Using Genetic Programming to Improve Software Effort Estimation Based on General Data Sets

Martin Lefley and Martin J. Shepperd

School of Design Engineering and Computing, University of Bournemouth, Talbot Campus, Poole, BH12 5BB, UK
mlefley@bournemouth.ac.uk

Abstract. This paper investigates the use of various techniques including genetic programming, with public data sets, to attempt to model and hence estimate software project effort. The main research question is whether genetic programs can offer 'better' solution search using public domain metrics rather than company specific ones. Unlike most previous research, a realistic approach is taken, whereby predictions are made on the basis of the data available at a given date. Experiments are reported, designed to assess the accuracy of estimates made using data within and beyond a specific company. This research also offers insights into genetic programming's performance, relative to alternative methods, as a problem solver in this domain. The results do not find a clear winner but, for this data, GP performs consistently well, but is harder to configure and produces more complex models. The evidence here agrees with other researchers that companies would do well to base estimates on in house data rather than incorporating public data sets. The complexity of the GP must be weighed against the small increases in accuracy to decide whether to use it as part of any effort prediction estimation.

1 Introduction

Reliable predictions of project costs — primarily effort — are greatly needed for better planning of software projects, but unfortunately size and costs are seldom, if ever, proportionally related [8]. There has been extensive research into software project estimation, with researchers assessing a number of approaches to improving prediction accuracy. One of the first methods to estimate software effort automatically was COCOMO [4], where in its simplest form effort is expressed as a function of anticipated size as:

$$E = aS^b \tag{1}$$

where E is the effort required, S is the anticipated size and a and b are domain specific parameters. Independent studies, for example, Kitchenham and Taylor [14] and Kemerer [12] found many errors considerably in excess of 100% even after model calibration.

E. Cantú-Paz et al. (Eds.): GECCO 2003, LNCS 2724, pp. 2477–2487, 2003.

Subsequently attempts have been made to model data automatically based on local collection, for example, Kok *et al.* [15] published encouraging results using stepwise regression. Such linear modelling can cover only a small part of the possible solution space, potentially missing the complex set of relationships evident in many complex domains such as software development environments.

A variety of machine learning (ML) methods have been used to predict software development effort. Good examples include artificial neural nets (ANNs) [3, 28], case based reasoning (CBR) [9, 23] and rule induction (RI) [20]. Hybrids are also possible, e.g. Shukla [24] reports better results from using an evolving ANN compared to a standard back propagation ANN. Dolado and others [6, 7] analyse many aspects of the software effort estimation problem and present encouraging results for a genetic programming (GP) based estimation using a single input variable. Burgess and Lefley [5] also had some success using GP based estimation.

One characteristic to all ML methods is the need for training data. However, recent software engineering research has found that shared or public data sets are much less effective than restricting the prediction system to potentially very few local cases [11, 21]. The issue at stake is whether a company can improve prediction accuracy by incorporating results from other companies. Generally the more data available to a learner, the better it can model behaviour. However, no matter how good the control, some metrics are likely to be measured differently across companies and the working environments may have a marked difference. Thus there will be some distortion of the models accuracy. The results reported by [11, 21] for non-evolutionary models found the larger data sets to provide less accurate estimates. This suggests that companies should only use their own data, assuming they have sufficient examples close to a new case to make an estimate. Their research used models based on regression, CART, robust regression, stepwise ANOVA and analogy based estimation. Genetic programming offers an evolutionary solution to estimation problems that may better take into account the source of data. For example a variable indicating in house or external could be used as a multiplier to ignore data from outside sources and so has the potential to build a prediction system at least as accurate as one based on only internal data.

To summarise, the aim of this paper is to consider the question: can the use of evolutionary models overcome the problems of using non-local data to build project effort prediction systems. The ability to do this is of particular interest to environments that have collected little local data, a seemingly common occurrence within software engineering. The remainder of the paper describes the data set used for our analysis, considers how prediction accuracy might be assessed and then reviews the application of ANNs, nearest neighbour (NN) and GP methods. Next we provide more detailed information on our GP method. This is followed by results, conclusions and suggestions for further work.

