (UFP)

adjusted function points (FP)

unadjusted function points with the 14 general system characteristics.

For the first two networks, with the large range in productivity and with system size as the only input, the estimation error was large. The results are shown in Table 4. In both cases the MARE was similar. The network with the lowest prediction error required 10 hidden neurons using UFP and 15 hidden units using FP as input.

Other networks were developed using a subset of the 14 gsc together with the UFP count as inputs. In

Table 3. Regression Analysis Results

	MARE	≤ 25%	> 25- ≤ 50%	> 50%
UFP	1.002	34%	28%	38%
FP	0.903	30%	30%	40%
UFP Log-Linear	0.790	40%	22%	38%
FP Log-Linear	0.733	40%	28%	32%
UFP + gscs	1.234	24%	26%	50%
Log UFP + gscs	0.623	36%	36%	28%

Table 4. ANN Prediction Error

Inputs	MARE
Unadjusted function points	0.590
Function points	0.585
UFP with gsc's and programming environment	0.352

a systematic manner the effect of the gsc on the estimation error was determined. Initially the individual effect was determined, after which several gsc were evaluated in various combinations. In numerous instances the individual gsc did not reduce the estimation error. Trials with the data also indicated that including the programming environment (3GL or 4GL) as an input consistently reduced the estimation error.

The results of including all the gsc together with the programming environment as inputs into the models are shown in Table 4. The network with the lowest prediction error required 5 hidden neurons and had a MARE of 0.35.

An analysis of the estimation errors for the best network indicates that 22 percent were within 10 percent of the actual effort, while 76 percent were within 50 percent. The results of this analysis are shown in Table 5.

With the fifty test cases the CBR model had a MARE of 0.362 with 18 percent of the estimates within 10 percent of the actual effort, while 80 percent were within 50 percent. The results are given in Table 5.

DISCUSSION OF RESULTS

Regression models do not perform very effectively in modelling the complexities involved in software development projects. The data used in this analysis is typical of software developments, with an extreme range of productivity (from .008 to .696 UFP/hr) and a considerable range of possible causes of productivity variation. The best of the regression models used here had a MARE of 0.62 with 36 percent of the estimates within 25 percent of the actual development effort.

Both artificial neural networks and case-based reasoning appear to perform somewhat better at providing estimates. The best ANN model had a MARE of 0.35 and the best CBR model a MARE of 0.36. To establish whether only the means of these two models are similar or whether estimates of the different project estimates of the two approaches are similar, a paired two sample for means t-test was conducted on the relative AREs. The null hypothesis was that the difference between the AREs of the case-based reasoning and neural network models are equal to zero (H_0: $\mu = 0$). As the calculated t statistic was well below the critical t value the H_0 was not rejected. This indicated that even though there are apparent differences in project effort estimates between the two approaches, these individual estimates are not statistically significantly different.

Although neither of these is particularly good, the noise in the dataset is considerable. The ANN model appears capable of effectively capturing the parameters influencing productivity but does so at the expense of comprehensibility, i.e., it is not possible to obtain any explanation or justification of an estimate. Case-based reasoning provides a reasonably good estimate but has the advantage that the actual cases used in the estimation process can be retrieved and if necessary adapted further by any individual involved in the estimation process, i.e., they allow the added use of human intuition and judgment.

The conclusion of this research is that artificial intelligence models are capable of providing adequate estimation models. Their performance is to a large degree dependent on the data on which they are trained, and the extent to which suitable project data is available will determine the extent to which adequate effort estimation models can be developed. CBR allows the development of a dynamic case base with new project data being automatically incorporated into the case base as it becomes available

Table 5. Summary of All Results

	Regression		ANN		CBR	
Estimation Error	Frequency %	Cumulative %	Frequency %	Cumulative %	Frequency %	Cumulative %
0–0.1	10%	10%	22%	22%	18%	18%
0.1–0.25	26%	36%	22%	44%	24%	42%
0.25–0.5	36%	72%	32%	76%	38%	80%
0.5–1.0	12%	84%	18%	94%	16%	96%
> 1.0	16%	100%	6%	100%	4%	100%
MARE	0.623		0.352		0.362	

while ANNs will require retraining to incorporate new data. Practical estimation within an organization is therefore critically dependent on good record keeping for all software projects developed by that organization. Such records must include not only all data relating to staffing levels and skills, time spent in development, the development environment and tools available, etc. but must also include information on all estimates and adaptations to estimates at various stages of the project. Organizations without a suitable case base initially could use industry data for initial projects (e.g., the Australian Software Metrics Association database) but should develop their own store of projects as the basis for future estimates.

