A Comparison of Alternatives to Regression Analysis as Model Building Techniques to Develop Predictive Equations for Software Metrics

Mr Andrew R. Gray
Dr Stephen G. MacDonell[1]
Computer and Information Science
University of Otago

March 1996

Abstract

The almost exclusive use of regression analysis to derive predictive equations for software development metrics found in papers published before 1990 has recently been complemented by increasing numbers of studies using non-traditional methods, such as neural networks, fuzzy logic models, case-based reasoning systems, rule-based systems, and regression trees. There has also been an increasing level of sophistication in the regression-based techniques used, including robust regression methods, factor analysis, resampling methods, and more effective and efficient validation procedures. This paper examines the implications of using these alternative methods and provides some recommendations as to when they may be appropriate. A comparison between standard linear regression, robust regression, and the alternative techniques is also made in terms of their modelling capabilities with specific reference to software metrics.

[1] Address correspondence to: Dr S. MacDonell, Lecturer, Department of Information Science, University of Otago, P.O. Box 56, Dunedin, New Zealand. Fax: +64 3 479 8311 Email: stevemac@commerce.otago.ac.nz

1 Introduction

Almost all research carried out to develop predictive software metrics has focused on linear regression analysis for implementation, often after using various transformations to permit non-linearities, leading to models expressed as mathematical equations. Such models include SLIM (Putnam, 1978), and COCOMO (Boehm, 1981). While there are many advantages to using such techniques, especially in the simplicity of model building and implementation, it can be argued that by using other techniques, to be discussed in this paper, more useful models may be derived in at least some circumstances. As Briand et al. (1992) state, classical statistical methods have limited model building capabilities with regard to software development models. With this in mind a thorough consideration of the alternative techniques available can be regarded as essential for selecting the methods that are most suited to the particular model development task at hand.

Given the large expenditures made by many companies for the development of software, even small increases in prediction accuracy are likely to be worthwhile. Underestimating costs can lead to accepting projects that do not provide sufficient returns or that overrun schedules, possibly with catastrophic consequences. Overestimating costs can lead to sound projects being rejected, and can lead to gaps between one project ending and another starting. This idle time can be expensive in competitive time-to-market industries. Either way, it is clear that more accurate estimates have considerable value to a corporation involved in software development. Once an estimation model has been derived it is important that the limitations of the techniques used to develop and implement the model are understood in order to ensure that it is only used within its limitations. For example, extrapolations outside the range of data used for development should not be attempted.

Three broad areas of concern can be cited with regard to software development models, with the first set of difficulties perhaps the most problematic. Software engineering data sets often have a number of qualities that make analysis difficult including missing data, large numbers of variables (leading to lower degrees of freedom), strong collinearity between the variables, heteroscedasticity, complex non-linear relationships, outliers, and small data sets. These factors all make the modelling process that much more difficult and the models derived by the process less reliable. Some of these problems can be at least partially overcome. For example heteroscedasticity can be reduced, or even eliminated, by various transformations; and collinearity can be removed by factor analysis. Other problems, such as missing data and a low number of degrees of freedom (arising from a large number of parameters to be estimated as compared to the number of available observations) cannot be easily overcome, and certain techniques may be unsuitable for a particular data set affected by these problems.

A second area of concern is the acceptability and validation of the models. This includes the issue of the model explaining its predictions. Software metrics expressed as equations are often less than meaningful, especially with several variables being used in the model, sometimes including interaction terms and non-linear transformations, and without sufficient semantic meaning attached to the model a satisfactory level of validation is unlikely to be achieved. This problem is made more serious by the small data sets commonly used for developing these models. An absence of validation can result in minimal or no acceptance of a model and its underlying metric and therefore the model may not be used for project planning purposes. With small data sets it is entirely possible for a model to be developed that while it may fit the data, violates common sense, for example slopes may be counter-intuitive to what would be expected. This can include a slope being of a different magnitude than expected, especially in relation to other variables, or a slope may even have the opposite sign to what common-sense would suggest.

The final area of concern considered here is that of generalisability. Since the first predictive software metrics were derived, attempts have been made without great success to apply the models associated with them, without recalibration, to other types of projects within the developing organisation and even to other organisations, the use of *standard* COCOMO coefficients being a case in point. The necessity of being able to easily recalibrate a model for another environment is supported by numerous authors, including Jeffery and Low (1990). Kemerer (1987) found that models that were uncalibrated resulted in relative errors of up to 600%. Linear regression models are easily regenerated, but they do not always generalise well given their susceptibility to influence by outliers.

Each modelling technique to be discussed here contributes in at least one way to resolving some of these problems in a manner that can be seen as (or at least argued as being) superior to standard linear regression analysis. This is not to say that regression analysis should not be used, but rather that the best technique for the specific problem at hand may not always be regression and without being aware of the alternatives the best model may not always be developed.

As well as using linear least squares regression some researchers have recognised the problems caused by outliers in software data sets and have used robust regression methods (MacDonell, 1993; Miyazaki et al., 1994). In other cases linear regression has been used in conjunction with more powerful statistical analysis techniques such as factor analysis (Coupal and Robillard, 1990; Mata-Toledo and Gustafson, 1992; Subramanian and Breslawski, 1993). The presence of these techniques in the literature indicates the increasing statistical sophistication to be found in the field of software metrics.

Recently a number of researchers have begun to use neural networks as an alternative to regression analysis (Sheppard and Simpson, 1990; Karunanithi et al., 1992; Hakkarainen et al., 1993; Wittig and Finnie 1994; Kumar et al., 1994; Wittig 1995; Srinivasan and Fisher, 1995; Khoshgoftaar and Lanning, 1995; Sherer, 1995). While this use of a new modelling technique is to be encouraged, many of the studies examined in the literature contain methodological flaws in their application of neural networks. This can partly be explained by the lack of understanding of the statistical concepts underlying neural networks, and also by the widely propagated myth of neural networks as *automatic problem solvers*. Sarle (1994) observes that neural networks are often marketed as being usable without any experience, where in reality using a neural network to solve a problem requires an equivalent amount of knowledge to what would be needed to solve that problem using statistical analysis, as well as the ability to recognise the relationships between neural networks and statistics. In the case of the backpropagation trained network, that would require some knowledge of multiple non-linear regression as well as an understanding of neural networks.

Other techniques that have featured in recent publications include fuzzy systems (Bastani et al., 1993; Kumar et al., 1994), case-based reasoning (Mukhopadhyay et al., 1992), regression trees and classification trees (Selby and Porter, 1988; Porter and Selby, 1990; Srinivasan and Fisher, 1995), and rule-based systems (Ramsey and Basili, 1989; Lakhotia, 1993; Griech and Pomerol, 1994). While these techniques are less frequently used than neural networks, they are all useful model building methods, especially regression trees and case-based reasoning.

In addition to the above selection of techniques which have already been used in published studies for software metrics, an additional technique is considered in this paper, that of neural network and fuzzy logic hybrids, here referred to as neuro-fuzzy systems. While they have not as yet, to the authors' knowledge, been used for deriving models for software metrics they are seen as especially worthy of consideration since they promise the benefits of both neural networks and fuzzy systems without all of the associated drawbacks to these techniques, and are the subject of a study currently being performed by the authors.

2 Requirements for Software Metric Modelling Techniques

A number of requirements can be stated regarding modelling techniques for software metrics. While it may be unrealistic to expect any given technique to satisfy all requirements, the chosen technique should best satisfy those requirements most important to the current problem. The relative importance of each requirement will vary depending on the nature of the data and the intended purpose of the model.