2 The Data Set

The data used for the case study is often referred to as the 'Finnish data set' and is made up of project data collected by the software project management consultancy organisation SSTF. This data set contains 407 cases described by more than 90 features including metrics such as function points. This data set used in this paper is

quite large for this type of application due to the fact that it comprises project data from many organisations interested in benchmarking their software productivity. The features are a mixture of continuous, discrete and categorical. However, there are missing data values and features not be known at prediction time and so are not included in the development of a prediction system. Removing features with missing values or values that cannot be known until after the prediction is required, leaves a subset of 83 features used for this case study. The data set exhibits significant multicollinearity, in other words there are strong relationships between features as well as with the dependent variable. More background information on this can be found in [21] which describes an earlier version of the same data set.

The total project effort variable was removed and used as the output or dependent variable. This is used to model the training data and to evaluate the test data. Where data was in the form of a string it was changed to an index, so that it could be processed numerically. This included the company number; to assist the models the chosen company was allocated the special code index 0. All values were scaled by range to fall between 0 and 1. In order to provide a realistic assessment of the hypothesis, the data was split chronologically into training and test sets. The data was sorted by start dates in such a way that a reasonable number of the projects from one company were available for training and testing. Thus we model the situation on 15th October 1991 when the selected company had completed 48 projects, with another 15 yet to commence. At this cut off date, there was data available from another 101 projects from other companies apart the one chosen for modelling. Data from the other companies beyond the cut off date was discarded.

3 Comparing Prediction Techniques

Our chief requirement is to obtain a quantitative assessment of the accuracy of the different prediction techniques. This achieved by comparing software project effort estimates made over a number of cases with known actual effort. Typically these cases are not made available during the configuration and training stages in order to avoid bias. There are many ways for calculating and combining the accuracy measures obtained, with no simple way of ranking these techniques, even for a given domain [13]. Most learners use an evaluation or fitness function to determine error to decide on future search directions. The choice of method to make this evaluation is crucial for accurate modelling. All of the learners considered here use root average mean square error. This is important also when considering accuracy estimates as the evaluation function will bias search towards similar evaluation metrics.

A number of accuracy — strictly speaking residual — summary statistics can be used to make comparisons between models. Each captures different aspects of the notion of model accuracy. Ultimately the decision should be made by the user, based on estimates of the costs of under or over estimating project effort, how risk averse they are and so forth. All are based on the calculated error or residual, that is the difference between the predicted and observed output values for a project. Most use an average performance over the whole set so we also include the worst case as an illustration of a less generalised metric. We also assess results using AMSE, adjusted mean square error (see Equation 2). This is the sum of squared errors, divided by the

product of the means of the predicted and observed outputs. For further details on these metrics, see [5].

$$\text{AMSE} = \sum_{i=1}^{i=n} \frac{(E_i - \hat{E}_i)^2}{\left(\overline{E}_i * \overline{\hat{E}}_i\right)} . \tag{2}$$

In summary the accuracy metrics used for this research are:

1. Correlation coefficient (Pearson) of actual and predicted
2. Adjusted mean square error - AMSE
3. Percentage of predictions within 25% - Pred(25)%
4. Mean magnitude of relative error - MMRE
5. Balanced mean magnitude of relative error - BMMRE
6. Worst case error as %

We also consider qualitative factors adapted from Mair *et al*. [20]: accuracy, explanatory value and ease of configuration. The ease of set up and quality of information provided for the estimator is of great importance. Empirical research has indicated that end-users coupled with prediction systems can outperform either prediction systems or end-users alone [25]. The more explanations given for how a prediction was obtained, the greater the power given to the estimator. If predictions can be explained, estimators may experiment with "what if" scenarios and meaningful explanations can increase confidence in the prediction.

