Full citation: Gray, A.R., & MacDonell, S.G. (1997) A comparison of techniques for developing predictive models of software metrics, Information and Software Technology 39(6), pp.425-437.
doi: 10.1016/S0950-5849(96)00006-7

A Comparison of Techniques for Developing Predictive Models of Software Metrics

Andrew Gray and Stephen G. MacDonell

Software Metrics Research Lab
Department of Information Science
University of Otago, PO Box 56
Dunedin, New Zealand
agray@commerce.otago.ac.nz, stevemac@commerce.otago.ac.nz

Abstract

The use of regression analysis to derive predictive equations for software metrics has recently been complemented by increasing numbers of studies using non-traditional methods, such as neural networks, fuzzy logic models, case-based reasoning 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, and more effective validation procedures. This paper examines the implications of using these methods and provides some recommendations as to when they may be appropriate. A comparison of the various techniques is also made in terms of their modelling capabilities with specific reference to software metrics.

1. INTRODUCTION

A significant proportion of 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 Putnam's equations[1] and COCOMO[2] as typical examples. 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. As Briand et al.[3] 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 a particular model development task.

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 an acceptance of projects that do not provide sufficient returns or that overrun budgets and schedules. 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 any organisation 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. Outside of accuracy considerations, issues such as ease of use and interpretability can impact on the usefulness of particular modelling techniques.

1.1. Difficulties in Analysis and Modelling

Three broad areas of concern can be cited with regard to software development predictive models, with the first set of difficulties perhaps the most serious. Software engineering data sets often have a number of characteristics 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 size. 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 modelling techniques may be unsuitable for a particular data set affected by these problems.

A second area of concern is the acceptability and validation of 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. 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. With small data sets it is possible for a model to be developed that fits the data, but that violates common sense - for example, slopes may be counter-intuitive to those 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 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[4,5]. Kemerer[5], for example, 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 discussed here contributes in at least one way to resolving some of these problems in a manner that can be seen 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. An awareness of other approaches should help to ensure that the most appropriate model is developed through employing the most suitable of the various alternatives. In some cases the combination of methods may be useful, each providing estimates that can then be combined in some fashion.

The remainder of the paper continues with a discussion of each modelling technique in turn. An example is provided for each method (apart 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.

2. LEAST SQUARES REGRESSION

Linear least squares regression analysis is still the most common technique used, as observed in the literature[6]. 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, each exercising the same extent of influence on the regression equation. In data sets that contain outlier observations, this equivalent influence can have a marked (and undesirable) effect on the form of the derived regression line (see the next section for further details).

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 (in the statistical sense, for example there are no outliers or significant heteroscedasticity),
- a small number of independent variables are sufficient, after transformations if necessary, to linearly predict the possibly transformed output variable(s) so as to enable an interpretable representation, 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[3]. 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[6].

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*[7] or more generally, *data mining*[8]. Courtney and Gustafson[7] 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[9], and Snee[10] 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 data sets that have not been withheld for validation, while common practice[6], 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[9]. While the small size of data sets often reported in the literature prevents data splitting, other techniques are available, although not widely used[6]. These alternatives to data splitting are the PRESS statistic[11] and resampling-based methods such as the bootstrap[12,13].

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[14-16]. 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[17] for a discussion of methods of detecting collinearity in regression-based models.

3. ROBUST REGRESSION ANALYSIS

Robust regression analysis has been used in MacDonell[18] and Miyazaki et al.[19] 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[20]) 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[20] provides estimates that cannot be affected to an arbitrary degree by up to 50 percent contamination (i.e. data values that do not reflect the underlying system being modelled for reasons that may include measurement error and an unusual system); 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[21].

