Available online at www.sciencedirect.com

SCIENCE DIRECT®

The Journal of Systems and Software 76 (2005) 147–156

The Journal of Systems and Software

www.elsevier.com/locate/jss

Application of neural networks for software quality prediction using object-oriented metrics

Mie Mie Thet Thwin *, Tong-Seng Quah

School of Electrical and Electronic Engineering, Info. Comm. Research Lab, s2-1-B4-04, Nanyang Technological University, Singapore 639798, Singapore

Received 21 July 2003; received in revised form 11 April 2004; accepted 2 May 2004
Available online 7 June 2004

Abstract

This paper presents the application of neural networks in software quality estimation using object-oriented metrics. In this paper, two kinds of investigation are performed. The first on predicting the number of defects in a class and the second on predicting the number of lines changed per class. Two neural network models are used, they are Ward neural network and General Regression neural network (GRNN). Object-oriented design metrics concerning inheritance related measures, complexity measures, cohesion measures, coupling measures and memory allocation measures are used as the independent variables. GRNN network model is found to predict more accurately than Ward network model.

1. Introduction

Many object-oriented metrics have been proposed over the last decade. Prediction models using object-oriented design metrics can be used for obtaining assurances about software quality. In practice, quality estimation means either estimating reliability or maintainability. The estimated number of defects can also be normalized by a size measure to obtain a defect density estimate. Maintainability is typically measured as change effort. Change effort can mean either the average effort to make a change to a class, or the total effort spent on changing a class.

A variety of statistical techniques are used in software quality modeling. Models are often based on statistical relationships between measures of quality and measures of software metrics. However, relationships between static software metrics and quality factors are often complex and nonlinear, limiting the accuracy of conventional approaches. Artificial neural networks are adept at modeling nonlinear functional relationships that are difficult to model with other techniques, and thus, are attractive for software quality modeling.

We conduct our study in the object-oriented paradigm. However since the object-oriented paradigm exhibits different characteristics from the procedural paradigm, different software metrics have to be defined and used.

Our neural network model aims to predict object-oriented software quality by estimating the number of faults and the number of lines changed per class. We used software metrics including both object-oriented metrics and traditional complexity metrics. Object-oriented metrics used include inheritance related measures, cohesion measures, coupling measures and memory allocation measures.

We also introduce using Ward neural network and General Regression neural network to improve prediction result for estimating software quality. Ward neural network is a backpropagation network with different activation functions. They are applied to hidden layer slabs to detect different features in a pattern processed through a network to lead to better prediction. We use a Gaussian function in one hidden slab to detect features in the mid-range of the data and a Gaussian complement in another hidden slab to detect features for the upper and lower extremes of the data. Thus, the output

* Corresponding author. Tel.: +65-90464662; fax: +65-7922971.
E-mail address: miethet@pmail.ntu.edu.sg (M.M.T. Thwin).

0164-1212/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2004.05.001

layer will get different "views of the data". Combining the two feature sets in the output layer leads to a better prediction.

Another architecture that we have chosen is the General Regression Neural Network (GRNN). Specht (1991) states that it is a memory-based network that provides estimates of continuous variables and converges to the underlying (linear or nonlinear) regression surface. This is a one-pass learning algorithm with a highly parallel structure. Even with sparse data in a multidimensional measurement space; the algorithm provides smooth transitions from one observed value to another.

2. Related work

There is great interest in the use of object-oriented approach in software engineering. With the increasing use of object-oriented methods in new software development there is a growing need to both document and improve current practices in object-oriented design and development.

Many measures have been proposed in the literature to capture the quality of object-oriented (OO) code and design for detecting fault-proneness of classes (Briand et al., 2002; Cartwright and Shepperd, 2000; Emam et al., 2001; Fioravanti and Nesi, 2001; ReiBing, 2001). Many investigations using statistical methods have been made to predict software quality.

One such set of object-oriented metrics is the set proposed by Chidamber and Kemerer (1994). Chidamber and Kemerer also reported empirical data from two commercial organizations and suggested ways in which the metrics could be used to manage OO design efforts. In their paper it was suggested that the primary use of the metrics by managers would be to identify outlying values that might reflect suboptimal design practice.

Emam et al. (2001) have constructed a model to predict which classes in a future release of a commercial Java application will be faulty. The model was then validated on a subsequent release of the same application. Their results indicated that the prediction model had a high accuracy.