4 Machine Learning and Software Effort Estimation

Many machine learning methods have been used for effort estimation, though there are few comparisons. Work by Burgess and Lefley [5] found that GPs performed similarly to other advanced models. As part of this investigation we consider different learners to benchmark the achievements of the GP system. Results are presented for models based on

- Random
- Least squares regression
- Nearest neighbour
- Artificial neural network
- Genetic programming
- Average of all non-random estimators

The random method uses the substitution of an output variable, that is effort, randomly selected from the training set and is included as a benchmark.

4.1 Artificial Neural Networks (ANNs)

ANNs are parallel systems inspired by the architecture of biological neural networks, comprising simple interconnected units, called artificial neurons. The neuron com-

putes a weighted sum of its inputs and generates an output activated by a stepped function. This output is excitatory (positive) or inhibitory (negative) input fed to other neurons in the network. Feed-forward multi-layer perceptrons are the most commonly used form of ANN, although many more neural network architectures have been proposed. The ANN is initialised with random weights. The network then 'learns' the relationships implicit in a data set by adjusting the weighs when presented with a combination of inputs and outputs that are known as the training set.

Studies concerned with the use of ANNs to predict software development effort focus on comparative accuracy with algorithmic models, rather than on the suitability of the approach for building software effort prediction systems. An example is the investigation by Wittig and Finnie [28]. The results from two validation sets are aggregated and yield a high level of accuracy viz. Desharnais set MMRE=27% and ASMA set MMRE=17%, although some outlier values are excluded. Mair *et al.* [20] show neural networks offer accurate effort prediction, though not as good as Wittig and Finnie, concluding that they are relatively difficult to configure and interpret.

A number of experiments were needed to determine an appropriate number of hidden nodes, with 20 being used for this investigation. Other parameters such as learning rate were set to values chosen by experience and experimentation. It was found that nearby values for parameters only slightly affected convergence performance.

4.2 Nearest Neighbour Techniques

Examples of successful case based reasoning (CBR) tools for software project estimation include: cEstor [26], a CBR system dedicated to the selection of similar software projects for the purpose of estimating effort, and more recently, FACE [2] and ANGEL [22]. We use a simple NN technique and a weighted (by the reciprocal of the Euclidean distance) average of three neighbours as simple form of CBR. The Euclidean distance is determined by the root sum of squares of the difference between the scaled input variables for any two projects.

5 Background to Genetic Programming

Genetic Algorithms were developed as an alternative technique for tackling general optimisation problems with large search spaces. They have the advantage that they do not need any prior knowledge, expertise or logic related to the particular problem being solved. Occasionally, they can yield the optimum solution, but for most problems with a large search space, a good approximation to the optimum is a more likely outcome. The basic ideas used are based on the Darwinian theory of evolution, which in essence says that genetic operations between chromosomes eventually leads to fitter individuals which are more likely to survive. Thus, over time, the population of the species as a whole improves. However, not all of the operations used in the computer analogy of this process necessarily have a biological equivalent.

Genetic Programming is an extension of Genetic Algorithms, which removes the restriction that the chromosome representing the individual has to be a fixed length binary string. In general for GP, the chromosome is some type of program, which is then executed to obtain the required results. One of the simplest forms of program, used for this

application, is a binary tree containing operators and operands. This means that each solution is an algebraic expression that can be easily evaluated. Koza "offers a large amount of empirical evidence to support the conclusion that GP can be used to solve a large number of seemingly different problems from many different fields" [16].