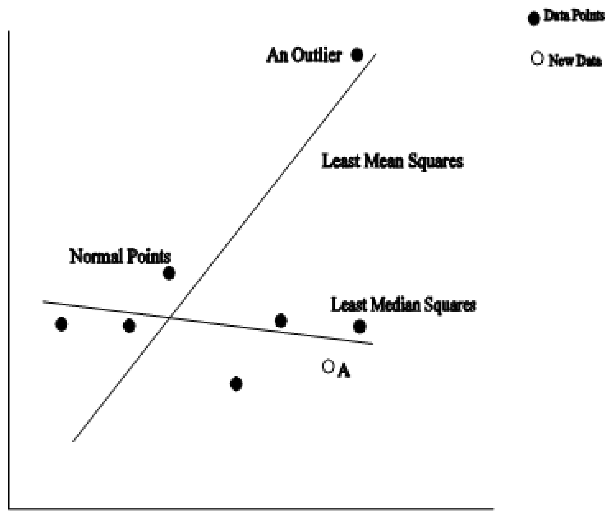

Figure 1. The influence of outlier data points on regression

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[22]. Some of the weaknesses of the method are discussed in Hettmansperger and Sheather[23].

An important point to remember here is that robust regression, as with any outlier detection method (see Rousseeuw and van Zomeren[24] 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[25] where a large value would indicate that the observation is a large number of standard deviations from the mean. This indicates that, for an approximately normal distribution, the observation is unlikely to have occurred simply by chance. 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[19].

4. NEURAL NETWORKS

The most common model-building technique used in the literature as an alternative to least mean squares regression is back-propagation trained feed-forward neural networks, often referred to simply as back-propagation 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[26-35]. 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[36] provide guidelines for model development of software metrics using various neural network architectures, and Li[37] provides a good general introduction to neural network applications. Neural networks have been used successfully in many software metric modelling studies, including Wittig[29] where prediction accuracy was within 10% (although the use of accuracy, while suggested here as the most important requirement for most project managers, as the sole determinant of a model's goodness is not recommended, with other factors such as ease of use and interpretability also being important).

The example depicted 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 back-propagation 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.

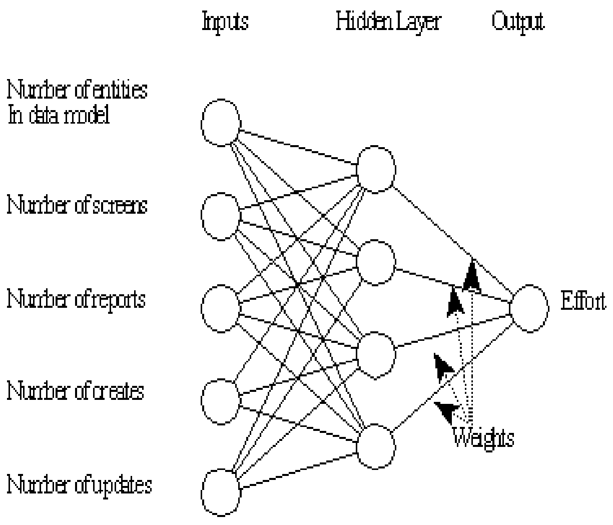

Figure 2. An example neural network structure for effort prediction

Back-propagation 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 that 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[35]. 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[38].

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 (here well-behaved is used in the mathematical sense for describing a function), 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[39]. 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 cannot be implemented.

It has been further shown that such networks are not just capable of representing such mappings, but these mappings are always learnable[40]. The problem here is that even if the architecture provides a sufficient number of hidden layer neurons, the back-propagation 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[41]. Better learning algorithms that do not have such problems with local minima and often train much faster have been developed (see, for example, Baba[42] and Baldi and Hornik[43]). As an alternative, schemes are available that can initialise the network weights so as to avoid the false minima[44-46]. 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 relatively slow and unreliable back-propagation 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, Kohonen 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[29].

A failing of neural networks is that they operate as ‘black boxes’ and provide the user with no information about how outputs are reached[47]. As stated by Davis et al.[48], the ability to generate explanations is important in order to gain user acceptance of artificial intelligence techniques. In fact the importance of explanation extends to all techniques discussed here. In terms of the ‘black box’ nature of a neural network, this can make it difficult to test the 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 less problematic with regression equations where the signs and relative magnitudes of the coefficients can be checked to ensure that the predicted output will vary in the correct way with respect to the inputs. Another problem with neural networks is catastrophic forgetting[49] where training on new data causes the network to lose existing knowledge, although given the relatively small sizes of software metric data sets this is unlikely to be problematic.