Fioravanti and Nesi (2001) have extracted over 200 different object-oriented metrics to identify a suitable model for detecting fault-proneness of classes. They came to the conclusion that only a few of them were relevant for identifying fault-prone classes.

A set of object-oriented metrics in terms of their usefulness in predicting fault-proneness, an important software quality indicator is empirically validated in Ping et al. (2002). Their validation is carried out using two data analysis techniques: regression analysis and discriminant analysis.

Briand et al. (2000) performed an empirical study on the relationships between existing object-oriented coupling, cohesion, and inheritance measures and the probability of fault detection in system classes during testing. Their univariate analyses have shown that many coupling and inheritance measures are strongly related to the probability of fault detection in a class. Their multivariate analysis results showed that by using some of the coupling and inheritance measures, very accurate models could be derived to predict in which classes most of the faults actually lie.

Most of these prediction models are built using statistical models. Neural networks have seen an explosion of interest over the years, and are being successfully applied across a range of problem domains, in areas as diverse as finance, medicine, engineering, geology and physics. Indeed, anywhere that there are problems of prediction, classification or control, neural networks are being introduced. A neural network can be used as a predictive model because it is very sophisticated modeling technique capable of modeling complex functions.

Khoshgoftaar et al. (2000) presented a case study of real-time avionics software to predict the testability of each module from static measurements of source code. They found that the neural network is a promising technique for building predictive models, because they are able to model nonlinear relationships.

Our neural network model aims to predict object-oriented software quality by estimating the number of faults per class and the number of lines changed per class. We also introduce using Ward neural network and General Regression neural network to improve prediction results for estimating software quality.

3. Design of the study

3.1. Neural network modeling

The first neural network architecture that we have chosen is the Ward Network (NeuroShell 2, 2001) as shown in Fig. 1. Backpropagation network is the most popular network for practical applications. We chose three layer Backpropagation Ward neural network

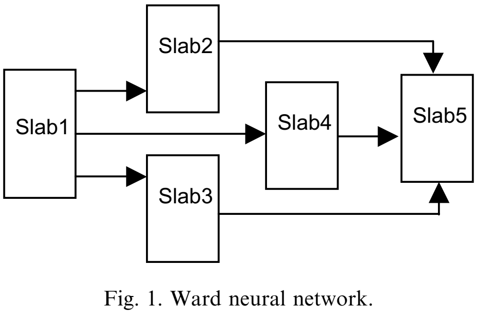


Fig. 1. Ward neural network.

because it is an effective network for most applications and trains much more quickly than 4 or 5 layer networks. We use three different activation functions. It has three slabs (slab2, slab3 and slab4) in the hidden layer. Hidden layers in a neural network are known as feature detectors. A slab is a group of neurons. Each slab in the hidden layer has a different activation function; this offers three ways of viewing the data. We use a linear function for the output slab (slab5). The hyperbolic tangent (tanh) function is used in one slab of hidden layer (slab3) because it is better for continuous valued outputs especially if the linear function is used on the output layer. The Gaussian function is used in another slab of the hidden layer (slab2). This function is unique, because unlike the others, it is not an increasing function. It is the classic bell shaped curve, which maps high values into low ones, and maps mid-range values into high ones. The Gaussian Complement is used in the third slab of the hidden layer (slab4) to bring out meaningful characteristics in the extremes of the data. We use a smaller learning rate 0.1 and momentum 0.1 as our network is a predictive network where outputs are continuous values rather than categories.

Another neural network architecture that we have chosen is the General Regression Neural Network (GRNN) as shown in Fig. 2. GRNN is based on a one-pass learning algorithm with a highly parallel structure. GRNN is a powerful memory based network that could estimate continuous variables and converge to the underlying regression surface. The strength of GRNN is that it is able to deal with sparse data effectively. GRNNs feature fast training times, can model non-linear functions, and have been shown to perform well in noisy environments given enough data. Specht (1991) has shown that the algorithm in GRNN is able to provide a smooth transition from one observed value to another, even with sparse data in a multidimensional measurement space. GRNN applications are able to produce continuous valued outputs. For GRNN networks, the number of neurons in the hidden layer (Slab2) is usually the number of patterns in the training set because each pattern in the training set is represented by one neuron. The number of neurons in the input layer (Slab1) is the number of inputs, and the number of neurons in the output layer (Slab3) corresponds to the number of outputs.

