Adaptive Self-Organizing Map Clustering for Software Fault Prediction

Atchara Mahaweerawat*,+, Peraphon Sophatsathit+, Chidchanok Lursinsap+
*Department of Mathematics, Statistics and Computer Science, Faculty of Science
Ubonratchathani University, Ubonratchathani 34190, Thailand
matchara@sci.ubu.ac.th
+Advanced Virtual and Intelligent Computing Center (AVIC)
Department of Mathematics, Faculty of Science
Chulalongkorn University, Bangkok 10330, Thailand
peraphon.s@chula.ac.th, lchidcha@pioneer.netserv.chula.ac.th

***Abstract*— This paper presents a new approach for predicting software faults by means of two-level clustering with unknown number of clusters. We employed Self-Organizing Map method and our proposed clustering approach in the first and second level, respectively, to classify historical and development data into clusters. Next we applied the Radial-Basis Function Network to predict software faults occurred in cluster components. In so doing, we were able to predict software faults reasonably accurate.**

I. INTRODUCTION

Software reliability is the probability of failure free operations of a computer program executing in a specified environment for a specified time[1]. Software reliability is considered a software quality factor that aids in predicting software quality using standard predictive models. There are many approaches for predicting software quality, most of which yield some forms of quality indicators. One popular indicator is known as software fault. Software fault prediction utilizes historical and development data to arrive at a conclusive decision whether the software in question is at fault.

Recent researches [2], [3], [4], [5] in neural networks and statistics proliferate various techniques and algorithms to enhance the accuracy and efficiency of fault prediction considerably. This is due to the fact that myriad of data are collected as a result of software reliability study, some preprocessing may be required to classify those representing data in clusters such that more accurate fault patterns can be extracted from each cluster.

Our definition of a cluster is a group of entities with similar properties [6]. Thus, a set of points can be grouped into one or more clusters, depending on the properties of those points. Usually, the simplest and obvious property is distance among the points. Therefore, a qualified candidate is determined by the distance between the point in question and a neighboring point which must be smaller than the distance to the rest of the points.

In neural networks, there are techniques for data clustering such as Self-Organizing Map (SOM) and Radial-Basis Function Network (RBFN). The SOM technique can be used as a vehicle for analysis of multidimensional data [7]. It maps high dimensional data into low dimensional (usually two-dimensional) space, whereby the structure of the original data set can be visualized to identify potential clustering. However, as the number of SOM units grows, it is too time-consuming to continue with SOM carrying out quantitative analysis of data sets. The variation among some smaller units may not be significant enough to warrant their autonomy. As a consequence, we proposed a new data clustering approach to overcome SOM grouping for large number of map units.

The RBF technique, on the other hand, is suitable for interpolation problem which is viewed as a curve fitting problem in high-dimensional space [8]. The learning process involved is equivalent to finding a surface in a multidimensional space that provides the best fit to the training data, with the criteria for 'best fit' being measured in some statistical sense. Thus, we employed the SOM technique, incooperation with the proposed clustering approach, and the RBF technique to predict software fault for this study.

Based on the aforementioned framework, we constructed a model in our experiment to predict the number of software faults ranging from zero to seven inclusive. The experimental data were preprocessed based on a number of software metrics, namely, lines of code (LOC), non-commented lines of code ($NCLOC$), Halstead program length (N), Halstead volume (V), McCabe cyclomatic complexity ($V(G)$), Halstead number of unique operands (n_2), Halstead total number of operands (N_2), Henry&Kafura fan-in (fan_{in}) and fan-out (fan_{out}), Henry&Kafura information flow (IF), and density of comments (DC). These software metrics can be derived by the followings:

- McCabe cyclomatic complexity (V(G)) [9]
  $V(G) = e - n + 2p$
  where
  $V(G) =$ cyclomatic number of G
  $e =$ number of edges
  $n =$ number of nodes
  $p =$ number of unconnected paths of the graph
- fan_{in}: the fan_{in} of a module M is the number of local flows that terminates at M, plus the number of data structures from which information is retrieved by M [10]
- fan_{out}: the fan_{out} of a module M is the number of

local flows that emanates from M, plus the number of data structures that is updated by M [10]
- The information flow (IF) is measured by the formula [10] $IF = (fan_{in} \times fan_{out})$
- Halstead's software science [9] is defined as
  Lenght(N) $= N_1 + N_2 = n_1 log_2(n_1) + n_2 log_2(n_2)$
  Volumn(V) $= N log_2(n) = N log_2(n_1 + n_2)$
  where
  $n_1 =$ the number of distinct operators that appear in a program
  $n_2 =$ the number of distinct operands that appear in a program
  $N_1 =$ the total number of distinct operators occurrences
  $N_2 =$ the total number of distinct operands occurrences

- Density of comments (DC) [10]
  $DC = CLOC/LOC$
  where CLOC is the number of comment lines of program text.

We classified the data into several groups using the SOM technique and the proposed approach. These groups of data were then fed into the models for software fault prediction. The results were analyzed to find some fundamental factors that contributed to fault occurrence.

This paper is organized as follows. Section II explains the concepts and the algorithm of Self Organizing Map. Section III discusses an approach for data clustering technique after applying the SOM method. Section IV derives the radial-basis membership function. Section V describes how to construct the fault preditive model by employing Radial-Basis Function Network and the analysis of the result so obtained. Some final remarks are given in Section VI.

II. Self Organizing Map (SOM)

A. Concepts

There are two aspects involved in SOM method, namely, visualization and abstraction [11]. The first aspect offers a powerful tool to visualize high-dimensional data by simple geometric relationships on low-dimensional display. The other aspect compresses information yet preserving the most important topological and metric relationships of the primary data elements. From the theoretical standpoint, SOM is a neural network algorithm that maps nonlinear projection characteristics of high-dimensional space of sensory input signals onto a low-dimension array of neurons [11]. The goal of SOM method is to detect features from input space by transforming an incoming signal pattern of arbitrary dimensions to a one- or two-dimensional discrete map [8].

The SOM network consists of a 2-D matrix of neurons. Each neuron is identified by its location and weight vector [7]. The network is based on competitive learning, i.e., *the output neurons of the network compete among themselves to be activated or fired, with the result that only one output neuron, or one neuron per group, is on at any time*. All the output neurons that win the competition are called *winner-take-all* neurons [8].

B. The SOM Algorithm

The SOM algorithm may be described as a nonlinear, ordered, and smooth mapping of high-dimensional input data domain onto elements of a regular, low-dimensional array [11]. The mapping process can be explained by the SOM algorithm as follows:

Let

$W_{i,j} \in \Re^n$ be the weight vector of the neuron with coordinate (i, j)
where $i = 1, 2, \ldots, COL$
$j = 1, 2, \ldots, ROW$
COL is the total columns
ROW is the total rows in the map

$X_k \in \Re^n$ $k = 1, 2, \ldots, p$ be the input pattern
where p is total input patterns

W_{i_0,j_0} be the weight vector of the winning neuron

1) Initialize the weight vectors with small arbitrary values
2) Apply an input pattern to the network
3) Find the winning neuron according to the equation

$$\|X(t) - W_{i_0,j_0}(t)\| = min_{i,j} \|X(t) - W_{i,j}(t)\| \quad (1)$$

   where t is the iteration number
4) Adjust weight vectors of the winning neuron and the neighbor neurons

$$W_{i,j}(t+1) = W_{i,j}(t) + \alpha(t) N(i, j, i_0, j_0, t)[X(t) - W_{i,j}(t)] \quad (2)$$

   where
   $\alpha(t)$ is the learning rate for iteration t
   $N(i, j, i_0, j_0, t)$ is the neighborhood function
5) Repeat steps 2-4 until all input patterns are used
6) Repeat steps 2-5 for the number of epochs

Network training is carried out by repeating the above process for the predefined number of epochs.

