APPLICATION OF NEURAL NETWORK FOR PREDICTING SOFTWARE DEVELOPNIENT FAULTS USING OBJECT-ORIENTED DESIGN METRICS

Me Me *Thet Thwin*　　**Tong-Seng** *Quah*

School **of** Electronic & Electrical Engineering
Nanyang **Technological University**
pl 15285@ntu.edu.sg

ABSTRACT

In this paper, we present the application of neural network for predicting software development faults including object-oriented faults. Object-oriented metrics can be used in quality estimation. In practice, quality estimation means either estimating reliability or maintainability. In the context of object-oriented metrics work, reliability is typically measured as the number of defects. Object-oriented design metrics are used as the independent variables and the number of faults is used as dependent variable in our study. Software metrics used include those concerning inheritance measures, complexity measures, coupling measures and object memory allocation measures. We also test the goodness of fit of neural network model by comparing the prediction result for software faults with multiple regression model. Our study is conducted on three industrial real-tirne systems that contain a number of natural faults that has been reported for three years [1].

1. INTRODUCTION

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. It can learn by example. In order to make a neural network useful, the user needs to gather representative data, and then invokes training algorithms to train the neural network.

Only a few applications of artificial network to sofiware quality have appeared in the literature [15]. Khoshgofiarr et al. [15] introduced the use of the neural networks as a tool for predicting sofiware quality. They compared the neural-network model with a nonparametric disciminant model, and found the neural network model had better predictive accuracy. Their model used domain metrics derived from the complexity metric data. These metrics are not adequate for detecting object-oriented faults. Since the object-oriented paradigm exhibits different characteristics from the procedural paradigm, software metrics in object-oriented paradigm need to be used.

With the increasing use of object-oriented methods in new software development there is a growing need to improve current practice in object-oriented design and development.

Possible problems in system designs can be detected during development process. Meaningful soflzware metrics are used for measuring project progress and quality. They provide help for developers and managers to promote better designs, more reusable code, and better estimates. The software metrics can be divided into project metrics and design metrics. Project metrics are used to predict project needs, such as staffing levels and total effort. They also measure the dynamic changes that have taken place in the state of the project, such as “how much has been done” and “how much is lefi to do”. These metrics are more global and less specific than the design metrics. Design metrics are measurements of the static state of the project design at a particular point in time. These metrics are more localized and prescriptive in nature. They look at the quality of the way the system is being built. The quality characteristics are reliability, maintainability, extendibility, usability and reusability [2].

Object-oriented metrics can be used in quality estimation. In practice, quality estimation means either estimating reliability or maintainability. In the context of object-oriented metrics work, reliability is typically measured as the number of defects. These can be pre-release or post release. The estimated number of defects can also be normalized by a size measure to obtain a defect density estimate.

We aim to build an efficient predictive model by combining power of neural network and object-oriented software metrics.

2. RELATED WORK

Toshihiro Kamiya et al. [3] presented a method to, estimate the fault-proneness of 'the class in-the early phase, using several complexity metrics for" object—' oriented software. They introduced four checkpoints into the analysis/design/irnplementation phase, and estimate the fault-prone classes using the applicable metrics at each, checkpoint. They estimate thefault-proness by using the multivariate logistic regression analysis.

Emanm and Melo [4] have performed to construct a logistic regression 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. They used ten design metrics, two defined by» Chidamber and Kemerer [5] and eight by Briand et al [6].

In [7], the relationships between existing object-oriented coupling, cohesion, and inheritance measures and the probability of fault detection in system classes during testing explored empirically. Their univariate regression analysis have shown that many coupling and inheritance measures are strongly related to'the probability of fault detection in a class. Their multivariate'regression 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. In order to draw more general conclusions and confirm the, results, they conducted the same study again using an industrial system developed by professional in [8].

L. Briand et al. [9] built a fault-proneness prediction model based on a set of OO measures using data collected from a mid-sized Java system using logistic regression analysis, and then applied the model to a different Java system developed by' the same team. They then evaluated the accuracy the model's prediction in that system and the model's economic viability using a cost-benefit model.