Given the type of data set usually available, the modelling technique used will normally be required to deal with complex relationships and *messy data*. This *messiness* in software engineering data sets is often caused by collinearity (independent variables being related to one another) and heteroscedasticity (non-constant variance in the dependent variable with respect to one or more of the independent variables), as well as the high frequency of outliers. Another problem is that of missing data values. Thus, any technique used for developing models of metrics needs the capability of determining the important variables, finding appropriate transformations, discarding or weighting the outliers as less important where this is justified, and estimating missing data values.

Ideally, a modelling technique will also support easy recalibration as new data becomes available or the model is applied to new situations. Jeffery and Low (1990) found that model calibration was required to obtain a satisfactory level of accuracy in prediction for linear models. We would expect that the reasons for this requirement, the changes in project factors, will be found irrespective of the modelling technique employed, although different types of models will experience different levels of difficulty if used uncalibrated.

The remainder of the paper continues with a discussion of each modelling technique in turn. An illustrative example is provided for each method (aside from least squares regression) to demonstrate its usefulness for modelling software metrics. This is followed by a section comparing the techniques based on a number of criteria considered important for the modelling task. Finally, the paper concludes with some general observations and suggestions for additional research.

3 Least Squares Regression

Linear least squares regression analysis is still the most common technique used in the literature (MacDonell and Gray, 1996). Much of the appeal of this technique lies with its simplicity and also its easy accessibility from many of the popular statistical packages. Linear least squares regression operates by estimating the coefficients in order to:

$$\underset{\hat{\theta}}{minimise} \sum_{i=1}^{n} r_i^2$$

where r_i is equal to the residual between the observed data and the model's prediction for the ith observation. Thus all observations are taken into account, allowing for a single outlier to have a marked influence on the regression line derived.

Least squares regression is well suited for use in situations where:

- many degrees of freedom are available (that is, there are many more observations than parameters to be estimated),
- the data is well-behaved (no outliers or heteroscedasticity),
- a small number of independent variables are sufficient, after transformations if necessary, to linearly predict the possibly transformed output variable(s), and
- there is no missing data.

This places a severe restriction on the use of this technique for software engineering data sets that rarely meet all of these conditions (Briand et al., 1992). At the very least, robust regression or some form of outlier detection should be used to improve the accuracy of the estimates. Even the use of transformations on the data set can be capable of producing a much more useful model. Despite this, the majority of papers do not mention any attempts to use transformations to improve the data model fit (MacDonell and Gray, 1996).

Experiments involving linear regression often become a matter of finding some combination of available variables linearly correlated to the output variable, sometimes after trying various transformations. This has been referred to as the *shotgun approach* (Courtney and Gustafson, 1993) or more generously, *data mining* (Lovell, 1983). Courtney and Gustafson also discuss the dangers of relying on correlation coefficients where hypotheses have not been proposed in advance. The stating of the hypothesis to be tested before any experimental work is carried out is required for unbiased results for all techniques to be discussed in this paper.

One of the most important parts of developing a model when using a number of different trials is the validation of the finally selected model. Data splitting for linear regression is discussed in Picard and Berk (1990), and Snee (1977) provides a more general discussion of data splitting and model validation issues. Data should ideally be divided into three sets, a set of usually half to three-quarters of the data for developing the models, a second set for selecting the best model based on all models' performance for this set, and the remainder for testing the best model's fit. It is only the performance on this third data set that provides an unbiased estimate of the predictive capabilities of the final model. Stating the levels of performance on non-validation data sets, while common practice (MacDonell and Gray, 1996), provides exaggerations of the model's predictive ability. If only one model is being tested then the data should be split into two sets, one for development, the other for validation of accuracy. Most of the data in this case should be used for development, and only a quarter to third for validation (Picard and Berk, 1990). While the small size of data sets often reported in the literature prevents data splitting, other techniques are available, although not widely used (MacDonell and Gray, 1996). Other

alternatives to data splitting are the PRESS statistic (Allen, 1971) and resampling-based methods such as the bootstrap (Efron, 1979; Young, 1994).

Collinearity inflates the error terms of estimates, leading to less reliable models. Such relationships are common in software data sets, for example the number of entities and the number of attributes in a data model can be quite reasonably argued as being related on the basis that a change in the number of entities would also be reflected in the number of attributes present in the data model. While the relationship here will probably not hold perfectly (since normalisation may change the number of entities greatly without much change to the number of attributes in the data model) there is still correlation between the two variables. This problem can be dealt with by reducing the number of dependent predictors used for estimation. Factor analysis and other data reduction techniques can be used to reduce the number of influencing components under consideration when developing a software metric model (Coupal and Robillard, 1990). This can be used to *group* variables that measure the same aspect into single factors, each representing a major dimension within the data. In the data model example mentioned above, a single data model size factor may be extracted as a combination of the numbers of entities and attributes. See Stewart (1987) for a discussion of methods of detecting collinearity in regression-based models.

4 Robust Regression Analysis

Robust regression analysis has been used in MacDonell (1993) and Miyazaki et al. (1994) to screen for outliers in software metric models. The general idea behind robust regression is that by changing the error measure (from least squares) the model can be made more resilient to outlying data points. Many different robust regression models exist, often based on median, rather than mean, measures of error (for example, Rousseeuw, 1984) or on some middle portion of the errors (for example, ignoring the top and bottom ten percent of errors, and using least squares on the remaining eighty percent).

The use of robust regression is especially attractive in software development data sets since they are often very small, and therefore extremely sensitive to the abnormal observations they contain, and often contain errors in measurement. On the other hand, the small size of the data sets available can make researchers reluctant to give up an observation since this also reduces the statistical validity of models they develop. *Least Median Squares* regression (Rousseeuw, 1984) provides estimates which cannot be affected to an arbitrary degree by up to 50 percent contamination, that is to say it has a *breakdown point* of 0.5. This compares to Least Squares regression's estimates which can be arbitrarily affected by a *single* outlying observation, which

is a breakdown point of 0. This improvement is achieved by using the following method of estimating the coefficients:

$$\underset{\hat{\theta}}{minimise}\ \underset{i}{median}\ r_i^2$$

with r_i equalling the residual as with least squares above. The median residual can only be arbitrarily affected if at least half of the data changes. This method can be used to find points that deviate from the median regression line, which may suggest that these points are worthy of consideration to ensure that they are not outliers (Massart et al., 1986).

