# A Novel Method for Early Software Quality Prediction Based on Support Vector Machine

Fei Xing[1], Ping Guo[1,2], and Michael R. Lyu[2]
[1]*Department of Computer Science*
*Beijing Normal University, Beijing, 100875, China*
[2]*Department of Computer Science & Engineering*
*The Chinese University of Hong Kong, Shatin, NT, Hong Kong, SAR of China*
*xsoar@163.com, pguo@bnu.edu.cn, lyu@cse.cuhk.edu.hk*

## Abstract

*The software development process imposes major impacts on the quality of software at every development stage; therefore, a common goal of each software development phase concerns how to improve software quality. Software quality prediction thus aims to evaluate software quality level periodically and to indicate software quality problems early. In this paper, we propose a novel technique to predict software quality by adopting Support Vector Machine (SVM) in the classification of software modules based on complexity metrics. Because only limited information of software complexity metrics is available in early software life cycle, ordinary software quality models cannot make good predictions generally. It is well known that SVM generalizes well even in high dimensional spaces under small training sample conditions. We consequently propose a SVM-based software classification model, whose characteristic is appropriate for early software quality predictions when only a small number of sample data are available. Experimental results with a Medical Imaging System software metrics data show that our SVM prediction model achieves better software quality prediction than some commonly used software quality prediction models.*

## 1. Introduction

Modern society is fast becoming dependent on software products and systems. High reliability is one of the most important problem facing the software industry. A software quality model is a tool for focusing software enhancement efforts. Such models yield timely predictions on a module-by-module basis, enabling one to target high-risk modules.

Software metrics represent a quantitative description of program attributes and they play a critical role in predicting the quality of the resulting software [1]. Software complexity metrics have been shown to be closely related to the distribution of faults in program modules. That is, there is a direct relationship between some complexity metrics and the number of faults later found during test, validation and operation [2, 3]. Consequently, investigating the relationship between the number of faults in a program and its software complexity metrics attracts attentions from many researchers.

Software complexity metrics can be used as input variables of quality prediction model to predict the fault number, but predicting the exact number of faults in each module is often not necessary. Several different techniques have been proposed to develop predictive software metrics for the classification of software program modules into fault-prone and non fault-prone categories. These techniques include discriminant analysis [4, 5], factor analysis [6], boolean discriminant functions [7], classification trees [8, 9], pattern recognition (Optimal Set Reduction, OSR) [4, 10], EM algorithm [11], feedforward neural networks [12], random forests [13], and many other methods [14]. With these predictive models, developers and managers can focus resources on the most fault-prone modules early and prevent problems of poor quality later in the software life cycle.

To build a predictive model, the number of changes (faults) is usually required. However, to obtain the dependent criterion variables, we need to take a long time for collecting the feedback of test and validation results. For example, for the Medical Imaging System (MIS) software presented later in this paper, the actual number of changes (faults) in that program was collected during a three-year observation period. As software complexity metrics can be obtained relatively early in the software life-cycle, it is worthy to explore new techniques for early prediction of software quality based on software complexity metrics. On the other hand, the relationships between software metrics and the classification of program modules are often compli-

Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering (ISSRE 2005)
1071-9458/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY

cated and nonlinear, limiting the accuracy of conventional approaches. So it is difficult to model with the traditional methods, and an appropriate nonlinear model needs to be developed to solve the problem.

Support Vector Machine (SVM) [15] is a new technique for data classification, which has been used successfully in many object recognition applications [16, 17, 18, 19]. SVM is known to generalize well even in high dimensional spaces under small training sample conditions and it is adaptive to model nonlinear functional relationships that are difficult to model with other techniques. All these characteristics make SVM appropriate for software quality modeling as such conditions are typically encountered.

## 2. Support Vector Machine

Here we briefly review the basics of SVM first. SVM was introduced by Vapnik in the late 1960s on the foundation of statistical learning theory [20]. In theory, the SVM classification can be traced back to the classical structural risk minimization (SRM) approach, which determines the classification decision function by minimizing the empirical risk.

### 2.1. The Optimal Separating Hyperplane

SVM employs a linear model to implement nonlinear class boundaries through some nonlinear mapping of the input vectors $\mathbf{x}$ into the high-dimensional feature space $\mathscr{F}$ via a nonlinear mapping $\phi$. The optimal separating hyperplane is determined by giving the largest margin of separation between different classes. For the two-class case, this optimal hyperplane bisects the shortest line between the convex hulls of the two classes. The data are separated by a hyperplane defined by a number of support vectors. The SVM attempts to place a linear boundary between the two different classes, and orient the boundary in such a way that the margin is maximized. The boundary can be expressed as follows:

$$(\mathbf{w}\cdot\mathbf{x})+b=0, \qquad \mathbf{w}\in R^N,\ b\in R, \tag{1}$$

where the vector $\mathbf{w}$ defines the boundary, $\mathbf{x}$ is the input vector of dimension $N$ and $b$ is a scalar threshold.

The optimal hyperplane is required to satisfy the following constrained minimization as

$$\min\{\frac{1}{2}\|\mathbf{w}\|^2\} \tag{2}$$

$s.t.$

$$y_i(\mathbf{w}\cdot\mathbf{x}_i+b)\geq 1,\ i=1,2,\ldots,l, \tag{3}$$

where $(\mathbf{x}_i, y_i)$ is the training set, and $l$ is the number of training sets.

