Prediction of Software Readiness Using Neural Network

Jon T.S. Quah, Mie Mie Thet Thwin

Abstract-- **In this paper, we explore the behaviour of neural network in predicting software readiness. Our neural network model aims to predict the number of faults (including object-oriented faults) of a software under development. We use Ward neural network that is a backpropagation network with different activation functions. Different activation functions are applied to hidden layer slabs to detect different features in a pattern processed through a network. In our experiments, hyperbolic tangent, Gaussian, Gaussian-complement and linear functions are used as activation functions to improve prediction. This paper also compares the prediction results from multiple regression model and neural network model. Object-oriented design metrics are used as the independent variables in our study. Our study is conducted on three industrial real-time systems that contain a number of natural faults that has been reported over a period of three years.**

***Index Terms*—Neural Networks, Object-oriented Design Metrics, QA in Software Development, Readiness, Reliability**

I. Introduction

One of the critical questions of a software company is "Is the software ready to release now?" The answer should be based on the number of remaining errors and required number of changes in applications programs. Not only would a reasonable estimate of this helps one in determining when to stop testing, but also it could be used to estimate the maintenance costs after the program is placed into production, and also to estimate the program's reliability.

Khoshgoftarr et al. [17] introduced the use of the neural networks as a tool for predicting the number of faults in programs. They compared the neural-network model with a nonparametric disciminant model, and found the neural network model had better predictive accuracy. However, since their model used domain metrics derived from the complexity metric data, and such metrics are not adequate for detecting object-oriented faults, their model cannot be applied to object-oriented programs. Since the object-oriented paradigm exhibits different characteristics from the procedural paradigm, software metrics in object-oriented paradigm need to be used.

Our neural network model is aimed to predict the number of faults including object-oriented faults. We used software metrics including both object-oriented metrics and traditional complexity software metrics. They can be broadly classified into metrics concerning with inheritance related measures, complexity measures, coupling measure and memory allocation measure.

We use Ward Network for our experiments. It is a backpropagation neural network with different activation functions being applied to hidden layer slabs to detect different features in a pattern processed through a network to lead to a better prediction. The network design use a Gaussian function in one hidden slab to detect feature in the mid-range of the data and use a Gaussian complement in another hidden slab to detect features for the upper and lower extremes of the data. Thus, the output layer will get "different views of the data". Combining the two feature sets in the output layer may lead to a better prediction. Finally, we perform a comparative study using result obtain from neural network modeling and those from multiple regression models. The former is found to outperform the latter.

II. Related Work

A. Object-Oriented Metrics

There is great interest in the use of object-oriented approach to 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 practice in object-oriented design and development [6]. Many measures have been proposed (and validated using statistical methods) in the literature to capture the structural quality of object-oriented (OO) code and design [3]-[5], [8], [9], [21].

One such set of metrics is the set proposed by Chidamber and Kemerer [8]. Six metrics were proposed. A later paper refined the metrics and collected empirical data at two sites, and also presented the empirical distributions [9]. Chidamber and Kemerer (CK) also reported empirical data from two commercial organizations and suggest ways in which the metrics can be used to manage OO design efforts [9].

In [4], Lionel Briand, Prem Devanbu and Walcelio Melo proposed a comprehensive suite of measures to quantify the level of class coupling during the design of object-oriented systems. That suite has taken into account the different OO design mechanisms provided by the C++ language. They proposed eighteen OO design metrics.

Saida Benlarbi and Walcelio L. Melo[2] defined and empirically investigated the quality impact of polymorphism on OO design. They validated their measues by evaluating

their impact on class fault-proneness, a software quality attribute.

Wei Li and Sallie Henry [20] have carried out the statistical analyses of a prediction model incorporating CK metrics. Their result has shown that there is a strong relationship between metrics and maintenance effort in object-oriented systems.

Basili et al. have reported on results of using the CK metrics suite to predict the quality (fault proneness) of student C++ programs [1].