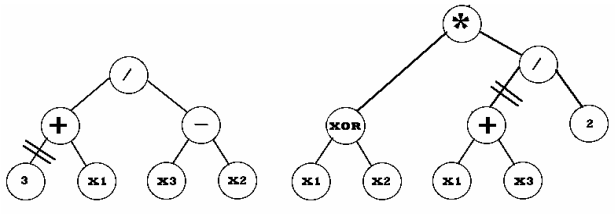

Fig. 1. Illustration of crossover operator before operation. The double line illustrates where the trees are cut

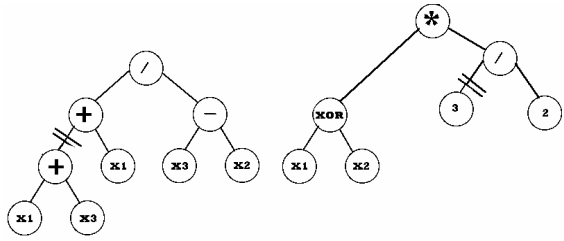

Fig. 2. Illustration of crossover operator after operation. The double line illustrates where the sub-trees have been swapped.

The main genetic operations used are crossover and mutation. The crossover operator chooses a node at random in each chromosome and the branches to those nodes are cut. The sub trees produced below the cuts are then swapped. The method of performing crossover is easily illustrated using an example (see figures 1 and 2). Although this example includes a variety of operations, only the set { + , - , * , / } were made available for our experiments. More complex operators can, of course, be developed by combining these simple operations. Simple operators eliminate the bounding problems associated with more complex operations such as XOR, which is not defined for negative or non-integer values. The multiply is a protected multiply which prevents the return of any values greater than 10^{20}. Similarly there is a protected divide, division by 0 returning 1000. This is to minimise the likelihood of real overflow occurring during the evaluation of the solutions.

The initial solutions were generated using "ramped half and half". This means that half the trees are constructed as full trees, i.e. operands only occur at the maximum depth of the tree, and half are trees with a random shape. Often elitism, where the top *x*% of the solutions, as measured by fitness, is copied straight into the next generation, is used to maintain the best solutions, where *x* can be any value but is typically 1 to 10.

Since trees tend to grow as the algorithm progresses, a maximum depth is normally imposed (or maximum number of nodes) in order that the trees do not get too large. This is often larger than the maximum size allowed in the initial population. Any trees produced by crossover that are too large are discarded. The reasons for imposing the size limit is to save on both the storage space required and the execution

time needed to evaluate the trees. There is also no known evidence that allowing very large trees will lead to better results and smaller trees are more likely to generalise. Key parameters that have to be determined, in order to get good solutions without using too much time or space, are: the best genetic operators, the population size, maximum tree size, number of generations, etc. More information related to Genetic Programming can be found in [1, 17, 18].

6 Applying GP to Software Effort Estimation

The software effort estimation problem may be modeled as a symbolic regression problem, which means, given a number of sets of data values for a set of input parameters and one output parameter, construct an expression of the input parameters which best predicts the value of the output parameter for any set of values of the input parameters. In this paper, GP is applied to the data set already described in section 2, with the same 149 projects in the Training Set and the remaining 15 projects in the Test Set in a similar way to the other ML methods. Ease of configuration depends mainly on the number of parameters required to set up the learner. For example, a nearest neighbour(NN) system needs parameters to decide the weight of the variables and the method for determining closeness and combining neighbours to form a solution.

Koza [16] lists the number of control parameters for genetic programming as being 20 as compared to 10 for neural networks, but it is not always easy to count what constitutes a control parameter. However, it is not so much the number of the parameters as the sensitivity of the accuracy of the solutions to their variation. It is possible to search using different parameters but this will depend on their range and the granularity of search. In order to determine the ease of configuration for a genetic program, we test empirically whether parameter values suggested by Koza offer sufficient guidance to locate suitably accurate estimators. Similarly, all of our solutions use either limited experimentation or commonly used values for control parameters. The parameters chosen after a certain amount of experimentation with the Genetic Programming system are shown in Table 1.