Using standard Lagrangian duality techniques, one arrives at the following dual Quadratic Programming (QP) problem:

$$\max\{\sum_{i=1}^{l}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{l}\alpha_i\alpha_j y_i y_j(\mathbf{x}_i\cdot\mathbf{x}_j)\} \tag{4}$$

$s.t.$

$$\sum_{i=1}^{l}y_i\alpha_i=0, \tag{5}$$

$$\alpha_i\geq 0,\ i=1,2,\ldots,l, \tag{6}$$

where the $\alpha_i$ are the Lagrangian multipliers and are nonzero only for the support vectors. Thus, the hyperplane parameters $(\mathbf{w},b)$ and the classifier function $f(\mathbf{x};\mathbf{w},b)$ can be computed by an optimization process. The decision function is obtained as follows:

$$f(\mathbf{x})=\text{sign}\{\sum_{i=1}^{l}y_i\alpha_i(\mathbf{x}\cdot\mathbf{x}_i)+b\}. \tag{7}$$

### 2.2. The Generalized Optimal Separating Hyperplane

For the linearly non-separable case, the minimization problem needs to be modified to allow misclassified data points. This modification results in a soft margin classifier that allows but penalizes errors by introducing positive slack variables $\xi_i$ $(i=1,2,\ldots,l)$ as the measurement of violation of the constraints:

$$\min\{\frac{1}{2}\|\mathbf{w}\|^2+C(\sum_{i=1}^{l}\xi_i)\} \tag{8}$$

$s.t.$

$$y_i(\mathbf{w}\cdot\mathbf{x}_i+b)\geq 1-\xi_i,\ i=1,2,\ldots,l, \tag{9}$$

where $C$ is used to weight the penalizing variables $\xi_i$, and a larger $C$ corresponds to assigning a higher penalty to errors.

The solution to this minimization problem is identical to the separable case except for a modification of the bounds of the Lagrange multipliers. Equation (6) is thus changed to:

$$0\leq\alpha_i\leq C,\ i=1,2,\ldots,l. \tag{10}$$

### 2.3. Generalization in High Dimensional Feature Space

In the cases where the linear boundary in input spaces is not enough to separate two classes properly, it is possible to create a hyperplane that allows the linear separation in a higher dimension space $\mathscr{F}$. The method consists of projecting the data in a higher dimension space $\mathscr{F}$ via a nonlinear mapping $\phi$, where they are considered to become


1071-9458/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY

linearly separable. The transformation into a higher dimensional feature space is relatively computation-intensive. A kernel can be used to perform this transformation, and the dot product in a single step providing the transformation can be replaced by an equivalent kernel function. This helps in reducing the computational load and retaining the effect of higher-dimensional transformation at the same time. The kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$ is defined as follows [15]:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j). \tag{11}$$

There are some commonly used kernel functions:

1. Polynomial: $K(\mathbf{x}_i, \mathbf{x}_j) = [(\mathbf{x}_i \cdot \mathbf{x}_j) + 1]^q$
2. Radial basis: $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma^2)$
3. Sigmoid: $K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\nu(\mathbf{x}_i \cdot \mathbf{x}_j) + c)$

## 2.4. SVM with Risk Feature

Generally speaking, the loss incurred for making an error is greater than the loss incurred for being correct, and minimizing risk means that we try to find decision procedure that minimizes serious errors. We therefore introduce risk control into SVM, which takes into account the cost of different types of errors by adjusting the error penalty parameter $C$ to control the risk. We suppose the first $k$ training sets belong to class 1 and the remaining $l-k+1$ training sets belong to class 2. Equation (8) is then converted to:

$$\min\{\frac{1}{2}\|\mathbf{w}\|^2 + C_1(\sum_{i=1}^{k} \xi_i) + C_2(\sum_{i=k+1}^{l} \xi_i)\} \tag{12}$$

$s.t.$

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i,\ i = 1, 2, \ldots, l, \tag{13}$$

where $C_1$ is the error penalty parameter of class 1 and $C_2$ is the error penalty parameter of class 2.

The solution to this minimization problem is also similar to that of the separable case, but the bounds of the Lagrange multipliers is changed to:

$$\begin{aligned} &0 \leq \alpha_i \leq C_1,\ i = 1, 2, \ldots, k, \\ &0 \leq \alpha_i \leq C_2,\ i = k+1, k+2, \ldots, l. \end{aligned} \tag{14}$$

Various effects of adjusting the error penalty parameter $C$ are shown in Figs. 1, 2 and 3, respectively. It is noted that when the class 1 and 2 have the same value of parameter $C$, the optimal separating hyperplane is achieved.

## 2.5. Transductive SVM

Traditional SVM only employs labeled training samples to build a classifier. As a kind of semi-supervised learning methods, Transductive Support Vector Machines (TSVM) [21] take into account a particular test set as well as training set, and try to minimize misclassifications of only those particular examples.