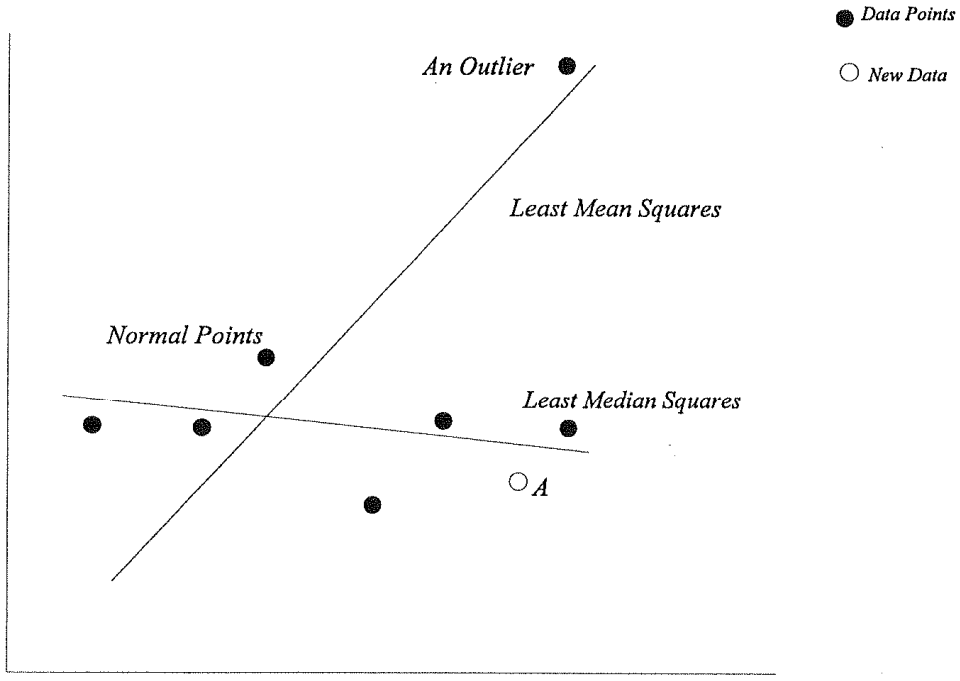

Figure 1

As can be seen in figure 1, a single outlier has a dramatic effect on the regression line under Least Mean Squares, but none under Least Median Squares. Although, mathematically, the Least Mean Squares line is minimising its error function it is easy to see that the model would be of little use in predicting a new point like A. Additional information about Least Median Squares regression can be obtained from Rousseeuw and Leroy (1987). Some of the weaknesses of the method are discussed in Hettmansperger and Sheather (1992).

An important point to remember here is that robust regression, as with any outlier detection method (see Rousseeuw and van Zomeren (1990) for some other outlier detection techniques), can only be used to indicate suspicious data points. The term outlier refers to the observation having a large t or t^* value (Chatterjee and Hadi, 1986). The fact that a data point qualifies as an outlier under this definition is not however sufficient justification for removing it from the sample. Here it is important to ensure that the population being studied is properly defined. The points flagged as appearing to be different from the majority should never be rejected simply on the basis that they have an adverse effect on the model's fit. Other justification must be found, such as an unusual project, before any removal of data is carried out. The unjustified exclusion of data points has a serious biasing effect on the model developed. For example, often some examples of larger systems in a data set of mainly medium sized systems will stand out as potential outliers, especially under a linear model. Here, by rejecting the larger systems the population is being reduced to medium sized systems. If the model is to be used for such large systems then they cannot be removed. By including these observations, however, the model's accuracy for predicting medium systems may be inadequate. A possible solution here would be to divide the data set into medium and large systems and then develop separate models for each population.

Some of the reasons for the large number of outliers in software development data sets include the lack of agreement on terminology leading to differing definitions for variables, the inconsistencies between data counters, the wide variety of software development processes and the wide range of system sizes (Miyazaki et al., 1994).

5 Neural Networks

The most common model-building technique used in the literature as an alternative to Least Mean Squares regression is backpropagation trained feed-forward neural networks, often referred to simply as backpropagation networks although this is strictly speaking incorrectly confusing the architecture and the training method. Even though a large number of different neural network architectures and training algorithms exist, almost all published studies involving software metric models have been limited to this type. This can be seen as a reflection of the lack of understanding of neural network techniques by many software metric researchers which is understandable given the tremendous growth in the neural network field in the past decade. Gray and MacDonell (1996) provide guidelines for model development of software metrics using various neural network architectures, and Li (1994) provides a good general introduction to neural network applications. Neural networks have been used successfully in many software metric modelling studies, including Wittig (1995) where prediction accuracy was within 10%.

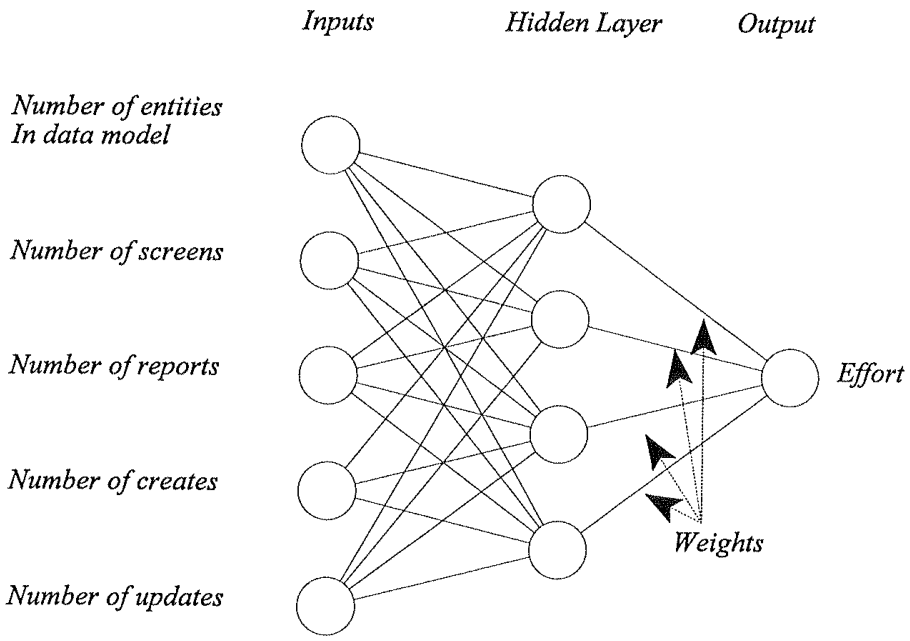

Figure 2

The example in figure 2 demonstrates a neural network model for a metric that predicts the development effort required for a system with a given set of design requirements. The network shown is a feed-forward network that could be trained using the backpropagation algorithm so as to determine weights that attempt to minimise the predictive error. Once the network's weights have been determined new instances can be presented as inputs to have the network make an estimate for the effort required.

Backpropagation trained feed-forward neural networks are developed by first selecting an appropriate architecture of neurons. This includes how many layers of neurons will be used, the number of neurons in each layer, and how the neurons will be connected to each other. Other decisions are also possible regarding the precise nature of neurons, such as their transfer function, and parameters for the training algorithm. Once the architecture has been created the network is trained by presenting it with a series of inputs and the correct output from the training data. As with all empirically-based modelling techniques data should be withheld for verification and validation purposes. The network learns by adjusting its weights to decrease the distance between its predicted output and the actual output. This process of training continues until the network's ability to generalise, as measured by its predictive performance on new data, is optimal.

This means stopping before the network has learned the training data completely and has overtrained, thus losing its important ability to generalise. Usually, various different architectures will be tried and the best tested on the validation data set to ensure good generalisability.

The popularity of this single method has also been noted in other disciplines and can be explained by its *apparent* simplicity and robustness. Often papers using this technique will be based on data sets which are so small or affected with collinearity that normal statistical methods would be unable to produce any valid results. The authors will even on some occasions acknowledge these problems and suggest neural networks as the solution. In fact, it is long acknowledged within the neural network community that neural networks are not immune to statistical problems since they are in many cases equivalent to standard statistical techniques. For example, a feed-forward neural network is equivalent to a multiple nonlinear regression model (Sarle, 1994). In this way they are subject to the same problems as any nonlinear regression, such as too many free parameters for the data set size (low degrees of freedom), collinearity between variables, outliers in the data, and missing data values. Some researchers in the neural network community have even claimed that neural networks are *less* resilient to these problems than their corresponding statistical functions. For a more complete examination of the statistical properties of neural networks see Cheng and Titterington (1994).

The phrase *universal approximator* is often used to describe this form of neural network, suggesting that it can capture any relationship that may exist between the variables. There are several conditions attached to the proof that feed-forward neural networks are capable of representing any *well-behaved* relationship, firstly that the proofs for multilayer perceptrons being universal approximators only prove that there exists some two layer (the number of layers refers to the number of connections, so a two layer network really has an input, hidden and an output *slab* of neurons) network that can approximate any well-behaved function to an arbitrary degree of accuracy (Hornik et al., 1989). The proofs do not specify the number of neurons required in the hidden layer. Since training time and the minimum data set size for valid learning increases with the number of neurons, the theoretical existence of a network capable of capturing a relationship is of little use if it can not be implemented.