Lionel C. Briand et al. [6] have empirically explored the relationship between existing object-oriented coupling, cohesion, and inheritance measures and the probability of fault detection in system classes during testing.

Mei-Huei Tang, Ming-Hung Kao and Mei-Hwa Chen [23] proposed a set of new metrics: Inheritance Coupling (IC), Coupling Between Method (CBM), Number of Object/Memory Allocation (NOMA) and Average Method Complexity (AMC) to identify the dynamic behavior of the program and the scenarios that the instances of the classes are referenced in the program. Emanm et. al.[10],[11] have constructed a model to predict which classes in a future release of a commercial Java application will be faulty.

L. Briand et al. [7] built a fault-proneness prediction model based on a set of OO measures using data collected from a mid-sized Java system, 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.

B. Neural Network Model

The usefulness of connectionist models for software reliability growth prediction was illustrated in [14], [15]. In [14], the applicability of the connectionist approach was explored using various network models, training regimes, and data representation methods. An empirical comparison was made between that approach and five well-known software reliability growth models using actual data sets from several different software projects. Their result showed that neural networks could be used for predicting software reliability growth.

SungBack Hong and Kapsu Kim [12] studied based on hybrid metrics which include internal and external module complexity. Using the neural network model, they classified function block as either fault-prone or not fault prone based on the proposed metrics. They can predict whether the newly added function block is fault prone or not fault prone.

Khoshgoftaar et al. [16] 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 neural network is a promising technique for building the predictive models, because it is able to model nonlinearities in relationship.

III. DESIGN OF THE EMPIRICAL STUDY

A. Dependent and Independent Variables

The objective of this study is to estimate the software readiness using neural network model. To estimate the readiness, how many faults are remaining in the programs is considered in this study. The number of faults is used as dependent variable and the measures of coupling, inheritance, complexity, object/memory allocation are used as independent variables.

1) Inheritance related measures

A class hierarchy is the result of the inheritance relationship between classes of a system. Inheritance, which is emphasized in OO programming languages, is a binary, asymmetric connection between two classes. Class inheritance enables methods of one class to be accessed by its subclasses. As a result of inheritance, classes with more general attributes and methods are usually located in the upper part of the hierarchy whereas classes dealing with specific operations are consequently lowered down to the hierarchy. In our study, the following two metrics are considered.

- Depth of Inheritance Tree (DIT). The DIT of a class is the length of the longest path from the class to the root in the inheritance hierarchy.
- Number of Children (NOC). The NOC metric measures the number of immediate descendants of a particular class.

2) Coupling measures

Coupling refers to the degree of inter dependence among the components of a software system. Good software design should obey the principle of low coupling. Strong coupling makes a system more complex; highly interrelated modules are harder to understand, change or correct. By minimizing coupling, one can avoid propagating errors across modules. These metrics are as follows:

- Coupling Between Objects (CBO). According to the definition of this measure, a class is coupled to another if methods of one class use methods or attributes of the other, or vice versa. CBO for a class is then defined as the number of other classes to which it is coupled.
- Response For a Class (RFC). This is the number of methods that can potentially be executed in response to a message received by an object of that class.
- Inheritance Coupling (IC). The 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). The CBM provides the total number of new/redefined methods to which all the inherited methods are coupled. CBM measures the total number of function dependency relationships between the inherited methods and new/redefined methods.

3) Complexity measures

Weighted Methods per Class and Average Method Complexity are used to evaluate the complexity of an individual class.

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

4) Object/memory allocation measures

A class with more object/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 (NOMA). It measures the total number of statements that allocate new objects or memories in a class.

B. Neural Network Modeling

The Ward Network[22] is used in this research. It is a Backpropagation network that has three slabs (slab2, slab3 and slab4) 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 apply linear function to the output slab (slab5) as it is useful for this study where the output is a continuous variable. The output node does not represent categories. Although the linear

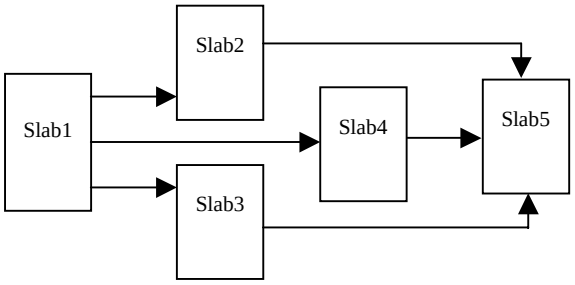

Fig. 1. Ward Neural Network.