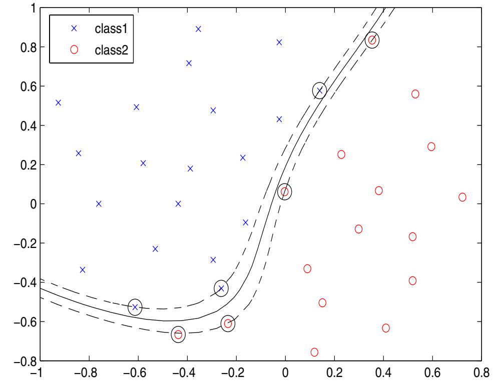


Figure 1. Optimal Separating Hyperplane, $C_1 = C_2 = 20000$

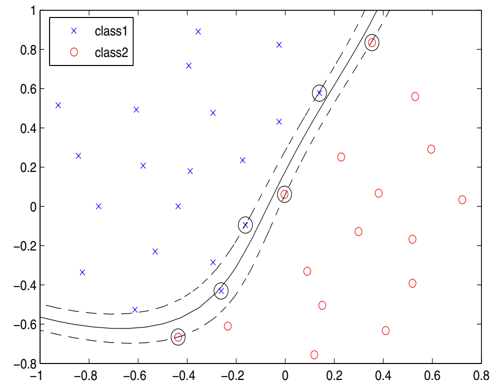


Figure 2. Separating Hyperplane with Risk Feature, $C_1 = 10000, C_2 = 20000$

Besides of $l$ labeled training samples $(\mathbf{x}_i, y_i)$, we consider another $m$ unlabeled samples $x_i^*$. To find a labeling $y_i^*$ of the test sample, the hyperplane $< \mathbf{w}, b >$ should separate both training and test data with the maximum margin:

$$Minimizeover(y_i^*, \mathbf{w}, b) : \frac{1}{2}\|\mathbf{w}\|^2 \tag{15}$$

$s.t.$

$$\begin{aligned} &y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1,\ i = 1, 2, \ldots, k, \\ &y_i^*(\mathbf{w} \cdot \mathbf{x}_i^* + b) \geq 1,\ i = 1, 2, \ldots, m. \end{aligned} \tag{16}$$

To be able to handle non-separable data cases, we can introduce slack variables $\xi_i^*(i = 1, 2, \ldots, m)$ similar to the way we handle the traditional SVM. That is, the learning process of TSVM can be formulated as the following opti-


1071-9458/05 $20.00 © 2005 **IEEE**
IEEE COMPUTER SOCIETY

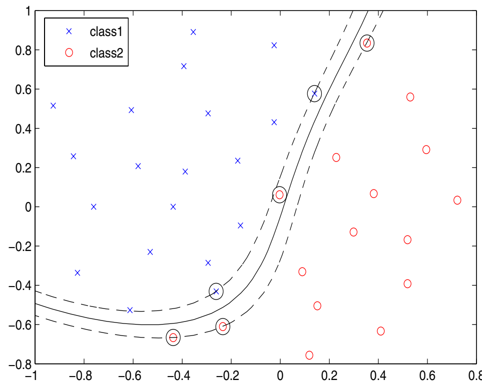


Figure 3. Separating Hyperplane with Risk Feature, $C_1 = 20000, C_2 = 10000$

mization problem:

$$\textit{Minimizeover} \quad (y_i^*, \mathbf{w}, b, \xi_i, \xi_i^*) : \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i + C^*\sum_{i=1}^{m}\xi_i^* \quad (17)$$

$s.t.$

$$\begin{aligned} & y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i,\ i = 1,2,\ldots,k, \\ & y_i^*(\mathbf{w}\cdot\mathbf{x}_i^* + b) \geq 1 - \xi_i^*,\ i = 1,2,\ldots,m, \\ & \xi_i \geq 0 \\ & \xi_i^* \geq 0 \end{aligned} \quad (18)$$

where $C$ and $C^*$ are user-specified parameters. $C^*$ is called the "effect factor" of the unlabeled examples and $C^*\xi_i^*$ is called the "effect term" of the $i$th unlabeled example in the above function. To solve this optimization equation, algorithms can be referenced from [22].

## 3. Other Models

### 3.1. Discriminant Analysis

Several researchers have applied discriminant techniques to study software complexity metrics. One of several discriminant techniques may be appropriate for a given analysis. Among them quadratic discriminant analysis (QDA) is the most widely used in the field of pattern recognition when sufficient training samples could be supplied.

Given the data points $D = \{\mathbf{x}_i\}_{i=1}^N$, we can apply Bayesian decision rule to classify the data $\mathbf{x}$ into $j$th class:

$$j^* = \arg\min_j d_j(\mathbf{x}), \qquad j = 1,2,\cdots,k \quad (19)$$

with

$$d_j(\mathbf{x}) = (\mathbf{x}-\mathbf{m}_j)^T\Sigma_j^{-1}(\mathbf{x}-\mathbf{m}_j) + \ln|\Sigma_j| - 2\ln\alpha_j \quad (20)$$

where $\alpha_j$ is the *prior* probability, $\mathbf{m}_j$ is the mean vector, and $\Sigma_j$ is the covariance matrix of the $j$th class.

Equation (20) is often called the discriminant function for the $j$th class in the literature [23]. Furthermore, if the *prior* probability $\alpha_j$ is the same for all classes, the term $2\ln\alpha_j$ can be omitted and the discriminant function reduces to a simpler form [24].