5. FUZZY SYSTEMS

Fuzzy systems have only been used in a few publications for software development models[30,50] 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[51] provide a good introduction to fuzzy systems. Thus an input into a fuzzy system can be either 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[52-54].

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 which can be obtained from experts’ understandings of the relationships being modelled and refined (or even obtained in the first case) using various data-driven adaptation techniques. 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 (note that this defuzzification is based on several other rules not shown).

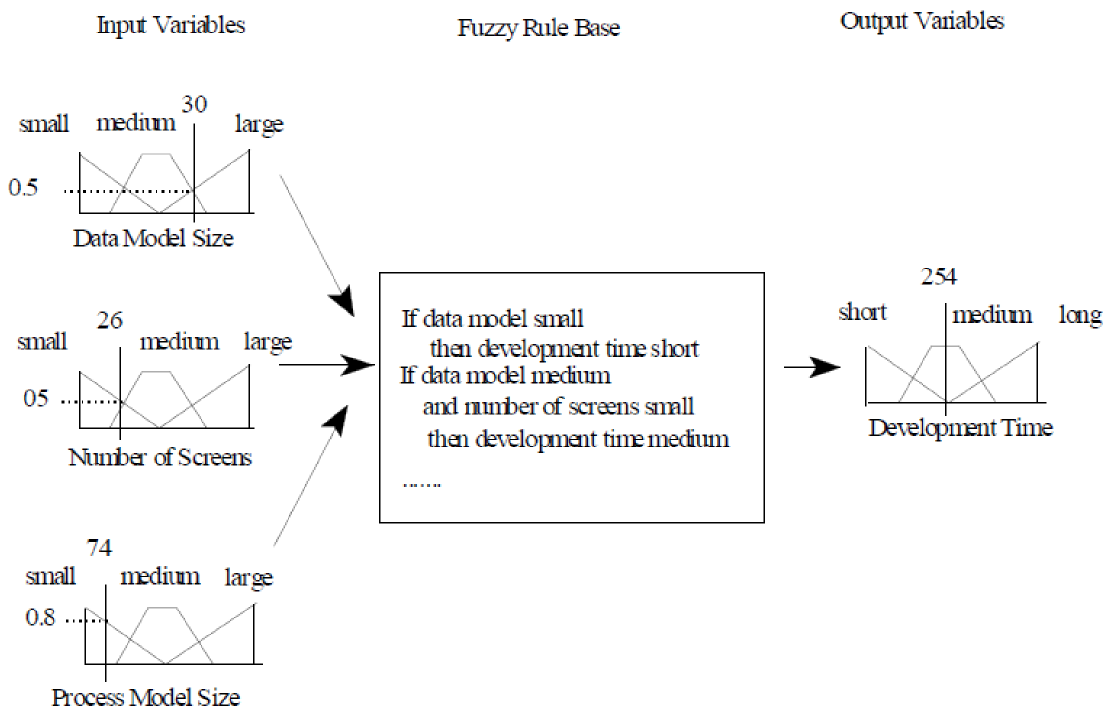

Figure 3. A fuzzy system for duration estimation

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. On the negative side fuzzy systems suffer from some limitations, including the difficulty of specifying a system with very high accuracy while maintaining a degree of meaningfulness (generally more accuracy requires more rules, with greater numbers of rules leading to more complex and less interpretable systems). 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[53]. They suggest that this then allows for an expert to fine-tune and add to the resulting system rather than starting from scratch.

6. 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[55,56]. 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 (with four layers of 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[36].

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.