III. Clustering of the Self Organizing Map

Conventional SOM technique employs the same predefined number of weight vectors or neuron units as the number of clusters to organize group mapping. Nevertheless, if the number of clusters is not known prior to organizing process commences, the U-matrix display [12], [7], [13] may yield blurry boundaries that are difficult to determine the exact number of clusters. As a consequence, we incorporated the two-level SOM approach [14] using nearly equi-numbered neuron-to-data units to algorithmically adjust the data grouping. We considered a widely adopted optimal clustering definition which is a partition minimizing the distance among the data

points within the cluster, as well as maximizing the distance between clusters [14]. The within-cluster and between-clusters distances can thus be defined in several ways as depicted in the table I [14] shown below.

TABLE I
CLUSTER DISTANCE SPACING

Within-cluster distance	$S(Q_k)$ and $x_j \in Q_l, k \neq l$
Between-clusters distances	$d(Q_k, Q_l); x_i, x_{i'} \in Q_k$, $i \neq i'$,
N_k	is the number of samples in cluster Q_k
and	$c_k = 1/N_k \Sigma_{x_i \in Q_k} x_i$

Within-cluster distance	$S(Q_k)$
average distance	$S_a = \frac{\Sigma_{i,i'} \left\|\left\| x_i - x_{i'} \right\|\right\|}{N_k(N_k-1)}$
nearest neighbor distance	$S_{nn} = \frac{\Sigma_i min_{i'} \left\{ \left\|\left\| x_i - xi' \right\|\right\| \right\}}{N_k}$
centroid distance	$S_c = \frac{\Sigma_i \|x_i - c_k\|}{N_k}$

Between-cluster distances	$d(Q_k, Q_l)$
single linkage	$d_s = min_{i,j} \{\|x_i - x_j\|\}$
complete linkage	$d_{co} = max_{i,j} \{\|x_i - x_j\|\}$
average linkage	$d_a = \frac{\Sigma_{i,j} \|x_i - x_j\|}{N_k N_l}$
centroid linkage	$d_{ce} = \|c_k - c_l\|$

We opted for Euclidean norm to define the distance norm $\|\cdot\|$ due to its popularity with SOM.

Many clustering methods such as k-means and Isodata method [15] need to know the number of clusters in advance to properly group input data. Our approach however, does not require a predefined cluster number. The proposed approach uses the average distance among member points in the same cluster and average number of linkages between-cluster distance to determine the number of clusters required. We established a pair-wise distance matrix of all weights and data points for subsequent computations look up. The algorithm for computing cluster number is as follows:

1) Calculate the distance between each weight unit and other weight units according to:

$$d_{(p,q)} = \|W_p - W_q\| \tag{3}$$

where

W_p is weight unit p
W_q is weight unit q
$d_{(p,q)}$ is the distance between weight units p and q

2) For each weight unit, find another weight unit which has minimum distance from itself to the weight unit under consideration. This can be done by looking up the weight unit which satisfies equation below.

$$\|W_p - W_{pmin}\| = min_{q=1}^{n} \|W_p - W_q\| \tag{4}$$

where

W_{pmin} is the weight unit with minumum distance from weight unit p to itself
n is the number of all other weight units

3) Define a threshold parameter (d_{mem}). This threshold is the maximum distance in the pair-wise distance matrix that is used in cluster member computation [16].
4) Find a very close pair of weight units satisfying this condition:

IF *W_p has minimum distance from itself to W_q*

AND *W_q has minimum distance from itself to W_p*

AND $d_{(p,q)} \leq d_{mem}$

THEN *(W_p, W_q) is a very close pair of weight units*

The result is a set of clusters, each of which containing two (at most 3 in case of a tie) points with the smallest average distance, that could be subsequently merged.
5) Merge two clusters to one. A cluster can be merged to another cluster if the average distance between members of both clusters is not greater than the member threshold value.