REFERENCES

Albrecht, A. J., and Gaffney, J. E., Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. *IEEE Transactions on Software Engineering*, 9(6), 639–648 (1983).

Betteridge, R., Successful Experience of Using Function Points to Estimate Project Costs Early in the Life Cycle. *Information and Software Technology*, 34(10), 655–658, (1992).

Desharnais, J. M., et al., Adjustment Model for Function Points Scope Factors—A Statistical Study. *IFPUG Spring Conference*, Florida (1990).

Ferens, D. V., and Gurner, R. B., An Evaluation of Three Function Point Models for Estimation of Software Effort. *IEEE National Aerospace and Electronics Conference—NAECON92*, 2, 625–642 (1992).

Finnie, G. R., and Wittig, G. E., AI Tools for Software Development Effort Estimation. *Proceedings of the Conference on Software Engineering: Education and Practice*, University of Otago, 113–120 (1996).

Function Point Counting Practices Manual, Release 4.0 *International Function Point Users Group*, Blendonview Office Park, 5008-28 Pine Creek Drive, Westerville, OH 43081-4899, USA, 1994.

Jeffery, D. R., and Low, G. C., Calibrating Estimation Tools for Software Development. *Software Engineering Journal*, 215–221 (July 1990).

Jeffery, D. R., Low, G. C., and Barnes, M., A Comparison of Function Point Counting Techniques. *IEEE Transactions on Software Engineering*, 19(5), 529–532 (1993).

Kemerer, C. F., An Empirical Validation of Software Cost Estimation Models. *Communications of the ACM*, 30(5), 416–429 (1987).

Marir, F., and Watson, I. D., Case-based Reasoning: A Categorized Bibliography. *The Knowledge Engineering Review*, 9(4) (1994).

Matson, J. E., and Mellichamp, J. M., An Object-oriented Tool for Function Point Analysis. *Expert Systems*, 10(1), 3–14 (1993).

Matson, J. E., Barrett, B. E., and Mellichamp, J. M., Software Development Cost Estimation Using Function Points. *IEEE Transactions on Software Engineering*, 20(4), 275–287 (1994).

Miyazaki, Y., Terakado, M., Ozaki, K., and Nozaki, H., Robust Regression for Developing Software Estimation Models. *Journal of Systems and Software*, 27, 3–16 (1994).

Mukhopadhyay, T., Vicinanza, S. S., and Prietula, M. J., Examining the Feasibility of a Case-Based Reasoning Model for Software Effort Estimation. *MIS Quarterly*, 16(2), 155–171 (1992).

Ramsey, C. L., and Basili, V. R., An Evaluation of Expert Systems for Software Engineering Management. *IEEE Transactions on Software Engineering*, 15(6), (1989).

Rumelhart, D. E., Hinton, G. E., and Williams, R. J., Learning Internal Representations by Error Propagation. In *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, Cambridge, MA, MIT Press, 1, 318–362 (1986).

Srinivasan, K., and Fisher, D., Machine Learning Approaches to Estimating Software Development Effort. *IEEE Transactions on Software Engineering*, 21(2126-137) (1995).

Symons, C. R., *Software Sizing and Estimating MkII FPA*, John Wiley & Sons, Chichester, 1991.

Treigueiros, D., and Berry, R., The Application of Neural Network Based Methods to the Extraction of Knowledge from Accounting Reports. *Proceedings of 24th Annual Hawaii International Conference on System Sciences*, IV, 137–146 (1991).

Watson, I., and Marir, F., Case-based Reasoning: A Review. *The Knowledge Engineering Review*, 9(4), 327–354 (1994).