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 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. Gaussian function is used in another slab of 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. Gaussian Complement is used in yet another slab of hidden layer (slab4) 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. Neural network summary is presented in Table I.

Table I. Neural network summary

	Number of neurons				
	Slab1	Slab2	Slab3	Slab4	Slab5
System A	9	3	3	3	1
System B	9	4	4	4	1
System C	9	3	3	3	1

IV. PREDICTION RESULT

The applications used in this study are subsystems of an HMI (Human Machine Interface) software, which is a fully networked Supervisory Control and Data Acquisition system [23]. 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.

To measure the goodness of fit of the model, we use the coefficient of multiple determination (R-squared) that is a statistical indicator. It compares the accuracy of the model to the accuracy of a benchmark model where the prediction is just the mean of all of the samples. 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 II shows the values of R-square that are obtained from the regression model and neural network model. Table III, IV and V show the comparison of prediction results of number of faults in a particular class from regression model and neural network model with the actual number of faults which are reported for three years. These comparison results are also shown in Fig. 5, Fig. 6 and Fig. 7 graphically.

Table II. R-square values for three systems

	Regression	Neural Network
System A	0.956	0.9763
System B	0.789	0.9306
System C	0.976	0.9934

Fig. 2, Fig. 3 and Fig. 4 display the contribution factor bar graphs for three neural networks. Contribution factor of a variable is a measurement of the importance of that variable in predicting the network's output, relative to the other input variables in the same network. A higher number indicates that the variable is contributing more to the prediction or classification. Obviously, if a certain variable is highly correlated with the output, the variable will have a high

contribution factor. The contribution factor is developed from an analysis of the weights of the trained neural network [22].