It has been further shown that such networks are not just capable of representing such mappings, but these mappings are always learnable (White, 1990). The problem here is that even if the architecture provides a sufficient number of hidden layer neurons, the backpropagation training algorithm, which is a simple gradient descent algorithm, is notoriously unreliable at finding the globally optimal set of weightings and often falls into local minima which may not provide an accurate mapping between inputs and outputs (Sutton, 1989). Better learning algorithms that do

not have such problems with local minima and often train much faster have been developed (for example, Baldi and Hornik, 1989; Baba, 1989). As an alternative, schemes are available that can initialise the network weights so as to avoid the false minima (Wessels and Barnard, 1992; Denoeux and Lengellé, 1993; Weymaere and Martens, 1994). These can be used in place of the normal method of initialising the network with small random values. Modern neural network literature abounds with better training algorithms and initialisation techniques but still the vast majority of applications continue to use the slow and unreliable backpropagation method with the small random value initialisation rule. This can be seen as a reflection of the slow dissemination of information in comprehensible formats to those applying neural networks rather than investigating their properties.

Other architectures that are suitable for neural networks include cascade correlation networks, Kohonon networks, and radial basis function networks. The only one of these that the authors are aware of having been used for software metrics is cascade correlation (Wittig, 1995).

A failing of neural networks is that they operate as 'black boxes' and provide the user with no information about how outputs are reached (Diederich, 1990). As stated by Davis et al. (1977), the ability to generate explanations is important in order to gain user acceptance of artificial intelligence techniques. This can make it difficult to test a neural network's output gradient vectors with respect to the various inputs to ensure that the relationships are sensible (in other words increasing or decreasing as appropriate and with a suitable relative magnitude). This is a simple matter with regression equations where the signs and relative magnitudes of the coefficients can be easily checked to ensure that the predicted output will vary in the correct way with respect to the inputs.

While certain types of neural networks are prone to the phenomenon of *catastrophic forgetting* (Robins, 1995), where training the neural network on new data causes the network to forget previously learned data, this can be overcome by repeating the training with a new unified data set. Given the small data sets that are often used for software metrics and the non-real time nature of the models it is recommended that using the combined data set to redevelop the neural network should not present problems in terms of training time.

6 Fuzzy Systems

Fuzzy systems have only been used in a few publications for software development models which is surprising given their rapid adoption into other areas. A fuzzy system is a mapping between linguistic terms, such as "very small", attached to variables. Munakata and Jani (1994) provide a good introduction to fuzzy systems. Thus an input into a fuzzy system can be either

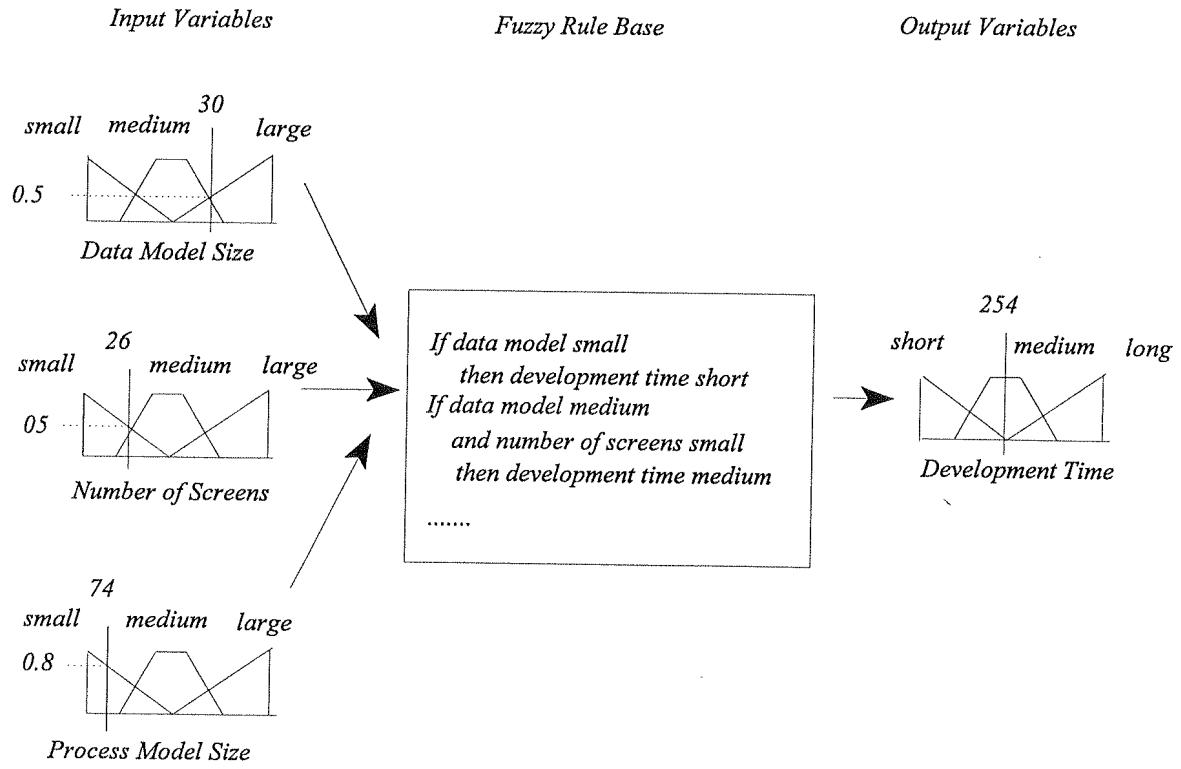

Figure 3

numerical or linguistic, with the same applying to the output. A number of different types of fuzzy systems have been shown to act as universal approximators in the same sense as neural networks (Kosko, 1994; Wang and Mendel, 1992; Castro, 1995).

A fuzzy system as considered here, although as noted above there are different types, is made up of three main components. The first, the membership functions, represent how much a given numerical value for a particular variable fits the term being considered. The second component is the rule base. This performs the mapping between the input membership functions and the output membership functions. The greater the input membership degree, the stronger the rule fires, and thus the stronger the pull towards the output membership function. Since several different output memberships could be contained in the consequents of rules fired, a defuzzification process, the third component, is carried out to combine the outputs into a single label or numerical value as required.

This approach is demonstrated in figure 3. In this simple example numerical inputs are provided for the data model size (30), number of screens (26), and process model size (74). These numerical values are plotted on the membership functions, with the height of intersection with the membership curve indicating the degree to which the value belongs to the respective label.

For the particular type of fuzzy logic system described here this step is called fuzzification. In this case the data model size is medium to a degree of 0.5 and large to a degree of 0.5, while the process model size is small to a degree of 0.8. The membership degree determines how much weight to give to the rules involving the membership label in its antecedent. There are various methods for weighting the rules in this case. The consequents of each rule are then combined, for the type of fuzzy system described here this process is called defuzzification, and a single output value is determined, in this case 254.

The most obvious strength of fuzzy systems is that by using linguistic mappings a highly intuitive model can be created that anyone, even without any training, can understand and if necessary criticise. As with neural networks there are a large number of different types of fuzzy system, and again when developing a model it is necessary to understand the various choices available. Many different schemes have been devised to extract fuzzy membership functions and rules directly from data including that described by Wang and Mendel (1992). They suggest that this then allows for an expert to fine-tune and add to the resulting system rather than starting from scratch.