The primary advantage of GRNN is the speed at which the network can be trained. There are no training parameters such as learning rate and momentum in backpropagation network, but there is a smoothing factor that is applied after the network is trained. Smoothing factor is the only adjustable parameter in GRNN. Therefore, making overtraining is less likely in GRNN. The smoothing factor allows the GRNN to interpolate between patterns or spectra in the training set. The smoothing factor determines how tightly the network matches its predictions to the data in the training patterns. For GRNN networks, the smoothing factor must be greater than 0 and usually ranges from 0.01 to 1 with good results.

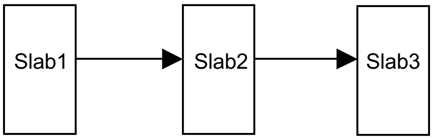


Fig. 2. GRNN network.

3.2. Principal component analysis

If a group of variables in a data set are strongly correlated, these variables are likely to measure the same underlying dimension (i.e., class property) of the object. Many object-oriented metrics have high correlation with each other. Principal components analysis transforms raw data into variables that are not correlated to each other. If correlated raw software metrics were used directly, the neural network models did not train satisfactorily, but transforming raw data with principal components analysis facilitated training. Principal component analysis (PCA) is a standard technique to identify the underlying, orthogonal dimensions that explain relations between the variables in a data set. Principal components (PCs) are linear combinations of the standardized independent variables. It is also a data reduction technique. The varimax rotation method was used in this study. It is an orthogonal rotation method that minimizes the number of variables that have high loadings on each factor. It simplifies the interpretation of the factors. We selected only PCs whose Eigen values are larger than 1.0. We preformed PCA using SPSS software.

4. Prediction of number of faults

4.1. Description of data

Faults appear when a program does not perform according to users' specification during testing and operations stages. The applications used in this prediction are three subsystems of HMI (Human Machine Interface) software, which is a fully networked Supervisory Control and Data Acquisition system. This software, which consists of more than 200 subsystems and 3 million lines of code, has been used by many manufacturing companies for several years. Although each subsystem selected plays a different role in the system and performs a different functionality, they share some similar characteristics that meet our selection criteria. Subsystem A is a user interface-oriented program that allows customers to configure the basic product operations and device communications. It consists of 20 classes that define 256 new, re-defined or virtual functions, and approximately 5600 lines of code in length. Subsystem B is a real time data logging process that collects data as needed and logs data into the database, based on the user configuration. This subsystem defines 48 classes and 353 new, re-defined or virtual functions, comprising approximately 21,300 lines of code. Subsystem C is a communication-oriented program that acts as a router, not only delivering messages between processes within the same host but also forwarding messages to other hosts. This subsystem defines 29 classes and 293 new, re-defined or virtual functions and contains approximately 16,000 lines of code (Tang et al., 1999).

4.2. Selection of metrics

As discussed in Section 1, we are introducing the research on software defects prediction into the object-oriented paradigm using neural networks. We have selected the object-oriented metrics that have a strong relationship with software quality. To indicate the presence of software defects the following existing metrics (Chidamber and Kemerer, 1994 ; Tang et al., 1999) are used.

Depth of Inheritance Tree (DIT) of a class is the length of the longest path from the class to the root in the inheritance hierarchy. This determines the complexity of a class based on its ancestors, since a class with many ancestors is likely to inherit much of the complexity of its ancestors. The deeper a class is in the hierarchy, the greater the number of methods it is likely to inherit making it more complex to predict its behavior. Therefore, the more likely it is to contain a fault.

Number of Children (NOC) measures the number of immediate descendants of a particular class. This measures an amount of potential reuse of the class. The more reuse a class might have, the more complex it may be, and the more classes are directly affected by changes in its implementation. This increases the magnitude of ripple effects. Therefore we selected the NOC metric to predict the number of faults.

Coupling Between Objects (CBO) is defined as the number of other classes to which a class is coupled. Coupling measures the degree of inter dependence among the components of a software system. High coupling makes a system more complex; highly interrelated modules are harder to understand, change or correct and thus likely to be more fault-prone.

Response For a Class (RFC) is the number of methods that can potentially be executed in response to a message received by an object of that class. The response set of a class consists of the set of M methods of the class, and the set of methods directly or indirectly invoked by methods in M. The number of methods that could potentially respond to a message indicates the complexity of that class. Therefore RFC can be used as a predictor variable for the number of faults.