Table 1. Main parameters used for the GP system

Parameter	Value
Size of population	1000
Number of generations	500
Number of runs	10
Maximum initial full tree depth	5
Maximum no. of nodes in a tree	64
Percentage mutation rate	5%
Percentage of elitism	5%

No function ‘seeding’ was used and only a simple mutation operator was available based on substitution of parameters. The initial solutions were generated using "ramped half and half". This means that half the trees are constructed as full trees, i.e.

operands only occur at the maximum depth of the tree, and half are trees with a random shape.

The GP system, as with all the systems considered here, is written in power basic running on a 1GHz Pentium PC with 512Mbytes of RAM. However, the run-time memory usage is less than 1Mbytes to store the trees and population data required for each generation. Careful garbage management ensures this does not grow with successive generations. One run of 500 generations takes about 40 minutes processor time for the full data set, 25 minutes using the smaller, company specific set.

The In order to appreciate the performance of the GP the test is also performed on some other suitable techniques.

7 Results of the Comparison

This study evaluates various statistical and machine learning techniques, including a genetic programming tool, to make software project effort predictions. The main hypothesis to be tested is whether GP can significantly produce a better estimate of effort using a public data set rather than a company specific database. We make comparisons mostly based on the accuracy of the results but we also consider the ease of configuration and the transparency of the solutions. Note that in the comparison, learners that contain a stochastic element are tested over a population of solutions, independently generated with the best modeller (smallest average squared error) used.

7.1 Accuracy of the Predictions

The ANN and GP solutions required some experimentation to find good control parameters. All of the training sessions were successful and no problems with local minima were encountered. Table 2 lists the main results used for using general, company wide data. Table 3 shows the same results for the company specific data set.

Table 2. Comparing different prediction systems for the full data set. Best performing estimators (within 5% of best) are highlighted in bold type. Best performers across both data sets are shown in italics

	Estimated effort						
	Random	LSR	1-NN	3-NN	ANN	GP	Average
correlation	-0.161	0.846	0.390	0.497	0.806	**0.937**	*0.890*
AMSE	28.091	4.601	13.970	14.720	6.584	**1.981**	3.596
Pred(25)%	13.333	***40.000***	33.333	20.000	26.667	***40.000***	33.333
MMRE	166.39	46.925	85.401	59.192	68.856	***37.670***	43.467
BMMRE	238.35	73.629	110.022	79.049	85.478	62.797	***47.343***
Worst case MRE	609	150	332	185	195	**125**	167

Table 3. Comparing different prediction systems for the company specific data set. Best performing estimators (within 5% of best) are highlighted in bold type. Best performers across both data sets are shown in italics.

	Estimated effort						
	Random	LSR	1-NN	3-NN	ANN	GP	Average
correlation	-0.167	***0.927***	0.104	0.793	***0.951***	***0.933***	***0.939***
AMSE	77.762	2.523	19.347	5.868	2.015	12.672	***1.796***
Pred(25)%	6.667	26.667	26.667	20.000	***40.000***	33.333	***40.000***
MMRE	381.422	45.648	129.045	114.709	69.228	***37.959***	65.694
BMMRE	448.695	63.995	217.353	128.525	71.166	**54.015**	71.888
Worst case MRE	1865	174	436	495	254	***79***	198

The three strongest techniques overall appear to be LSR, ANN and GP with GP achieving the best (or within 5%) level of accuracy the most often. The NN techniques are consistently less accurate. Using an average of each technique performs well for the company specific data set but is less successful for the full data set.

In terms of our main hypothesis, that it is better to restrict data to locally relevant projects, the evidence is somewhat mixed. For the poorest techniques, most notably 1-NN, this is clearly not the case as in all cases the results from the full data set are to be preferred to the company specific data set. With ANN and LSR the results for the company specific data set are generally better than those for the full data set. This agrees with other researchers that companies would do well to base estimates on their own data rather than incorporating public data sets. The GP results are almost identical in terms of the correlation coefficients and pred(25) indicator, substantially improved for BMMRE and the worst case, yet worse for MMRE and AMSE. It may be that the latter two indicators are affected by a different distribution of under and over-estimates since they are both in asymmetric in their treatment of such estimates. Overall the results are weakly suggestive of the benefits of using preferring local data particularly for LSR, ANN and too a lesser extent GP. However, where local data is very limited, it is possible that the public data set could improve estimation accuracy.