7 Hybrid Neuro-Fuzzy Systems

Recently researchers have attempted to combine the strengths of neural networks and fuzzy systems while avoiding most of the disadvantages of each (Jang, 1993; Horikawa et al., 1992). This has resulted in a wide range of possibilities for hybridizing the two techniques. While all of these techniques are different in some way, they share the same basic principles: an adaptive system that can deal with easily comprehended linguistic rules and that permits initialisation of the network based on available knowledge.

The standard neuro-fuzzy hybrid system is based on inputs into the network being transformed into membership degrees that can then activate rules, leading to membership degrees for the outputs that can be defuzzified. Thus the five layers of neurons (technically slabs, since layers refers to connections) in such a network represent the crisp inputs, input fuzzy membership degrees, rule firing, output memberships, and crisp outputs. One problem with a fully trainable neuro-fuzzy system is that the membership functions can drift such that they no longer represent their linguistic label. One approach to avoiding this problem is the separation of the rule learning and membership extraction operations as discussed in Gray and MacDonell (1996).

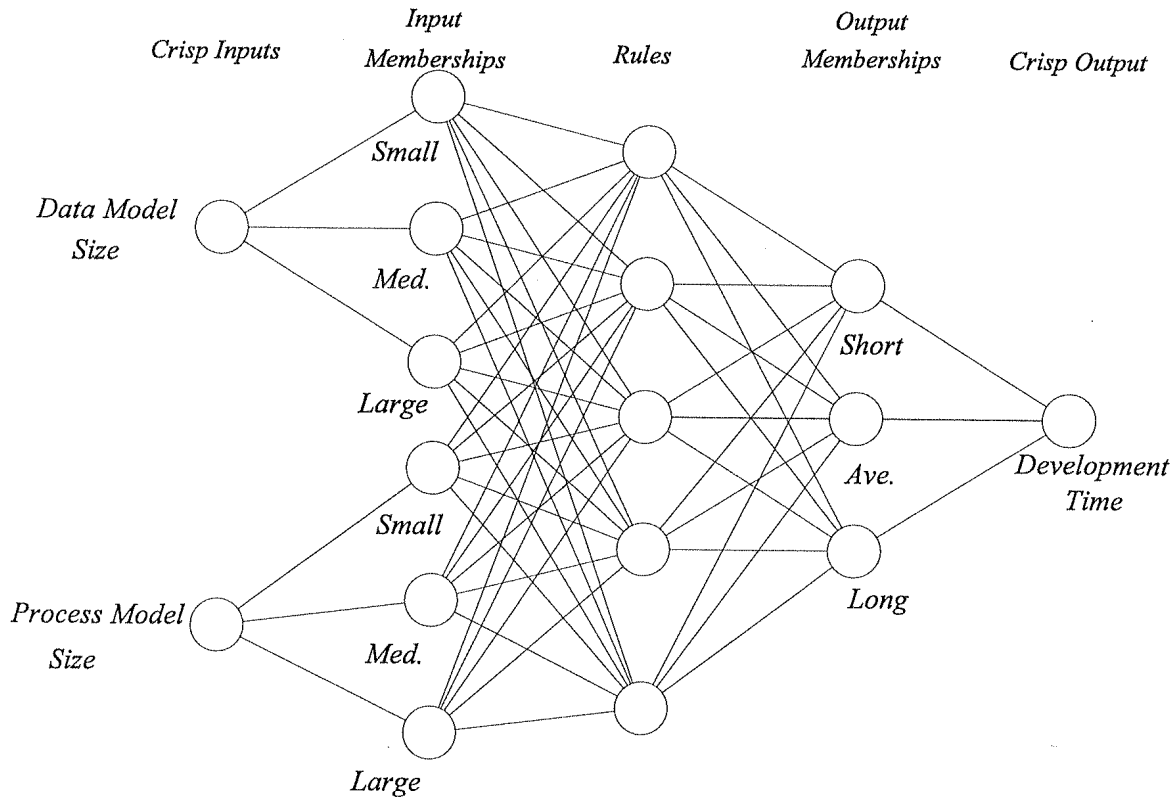

Figure 4

An example of a neuro-fuzzy system is presented in figure 4. Here two inputs (data model size and process model size) are presented to the network's input neurons which are then fed through the first layer of connections. The outputs to the second layer of connections represent the membership degrees, leading to the rules where positive input weights represent affirmative rules. The outputs from the rule neurons represent the degree to which the rule has been fired and determine the activation of the output membership neurons. The results from these output neurons are combined into a single numerical value.

Once a network has been trained it is possible to extract the rules contained within it which can then be checked for acceptability, and if desired used in a standard fuzzy system. The ability to extract rules in this manner can be used to check for catastrophic forgetting, where the network *learns* new relationships from new data but *forgets* the old relationships from old data (Robins, 1995).

Ironically, although neural networks and fuzzy systems are both universal approximators, standard neuro-fuzzy systems are not. Despite this, neuro-fuzzy systems are capable of

approximating well enough and alterations to the standard architecture are possible to ensure universal approximation should the model require this (Buckley and Hayashi, 1994). However it seems unlikely that any metric would be unable to be modelled using a neuro-fuzzy system since this inability seems limited to unusual functions.

8 Rule-Based Systems

Rule-based systems have been used in very few cases for modelling software development. Fuzzy rule systems are a superset of crisp rule systems and any such system can be simulated by a fuzzy system. For this reason it may be considered that crisp systems are redundant. However, the greater simplicity of a crisp-rule base can be seen as an attractive feature, especially where many input variables are involved.

A rule-based system is organised around a set of rules that are activated by facts being present in the working memory, and that activate other facts, as shown in figure 5. In this way chaining can occur with one rule enabling another rule to fire. This would, for example, allow for rules to be developed to recognise a *high error* module along the lines of:

IF module length > 40 LOC or
    IF module length>20 LOC AND development time > 2 hours
    THEN module is high error risk

Such a system has the disadvantage, compared to a fuzzy system, that all antecedents and consequents must be either true or false, with no degrees of true or false allowed. This can cause problems when a module with 21 LOC and a module with 20 LOC, both taking four development hours, are put through the above rule. The two modules are very similar but only the first will fire the rule.