Inheritance Coupling (IC) provides the number of parent classes to which a given class is coupled. A class is coupled to its parent class if one of its inherited methods is functionally dependent on the new or redefined methods in the parent class. When a data member, which is used by an inherited method, is modified by a new or redefined method, it is likely to introduce new faults into the inherited method.

Coupling Between Methods (CBM) provides the total number of new/redefined methods in which all the inherited methods are coupled. CBM measures the total number of function dependency relationships between the inherited methods and new/redefined methods. We have chose CBM to predict the number of faults because it can measure the functional dependency complexity at the methods level.

Weighted Methods per Class (WMC) is defined as a function of the number of all member functions and operators in each class. We have selected WMC to measure the complexity of an individual class.

Number of Object/Memory Allocations (NOMA) measures the total number of statements that allocates new objects or memories in a class. A class with more object/memory allocating activities tends to introduce more object management faults related to object copying, dangling reference, object memory usage faults and so on. Therefore NOMA can be used as a predictor variable for software quality.

First, we performed a preliminary analysis using multiple regressions. As shown in Appendix A, we got an R square value of 0.856. R square is a regression quality indicator that measures how much variance in the dependent variable is accounted for by the independent variables in the sample. About 85% of the variation in the criterion variable, the number of faults in the class, can be explained by the regression model with all predictors WMC, DIT, NOC, CBO, RFC, IC, CBM and NOMA. The adjusted R square corrected for the number of predictors equals 0.843. So therefore, 84% of the total variance in the number of faults is accounted for by the metrics in the population. The observed significance level is less than 0.001. We can conclude that the prediction of the number of faults from the above metrics is possible.

4.3. Experiments

Relationships between static software metrics and quality factors are often complex and nonlinear, limiting the accuracy of conventional approaches. Artificial neural networks are adept at modeling nonlinear functional relationships that are difficult to model with other techniques, and thus, are attractive for software quality modeling.

First, each data pattern was examined for erroneous entries, outliers, blank entries and redundancy. We standardized the metrics to a mean of zero and a variance of one for each metric. Many raw software metrics have incompatible units of measures. This step converts all of them to the unit of one standard variation. After standardizing the metric data, we performed the principal component analysis. Table 1 presents the relationship between the original object-oriented metrics and the domain metrics for HMI system.

For HMI system, PCA identified three sets of principal components (PCs), which capture 40.7%, 19.24% and 18.28% respectively of the data set variance, which gives a representation of about 78.22% of the population. Table 1 shows the coefficient measure for each rotated component, with coefficients larger than 0.6 set in boldface. The Eigen value, the percentage of the data set described, and the cumulative variance percentages are also shown. Based on the analysis of the coefficients associated with each metric within each of the three sets of rotated components, the PCs are interpreted as follows.

The first principal component shows high correlation between metrics with RFC, WMC, CBM, IC and NOMA. The second principal component is highly correlated with NOC and CBO. The third principal component shows highest correlation with DIT.

We divided our data into training, testing, and production sets using 3:1:1 ratio, which is the commonly accepted proportion used by most neural network researchers. We randomly extracted 19 patterns for the test set and another 19 patterns for the production set. The remaining 59 patterns are used as training set. We used the production data set to evaluate model performance.

The dependent variable was the number of faults per class and the independent variables were the three principal components identified above (out of eight software metrics). We used both the Ward network and GRNN network for predicting the number of defects. The Ward neural design summary is presented in Table 2.

Table 1
Rotated principal components for the HMI system

Metrics	PC1	PC2	PC3
WMC	**0.9068**	−0.0544	−0.1768
DIT	−0.0358	−0.0400	**0.9286**
NOC	−0.0274	**0.8710**	−0.0694
CBO	−0.0043	**0.8508**	0.0471
RFC	**0.9399**	−0.0818	−0.0919
IC	**0.6452**	0.1678	0.5140
CBM	**0.8636**	0.0811	0.2932
NOMA	**0.6216**	−0.1029	0.4508
Eigenvalues	3.256	1.539	1.462
% Variance	40.703	19.238	18.279
Cummulative % variance	40.703	59.940	78.219

Table 2
Ward neural network architecture used

	Slab1	Slab2	Slab3	Slab4	Slab5
No. of neurons	3	4	4	4	1

Table 3
GRNN neural network architecture used

	Input layer	Hidden layer	Output layer
No. of neurons	3	97	1

The GRNN usually requires one hidden node for each training sample. Therefore we used 97 neurons in the hidden layer, 3 neurons in the input layer and one neuron in the output layer as shown in Table 3. In this study, the value of the smoothing factor is 0.05507813.