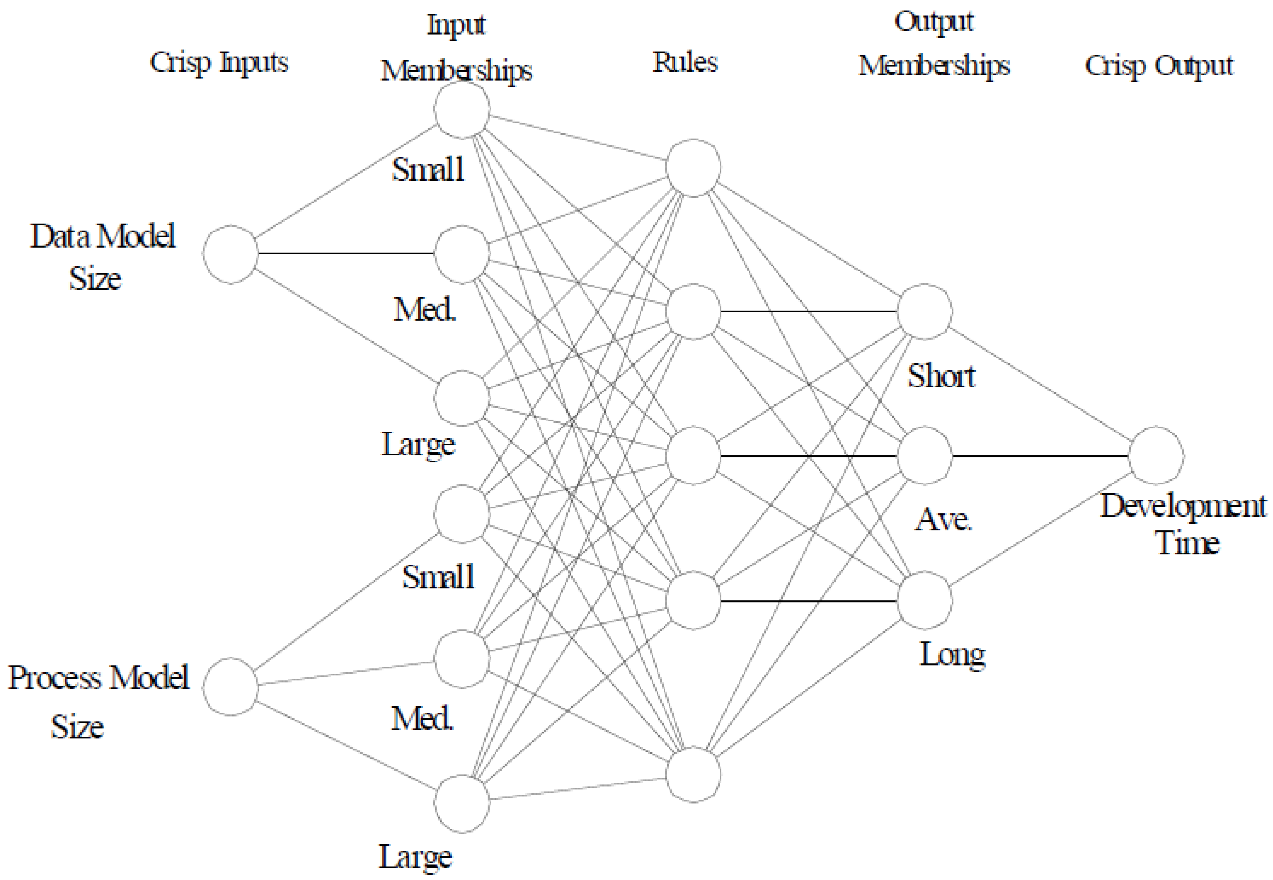

Figure 4. Neuro-fuzzy hybrid architecture for development time prediction

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[49].

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[57]. However it seems unlikely that any metric would involve sufficiently pathological relationships for this to be a problem.

7. RULE-BASED SYSTEMS

Rule-based systems have been used in very few cases for modelling software development[58-60]. 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.