In [10], empirical study was performed with the data from a commercial Java appliéation using logistic regression. They found that Depth of Inheritance Tree (DIT) is a good measure of familiarity and, has a quadratic relationship with fault-proneness. Their hypotheses were confirmed for Import Coupling to other classes, Export Coupling and Number of Children metrics. The Ancestor based Import Coupling metrics were not associated with fault-proneness after controlling for the confounding effect of DH.

Yida Mao, H.A. Sahraoui and H. Lounis [11] proposed an experiment to verify three hypotheses about the impact of three internal characteristics (inheritance, coupling and complexity) of OO applications on reusability. That verification was done through a machine-leaming approach **C4.5** [12]. To verify the hypotheses, they used six inheritance metrics, 18 design coupling metrics, 12 code coupling metrics and 13 complexity metrics. The results of their experiment show that the selected metrics can predict with a high level of accuracy the classes that can be potentially reusable. The four predictive models obtained give a satisfactory results (74% to 89% of accuracy).

3. DESIGN OF THE EMPIRICAL STUDY

The objective of this study is to estimate how many faults are remaining in the programs. The use of object-oriented software development techniques add new elements to software complexity in the software development process and in the software product. Traditional metrics are not adequate for detecting object-oriented faults. Object-oriented faults that are strongly related to the OO features and are introduced by these features such as inheritance and polymorphism. The object management faults that are related to object management such as object copying, dangling reference, object memory usage faults and so on [1]. Our neural network model aims to predict the number of all kinds of faults including object-oriented faults. We used software metrics including both object-oriented metrics and traditional complexity software metrics. In this study, we use software metrics concerning with inheritance related measures, complexity measures, coupling measures and memory allocation measures. To detect software faults, we selected the following metrics, which are defined by Mei-Huei Tang et al. [1], Chidamber and Kemerer [5].

3.1. Inheritance related measures

Inheritance is a kind of relationship among classes that enables programmers to reuse previously defined objects, including variables and operations. Inheritance decreases complexity by reducing the number of operations and attributes, but this abstraction of objects can make maintenance and design difficult. The following metrics are used to measure the depth and breadth of the inheritance hierarchy.

- 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.

- 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.

3.2. Coupling measures

Coupling metrics measure the degree of inter dependence among the components of a sofiware system. High coupling makes a system more complex; highly interrelated modules are harder to understand, change or correct. By minimize coupling, propagating errors across modules can be avoided. These metrics are as follows:

- Coupling Between Objects (CBO) is defined as the number of other classes to which it is coupled.
- 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.
- 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 class.
- Coupling Between Methods (CBM) provides the total number of new/redefmed methods to which all the inherited methods are coupled. CBM measures the total number of function dependency relationships between the inherited methods and new/redefmed methods.

3.3. Complexity measures

In this study, the following metrics are used to evaluate the complexity of a class.

- Weighted Methods per Class (WMC) is defined as being the number of all member functions and operators defined in each class.
- Average Method Complexity (AMC) provides the average method size for each class.

3.4. Object/memory allocation measures

A class with more obj eat/memory allocating activities tends to introduce more the object management faults that are related to object management such as object copying, dangling reference, object memory usage faults and so on.

- Number of Object/Memory Allocation metric measures the total number of statements that allocates new object or memories in a class.

4. ElVIPHERICAL RESULTS

The applications used in this study are three subsystems of an HMI (Human Machine Interface) sofiware, which is a fully networked Supervisory Control and Data Acquisition system [1]. Subsystem A is a user interface-oriented program that consists of 20 classes. Subsystem B is real-time data logging process that defines 48 classes. Subsystem C is a communication-oriented program that defines 29 classes.

We divided the data into training set, test set and production set. We extract 19 patterns of the total patterns to make the test set to prevent over training network so they will generalize well on new data and 19 patterns to make the production set to be used to test the network's results with the data the network has never seen before. The remainder 59 patterns are set for the training set.