4.4. Experiment results

To measure the goodness of fit of the model, we use the coefficient of multiple determination (R square), the coefficient of correlation (r), mean square error, mean absolute error, minimum absolute error and maximum absolute error. These statistical measures are shown in Table 4. The correlation of the predicted change and the observed change is represented by the coefficient of correlation (r). An r value of 0.9476 in Ward neural network and 0.9531 in GRNN network represent high correlations for cross-validation. The number of observations is 97. The significance level of a cross-validation is indicated by the p value. A commonly accepted p value is 0.05. In our experiment, the two tailed probability p value is less than 0.001 in the both cross-validations. This shows a high degree of confidence for the successful validations. The results clearly indicate a close relationship between metrics (independent variables) and the number of faults per class in software applications (dependent variable).

Table 4
Experimental result for the HMI system

	Ward	GRNN
R square	0.8715	0.9077
r (correlation coefficient)	0.9476	0.9531
Mean square error	1.584	1.138
Mean absolute error	0.823	0.765
Min absolute error	0.001	0
Max absolute error	6.211	4.295
t Values	28.88816	28.88154
p Values	<0.001	<0.001

4.5. Results from a 10-cross-validation study

We performed a 10-cross-validation of the prediction neural network model in Tables 2 and 3. It is a common technique to evaluate learning algorithms on a dataset. The 97 data points were randomly split into 10 partitions or folds of roughly equal size (eight partitions of 10 data points each, one partition of nine data points and one partition of eight data points). For each fold, the following processing is performed. For each iteration, the selected fold becomes the test set, and the other 9 folds are combined into the training set. Create a neural network and train it on the training set and test it on the test set. We performed 10 times and took a mean accuracy to obtain the estimated performance of the neural network. The results are summarized in Table 5 for the GRNN network and in Table 6 for the Ward neural network.

The average value of the correlation between predicted faults and actual faults is 0.94147 in the GRNN network and 0.93515 in the Ward network. They are statistically significant. The average squared multiple correlation is 0.88008 in the GRNN network and 0.85966 in the Ward network. Therefore about 88% and 86% of the variance, respectively, in the number of faults can be accounted for by the two predictors.

Table 5
Results from 10-cross-validation for the GRNN network

Partition	*R* squared	Correlation coefficient *r*
1	0.9151	0.9576
2	0.8863	0.9428
3	0.8563	0.9286
4	0.8963	0.9481
5	0.8485	0.9238
6	0.9486	0.9743
7	0.9355	0.9677
8	0.6425	0.8362
9	0.9137	0.9567
10	0.9580	0.9789
Average	0.88008	0.94147

Table 6
Results from 10-cross-validation for the Ward Network

Partition	*R* squared	Correlation coefficient *r*
1	0.8834	0.9401
2	0.8749	0.9377
3	0.7417	0.9043
4	0.8593	0.9327
5	0.8593	0.9327
6	0.8724	0.936
7	0.8765	0.9442
8	0.8765	0.9442
9	0.8825	0.9427
10	0.8701	0.9369
Average	0.85966	0.93515

5. Prediction of maintenance effort

5.1. Description of data

This investigation attempted to predict the maintenance effort. The commercial software product QUES(Quality Evaluation System) data is used in this investigation (Wei and Henry, 1993). The maintenance effort is measured by counting the number of lines changed per class. A line change could be an addition or a deletion. A change of the content of a line is counted as a deletion followed by an addition. The QUES system was designed and developed with Class-Ada.

5.2. Selection of metrics

We have selected the software metrics that have a strong relationship with software maintainability. To predict the maintenance effort, DIT, RFC and WMC that are previously defined in Section 4.2 and the following metrics are used.

Message Passing Coupling (MPC) gives an indication of how many messages are passed among objects of the classes. The number of messages sent out from a class indicates how dependent the implementation of the local methods is on the methods in other classes.