IF $\frac{\Sigma_{a,b} \|W_a - W_b\|}{n_A \times n_B} \leq d_{mem}$

THEN *Cluster A and Clust B can be merged to one cluster*

where

W_a is the weight unit from cluster A
W_b is the weight unit from cluster B
d_{mem} is the member treshold
n_A is the total members of cluster A
n_B is the total members of cluster B

6) Repeat the above cluster merging step. The above cluster merging process is repeated to combine any clusters whose average distances among the members of participating clusters is not greater than the member threshold value. The procedure terminates when no more cluster is eligible.
7) Assign the remaining weight units to all merged clusters. After the last step, if there are weight units which cannot be grouped into the clusters, each of which will be assigned to an arbitrary cluster or grouped as a new cluster based on the distance from itself to the weight unit of the existing cluster. The procedures are as follows:

a) Identify the closest weight unit (W_{uc}) of an unclassified weight unit (W_u)
b) Determine whether that closest weight unit of the unclassified weight unit is a member of any cluster. If so and the average distance from the unclassified weight unit and the members of a cluster is not greater than the member threshold value (d_{mem}), assign the unclassified weight unit to this cluster.

IF $d_{(w_u,w_{uc})} \leq d_{mem}$

AND $W_{uc} \in C_A$

AND $\frac{\Sigma_a \|W_u - Wa\|}{n_A} \leq d_{mem}$

THEN $W_u \in C_A$

where

W_u	is an unclassified weight unit
W_{uc}	is a very close weight unit of W_u
W_a	is the weight unit in cluster A
$d_{(w_u,w_{uc})}$	is the distance between W_u and W_{uc}
C_A	is cluster A

8) Repeat the unclassified weight unit assignment steps until all weight units are assigned to cluster or all remaining weight units do not satisfy the above conditions.
9) Group the remainder unclassified weight units into a new cluster. The remaining unclassified weight units can be combined into the same new cluster if the average distance among them is not greater than the member threshold value.

IF $d_{(w_{u1},w_{u2})} \leq d_{mem}$

THEN *W_{u1} and W_{u2} can be combined into C_{new}*

where

W_{u1}, W_{u2}	are unclassified weight units
$d_{(w_{u1},w_{u2})}$	is the distance between W_{u1} and W_{u2}
C_{new}	is a new cluster

10) Repeat for the remaining unclassified weight unit grouping steps until all weight units get assigned to a cluster or all remaining weight units do not satisfy the above conditions.
11) Define a new cluster for each unclassified weight unit. For each unclassified weight unit, assign a new cluster to it. The number of new clusters generated in this step will be the same as the number of unclassified weight units, that is, each new cluster has only one weight unit.

Each resulting clusters will be subject to subsequent RBFN construction for test runs.

IV. Radial-Basis Function Network (RBFN)

A. The Interpolation Problem

The radial-basis function networks (RBFN) mention that the problem of curve-fitting is approximation in high dimensional spaces. In this case, the learning process is equivalent to finding an interpolating surface in the multidimensional space that provides the best fit to the training data, measured by pre-selected statistical criteria.

The curve-fitting or interpolation problem can be stated as follows:

Given a set of N different points $\{x_i \epsilon \mathcal{R}^{m_0} \,|i = 1, 2, ..., N\}$ and a corresponding set of N real numbers $\{d_i \epsilon \mathcal{R}^1 \,|i = 1, 2, ..., N\}$, find a function $F : \mathcal{R}^N \rightarrow \mathcal{R}^1$ that satisfies the interpolation condition:

$$\mathcal{F}(x_i) \quad = \quad d_i \quad i = 1, 2, \ldots, N \tag{5}$$

B. Radial-Basis Functions

The radial-basis functions (RBF) technique suggests that the interpolation function $\mathcal{F}$ should be constructed in the following form

$$\mathcal{F}(x) \quad = \quad \sum_{i=1}^{N} w_i \varphi(\|x - x_i\|) \tag{6}$$

where $\{\varphi(\|x - x_i\|) \,|i = 1, 2, \ldots, N\}$ is a set of N arbitrary (generally nonlinear) functions; radial-basis functions; and $\|\cdot\|$ is the Euclidean norm. The known data points $\{x_i \epsilon \mathcal{R}^{m_0} \,|i = 1, 2, ..., N\}$ are defined to be the centers of the radial-basis functions.