The parameters in Eq. (19) and Eq. (20) can be estimated with the traditional maximum likelihood estimator:

$$\alpha_j = \frac{n_j}{N}, \quad (21)$$

$$\mathbf{m}_j = \frac{1}{n_j}\sum_{i=1}^{n_j}\mathbf{x}_i, \quad (22)$$

$$\Sigma_j = \frac{1}{n_j}\sum_{i=1}^{n_j}(\mathbf{x}_i-\mathbf{m}_j)(\mathbf{x}_i-\mathbf{m}_j)^T. \quad (23)$$

Now $\mathbf{x}_i$ is a sample from class $j$ with probability one, and $n_j$ is the training sample number of class $j$. When an unbiased estimation is used, then

$$\Sigma_j = \frac{1}{n_j-1}\sum_{i=1}^{n_j}(\mathbf{x}_i-\mathbf{m}_j)(\mathbf{x}_i-\mathbf{m}_j)^T, \quad (24)$$

which is called the sample covariance matrix in the literature [25].

Using the classification rule in Eq. (19) and Eq. (20) with the above covariance estimation is known as quadratic discriminant analysis.

### 3.2. Classification Tree

A classification tree (CT) can be used to predict the class membership of objects on the basis of one or more predictor variables. It is widely used in pattern recognition filed, whereas Khoshgoftaar introduced the classification and regression trees (CART) algorithm to software quality prediction [26]. The CT as well as constructing CT algorithms are described in the literature as the following [27].

The tree consists of a set of decision rules, which are applied in a sequential manner, until each object has been assigned to a specific class. The first decision rule, applied at the "root node" of the tree to the values of all objects along one or more predictor variables, has two possible outcomes: Objects are either sent to a terminal node (leaf), upon which a class is assigned, or to an intermediate node, upon which another decision rule applies. Ultimately, all objects are sent to a terminal node and assigned a class label. The simplest type of CT is the binary tree, in which the splits are binary (that is, each parent node is attached to two daughter nodes) and the decision rules are univariate. CTs can be constructed based on continuous or discrete predictor variables, or on a mixture of both (when univariate splits


1071-9458/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY

are applied), and the trees are generally constructed by recursive partitioning (i.e., a given predictor variable can be engaged in more than one decision rule).

To construct CTs, there are two commonly-used algorithms. One algorithm is classification and regression trees (CART), which was developed by Breiman *et al* [28], and the other is the quick, unbiased, efficient statistical trees (QUEST), which was developed by Loh and Shih [29]. The CART algorithm finds the optimal univariate splits by carrying out an exhaustive search of all possible splits, whereas by applying a modified form of discriminant analysis QUEST algorithm finds the optimal univariate or multivariate splits. These algorithms have some different features. It is reported that the CART algorithm is biased toward selecting predictor variables having more levels, whereas the QUEST algorithm avoids this bias, and is therefore more appropriate when some predictor variables have few levels while other predictor variables have many levels [29]. Conversely, an advantage of CART is that it is a non-parametric classifier, i.e., no assumptions are made about the distributions of the variables. Thus, CART analysis can be used when the assumptions of linear discriminant analysis (LDA) and binary logistic regression (BLR) have not been satisfied.

The optimal CT is one that minimizes costs. When the prior probabilities of objects belonging to different classes are set proportional to the class size, and if misclassification costs are set to be the same for every class, then minimizing costs is equivalent to minimizing the overall proportion of misclassified objects. However, if the prior probabilities are set according to previous knowledge, or if different misclassification costs are used, then minimizing costs does not correspond exactly to minimizing the misclassification rate. Unequal misclassification costs are used when one kind of misclassification is considered "worse" than another. For example, incorrectly predicting a fault-prone module to be non fault-prone (Type II error) might be considered worse than incorrectly predicting a non fault-prone module to be fault-prone (Type I error). In such a case, when assessing accuracy of the classification, misclassifications of the former type could be penalized more than misclassifications of the latter.

When constructing a CT, if it is grown until all terminal nodes are homogeneous, the resulting tree is likely to overfit the data, and will therefore suffer a lower accuracy of classification when applied to new objects. To avoid this case, one should therefore apply a stopping rule so that splitting stops at nodes that are either completely homogenous, or have no more than a specified number of objects in the case of nodes containing more than one class. Other strategies include specifying the size of the tree to be grown (i.e., the number of terminal nodes increases), or defining the minimum heterogeneity that a node must have in order to be split.

Even when a stopping rule is applied, the resulting tree may not be the best tree, in the sense of maximizing accuracy of classification while at the same time minimizing complexity. Thus, in order to obtain the optimal tree, we should derive procedures for pruning trees, i.e., for successively prune the least important splits until the best tree is produced. In minimal cost-complexity pruning, a nested sequence of optimally pruned subtrees is generated when the tree of the maximum size is pruned to the root node. The maximum size is determined by the stopping criterion. The sequence is optimally pruned since there is no other tree of the same size with lower cost for every size of tree in the sequence. At first, the learning sample cost decreases as the size of the tree increases. However, the cost generally decreases slowly as the first terminal nodes are removed, until a point is reached when the cost rises rapidly upon removal of additional nodes. This turning point can be used to define the best-sized tree. Alternatively, the CV costs can be used to identify the best tree in the sequence if cross-validation (CV) is performed at each step of the pruning process. This is called as minimal cost-complexity CV pruning. Generally, the CV cost falls slowly to a minimum value as terminal nodes are removed, and then rises rapidly as the last few nodes are removed. Thus, the best tree can be defined as the tree closest to the minimum, i.e., the tree with the minimum CV cost. In addition, Breiman *et al* suggested that the best-sized tree can be identified as the smallest tree whose CV cost does not exceed the cost of the minimum CV cost tree plus one standard error of this tree's CV cost [28].