Lack of Cohesion in Methods (LCOM) is the number of pairs of methods in the class using no attributes in common, minus the number of pairs of methods that do. If this difference is negative, LCOM is set to zero. The cohesion of a class is characterized by how closely the local methods are related to the local instance variables in the class. It is harder to maintain a class that has a larger LCOM metric value because, if all the methods defined in a class access many independent sets of data structures encapsulated in the class, the class may not be well designed and partitioned. LCOM can be used to predict the maintainability of the class.

Data Abstraction Coupling (DAC) is the number of attributes in a class that have as their type another class. This metric measures the coupling complexity caused by abstract data type (ADT). The number of variables having ADT type may indicate the number of data structures dependent on the definitions of other classes. It is more difficult to maintain the class that has more ADT's.

The number of local methods (NOM) defined in a class indicates the operation property of a class. The more methods a class has, the more complex will be the class's interface.

Size metrics have been used as software metrics for a long time. Two size metrics are used in this prediction. The first size metric (SIZE1) is the traditional line of code metric, which is calculated by counting the number of semicolons in a class. The second size metric (SIZE2) is the total number of attributes and methods of a class.

We performed the preliminary analysis using multiple regression. As shown in Appendix B, we got an *R* square value of 0.734. About 73% of the variation in the criterion variable maintenance effort can be explained by the regression model with all predictors DIT, MPC, RFC, LCOM, DAC, WMC, NOM, SIZE1 and SIZE2. The adjusted *R* square is 0.694. So therefore, 69% of the total variance in the number of faults is accounted for by the metrics in the population. The observed significance level is less than 0.001. We can conclude that the prediction of maintenance effort from the above metrics is possible.

5.3. Experiments

As in this experiment, first, each data pattern was examined for erroneous entries, outliers, blank entries and redundancy. After standardizing the metric data, we performed the principal component analysis. Table 7 presents the relationship between the original object-oriented metrics and the domain metrics for QUES system.

For the QUES system, PCA identified three PCs, which capture 89% of the data set variance; Table 7 shows for each rotated component the coefficients of the measure, with coefficients larger than 0.6 set in boldface. The Eigen value, the percentage of the data set variance each PC describes, and the cumulative variance percentage are also provided. Based on the analysis of the coefficients associated with each metric within each of the three rotated components, the PCs are interpreted as follows.

The first component is highly correlated with NOM, SIZE2, RFC, LCOM, WMC, SIZE1 and DAC. NOM is the best representative, however, because it is less correlated with the other two components. The second component is most highly correlated with MPC. The third component is most highly correlated with DIT. This suggests that NOM, MPC and DIT metrics should be focused on in further analysis for this system.

Table 7
Rotated principal components for the QUES system

Metrics	PC1	PC2	PC3
DIT	0.060	0.027	**0.966**
MPC	−0.023	**0.966**	0.037
RFC	**0.877**	0.333	0.043
LCOM	**0.869**	−0.156	0.059
DAC	**0.796**	0.027	0.427
WMC	**0.832**	0.258	−0.27
NOM	**0.971**	−0.132	0.097
SIZE1	**0.812**	0.475	−0.089
SIZE2	**0.963**	−0.093	0.190
Eigenvalues	5.384	1.388	1.248
% Variance	59.826	15.424	13.863
Cummulative % variance	59.826	75.250	89.113

Table 8
Ward neural network architecture used for the QUES system

	Slab1	Slab2	Slab3	Slab4	Slab5
No. of neurons	3	3	3	3	1

Table 9
GRNN neural network architecture used

	Input layer	Hidden layer	Output layer
No. of neurons	3	71	1

We divided the data into training, testing, and production sets using a 3:1:1 ratio. Test set is used to prevent over training the network so they will generalize well. We used the production data set to evaluate model performance. The network's results can be tested with the data the network has never seen before.

We used the Ward network and the GRNN network for predicting the number of changes. Table 8 shows the summary of the Ward network design. In our General Regression neural network design, there were 71 neurons in the hidden layer, 3 neurons in the input layer and one neuron in the output layer as shown in Table 9. In this study, the value of the smoothing factor is 0.02460938.