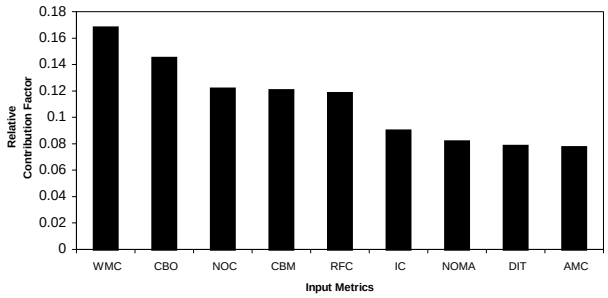

Fig. 2. Contribution factor of input metric variables in subsystemA

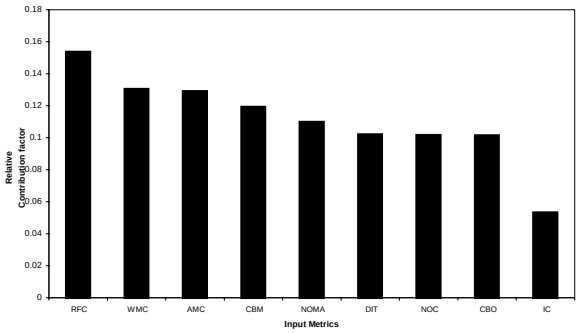

Fig. 3 Contribution factor of input metric variables in subsystemB

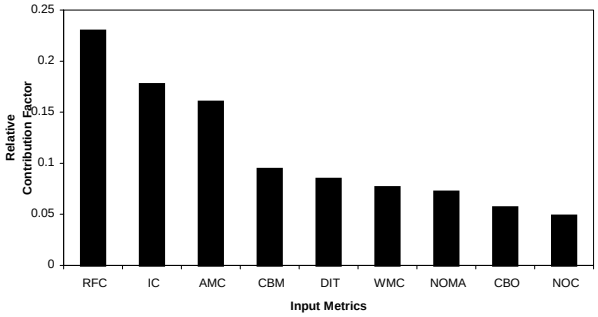

Fig. 4. Contribution factor of input metric variables in subsystemC

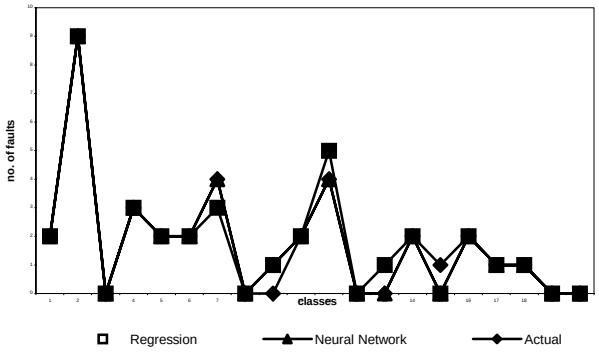

Fig. 5. Comparison chart of prediction result for subsystemA

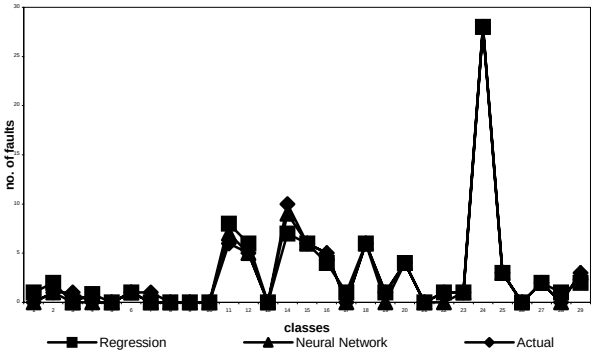

Fig. 6. Comparison chart of prediction result for subsystemB

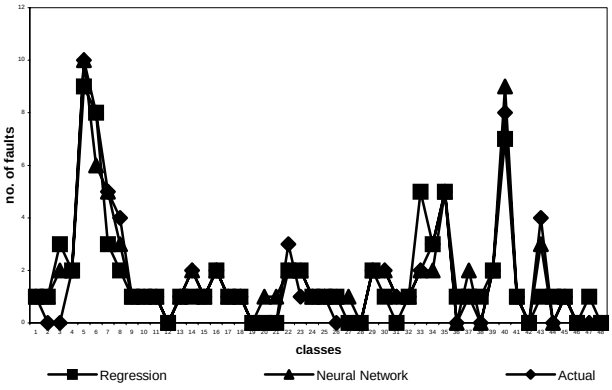

Fig. 7. Comparison chart of prediction result for subsystemC

Table III. Prediction result for Subsystem A

Class	Actual	Regression		Neural Network	
1	2	1.91414	2	2.204922	2
2	9	8.53001	9	8.815842	9
3	0	0.13821	0	0.460422	0
4	3	3.31796	3	2.977333	3
5	2	1.87382	2	2.062163	2
6	2	2.30366	2	2.249439	2
7	4	3.41991	3	4.120771	4
8	0	-0.13344	0	0.312293	0
9	0	0.72204	1	0.747336	1
10	2	2.31635	2	2.099284	2
11	4	4.67788	5	4.062977	4
12	0	-0.59464	-1	0.119285	0
13	0	0.89329	1	0.433413	0
14	2	1.83481	2	2.107817	2
15	1	0.19308	0	0.260461	0
16	2	1.7856	2	1.850852	2
17	1	0.77365	1	0.813616	1
18	1	1.02925	1	1.067508	1
19	0	0.08047	0	0.277788	0
20	0	-0.07607	0	0.381536	0