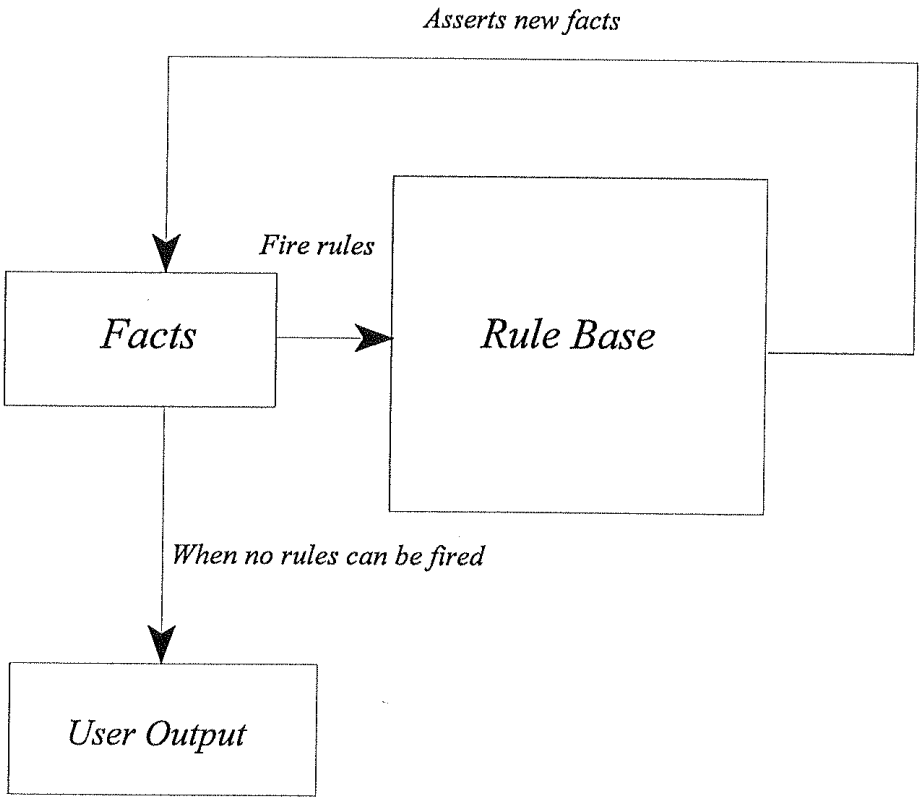

Figure 5

9 Case-Based Reasoning

Case-based reasoning is a method of storing observations, such as data about a project's specifications and the effort required to implement it, and then when faced with a new observation retrieving those stored observations closest to the new observation and using the stored values to estimate the new value, in this case effort. Thus a case-based reasoning system has a pre-processor to prepare the input data, a similarity function to retrieve the similar cases, a predictor to estimate the output value, and a memory updater to add the new case to the case base if required (Aha, 1991). This is shown in figure 6. Case-based reasoning systems are intended to mimic the process of an expert making a decision based on their previous experience (Mukhopadhyay et al., 1992). It was found by Vicinanza et al. (1991) that experience assisted with software development estimates and that experts at this used comparisons with past cases.

Problems have been encountered with some case-based reasoning systems. As stated by Breiman et al. (1993) they are intolerant of noise and irrelevant features. The authors also claim that the similarity function used has a strong influence on the algorithm's performance. This makes the creation of a case-based reasoning system a non-trivial task. However, extensions to standard case-based reasoning algorithms performed by Aha (1991) resulted in much more noise-tolerant systems.

An experiment by Mukhopadhyay et al. (1992) compared the performance of a case-based reasoning system, a human expert, and standard models using function points and COCOMO. The case-based reasoning system (ESTOR) and the expert were limited to using the standard inputs to the function point and COCOMO models. The performance of the case-based reasoning system exceeded that of the function point and COCOMO models, and was close to the level of the expert. The authors concluded that the case-based reasoning approach was worth further study due to its encouraging results.

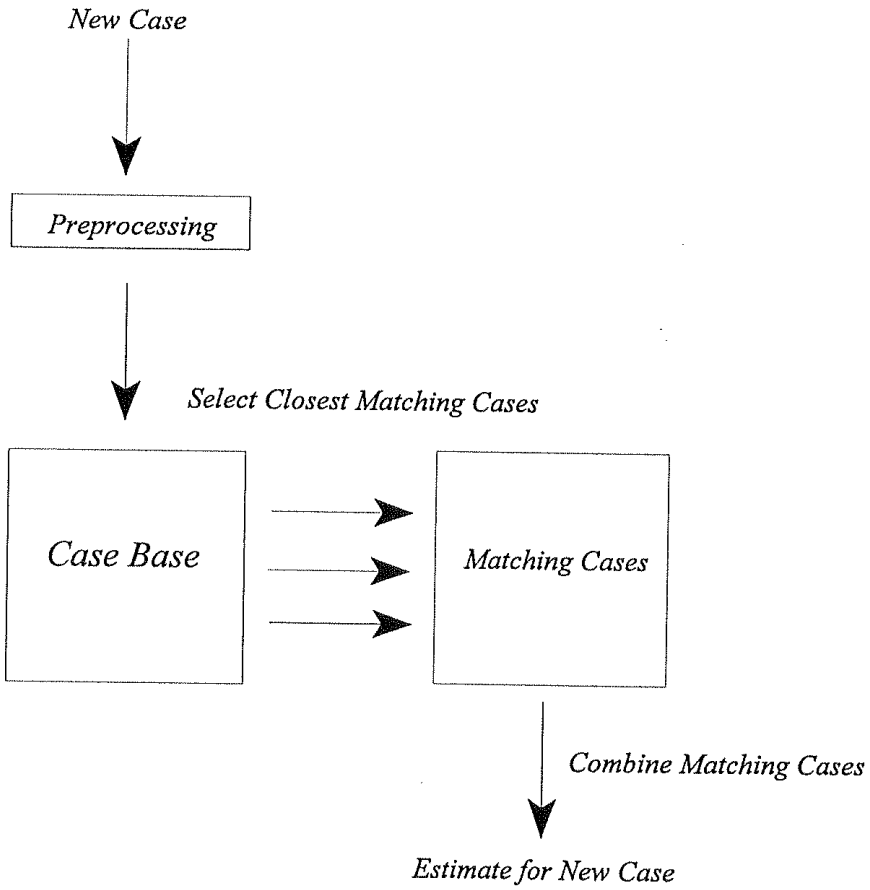

Figure 6

10 Regression and Classification Trees

Regression and classification trees, while based on the same principle, each have a different aim. Regression trees can be used when the output value to be predicted is from the interval domain, while classification trees (also known as decision trees) are used to predict the output class for an observation, that is to say, from the nominal or ordinal data scale. Both algorithms work by taking a known data set and learning the rules needed to classify it. For an overview of the techniques see Breiman et al. (1993).

As an illustration, refer to figure 7 where a regression tree algorithm has been used to extract the important rules that can be used to predict testing time for software developed using particular tools. The mean testing time for each class has been recorded as the leaf nodes. In order to create the tree the algorithm looks at which attributes can be used to best classify the data and iteratively constructs the tree, splitting nodes when required. While splits of two are most common, algorithms exist for splitting ranges into greater numbers of partitions. This method has the advantage of being easily comprehended and checked for logical errors.

Classification trees operate in much the same way as regression trees except that instead of interval scale data being attached to the leaf nodes, labels are used instead. Thus a classification tree could classify modules into various risk categories, for example 'high risk' and 'low risk.'

An experiment by Srinivasan and Fisher (1995) found that, using mean residual error as the performance measure, a regression tree approach was more successful than COCOMO or SLIM for estimating effort, although less successful than a backpropagation trained neural network (the most successful) and function points. An extension to the basic regression tree algorithm discussed by Srinivasan and Fisher was to replace the mean values at the leaf nodes with regression equations, allowing for a piece-wise regression equation over the domain. This could be a successful technique given the way the behaviour of metrics changes in relation to the scale of the project under consideration.