Inserting the interpolation conditions of Equation (5) in Equation (6), a set of simultaneous linear equations for the coefficients (weights) of the unknown w_i are expanded as follows:

$$\begin{bmatrix} \varphi_{11} & \varphi_{12} & \cdots & \varphi_{1N} \\ \varphi_{21} & \varphi_{22} & \cdots & \varphi_{2N} \\ \vdots & \vdots & & \vdots \\ \varphi_{N1} & \varphi_{N2} & \cdots & \varphi_{NN} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_N \end{bmatrix} \tag{7}$$

where

$$\varphi_{ji} \quad = \quad \varphi(\|x_j - x_i\|) \quad (j, i) = 1, 2, \ldots, N \tag{8}$$

Let

d	$= [d_1, d_2, \ldots, d_N]^T$ the desired output vector
w	$= [w_1, w_2, \ldots, w_N]^T$ linear weight vector
N	be the size of the training sample
Φ	denote an N-by-N matrix with element φ_{ji}

$$\Phi \quad = \quad \{\varphi_{ji} \,|(j, i) = 1, 2, \ldots, N\} \tag{9}$$

This matrix is called the *interpolation matrix* and Equation (9) can be written in compact form

$$\Phi w \quad = \quad d \tag{10}$$

The unknown weights(w) can be obtained by solving the following linear equation:

$$w \quad = \quad \Phi^+ d \tag{11}$$

where Φ^+ is the pseudo-inverse of Φ : $\Phi^+ = (\Phi^T \Phi)^{-1} \Phi^T$

This research employed the multivariate Gaussian function called *Green's function* with the following form:

$$G(x, x_i) = exp(-\frac{\|x - x_i\|^2}{2\sigma_i^2}) \quad (12)$$

C. Regularization Networks

The regularization network consists of three layers:

- *input layer* is composed of input nodes that is equal to the dimension m_0 of the input vector x,
- *hidden layer* is composed of nonlinear units that are connected directly to all nodes in the input layer,
- *output layer* consists of a single linear unit fully connected to the hidden layer.

There is one hidden unit for each data point x_i, $i = 1, 2, \ldots, N$, where N is the size of the training sample. Green's function is used as the activation function of individual hidden units. Therefore the output of the i^{th} hidden unit is $G(x, x_i)$. The output of the network is a linearly weighted sum of the outputs of the hidden units.

The regularization network models the interpolation function F as a linear superposition (linear weighted sum) of multivariate Gaussian functions whose size is equal to the number of the given sample input N:

$$F(x) = \sum_{i=1}^{N} w_i G(x, x_i) \quad (13)$$

or,

$$F(x) = \sum_{i=1}^{N} w_i exp(-\frac{\|x - x_i\|^2}{2\sigma_i^2}) \quad (14)$$

where w_i are the weights.

D. Generalized RBF Networks

In the regularization networks, the number of Green functions is equal to the number of the training examples. This causes computationally inefficient in practice, in the sense that it may require a very large number of basis functions.

In real world practical situations, finding the linear basis function weights needs to invert a very large $N \times N$ matrix which is computationally complex. To overcome this problem, the network complexity needs to be reduced to find a solution that approximates the solution produced by the regularization network. Let $F^*(x)$ be the approximated solution,

$$F^*(x) = \sum_{i=1}^{m_1} w_i \varphi(x) \quad (15)$$

where $\{\varphi(x \,|i = 1, 2, \ldots, m_1)\}$ is a new set of basis functions. Typically, the number of basis functions is less than the number of data points ($m_1 \leq N$), and w_i forms a new set of weights. Then,

$$\varphi_i(x) = G(\|x - t_i\|), \quad i = 1, 2, \ldots, m_1 \quad (16)$$

where the set of centers $\{t_i \,|i = 1, 2, \ldots, m_1\}$ is to be determined. Then $F^*(x)$ can be refined as

$$F(x) = \sum_{i}^{m_1} w_i G(x, t_i) \;\; = \sum_{i=1}^{m_1} w_i G(\|x - t_i\|) \quad (17)$$

V. The Experiment

The experiment was carried out on 118 software modules that were measured by 11 software metrics and normalized in the range 0 to 1. The data was separated into two sets, 89 modules for the training set and 29 modules for the test set. The training set was used to construct the SOM clusters and the RBF model. The test set was used for subsequent test runs.