Table IV. Prediction result for Subsystem B

Class	Actual	Regression		Neural Network	
1	1	0.57784	1	1.112232	1
2	0	0.87966	1	0.956696	1
3	0	2.81226	3	1.580051	2
4	2	2.45449	2	1.969271	2
5	10	9.30176	9	10.46048	10
6	8	7.88786	8	6.371044	6
7	5	3.06333	3	5.089035	5
8	4	1.50528	2	2.526809	3
9	1	1.17016	1	1.468316	1
10	1	0.53266	1	1.090944	1
11	1	1.09188	1	1.071907	1
12	0	0.39878	0	0.210687	0
13	1	0.77658	1	0.816	1
14	2	1.3464	1	1.554224	2
15	1	1.46517	1	0.792669	1
16	2	1.743	2	2.439447	2
17	1	0.58171	1	1.019993	1
18	1	0.80362	1	1.128049	1
19	0	0.38184	0	0.203242	0
20	0	0.39028	0	0.724395	1
21	0	0.32138	0	0.689982	1
22	3	2.42602	2	2.430226	2
23	1	1.71811	2	1.951091	2
24	1	0.94787	1	1.025429	1
25	1	1.11917	1	0.929054	1
26	0	0.99425	1	0.73153	1
27	0	0.48799	0	0.808253	1
28	0	0.37497	0	-0.12097	0
29	2	2.42924	2	2.124755	2
30	2	0.67527	1	1.958644	2
31	1	0.47276	0	0.936804	1
32	1	0.53935	1	1.160908	1
33	2	4.95835	5	2.196496	2
34	3	2.57792	3	2.218626	2
35	5	5.34077	5	4.74082	5
36	0	0.8026	1	-0.1157	0
37	1	0.53154	1	1.547117	2
38	0	0.53421	1	0.239422	0
39	2	2.05324	2	1.918721	2
40	8	6.7335	7	8.541753	9
41	1	0.79394	1	1.255774	1
42	0	-0.39619	0	0.150993	0
43	4	0.7219	1	2.614042	3
44	0	0.78841	1	-0.30424	0
45	1	1.35783	1	1.462041	1
46	0	0.48619	0	0.023016	0
47	0	0.54848	1	-0.24107	0
48	0	0.49637	0	-0.20256	0

V. Conclusion and Future Work

One of the easiest ways to judge whether a program is ready to be released is to measure its fault density, which is the number of faults per line of code. To decide whether new software version is reliable enough to ship, the estimated fault density is compared with the identified faults that have been detected for the new version of a software.

This paper presents a multiple regression model and a neural network model for predicting the number of faults in a particular class using three industrial real-time subsystems data. From the results shown above, object-oriented metrics appear to be useful to predict the number of faults. Moreover, neural network model can predict more accurately than regression model.

Our future research direction aims to estimate the software readiness using neural network model. To estimate the readiness, three factors will be considered in our future study: (1) how many faults are remaining in the programs (2) how much changes to correct the errors and (3) amount of time required to change the programs. Input to be used for modeling the neural network includes software metrics on polymorphism measures, inheritance related measures, complexity measures, cohesion measures, coupling measure, dynamic memory allocation measure, database operations measures and size measures.

Table V. Prediction result for Subsystem C

Class	Actual	Regression		Neural Network	
1	0	0.51228	1	0.317737	0
2	1	2.03362	2	0.614106	1
3	1	0.39443	0	0.472311	0
4	0	0.8373	1	0.338698	0
5	0	0.03332	0	0.286367	0
6	1	0.70069	1	0.625033	1
7	1	0.43744	0	0.453838	0
8	0	-0.42252	0	0.1502	0
9	0	0.12329	0	0.066168	0
10	0	0.15139	0	0.046733	0
11	6	7.77116	8	6.73549	7
12	5	6.17734	6	4.93935	5
13	0	0.29568	0	-0.16793	0
14	10	6.92235	7	8.649171	9
15	6	5.99938	6	5.622892	6
16	5	4.26267	4	4.546377	5
17	0	0.95936	1	0.424839	0
18	6	5.614	6	5.752192	6
19	1	0.58141	1	0.419876	0
20	4	4.18052	4	4.208034	4
21	0	0.03838	0	0.355994	0
22	1	0.57634	1	0.351029	0
23	1	0.67345	1	0.669363	1
24	28	28.13152	28	27.7595	28
25	3	2.7853	3	2.733317	3
26	0	0.28628	0	0.249484	0
27	2	1.90343	2	1.7681	2
28	0	0.75768	1	0.170828	0
29	3	2.2824	2	3.00577	3

ACKNOWLEDGMENT

The authors would like to thank Associate Professor Dr. Mei-Hwa Chen, Computer Science Department, University at Albany, State University of New York, for sharing their industrial real time operations data [23].