The Ward Network [17] is used in this investigation. It is a Backpropagation network that has three slabs in the hidden layer. Hidden layers in neural network are known as feature detectors. A slab is a group of neurons. When each slab in the hidden layer has a different activation function, it offers three ways of viewing the data. We use linear function to the output slab. It is useful for our problem where the output is a continuous variable. It does not represent categories. Although the linear function detracts from the power of the network somewhat, it sometimes prevents the network from producing outputs with more error near the min or max of the output scale. In other words the results may be more consistent throughout the scale with smaller learning rates, momentums, and initial weight sizes. We use hyperbolic tangent (tanh) function is used in one of the slabs of hidden layer because it is better for continuous valued outputs especially if the linear function is used on the output layer. Gaussian function is used in one of the slabs of hidden layer. 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. Gaussian Complement is used in one of the slabs of hidden layer to bring out meaningful characteristics in the extremes of the

data. The learning rate and momentum are set to 0.1 and initial weight is set to 0.3.

To measure of the goodness of fit of the model, we use the coefficient of multiple determination (R—square), the coefficient of correlation(r), r-square, mean square error, mean absolute error, minimum absolute error and maximum absolute error.

Coefficient of correlation (r) (Pearson's Linear Correlation Coefficient) is a statistical measure of the strength of the relationship between the actual versus predicted outputs. The r coefficient can range from -1 to +1. The closer r is to 1, the stronger the positive linear relationship, and the closer r is to –1, the stronger the negative linear relationship. When r is near 0, there is no linear relationship.

$$r = \frac{s_{xy}}{\sqrt{s_{xx} \cdot s_{yy}}}$$

$$s_{xx} = \sum x^2 - \frac{1}{n}(\sum x)^2$$

$$s_{yy} = \sum y^2 - \frac{1}{n}(\sum y)^2$$

s„y =Z勃一豊(Z篇)(Z刀

Where n equals the number of patterns, x refers to the set of actual outputs, and y refers to the predicted output.

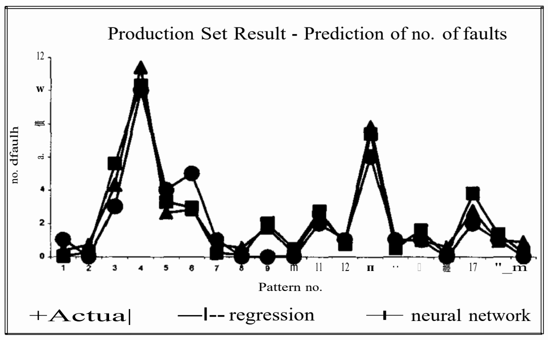
Coefficient of multiple determination (R2) is a statistical indicator. It compares the accuracy of the model to the accuracy of a trivial benchmark model wherein the prediction is just the mean of all of the samples. It can range from 0 to 1, and a perfect fit would result in an R-square value of 1, a very good fit near 1, and a very poor fit less than 0. It is calculated as follows:

$$R^2 = 1 - \frac{\sum (y-\hat{y})^2}{\sum (y-\bar{y})^2}$$

where y is the actual value for the dependent variable, $\hat{y}$ is the predicted value of y and $\bar{y}$ is the mean of the y values.

Table 1. Analysis result from two models

	Regression	Neural Network
R-square	0.806521172	0.943334217
r correlation câefflckæt)	0.91861	0.93667
r- square	0.84384	0.877346
Mean square error	1.2627	1.0419
Mean absolute error	0.854763	0.787345
Min absolute error	0.10609	0.010182
Max absolute error	2.57276	2.1703



[Figure 1. Prediction results of neural network model and regression model

Table 1 shows these values that are obtained from the regression model and neural network model for the production set. Figure 1 displays the comparison of prediction result from these models.

5. CONCUSIONS

This empirical study presents the prediction of faults in three industrial real-time systems using a multiple regression model and a neural network model. From the results presented above, object-oriented metrics appear to be useful to predict the number of faults. Moreover, neural network model can predict the number of faults more exactly than multiple regression model for sofiware engineering data.