## 4. Experiments

### 4.1. Data Description

In this section, we present a real project to which we apply SVM for quality prediction. The metrics data used for the application represents the results of an investigation of software for a Medical Imaging System (MIS). It was from the data and tool CD of the book [30] and widely disseminated. The total system consists of approximately 4500 routines amounting to about 400000 lines of code written in Pascal, FORTRAN, assembly, and PL/M. From the set of programs written in Pascal and FORTRAN, a random sample of 390 routines was selected for analysis. These routines include approximately 40000 lines of code. The software was developed over a period of five years, and was in commercial use at several hundred sites for a period of three years [6].

In MIS data set which was collected by Randy Lind [32], the number of changes made to a module, docu-


IEEE COMPUTER SOCIETY

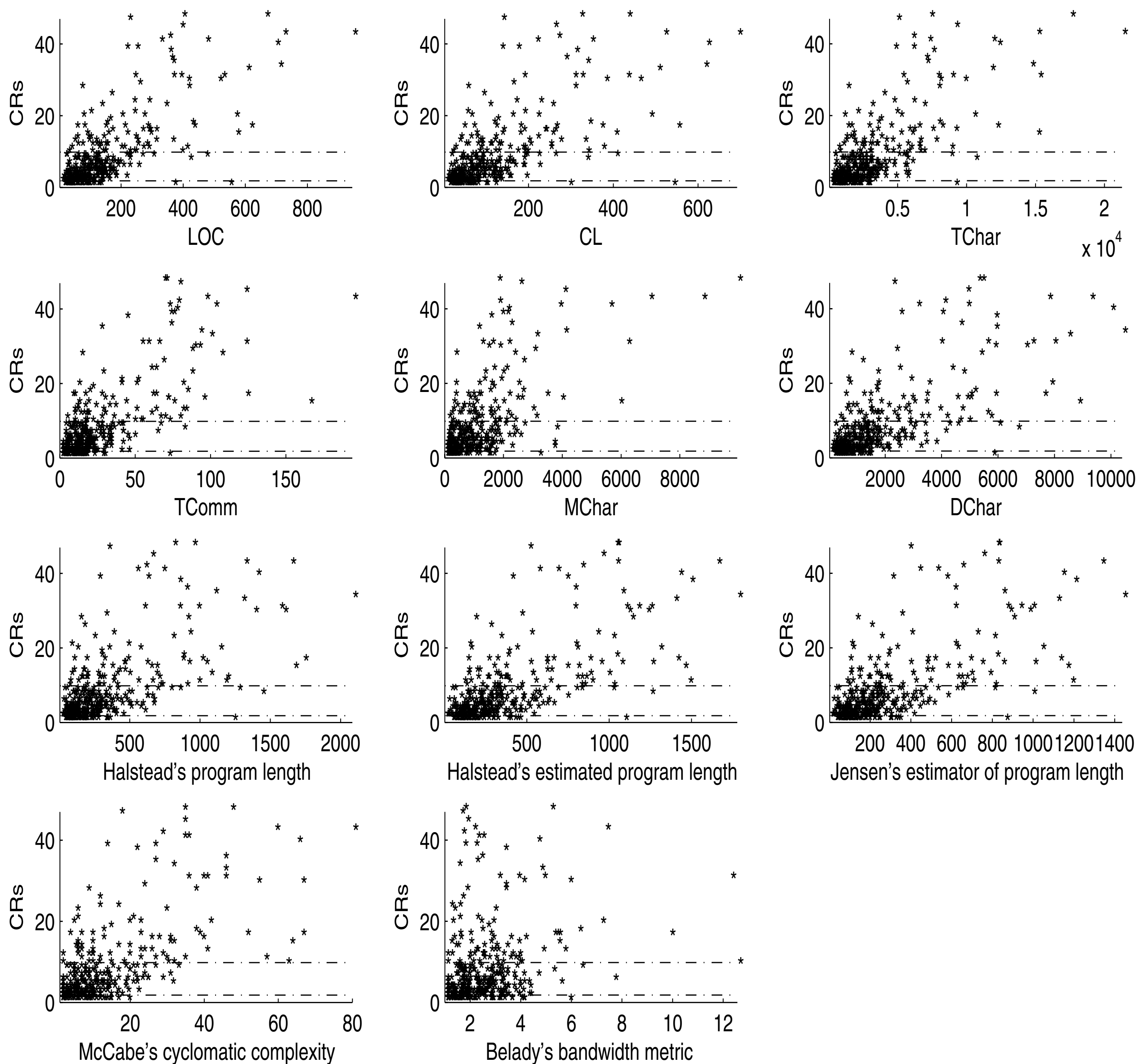


Figure 4. The relationship between the metrics the number of CRs

mented as Change Reports (CRs), was used as an indicator of the number of faults introduced during development [31]. The changes made to the routines were analyzed, and only those that affected the executable code were counted as faults (aesthetic changes such as comments were not counted).

Fig. 4 shows the relationships between the number of CRs and the software complexity metrics. We can find that the low software complexity metric corresponds to the small number of CRs. It is a obvious case in real project.