We first grouped data in to 57 groups with SOM, clustered the 57 groups with the proposed clustering approach to 15 clusters. We then applied the RBF model to each cluster to predict the software faults. A post-mortem analysis was performed to assess the accuracy of the model.

A. Data Clustering

SOM was applied to the training process covering two phases for fine grain classification [17]. The first phase was the ordering phase in which all weight vectors of the map units were ordered. In the second phase, the values of the weight vectors were fine-tuned to adjust the SOM structure based on the following parameters:

- size of the map: 11×8
- initial neighborhood radius: 10 (ordering phase), 3 (tuning phase)
- initial condition: randomly initiated connections
- type of neighborhood function: Gaussian function
- learning rate: 0.05 (ordering phase), 0.02 (tuning phase)
- termination criterion: 1000 iterations (ordering phase) 10000 iterations (tuning phase)

The resulting SOM grouping consisted of 57 weight units with their own members. Further investigation revealed that the remaining 31 weight units were singleton units. As such, we discarded the remaining 31 weight units and fed the other 57 weight units to the proposed clustering approach with the member threshold value of 0.7. This yielded 15 data clusters.

B. Cluster Weighted Centering

After data clustering phase, a fault predictive model was constructed using the RBFN to operate on each cluster. The GRBFN method is based on the concept that training neural network is a curve fitting process in high dimensional space [18]. Two kernel function (the Gaussian function in this case) parameters that made up the cluster weight used in the experiment are cluster center and its radius.

Center estimation is a process for determinating proper number of centers in the clusters and their specific location. In principle, the number of centers of a cluster is the same as the number of weight units in that cluster. The SOM algorithm is used to estimate the location of the centers. For example, for a cluster having 3 weight units obtained from SOM clustering

along with 15 data points (or software modules), we designate the weight units as the 3 centers of the cluster. Three new weight units are then introduced and the SOM process is re-applied. The resulting cluster formation will have the three newly introduced weight units as the new centers for the consider cluster.

The width of the Gaussian function (σ) for each cluster is defined as follows:

$$\sigma = \frac{d}{\sqrt{2M}} \tag{18}$$

where

M is the number of centers.
d is the maximum distance between the centers.

C. Fault Predictive Models

When the centers and the width of Guassian function are defined, the fault predictive model can be constructed with the RBFN for each cluster. The structure of our network consists of three layers encompassing 11 input nodes in the input layer, a number of hidden nodes in the hidden layer (which are determined by the number of centers in the clusters), and 3 output nodes in the output layer. The output value denotes the number of faults obtained from the experiment, displayed in binary format. For example, the values '000', '010', and '111' represent zero, two, and seven faults, respectively.

During the experiment, training data was fed to the RBFN to generate weights between the hidden layer and the output layer. Test data was then applied to the fault prediction process. If the prediction yields low accuracy, the clusters must be reorganized.

To reorganize the formation, the previous process is repeated with the number of centers incremented by one. This model restructuring by center-plus-one progression continues until the predicted accuracy is acquired or the number of centers reaches the number of training data points of that cluster.

Based on the above procedures, our model yields a 93% prediction accuracy of test data. Beside the correctness percentage, another evaluation criterion called the mean of absolute residual (MAR) [19] is also applied to the model with the value of 0.17. AR is defined as

$$AR = |y - \hat{y}| \tag{19}$$

where

y is the actual number of faults.
$\hat{y}$ is the predicted number of faults

D. Result Analysis

During RBF application to each cluster, the covariance matrix of the data and their corresponding eigen vector and eigen value were computed to assess any variation deviated from earlier training results. For multi-dimensional space, the eigen vectors and eigen values represent the axes of the ellipsoids and their lengths, respectively. The highest eigen value for each group of cluster represents the maximum spread of the corresponding feature.

Therefore, the software metrics corresponding to the highest eigen value axis will entail the maximum spread of software metrics.