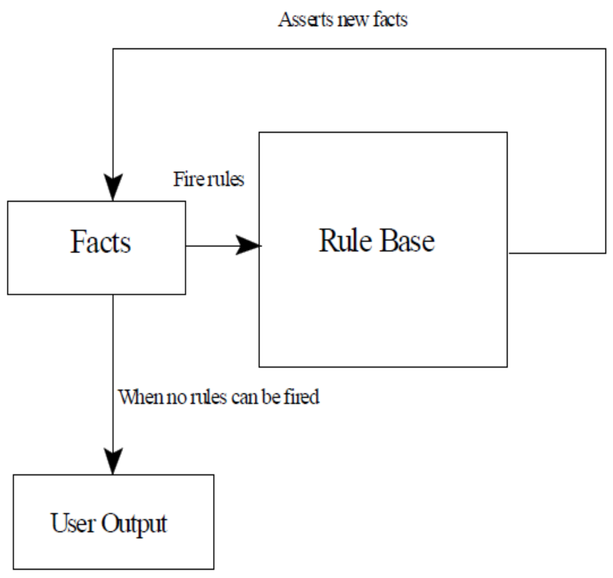

Figure 5. The generic structure of rule-based systems

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.

8. 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[61]. 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[62]. It was found by Vicinanza et al.[63] that experience assisted with software development estimates and that experts at this used comparisons with past cases.

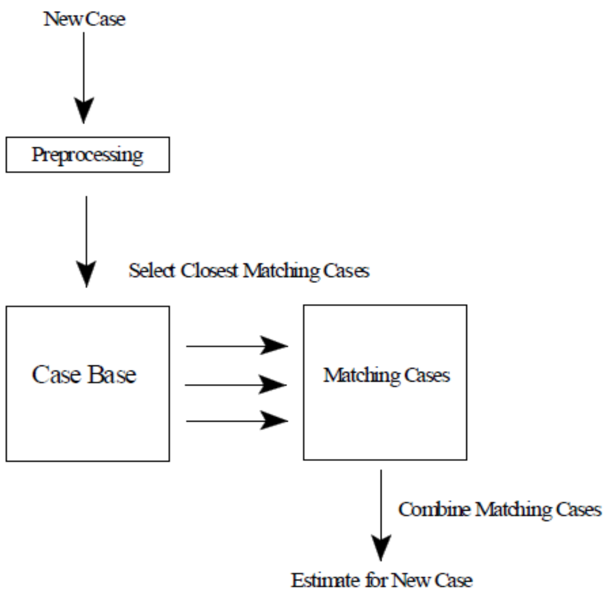

Figure 6. The case-based reasoning classification process

Problems have been encountered with some case-based reasoning systems. As stated by Breiman et al.[64] 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[61] resulted in much more noise-tolerant systems.

An experiment by Mukhopadhyay et al.[62] 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.

9. 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.[64] and Selby and Porter[65,66].

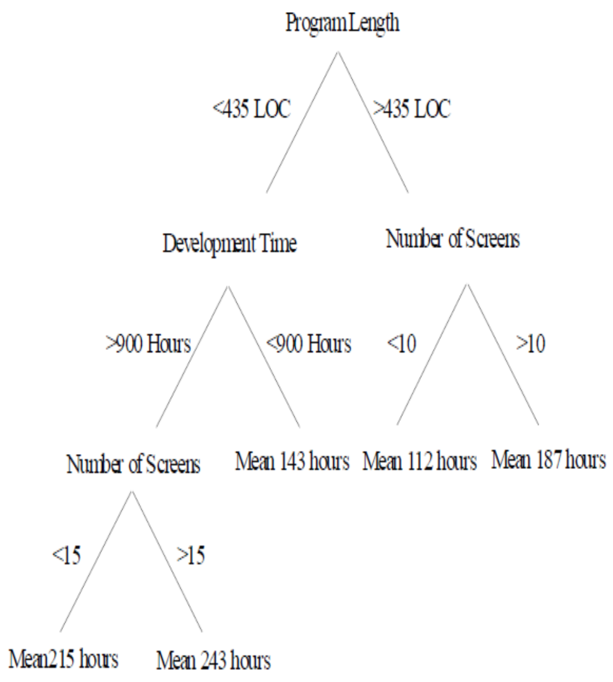

Figure 7. A development time classification model generated by a regression tree algorithm

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[32] 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 back-propagation 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.