7.2 Qualitative Assessment

It was fairly easy to configure the ANN but we have a fair degree of past experience in this area. Generally, different architectures, algorithms and parameters made little difference to results, but some expertise is required to choose reasonable values. Although heuristics have been published on this topic [10, 27], we found the process largely to be one of trial and error, guided by experience. ANN techniques would not be easy to use for project estimation by end-users as some expertise is required. We have found that investigation of using methods such as those reported in [19] can improve understanding of results from the ANN but that such information is limited.

GP has the most degrees of freedom in design, for example the choice of functions, mutation rate and strategy, percentage elitism, dealing with out of bounds conditions, creation strategy and setting maximum tree sizes and depths and so on. We found by using suggestions by Koza [16] and experimentation, we obtained convergence but again, at present, expertise is necessary. Many experiments were tried with various

approaches and a number are still pending. Such effort suggests that GP only be used if it provides greater accuracy, not supported by our quantitative results for this data. The resulting program, like most of the techniques here, may provide information on the relative importance of input variables. Typical GP solutions for this problem are provided below.

$$((X13 + X17 + (((((((X62 + X56) * (X20 + X33)) * (((0.25 * X34) * (X32 + X70 - X19)) + (X20 + X80))) - (X40 - X9)) - (((X4 * X69) + X39 + X47) + ((2.1 + X28 + X62) * X4 * X56))) - X75 + X2) + 2.1 + X28 + X62) + (X20 + X33))) + (3 * X56 + X9 - X21 - X19 + X62)) * 0.25 * X83 \quad (3)$$

$$X83 / (X45 * X4) \quad (4)$$

Equation 3 is not easy to comprehend, unlike equation 4, which is of forced, limited length. The latter type of solution was found to be marginally less accurate but might provide the software manager with clearer insight to the problem.

8 Conclusions and Future Work

There is no clear winner on accuracy but GP performed well for this problem. This is in good agreement with other research [5, 7] using different approaches and data sets. The company specific ANN also performed well and the average over all methods for this data is also particularly good. Generally the better techniques (LSR, ANN and GP) are slightly more accurate with the company specific database.

The results, however, highlight that measuring accuracy is not simple and researchers should consider a range of measures. Perhaps a useful conclusion is that based on the evidence from this experiment, the more elaborate estimation techniques such as ANNs and GPs provide better fitting models but this slight advantage must be weighed against the extra effort in design, and a simple LSR can still provide useful estimates.

Future work will center on a greater variety of GP models, though we need to ensure that by trying so many models leads to one model performing very well by chance over fit. The stochastic population nature of GP can be used to reduce this effect. Models that will be used include better mutation (by tree generation rather than parameter swapping), seeding (for example with the LSR) and parameter tuning.

Acknowledgements. The authors are indebted to STTF Ltd for making the Finnish data set available.

References

[1] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francome, *Genetic Programming: An introduction*. San Mateo, CA: Morgan Kaufmann, 1998.

[2] R. Bisio and F. Malabocchia, "Cost estimation of software projects through case base reasoning," presented at 1st Intl. Conf. on Case-Based Reasoning Research & Development, 1995.

[3] J. Bode, "Neural networks for cost estimation," *Cost Engineering*, vol. 40, pp. 25–30, 1998.

[4] B. W. Boehm, *Software Engineering Economics*. Englewood Cliffs, N.J.: Prentice-Hall, 1981.