6. ACKNOWLEDGMENTS

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 [1].

7. REFERENCES

[1] Mei-Huei Tang, Ming-ng Kao, Mei-Hwa Chen,"An empirical study on object-oriented metrics", Proceedings of the Sixth IEEE International Symposium on Software *Metrics,* pp. **242-249, 1999.**

[2] Mark Lorenz, Jeff Kidd, Object-Oriented Software *Metrics,* Prentice **Hall** Object-Oriented Series, 1994.

[3] T. Kamiya, Kusumoto, S., K. Inoue, "Prediction of fault-proneness at early phase in object-oriented development", Proceedings of the Second IEEE International Symposium on Object-Oriented ***Real-Time*** Distributed Computing, pp., 253-258, 1999.

[4] El Emam, W. Melo, 'The Prediction of Faulty Classes Using Object-Oriented Design Metrics", *Journal* of Systems and Software, Elsevier **Science,** 2001. (In press.). NRC **44178.** Also published as Technical Report, NRC/ERB-1064, November 1999. 24 pages. NRC **43609.**

[5] S.R. Chidamber, and C.F, Kemerer, "A Metrics Suite for Object Oriented Design", IEEE *Transactions* on *Sofware* Engineering, vol. 20, pp. 476-493, 1994.

[6] Lionel **Briand,** Prem Devanbu and Walcelio Melo, "An investigation into coupling measures for **C++-",** *Proceedings* of the 19th International Conference on *Software Engineering,* pp. **412-421, 1997.**

[7] L. Briand, J. **Wüst,** John W. Daly and V. Porter, "Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems", Journal of Systems and ***Sofiware,*** 51(2000) p 245-273.

[8] L.C. Briand, J. Wüst, S. lkonomovski and H. Lounis, "Investigating Quality Factors in Object-Oriented Designs: an Industrial Case Study", Proceedings of the 21st IEEE International Conference on Sofiware Engineering ICSE'99, pp. 345-354, 1999.

[9] Lionel C. Briand, Walcelio Melo and Juergen Wuest, "Assessing the Applicability of Fault-Proneness Models Across Object-Oriented Software Projects", International Software Egineering Research Network (ISERN), 2000, SERN-00-06 Version 2. Accepted for publication in IEEE Tansactions on Sofiware Engineering.

[10] D. Glasberg, K. El Emam, W. Melo, and N. Madhavji, "Validating Object-oriented Design Metrics on a Commercial Java Application". Technical Report, NRC/ERB-1080, NRC 44146, 2000.

[11] Yida Mao, H.A. Sahraoui, and H. Lounis, "Reusability hypothesis verification using machine learning techniques: a case study", Proceedings of the 13th IEEE International Conference on Automated Software Engineering, pp. 84-93, 1998.

[12] Quinlan, J. R., ***C4.5*** *: programs* for machine learning, Morgan Kaufinann Publishers, San Mateo, Calif., 1993.

[13] T. M. Khoshgofiaar, A.S. Pandya, and H.B. More, "A neural network approach for predicting sofiware development faults", Proceedings of Third International Symposium on Software Reliability Engineering, pp. 83-89,1992.

[14] N. Karunanithi et al., "Prediction of software reliability using neural networks", Proceedings IEEE International Sym. Sofiware Reliability Engineering, pp 124-130, 1991.

[15] [T.M. Khoshgoftaar, E.B. Allen, J.P. Hudepohl, and S.]. Aud, "Application of neural networks to sofiware quality modeling of a very large telecommunications system", IEEE Transactions on Neural Network, vol. 8, pp. 902-909, 1997.

[16] T.M. Khoshgofiaar, R.M. **Szabo** and **P.].** Guasti, "Exploring the behaviour of neural network software quality models", Software Engineering Journal, vol. 10 , pp. 89-96, 1995.

[17] NeuroShell 2 Help, Ward Systms Group, Inc.