10. COMPARISON OF TECHNIQUES

So as to ensure that the most appropriate model-building method is selected in as many cases as possible a two-part comparison of the techniques described above is now provided. The first section considers the appropriateness of each method based on the conceptual requirements of modelling methods. This is followed by an empirical evaluation of some of the methods (least squares regression, robust regression and a neural network) so as to illustrate the variation in performance achieved with these approaches, highlighting the need to more effectively consider the specific characteristics of the form and nature of the data set, the modelling methods, and the resultant model.

10.1. Criteria-based Evaluation

Table 1 shows a comparison between the techniques with respect to some desirable modelling 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. With a neural network, an appropriate approximate transformation will be found by the network when training.

The next entry in the table refers to the model's robustness of estimation when faced with a data set 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 and regression trees) 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 considered. Related to the explanation of a model is the capability for a user to *see* how a model arrived at its conclusions. This can be important for the purpose of verification as well as theory building and gaining and understanding of the process being modelled. 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 incorporate complex models is related to the issue of model-free estimation and the ability to add expert knowledge. Finally the table covers each technique's capability to include known information into a model, that is to initialise a model with known facts (expert knowledge) and then use data to improve and refine it. Table 2 shows each technique in terms of its usage for software metrics.

10.2. Empirical Evaluation

The analysis presented here is a summary of previous work and is intended to illustrate the use of some of the techniques that have been discussed[67]. In addition it is hoped that this provides some appropriate model comparison and validation techniques. A data set of eighty-one project observations[68] (collected originally to investigate the effectiveness of function point analysis (FPA)) were made available for analysis using the various model-building methods. Each observation included values for the following independent (potential predictor) variables: unadjusted function points, adjusted function points, levels of experience with equipment and in project management, numbers of basic transactions, and number of data entities; the main dependent variable was development effort. (It should be noted here that this analysis is not directed at evaluating the actual usefulness of the data in this case; rather, it is being used here to illustrate the different results achieved when different methods are used to develop predictive models.) For each of the methods, the same randomly selected set of fifty-four observations was used for model development, leaving a set of twenty-seven observations available for validation. As stated previously, it is only through the use of such a 'hold-out' data set that a realistic and unbiased assessment of a model's likely performance on new real-world data can be made. Predictive accuracy of each model was evaluated using the mean magnitude of relative error (MMRE) and the threshold-oriented pred measure.

The magnitude of relative error (MRE) is a normalised measure of the variance between actual values (V_A) and fitted values (V_F):

$$MRE = \frac{|V_A - V_F|}{V_A}$$

Table 1. Comparison of techniques in terms of modelling capabilities

Technique	Model free	Can resist outliers	Explains output	Suits small data sets	Can be adjusted for new data	Reasoning process is visible	Suit complex models	Include known facts
Least Squares Regression	□	□	▨	□	□	■	□	▨
Robust Regression	□	■	▨	▨	□	■	□	▨
Neural Networks	■	□	□	□	▨	□	■	▨
Fuzzy Systems (Adaptive)	■	▨	■	■	▨	■	■	■
Hybrid Neuro-Fuzzy Systems	■	▨	■	▨	▨	▨	■	■
Rule Based Systems	□	N/A	■	N/A	N/A	■	■	■
Case-Based Reasoning	■	▨	■	▨	■	▨	■	□
Regression Trees	■	■	■	▨	■	▨	■	▨
Classification or Decision Tress	■	■	■	▨	■	▨	■	▨

Key	
Yes	■
No	□
Partially	▨

The mean MRE is therefore the mean value for this indicator over all observations in the validation sample. A lower value for MMRE generally indicates a more accurate model.

The pred measure provides an indication of *overall* fit for a set of data points, based on the MRE value attained for each data point:

$$pred(l) = \frac{i}{n}$$

where l is the selected threshold value for MRE, i is the number of data points with MRE less than or equal to l, and n is the total number of data points.

As an illustration, if pred(0.25) = 0.4, then we can say that 40% of the fitted values fall within 25% of their corresponding actual values.

The performance of each analysis approach using these adequacy indicators is summarised in Table 3. As can be seen the neural network models performed much better in terms of model accuracy than the regression-based models. This is expected, if only because of the non-linearities inherent in a neural network. A similar ranking of neural networks outperforming statistical models in terms of accuracy has been found in other research[69, 70].