5.4. Experiment results

The summary results in terms of the goodness of fit of the model are shown in Table 10. The correlation of the predicted change and the observed change is represented by the coefficient of correlation (*r*). The coefficient of correlation values of 0.747 in the Ward neural network and 0.8590 in the GRNN network represent high correlations for cross-validation. The number of

Table 10
Experimental result for the QUES system

	Ward	GRNN
R square	0.5545	0.7220
r (correlation coefficient)	0.747	0.8590
Mean square error	817.004	509.790
Mean absolute error	20.782	12.182
Min absolute error	0.094	0
Max absolute error	114.161	109.385
t Values	9.329047	13.98484
p Values	<0.001	<0.001

Table 11
Results from 10-cross-validation for the GRNN network

Partition	R squared	Correlation coefficient r
1	0.7553	0.8824398
2	0.7984	0.9102747
3	0.7539	0.9206519
4	0.604	0.781025
5	0.6032	0.78
6	0.7759	0.9027181
7	0.7215	0.8598256
8	0.7345	0.8906178
9	0.5709	0.7584194
10	0.7963	0.8946508
Average	0.71139	0.8580623

Table 12
Results from 10-cross-validation for the Ward network

Partition	R squared	Correlation coefficient r
1	0.5723	0.7696753
2	0.5046	0.7690904
3	0.551	0.7721399
4	0.6472	0.8093207
5	0.544	0.7495999
6	0.5787	0.7694803
7	0.6277	0.7988742
8	0.5517	0.7179136
9	0.5048	0.717844
10	0.5247	0.7358668
Average	0.56067	0.7609805

observations is 71. Two tailed probability p values less than 0.001 in both cross-validations shows a high degree of confidence for the successful validations. We conclude that the impact of model prediction is valid in the population.

5.5. Results from a 10-cross-validation study

Results from the 10-cross-validation of the prediction neural network model are shown in Tables 11 and 12. The 71 data points were randomly split into 10 partitions or folds of roughly equal size (nine partitions of seven data points each and one partition of eight data points).

The average value of the correlation between maintenance effort and actual maintenance effort is 0.8580623 in GRNN network and 0.7609805 in the Ward network. They are statistically significant. The average squared multiple correlation is 0.71139 in the GRNN network and 0.56067 in the Ward network. Therefore about 71% and 56%, respectively, of the variance in the maintenance effort can be accounted for by the two predictors.

6. Conclusion

The neural network modeling techniques used in this study are applicable to improving the quality of software products. This paper presents regression models and neural network models of the Human Machine Interface system and Quality Evaluation System, predicting the number of software defects and the maintenance effort. The independent variables were principal components of software design metrics. Tables 4 and 10 summarize the prediction results of the neural network models. The two tailed probability p values are less than 0.001 in both cross-validations. That shows a high degree of confidence for the successful validations.

In the prediction of the number of faults experiment, the correlation between predicted faults and actual faults is 0.9531 in the GRNN network and 0.9476 in the Ward network. In the maintainability prediction experiment, the correlation between predicted maintenance effort and actual maintenance is 0.8590 in the GRNN network and 0.747 in the Ward network. They are statistically significant. The GRNN network model is found to predict more accurately than the Ward network model.

From the results presented above, the object-oriented metrics chosen in this study appear to be useful in predicting software quality. These software metrics are significantly related to the number of faults and the maintenance effort. These prediction models can be used to forecast the group membership of modules from a subsequent release or a similar system developed in the same environment.

Our future research direction aims to estimate software readiness using neural network models. To estimate readiness, three factors will be considered in our future study: (1) how many faults are remaining in the programs (2) how many changes are required to correct the errors and (3) how much time is required to change the programs. Software metrics concerning with polymorphism, inheritance, complexity, cohesion, coupling, dynamic memory allocation, database operations and size will be used.

Acknowledgement

The authors would like to thank Associate Professor Dr. Mei-Hwa Chen, Computer Science Department, University at Albany, State University of New York, for allowing us to test our model with data they had collected from industrial real-time systems.

Appendix A. Regression analysis for HMI system

Model summary

Model	R	R Square	Adjusted R square	Std. error of the estimate	Change statistics				
					R Square change	F Change	df1	df2	Sig. F change
1	0.925[a]	0.856	0.843	1.39971	0.856	65.283	8	88	0.000

[a] Predictors: (Constant), NOMA, CBO, DIT, IC, NOC, WMC, CBM, RFC.