In addition to the change data, the following 11 software complexity metrics were measured for each of the modules:


1071-9458/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY

- Total lines of code including comments (LOC)
- Total code lines (CL)
- Total character count (TChar)
- Total comments (TComm)
- Number of comment characters (MChar)
- Number of code characters (DChar)
- Halstead's program length ($N$), where $N = N_1 + N_2$ and $N_1$ represents a total operator count, and $N_2$ represents a total operand count [33].
- Halstead's estimated program length ($\hat{N}$), where $\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$, and $\eta_1$ and $\eta_2$ represent the unique operator and operand count, respectively [33].
- Jensen's estimator of program length ($N_F$), where $N_F = \log_2 \eta_1! + \log_2 \eta_2!$ [34].
- McCabe's cyclomatic complexity ($v(G)$), where $v(G) = e - n + 2$, and $e$ is the number of edges in a control flowgraph representation of a program with $n$ nodes [35].
- Belady's bandwidth metric (BW), where

$$BW = \frac{1}{n} \sum_i iL_i$$

and $L_i$ represents the number of nodes at level $i$ in a nested control flowgraph of $n$ nodes [34]. This metric is indicative of the average level of nesting or width of the control flowgraph representation of the program.

In classifying a module to be fault-prone or non fault-prone, there are two types of errors that can be made in the partition. A Type I error is the case where we conclude that a program module is fault-prone when in fact it is not. A Type II error is the case where we believe that a program module is non fault-prone when in fact it is fault-prone. We denote the types of errors as T1ERR and T2ERR, respectively. Of the two types of errors, Type II error has more serious implications, since a product would be seem better than it actually is, and testing effort would not be directed where it would be needed the most. The nature of the impacts of these error types suggests that the Type II error rate is more important than the Type I error rate in considering the quality of a classification model.

In the experiment, we consider those modules with 0 or 1 CRs to be non fault-prone, and those with CRs from 10 to 98 to be fault-prone. For the MIS data used in the experiment, there are 114 non fault-prone modules and 89 fault-prone modules. The distribution of these software complexity metrics of MIS software in the first three principal

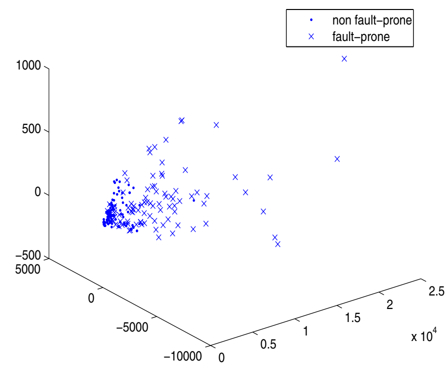


Figure 5. Distribution of MIS data set in first three principal components space

components (PCs) space is shown in Fig. 5. The bootstrap technique [36] is applied in the experiment. The total 203 samples are divided into two parts, where half independent random samples drawn from each class are used to train the SVM classifier, and the remaining half samples are used as test samples to calculate correct classification rate (CCR). The experiment is repeated 25 times with different random partitions and the mean and standard deviation of the classification accuracy are reported.

## 4.2. The Comparison of Several Methods

Quadratic discriminant analysis (QDA) is a widely used classification technique when sufficient training samples could be supplied. In the experiment, firstly QDA is used to classify the original data directly. Table 1 shows that for classification directly using QDA, the CCR could achieve 85.49%, and the Type I error and Type II error are 7.37% and 7.14%, respectively.

Principal components analysis (PCA) [37] can also be applied. Munson and Khoshgoftaar found that software complexity metrics are actually linear combinations of a small number of underlying orthogonal metric domains [38]. To reduce the interrelated effect, we adopt PCA to transform the original complexity metrics space into an orthogonal vector space. The principle of PCA is simple. Let us assume the data set has a covariance matrix $\Sigma$, which is a real symmetric matrix and can be decomposed as follows:

$$\Sigma = \mathrm{U}\Lambda\mathrm{U}^T, \tag{25}$$

where $\Lambda$ is a diagonal matrix with the eigenvalues $\lambda_1, \lambda_2, \cdots, \lambda_m$ on its diagonal, U is an orthogonal matrix where column $j$ is the eigenvector associated with $\lambda_j$, and $\mathrm{U}^T$ is the transpose of U. The $m$ eigenvectors in U give the coefficients that define $m$ uncorrelated linear combinations


1071-9458/05 $20.00 © 2005 IEEE
IEEE COMPUTER SOCIETY

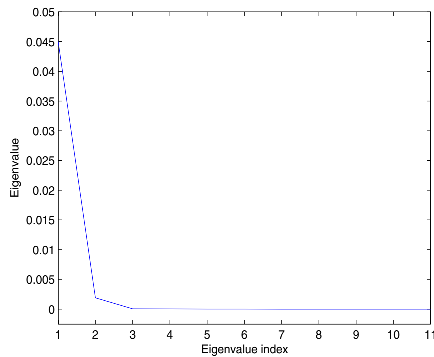


Figure 6. Eigenvalue in decreasing order

of the original complexity metrics. These orthogonal linear combinations are the principal components of Σ. An element $u_{ij}$ of U gives the coefficient of the $i$th complexity metric in the $j$th principle component. $\lambda_j$ gives the amount of complexity metric variance that is explained by the $j$th principle component. In principal components analysis, the eigenvalues along the diagonal of Λ form a decreasing series that explains all of the complexity data variance; that is, $\lambda_1 > \lambda_2 > \cdots > \lambda_m$, and $\sum_{j=1}^{m} \lambda_j$ gives the total variance in the complexity metric data.