[5] C. J. Burgess and M. Lefley, "Can genetic programming improve software effort estimation? A comparative evaluation," *Information & Software Technology*, vol. 43, pp. 863–873, 2001.
[6] J. J. Dolado, "Limits to methods in software cost estimation," presented at 1st Intl. Workshop on Soft Computing Applied to Software Engineering, Limerick, Ireland, 1999.
[7] J. J. Dolado, "On the problem of the software cost function," *Information & Software Technology*, vol. 43, pp. 61–72, 2001.
[8] S. Drummond, "Measuring applications development performance," in *Datamation*, vol. 31, 1985, pp. 102–8.
[9] G. R. Finnie, G. E. Wittig, and J.-M. Desharnais, "Estimating software development effort with case-based reasoning," presented at 2nd Intl. Conf. on Case-Based Reasoning, 1997.
[10] S. Huang and Y. Huang, "Bounds on the number of hidden neurons," *IEEE Trans. on Neural Networks*, vol. 2, pp. 47–55, 1991.
[11] R. Jeffery, M. Ruhe, and I. Wieczorek, "Using public domain metrics to estimate software development effort," presented at 7th IEEE Intl. Metrics Symp., London, 2001.
[12] C. F. Kemerer, "An empirical validation of software cost estimation models," *Communications of the ACM*, vol. 30, pp. 416–429, 1987.
[13] B. A. Kitchenham, S. G. MacDonell, L. Pickard, and M. J. Shepperd, "What accuracy statistics really measure," *IEE Proceedings - Software Engineering*, vol. 148, pp. 81–85, 2001.
[14] B. A. Kitchenham and N. R. Taylor, "Software cost models," *ICL Technical Journal*, vol. 4, pp. 73–102, 1984.
[15] P. Kok, B. A. Kitchenham, and J. Kirakowski, "The MERMAID approach to software cost estimation," presented at Esprit Technical Week, 1990.
[16] J. R. Koza, *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA: MIT Press, 1992.
[17] J. R. Koza, *Genetic Programming II: Automatic discovery of reusable programs*: MIT Press, 1994.
[18] J. R. Koza, F. H. Bennett, D. Andre, M. A. Keane, and (). *Genetic Programming III: Darwinian Invention and Problem Solving*. San Mateo, CA: Morgan Kaufmann, 1999.
[19] M. Lefley and T. Kinsella, "Investigating neural network efficiency and structure by weight investigation," presented at European Symp. on Intelligent Technologies, Germany, 2000.
[20] C. Mair, G. Kadoda, M. Lefley, K. Phalp, C. Schofield, M. Shepperd, and S. Webster, "An investigation of machine learning based prediction systems," *J. of Systems Software*, vol. 53, pp. 23–29, 2000.
[21] K. Maxwell, L. Van Wassenhove, and S. Dutta, "Performance evaluation of general and company specific models in software development effort estimation," *Management Science*, vol. 45, pp. 787–803, 1999.
[22] M. J. Shepperd and C. Schofield, "Estimating software project effort using analogies," *IEEE Transactions on Software Engineering*, vol. 23, pp. 736–743, 1997.
[23] M. J. Shepperd, C. Schofield, and B. A. Kitchenham, "Effort estimation using analogy," presented at 18th Intl. Conf. on Softw. Eng., Berlin, 1996.
[24] K. K. Shukla, "Neuro-genetic prediction of software development effort," *Information & Software Technology*, vol. 42, pp. 701–713, 2000.
[25] E. Stensrud and I. Myrtveit, "Human performance estimating with analogy and regression models: an empirical validation," presented at 5th Intl. Metrics Symp., Bethesda, MD, 1998.
[26] S. Vicinanza, M. J. Prietula, and T. Mukhopadhyay, "Case-based reasoning in effort estimation," presented at 11th Intl. Conf. on Info. Syst., 1990.
[27] S. Walczak and N. Cerpa, "Heuristic principles for the design of artificial neural networks," *Information & Software Technology*, vol. 41, pp. 107–117, 1999.
[28] G. Wittig and G. Finnie, "Estimating software development effort with connectionists models," *Information & Software Technology*, vol. 39, pp. 469–476, 1997.