VI. Conclusion

The application of neural networks in predicting software fault requires enormous amount of data. Analyzing these data is a major undertaking that must be carried out with the help of proper models. We proposed a systematic approach to categorize closely related data using SOM method in conjunction with proposed clustering approach, and the RBFN method to accentuate the analysis of software fault prediction accuracy. With proper fine-tuning, the number of faults for each component can be determined and applied to subsequent predictive quality. A notable characteristic of data which effects the number of clusters is fault scattering pattern. If the faults of the component under investigation do not correspond to the software metrics, fault prediction will not be accurate and meaningful.

We envision several approaches for reliability model improvement. First and foremost is metrics selection. It is difficult to find a proper mix of metrics for the model parameters. Second, determining meaningful features to extract fault pattern for subsequent analysis is not a straightforward formulation. Third, proper data clustering technique will enhance not only the efficiency of the training process, but also the performance of the model predictability precision. Accurate predictions obtained from such a good reliability model will be conducive toward higher software process efficiency and product quality.

References

[1] J. D. Musa, A. Iannino, and K. Okumoto, *Software Reliability Measurement, Prediction, Application*. the United States of America: McGraw-Hill Book Company, 1987.

[2] C. Anderson, A. V. Mayrhauser, and T. Chen, "Assessing neural networks as guides for testing activities," pp. 155–165, March 1996.

[3] W. A. Adnan and M. H. Yaacob, "An integrated neural-fuzzy system of software reliability prediction," pp. 154–158, December 1994.

[4] S. Hong and K. Kim, "Identifying fault prone modules: An empirical study in telecommunication system," pp. 179–183, March 1998.

[5] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Predicting the location and number of faults in large software systems," *IEEE Transactions on Software Engineering*, vol. 31, no. 4, pp. 340 – 355, 2005.

[6] P. Eklund and L. Kallin, "Fuzzy systems," Febuary 2000, lecture Notes prepared for courses at the Department of Computing Science at Umeå University, Sweden.

[7] W. Pedrycz, G. Succi, M. Reformat, P. Musilek, and X. Bai, "Self organizing maps as a tool for software analysis," pp. 93–97, May 2001.

[8] S. Haykin, *Neural Networks*. the United States of America: Prentice Hall, 1999.

[9] S. H. Kan, *Metrics and Models in Software Quality Engineering*. Massachusetts: Addison-Wesley, 1995.

[10] F. Lanubile, "Evaluating predictive models derived from software measure," *Journal of Systems and Software*, vol. 38, no. 1, pp. 225 – 234, 1996.

[11] T. Kohonen, E. Oja, O. Simula, A. Visa, and J. Kangas, "Engineering applications of the self-organizing map," pp. 1358–1384, October 1996.

[12] S. Kaski, J. Nikkila, and T. Kohonen, "Methods for interpreting a self-organizied map in data analysis," Brussels, Belgium, pp. 185–190, April 1998.

[13] A. Ultsch and H. P. Siemon, “Kohonen’s self organizing feature maps for exploratory data analysis,” Dordrecht, The Netherlands, pp. 305–308, 1990.
[14] J. Vesanto and E. Alhoniemi, “Clustering of the self-organizing map,” *IEEE Transactions on Neural Networks*, vol. 11, no. 3, pp. 586–600, May 2000.
[15] M. R. Anderberg, *Cluster Analysis for Applications*. New York: Academic Press, 1973.
[16] D. Merkl and A. Rauber, “Cluster connections: A visualization technique to reveal cluster boundaries in self-organizing maps,” May 1997.
[17] T. Kohonen, J. Hynninen, J. Kangas, and J. Laaksonen, “Self-organizing map program package,” Helsinki University of Technology, Laboratory of Computer and Information Science, Rakentajanaukio 2 C, SF-02150 Espoo, Findland, April 1995.
[18] R. Li, G. Lebby, and S. Baghavan, “Performance evaluation of gaussian radial function network classifiers,” pp. 355–358, April 2002.
[19] I. Myrtveit, E. Stensrud, and M. Shepperd, “Reliability and validity in comparative studies of software prediction models,” *IEEE Transactions on Software Engineering*, vol. 31, no. 5, pp. 380 – 391, 2005.