Of the two statistical approaches employed, their performance is roughly equivalent. This suggests that the influence of any outlier observations is not significant enough to have a disproportionate effect on the regression lines and that the data set is reasonably 'well-behaved' from the perspective of regression. Overall, however, the most effective model is that expressed by the neural network, with nearly half the MMRE of the other techniques and superior pred performance. If, as in much metrics data analysis, our model-building had been based only on statistical methods, the effectiveness of the resultant models would be less than optimal when compared to that obtained with the neural network. This further illustrates the need to consider a broader range of issues when selecting a method for analysis.

Table 2. Some applications of each technique

Technique	*Applications for Software Metrics*
Least Squares Regression	Simple models involving a small number of variables, where the relationships are linear or linear after transformation.
Robust Regression	Where the data contains a number of influential outliers and a general model that fits the "normal" projects is most desirable
Neural Networks (Multi-layer perceptron)	Metrics where accuracy is much more important than understanding the relationships.
Neural Networks (Kohonen)	Can be used to cluster systems in similar groups. This can help when modelling very different systems, and separate models need to be developed using other techniques.
Fuzzy Systems	Early estimation where sufficient information for more detailed models is not available (avoids overly precise commitment to specific values and represents the uncertainty of estimates at this stage). Can also be useful where data is only available in small quantities or not at all.
Hybrid Neuro-Fuzzy Systems	Provides a data-driven process for developing accurate fuzzy models and allows for the insertion and extraction of rules. This can assist with gaining understanding of the development process. Requires more data than a fuzzy system, but may operate better than a neural network.
Rule Based Systems	Deterministic systems where relationships exhibit very little stochastic behaviour.
Case-Based Reasoning	Situations where projects with similar independent variables tend to be similar in terms of dependent variables, but the relationship is very complex. Can also be a useful support tool for expert reasoning by analogy[71].
Regression Trees	As with Kohonen networks, regression trees allow different models to be used in a piecewise fashion for different types of system. Provides good explanation facilities.
Classification and Decision Trees	Suitable for non-numeric variables. Can be used to generate theory as well as models.

Table 3. Comparative analysis method performance

Method	MMRE	pred(0.10)	pred(0.25)
LS regression	0.86	0.15	0.41
LMS regression	0.85	0.07	0.41
Neural network	0.44	0.26	0.63

11. CONCLUSIONS

By considering a wide range of modelling techniques that may be suitable for developing predictive software metric models a project manager or researcher can be more confident that the best possible model (for practical purpose this will normally be the most accurate, although other considerations should be kept in mind) 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 these limitations have been discussed in this paper.

Given the set of criteria considered in Table 1, it is evident that some of the non-traditional analysis and modelling methods have significant potential in providing useful and robust predictive models. In particular, the case-based reasoning and regression tree approaches may be favoured in the first instance, as they are able to provide simple, effective and intuitively appealing methods of classification and estimation.

It must be noted, however, that the use of such methods cannot simply be embraced as a catch-all solution to current metrics data analysis problems. The techniques presented here are simply modelling devices that can assist in the gathering of information (for example, fuzzy systems), the derivation of complex relationships inherent in software development, and the presentation of results in a meaningful manner. Other goals, such as increasing the interpretability of models, may also be assisted through their use. The various techniques are, however, fields of research in themselves, and a reasonable level of understanding should be sought before they are used.

We are currently expanding the empirical comparisons to other techniques, especially fuzzy logic models and case-based reasoning (which has been modified based on the analogy approach[71]). In addition a flowchart approach for assisting with model selection is currently under preparation, along with software support for these techniques tailored for software metrics.