REFERENCES

[1] V.R. Basili, et al., "A Validation of Object-Oriented Design Metrics as Quality Indicators," IEEE Transactions on Software Engineering, vol. 22, pp. 751-761, 1996.

[2] S. Benlarbi, W.L Melo, "Polymorphism measures for early risk prediction," Proceedings of the 1999 International Conference on Software Engineering, pp. 334-344, 1999.

[3] J.M. Bieman, B.K. Kang, "Cohesion and Reuse in an Object-Oriented System," Proceeding of ACM Symposium on Software Reusability (SSR'94), 259-262.

[4] L. Briand, P. Devanbu, and W. Melo, "An investigation into coupling measures for C++", Proceedings of the 19th International Conference on Software Engineering, pp. 412-421, 1997.

[5] L. Briand, J. Daly, J. Wuest, "A Unified Framework for Coupling Measurement in Object-Oriented Systems," IEEE Transactions on Software Engineering, 1999.

[6] L. Briand, J. Daly, V. Porter, and J. Wuest, "Predicting Fault-Prone Classes based on Design Measures in Object-Oriented Systems," IEEE International Symposium on Software Reliability Engineering (ISSRE), 1998.

[7] L.C. Briand, W.L. Melo, J. Wust, "Assessing the applicability of fault-proneness models across object-oriented software projects," IEEE Transactions on Software Engineering, vol. 28, Issue: 7, pp. 706 –720, Jul. 2002.

[8] S.R. Chidamber, and C.F. Kemerer, "Towards a Metrics Suite for Object Oriented Design," Proceeding of the 6th ACM Conference on Object Oriented Programming, Systems, Languages and Applications, pp. 197-211, 1991.

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

[10] El Emam, W. Melo, J.C. Machado, "The Prediction of Faulty Classes Using Object-Oriented Design Metrics," Journal of Systems and Software, vol. 56, pp. 63-75, Feb. 2001.

[11] D. Glasberg, 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.

[12] S. Hong, and K. Kim, **"**Identifying fault-prone function blocks using the neural networks - an empirical study," Communications, Computers and Signal Processing, 1997. 10 Years PACRIM 1987-1997, IEEE Pacific Rim Conference on Networking the Pacific Rim, vol. 2, pp. 790-793, 1997.

[13] S. Hong, and K. Kim, **"**Identifying fault prone modules: an empirical study in telecommunication system," Proceedings of the Second Euromicro Conference on Software Maintenance and Reengineering, 179-183, 1998.

[14] N. Karunanithi, D. Whitley, and Y.K. Malaiya, "Prediction of software reliability using connectionist models**,"** IEEE Transactions on Software Engineering, vol. 18, 563-574, 1992.

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

[16] T. M. Khoshgoftaar, E.B. Allen, Z. Xu, "Predicting testability of program modules using a neural network", Proceedings of the 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology, pp. 57-62, 2000.

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

[18] T. M. Khoshgoftaar, D.L. Lanning, and A.S. Pandya, "A Comparative Study of Pattern Recognition Techniques for Quality Evaluation of Telecommunications Software," IEEE Journal on Selected Areas in Communications, vol. 12, pp. 279-291, 1994.

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

[20] W. Li, and S. Henry, "Object-Oriented Metrics that Predict Maintainability," Journal of Systems and Software, vol. 23, pp. 111-122, 1993.

[21] H. Lounis, and W. Melo, "Identifying and Measuring Coupling on Modular Systems," Procedings of the 8 th International Conference on Software Technology, 1997.

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

[23] Mei-Huei Tang, Ming-Hung 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.