The first few principal components typically explain a large proportion of the total observed variance. Thus, restricting our attention to the first few principal components can achieve a reduction in dimensionality with an insignificant loss of variance. From Fig. 6, we can find that the first two PCs have only 0.23% reconstruction error, so we choose them to define a 2-dimensional subspace and form 2-dimensional vectors in this subspace. Then QDA is used to complete the classification. The experimental results are shown in Table 1, where we can find that the composite classifier PCA+QDA achieves higher CCR 86.53% than QDA. Classification and regression trees (CART) analysis, a powerful nonparametric approach in classification or prediction, is also employed. We use PCA de-correlated data as input of CART, and obtain 83.02% CCR, which is lower than that of PCA+QDA. Consequently Type II error also declines compared with PCA+QDA, as shown in Table 1.

Furthermore, we discuss the approach in building software quality prediction model using SVM. Firstly, the original data are directly employed as the input to SVM. Considering the adaptive capacity of SVM, Radial Basis Function (RBF) is selected as the kernel function. From Table 1 we can see that direct classification by SVM could achieve 89.00% CCR. According to the applied hypothesis tests ($t$-test), the SVM method achieves a mean of the classification accuracy in the validation set, which is significantly ($\alpha$=0.05) larger than that of the QDA method ($p$-value is

Table 1. The comparison of several methods applied to software quality prediction

| Methods | CCR | Std | T1ERR | T2ERR |
|---|---|---|---|---|
| QDA | 85.49% | 0.0288 | 7.37% | 7.14% |
| PCA+QDA | 86.53% | 0.0275 | 4.90% | 7.52% |
| PCA+CART | 83.02% | 0.0454 | 9.59% | 6.41% |
| SVM | 89.00% | 0.0189 | 2.33% | 8.67% |
| PCA+SVM | 89.07% | 0.0209 | 2.06% | 8.87% |
| TSVM | 90.03% | 0.0326 | 2.11% | 7.86% |

Table 2. The classification accuracy comparison of the three kernels of SVM

| Kernel function | CCR | Std | T1ERR | T2ERR |
|---|---|---|---|---|
| Polynomial | 70.74% | 0.0208 | 0.45% | 28.81% |
| Radial basis | 88.68% | 0.0220 | 2.65% | 8.67% |
| Sigmoid | 88.75% | 0.0223 | 2.29% | 8.96% |

equal to $9.42 \times 10^{-10}$). But Type II error using SVM is higher than that of QDA. As a comparison, we also engage the dimension reduced data with PCA as the input of SVM. However, according to the hypothesis tests ($t$-test) applied, the classification result is not significantly improved ($p$-value is equal to 0.96). The reason is that SVM can simulate a non-linear projection which can map linearly inseparable data into a higher dimension space where the classes are linearly separable. So data space transform has little influence on SVM.

TSVM is a new method derived from SVM. When applying it to build the classifier, we find that the best CCR of 90.03% is achieved. But it is more time consuming for TSVM training than other methods.

### 4.3. The Comparison of the Three Kernels of SVM

For three kinds of commonly-used kernel functions that are mentioned in Section 2.3, we compare their performance in the experiment also. Table 2 shows the experimental result, where we can clearly see that Radial Basis kernel function and Sigmoid kernel function all achieve excellent performance, which are significantly better than Polynomial kernel function.

### 4.4. Experiments with the Minimum Risk

In practical case, Type II error often causes more serious consequence than Type I error does, so it is necessary to reduce Type II error. In Section 2.4, we introduce the


1071-9458/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY

Table 3. SVM with the risk feature

| $C_1$ | $C_2$ | CCR | Std | T1ERR | T2ERR |
|---|---|---|---|---|---|
| 5000 | 20000 | 86.53% | 0.0393 | 11.43% | 5.10% |
| 8000 | 20000 | 86.53% | 0.0275 | 4.90% | 7.52% |
| 10000 | 20000 | 89.00% | 0.0189 | 2.33% | 8.67% |
| 15000 | 20000 | 89.07% | 0.0209 | 2.06% | 8.87% |
| 20000 | 20000 | 89.07% | 0.0209 | 2.06% | 8.87% |

Table 4. The Bayesian decision with the minimum risk

| Risk ratio | CCR | Std | T1ERR | T2ERR |
|---|---|---|---|---|
| 1:1 | 85.94% | 0.0387 | 7.59% | 6.47% |
| 1:1.1 | 83.80% | 0.0326 | 11.06% | 5.14% |
| 1:1.2 | 78.73% | 0.0321 | 17.90% | 3.37% |
| 1:1.3 | 71.31% | 0.0436 | 27.12% | 1.57% |
| 1:1.4 | 59.98% | 0.0516 | 39.75% | 0.27% |

SVM theory combined with error risk control. In this experiment, we apply the theory to adjust Type II error. Table 3 shows the experimental result. When $C_1$ and $C_2$ are equal to 20000, 89.07% CCR is achieved, which is the optimal solution of SVM. But in this case Type II error is much larger than the Type I error. However, we can adjust $C_1$ to reduce Type II error. From Table 3, we find that when $C_1$ is reduced gradually, CCR also reduces, but this leads to the increase of Type I error. When $C_1$ is reduced to 5000, CCR is changed to 86.53%, where low Type II error 5.10% is achieved.