REFERENCES

1. Putnam, L.H., A General Empirical Solution to the Macro Software Sizing and Estimating Problem, *IEEE Trans. Soft. Eng*. 4, 345-361 (1978).
2. Boehm, B.W., *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, N.J., 1981.
3. 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).
4. Jeffery, D.R., and Low, G., Calibrating Estimation Tools for Software Development, *Soft. Eng. J*. 5, 215-221 (1990).
5. Kemerer, C.F., An Empirical Validation of Software Cost Estimation Models, *Comm. ACM* 30, 416-429 (1987).
6. MacDonell, S.G., and Gray, A.R., A Review of Model Building Techniques found in Software Metrics Literature, in preparation
7. Courtney, R.E., and Gustafson, D.A., Shotgun Correlations in Software Measures, *Software Eng. J.* 8, 5-13 (1993).
8. Lovell, M.C., Data Mining, *The Review of Economics and Statistics* LXV, 1-12 (1983).
9. Picard, R.R., and Berk, K.N., Data Splitting, *The American Statistician* 44, 140-147 (1990).
10. Snee, R.D., Validation of Regression Models: Methods and Examples, *Technometrics* 19, 415-428 (1977).
11. 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.
12. Efron, B., Bootstrap Methods: Another Look at the Jack-knife, *Annals of Statistics* 7, 1-26 (1979).
13. Young, G.A., Bootstrap: More than a Stab in the Dark?, *Statistical Science* 9, 382-415 (1994).
14. Coupal, D., and Robillard, P.N., Factor Analysis of Source Code Metrics, *J. Systems Software* 12, 263-269 (1990).
15. Mata-Toledo, R.A., and Gustafson, D.A., A Factor Analysis of Software Complexity Measures, *J. Systems Software* 17, 267-273 (1992).
16. Subramanian, G.H., and Breslawski, S., Dimensionality Reduction in Software Development Effort Estimation, *J. Systems Software* 21, 187-196 (1993).
17. Stewart, G.W., Collinearity and Least Squares Regression, *Statistical Science* 2, 68-100 (1987).
18. MacDonell, S.G., *Quantitative Functional Complexity Analysis of Commercial Software Systems*. Unpublished PhD Thesis, University of Cambridge, Cambridge, United Kingdom, 1993.
19. Miyazaki, Y., Terakado, M., and Ozaki, K., Robust Regression for Developing Software Estimation Models, *J. Systems Software* 27, 3-16 (1994).
20. Rousseeuw, P.J. Least Median of Squares Regression, *J. American Statistical Associatio*n 79, 871-880 (1984).
21. 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).
22. Rousseeuw, P.J., and Leroy, A.M., *Robust Regression and Outlier Detection*, John Wiley & Sons, New York, 1987.
23. Hettmansperger, T.P., and Sheather, S.J., A Cautionary Note on the Method of Least Median Squares, *The American Statistician* 46, 79-83 (1992)

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

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

26. 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).

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

28. 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).

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

30. 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).

31. 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).

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

33. 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).

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

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

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

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

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

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

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

41. 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).

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

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

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

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

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

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

48. 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).

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

50. 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).

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

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

53. Wang, L.-X., and Mendel, J.M., Generating Fuzzy Rules by Learning From Examples, *IEEE Trans. Systems, Man, and Cybernetics* 22, 1414-1427 (1992).

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

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

56. 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).

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

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

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

60. 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).

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

62. 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).

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

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

65. 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).

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

67. MacDonell, S.G., and Gray, A.R., Alternatives to Regression Models for Estimating Software Projects, in *Proceedings of the IFPUG Fall Conference*, IFPUG, Westerville OH, 279.1-279.15 (1996).

68. Desharnais, J-M., Analyse statistique de la productivitie des projects de development en informatique apartir de la technique des points des fonction. Master's Thesis, Universite du Montreal (1989).

69. Samson, B., Ellison, D., and Dugard, P., Software Cost Estimation using an Albus Perceptron (CMAC), To appear, *Information and Software Technology*.

70. Jorgensen, M., Experience with the Accuracy of Software Maintenance Task Effort Prediction Models, *IEEE Trans. Soft. Eng.* 21, 674-681 (1995).

71. Shepperd, M., Schofield, C., and Kitchenham, B., Effort Estimation Using Analogy, in *Proceedings 18th International Conference on Software Engineering*, IEEE Computer Society Press, Los Alamitos CA, 170-178 (1996).