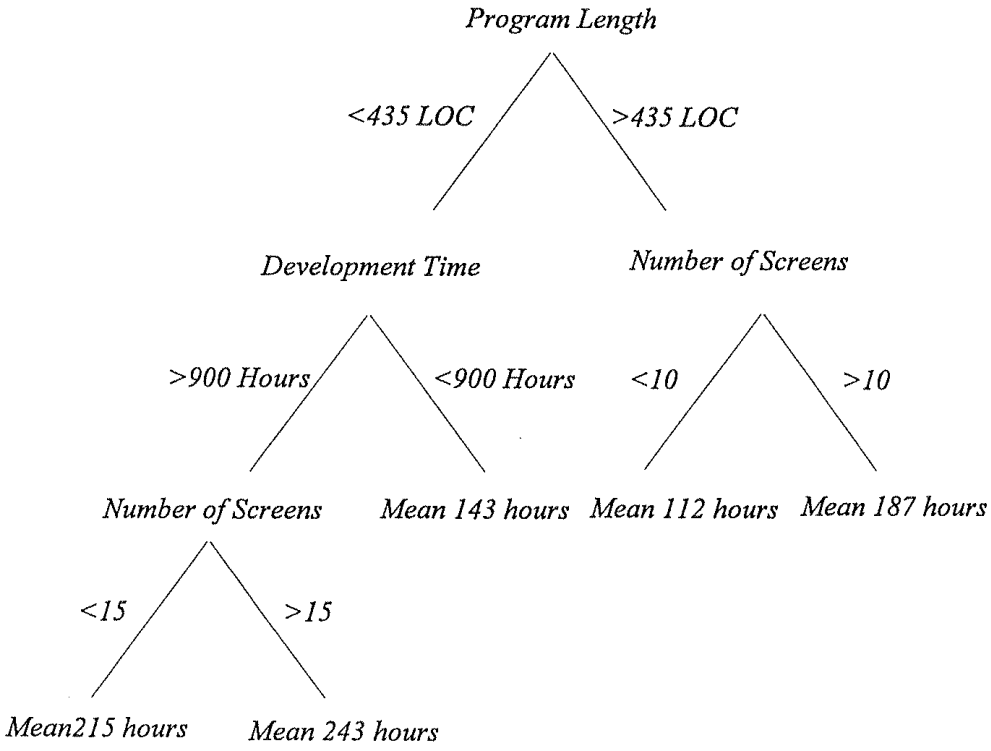

Figure 7

11 Comparison of Techniques

Table 1 shows a comparison between the techniques with respect to some desirable modelling technique attributes. It can be seen that not all techniques are suited to all types of problems. In addition, not all factors that may influence technique selection are listed here. Others may include available model-building software, expertise in each field, and the time available for development.

In the table the heading 'model free' refers to the ability of the modelling technique to determine its own structure, rather than relying on the developer to provide the form of the relationship between inputs and outputs. As an example, when developing a regression model it is necessary to specify which variables should be transformed and what type of transformation should be used, for example logarithmic. With a neural network, an appropriate approximate transformation will be found by the network when training.

Technique	Model Free	Can Resist Outliers	Explains Output	Suits Small Data Sets	Can be Adjusted For New Data	Black/ Grey/ White Box	Suit Complex Models	Include Known Facts
Least Squares Regression	No	No	Yes/No	No	No	White	No	No/ Yes
Robust Regression	No	Yes	Yes/No	No	No	White	No	No/ Yes
Neural Networks	Yes	No	No	No	Yes/ No	Black	Yes	No/ Yes
Fuzzy Systems (Adaptive)	Yes	Yes	Yes	Yes	No/Yes	White	Yes	Yes
Hybrid Neuro-Fuzzy Systems	Yes	Yes/ No	Yes	Yes/No	Yes/ No	Grey	Yes	Yes
Rule Based Systems	No	N/A	Yes	N/A	N/A	White	Yes	Yes
Case-Based Reasoning	Yes	Yes/No	Yes	No/Yes	Yes	Grey	Yes	No
Regression Trees	Yes	Yes	Yes	No/Yes	Yes	Grey	Yes	Yes
Classification or Decision Tress	Yes	Yes	Yes	No/Yes	Yes	Grey	Yes	Yes

Table 1

The next entry refers to the model's robustness of estimation when faced with a model containing outliers. Some techniques are capable of providing some explanation for their *reasoning* and this is noted in the next column. Small data sets are problematic for all modelling techniques, however by using expert knowledge as a supplement to the data (as in fuzzy systems) an accurate model can still be derived. Once a model has been developed, the issue of whether additional data can be added or whether the entire model must be regenerated on the combined data set must be faced. Related to the explanation of a model is the ability of a user to *see* how a model arrived at its conclusions. This can be important for the purpose of verification. Models can be black box (outputs are derived from inputs via a hidden process), white box (the process is visible and can be understood), or grey box (partially visible). The suitability of a technique to complex models is related to the issue of model free estimation and the ability to add expert knowledge. Finally the table covers the technique's ability to include known information into the model, that is to initialise the model and then use data to improve and refine it.

Some entries in the table are both yes and no. Since some techniques neither completely fail to deal with a particular issue, or succeed entirely, entries have been made as yes/no if the technique is fairly successful at dealing with the issue and no/yes for cases when the technique is only slightly successful.

12 Conclusions

By considering a wide range of modelling techniques that may be suitable for developing software metric models a project manager can be more confident that the best (for practical purpose this will normally be the most accurate) model possible has been developed. Even after the model has been developed it is important to keep in mind the inherent limitations of the technique used.

Some of the artificial intelligence techniques, especially neural networks, fuzzy models, and case-based reasoning, seem to be especially well suited to the model building problem. It is hoped that continuing research in applying these methods to metrics will improve the quality of predictions.

References

Aha, D.W., Case-Based Learning Algorithms, in *Proceedings of the DARPA Case-Based Reasoning Workshop*, Morgan Kaufmann, Washington, D.C., 147-158 (1991).

Allen, D.M., *The Prediction Sum of Squares as a Criterion for Selecting Predictor Variables*, Technical Report No. 23, Dept. Statistics, University of Kentucky, 1971.

Baba, N., A New Approach for Finding the Global Minimum of Error Function in Neural Networks, *Neural Networks* 2, 367-373 (1989).

Baldi, P, and Hornik, K., Neural Networks and Principal Component Analysis: Learning from Examples without Local Minima, *Neural Networks* 2, 53-58 (1989).

Bastani, F.B., DiMarco, G., and Pasquini, A., Experimental Evaluation of a Fuzzy-Set Based measure of Software Correctness Using Program Mutation, in *Proc. 15th Int. Conf. Software Engineering*, 45-54 (1993).

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

Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, C.J., *Classification and Regression Trees*, Chapman & Hall, New York, 1993.

Briand, L.C., Basili, V.R., and Thomas, W.M., A Pattern Recognition Approach for Software Data Analysis, *IEEE Trans. Soft. Eng.* 18, 931-942 (1992).

Buckley, J.J., and Hayashi, Y., Can Fuzzy Neural Nets Approximate Continuous Fuzzy Functions, *Fuzzy Sets and Systems* 61, 43-51 (1994).

Castro, J.L., Fuzzy Logic Controllers are Universal Approximators, *IEEE Trans. Systems, Man, and Cybernetics* 25, 629-635 (1995).

Chatterjee, S.C., and Hadi, A.S., Influential Observations, High Leverage Points, and Outliers in Linear Regression, *Statistical Science* 1, 379-416 (1986).

Cheng, B., and Titterington, D.M., Neural Networks: A Review from a Statistical Perspective, *Statistical Science* 9, 2-54 (1994).

Coupal, D., and Robillard, P.N., Factor Analysis of Source Code Metrics, *J. Systems Software* 12, 263-269 (1990).

Courtney, R.E., and Gustafson, D.A., Shotgun Correlations in Software Measures, *Software Eng. J.* 8, 5-13 (1993).

Davis, R., Buchanan, B.G., and Shortliffe, E., Production Rules as a Representation for a Knowledge-Based Consultation Program, *Artificial Intelligence* 8, 15-45 (1977).

Denoeux, T., and Lengellé, R., Initializing Back Propagation Networks with Prototypes, *Neural Networks* 6, 351-363 (1993).

Diederich, J., An Explanation Component for a Connectionist Inference System, in *Proc. 9th European Conf. Artificial Intelligence*, 222-227 (1990).

Efron, B., Bootstrap Methods: Another Look at the Jack-knife, *Annals of Statistics* 7, 1-26 (1979).

Gray, A.R., and MacDonell, S.G., Guidelines for the Development of Neural Network-Based Software Metric Models, in preparation.