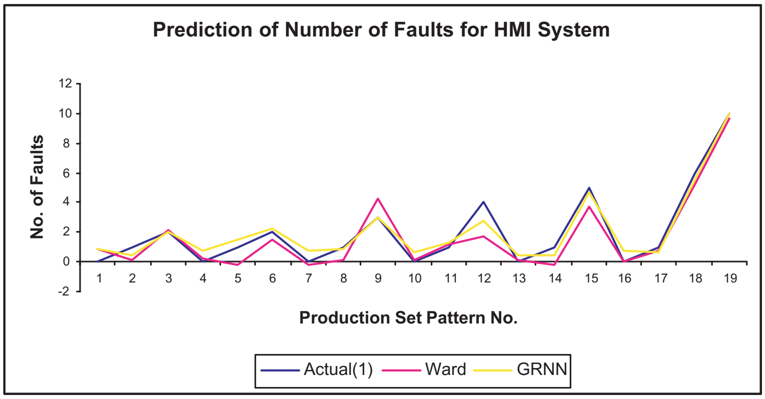
Appendix B. Regression analysis for QUES system

Model summary

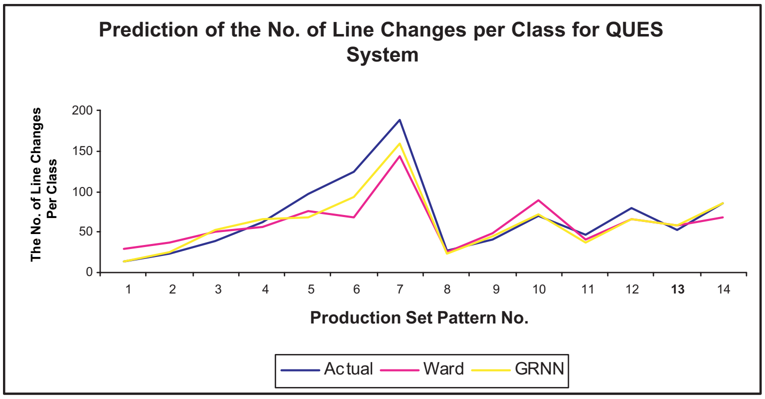
Model	R	R Square	Adjusted R square	Std. error of the estimate	Change statistics				
					R Square change	F Change	df1	df2	Sig. F change
1	0.856[a]	0.734	0.694	23.84591	0.734	18.663	9	61	0.000

[a] Predictors: (Constant), SIZE1, DIT, MPC, LCOM, DAC, WMC, RFC, NOM, SIZE2.

Appendix C



Appendix D



References

Briand, L. et al., 2000. Exploring the relationships between design measures and software quality in object-oriented systems. Journal of Systems and Software 51, 245–273.

Briand, L. et al., 2002. Assessing the applicability of fault-proneness models across object-oriented software projects. IEEE Transactions on Software Engineering 28, 706–720.

Cartwright, M., Shepperd, M., 2000. An empirical investigation of object oriented software system. IEEE Transactions on Software Engineering 26, 786–796.

Chidamber, S., Kemerer, C., 1994. A metrics suite for object oriented design. IEEE Transactions on Software Engineering 20, 476–493.

Emam, E. et al., 2001. The prediction of faulty classes using object-oriented design metrics. Journal of Systems and Software, 63–75.

Fioravanti, F., Nesi, P., 2001. A study on fault-proneness detection of object-oriented systems. In: Fifth European Conference on Software Maintenance and Reengineering, pp. 121–130.

Khoshgoftaar, T., et al., 2000. Predicting testability of program modules using a neural network. In: Proceedings of the 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology, pp. 57–62.

NeuroShell 2 Help, Ward Systms Group, Inc. http://www.wardsystems.com.

Ping, Y., et al., 2002. Predicting fault-proneness using OO metrics. An industrial case study. In: Proceedings of 6th European Conference on Software Maintenance and Reengineering, pp. 99–107.

ReiBing, R., 2001. Towards a model for object-oriented design measurement. In: Proceedings of the 5th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, pp. 71–84.

Specht, D., 1991. A general regression neural network. IEEE Transactions on Neural Networks 2 (6), 568–576.

Tang, M. et al., 1999. An empirical study on object-oriented metrics. In: Proceedings of the Sixth IEEE International Symposium on Software Metrics, pp. 242–249.

Wei, L., Henry, S., 1993. Object-oriented metrics that predict maintainability. Journal of Systems and Software, 111–122.