Griech, B., and Pomerol, J.-CH., Design and Implementation of a Knowledge-Based Decision Support System for Estimating Software Development Work-Effort, *J. Systems Integration* 4, 171-184 (1994).

Hakkarainen, J., Laamanen, P., and Rask, R., Neural Networks in Specification Level Software Size Estimation, in *Proc. 26th Hawaii Int. Conf. System Sciences*, 626-634 (1993).

Hettmansperger, T.P., and Sheather, S.J., A Cautionary Note on the Method of Least Median Squares, *The American Statistician* 46, 79-83 (1992)

Horikawa, S., Furnuhashi, T., and Ucikawa, Y., On Fuzzy Modelling Using Fuzzy Neural Networks with the Back-Propagation Algorithm, *IEEE Trans. Neural Networks* 3, 801-806 (1992).

Hornik, K., Stinchcombe, M., and White, H., Multilayer Feedforward Networks are Universal Approximators, *Neural Networks* 2, 359-366 (1989).

Jang, R.J.-S., ANFIS: Adaptive-Network-Based Fuzzy Inference System, *IEEE Trans. Systems, Man, and Cybernetics* 23, 665-685 (1993).

Jeffery, D.R., and Low, G., Calibrating Estimation Tools for Software Development, *Soft. Eng. J.* 5, 215-221 (1990).

Karunanithi, N., Whitley, D., and Malaiya, Y.K., Prediction of Software Reliability Using Connectionist Models, *IEEE Trans. Soft. Eng.* 18, 563-574 (1992).

Kemerer, C.F., An Empirical Validation of Software Cost Estimation Models, *Comm. ACM* 30, 416-429 (1987)

Khoshgoftaar, T.M., and Lanning, D.L., A Neural Network Approach for Early Detection of Program Modules Having High Risk in the Maintenance Phase, *J. Systems Software* 29, 85-91 (1995).

Kosko, B., Fuzzy Systems as Universal Approximators, *IEEE Trans. Computers* 43, 1329-1333 (1994).

Kumar, S., Krishna, B.A., and Satsangi, P.S., Fuzzy Systems and Neural Networks in Software Engineering Project Management, *J. Applied Intelligence* 4, 31-52 (1994).

Lakhotia, A., Rule-Based Approach to Computing Module Cohesion, in *Proc. 15th Int. Conf. Software Engineering*, 35-44 (1993).

Li, E.Y., Applications Artificial Neural Networks and their Business Applications, *Information & Management* 27, 303-313 (1994).

Lovell, M.C., Data Mining, *The Review of Economics and Statistics* LXV, 1-12 (1983).

MacDonell, S.G., and Gray, A.R., A Review of Model Building Techniques found in Software Metrics Literature, in preparation

MacDonell, S.G., *Quantitative Functional Complexity Analysis of Commercial Software Systems*. Unpublished PhD Thesis, University of Cambridge, Cambridge, United Kingdom, 1993.

Massart, D.L., Kaufman, L., Rousseeuw, P.J., and Leroy, A., Least Median of Squares: A Robust Method for Outlier and Model Error Detection in Regression and Calibration, *Analytica Chimica Acta* 187, 171-179 (1986).

Mata-Toledo, R.A., and Gustafson, D.A., A Factor Analysis of Software Complexity Measures, *J. Systems Software* 17, 267-273 (1992).

Miyazaki, Y., Terakado, M., and Ozaki, K., Robust Regression for Developing Software Estimation Models, *J. Systems 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, 155-171 (1992).

Munakata, T., and Jani, Y., Fuzzy Systems: An Overview, *Comm. ACM* 37, 69-76 (1994).

Picard, R.R., and Berk, K.N., Data Splitting, *The American Statistician* 44, 140-147 (1990).

Porter, A.A., and Selby, R.W., Evaluating Techniques for Generating Metric-Based Classification Trees, *J. Systems Software* 12, 209-218 (1990).

Putnam, L.H., A General Empirical Solution to the Macro Software Sizing and Estimating Problem, *IEEE Trans. Soft. Eng*. 4, 345-361 (1978).

Ramsey, C.L., and Basili, V.R., An Evaluation of Expert Systems for Software Engineering Management, *IEEE Trans. Soft. Eng.* 15, 747-759 (1989).

Robins, A., Catastrophic Forgetting, Rehearsal and Pseudorehearsal, *Connection Science* 7, 123-146 (1995).

Rousseeuw, P.J., and Leroy, A.M., *Robust Regression and Outlier Detection*, John Wiley & Sons, New York, 1987.

Rousseeuw, P.J., and van Zomeren, B.C., Unmasking Multivariate Outliers and Leverage Points, *J. American Statistical Association* 85, 633-639 (1990).

Rousseeuw, P.J. Least Median of Squares Regression, *J. American Statistical Associatio*n 79, 871-880 (1984).

Sarle, W.S., Neural Networks and Statistical Models, in *Proc. 19th Annual SAS Users Group Int. Conf*., 1538-1550 (1994).

Selby, R.W., and Porter, A.A., Learning from Examples: Generation and Evaluation of Decision Trees for Software Resource Analysis, *IEEE Trans. Soft. Eng.* 14, 1743-1757 (1988).

Sheppard, J.W., and Simpson, W.R., Using a Competitive Learning Neural Network to Evaluate Software Complexity, in *Proc. 1990 ACM SIGSMALL/PC Symp. Small Systems*, 262-267 (1990).

Sherer, S.A., Software Fault Prediction, *J. Systems Software* 29, 97-105 (1995).

Snee, R.D., Validation of Regression Models: Methods and Examples, *Technometrics* 19, 415-428 (1977).

Srinivasan, K., and Fisher, D., Machine Learning Approaches to Estimating Software Development Effort, *IEEE Trans. Soft. Eng,* 21, 126-137 (1995).

Stewart, G.W., Collinearity and Least Squares Regression, *Statistical Science* 2, 68-100 (1987).

Subramanian, G.H., and Breslawski, S., Dimensionality Reduction in Software Development Effort Estimation, *J. Systems Software* 21, 187-196 (1993).

Sutton, R.S., Two Problems with Backpropagation and Other Steepest-Descent Learning Procedures for Networks, in *Proc. Eight Annual Conf. Cognitive Science Society*, 823-831 (1989).

Vicinanza, S., Mukhopadhyay, T., and Prietula, M., Software Effort Estimation: An Exploratory Study of Expert Performance, *Information Systems Research* 2, 243-262 (1991).
Wang, L.-X., and Mendel, J.M., Generating Fuzzy Rules by Learning From Examples, *IEEE Trans. Systems, Man, and Cybernetics* 22, 1414-1427 (1992).

Wessels, L.F.A., and Barnard, E., Avoiding False Local Minima by Proper Initialization of Connections, *IEEE Trans. Neural Networks* 3, 899-905 (1992).

Weymaere, N., and Martens, J.-P., On the Initialisation and Optimisation of Multilayer Perceptrons, *IEEE Trans. Neural Networks*, 738-751 (1994).

White, H., Connectionist Nonparametric Regression: Multilayer Feedforward networks can Learn Arbitrary Mappings, *Neural Networks* 3, 535-549 (1990).

Wittig, G., *Estimating Software Development Effort with Connectionist Models*, Working Paper Series 33/95, Monash University 1995.

Wittig, G.E., and Finnie, G.R., Using Artificial Neural Networks and Function Points to Estimate 4GL Software Development Effort, *Australian Journal of Information Systems* 1(2), 87-94 (1994).

Young, G.A., Bootstrap: More than a Stab in the Dark?, *Statistical Science* 9, 382-415 (1994).