As a comparison, we also employ the Bayesian decision with the minimum risk to build a quality prediction model. The results are shown in Table 4, where the first column is the risk ratio of Type I error versus Type II error. We can observe that SVM with risk is superior to the Bayesian decision based on the minimum risk in the classification performance. Nevertheless, the Bayesian decision can adjust type II error to a very low value at the cost of lower CCR and higher Type I error.

## 5. Conclusions

A novel technique for software quality prediction is proposed in this paper. This methodology is based on support vector machine. The validity and robustness of SVM make it a meaningful tool for real-world applications in software quality prediction. Firstly, SVM is adaptive to modeling nonlinear functional relationships which are difficult to model with other techniques. Secondly, SVM generalizes well even in high dimensional spaces under small training sample conditions. Consequently, software quality prediction models can be built much earlier with SVM than other conventional techniques. Furthermore, as a new approach, Transductive SVM (TSVM) takes into account particular test samples as well as training samples in building classifiers. Thirdly, the SVM-based software quality prediction model achieves a relatively good performance. According to the hypothesis tests ($t$-test) applied, the classification results of the SVM are better than those of either QDA or a classification tree. Finally, we can control Type II error by adjusting the error penalty parameter $C$ of SVM, which can achieve better classification performance than using the minimum-risk-based Bayesian decision when considering both CCR and Type II error. To summarize, all the above characteristics show that the new approach which has never been explored in the software engineering fields offers a very promising technique in software quality prediction. We believe that our method can be extensively applied in many software engineering fields.

## Acknowledgements

This research work described in this paper was fully supported by a grant from the National Natural Science Foundation of China (Project No. 60275002) and a grant from Hong Kong Research Grants Council (Project No. CUHK4205/04E).

## References

[1] N. E. Fenton and S. L. Pfleeger, *Software metrics: a rigorous and practical approach*, 2nd ed. Boston, MA: PWS Publishing Co., 1997.

[2] V. Y. Shen, T. J. Yu, S. M. Thebaut, and L. R. Paulsen, "Identifying error-prone software – an empirical study," *IEEE Transactions on Software Engineering*, vol. SE-11, no. 4, pp. 317–324, apr 1985.

[3] T. M. Khoshgoftaar and E. B. Allen, "Classification of fault-prone software modules: Prior probabilities, costs, and model evaluation," *Empirical Software Engineering*, vol. 3, no. 3, pp. 275–298, September 1998.

[4] L. C. Briand, V. R. Basili, and C. Hetmanski, "Developing interpretable models for optimized set reduction for identifying high-risk software components," *IEEE Transactions on Software Engineering*, vol. SE-19, no. 11, pp. 1028–1034, 1993.

[5] J. Munson and T. Khoshgoftaar, "The detection of fault-prone programs," *IEEE Transactions on Software Engineering*, vol. SE-18, no. 5, pp. 423–433, 1992.

[6] T. Khoshgoftaar and J. Munson, "Predicting software development error using software complexity metrics," *IEEE Transactions on Software Engineering*, vol. 8, no. 2, pp. 253–261, 1990.

[7] T. M. Khoshgoftaar, "Improving usefulness of software


1071-9458/05 $20.00 © 2005 **IEEE**

IEEE COMPUTER SOCIETY

quality classification models based on boolean discriminant functions," in *International Symposium on Software Reliability Engineering*, (ISSRE '02), 2002, pp. 221–230.
[8] T. M. Khoshgoftaar, R. Shan, and E. B. Allen, "Improving tree-based models of software quality with principal components analysis," in *International Symposium on Software Reliability Engineering*, (ISSRE '00), 2000, pp. 198–209.
[9] T. M. Khoshgoftaar, V. Thaker, and E. B. Allen, "Modeling fault-prone modules of subsystems," in *International Symposium on Software Reliability Engineering*, (ISSRE '00), 2000, pp. 259–269.
[10] L. C. Briand, V. R. Basili, and W. M. Thomas, "A pattern recognition approach for software engineering data analysis," *IEEE Transactions on Software Engineering*, vol. SE-18, no. 11, pp. 931–942, 1992.
[11] P. Guo and M. R. Lyu, "Software quality prediction using mixture model with em algorithm," in *Proceedings of the First Asia-Pacific Conference on Quality Software*, (APAQS 2000), Hong Kong, 2000, pp. 69–78.
[12] 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, no. 2, pp. 279–291, 1994.
[13] L. Guo, Y. Ma, B. Cukic, and H. Singh, "Robust prediction of fault-proneness by random forests," in *International Symposium on Software Reliability Engineering*, (ISSRE '04), 2004, pp. 417–428.
[14] 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.
[15] C. Cortes and V. Vapnik, "Support-vector network," *Machine Learning*, vol. 20, pp. 273–297, 1995.
[16] S. Dumais, "Using svms for text categorization," *IEEE Intelligent Systems*, vol. 13, no. 4, pp. 21–23, 1998.
[17] M. Pontil and A. Verri, "Support vector machines for 3d object recognition," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 20, no. 6, pp. 637–646, 1998.
[18] H. Drucker, D. Wu, and V. N. Vapnik, "Support vector machines for spam categorization," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 1048–1054, 1999.
[19] F. Xing and P. Guo, "Classification of stellar spectral data using svm," in *International Symposium on Neural Networks* (ISNN'2004), ser. Lecture Notes in Computer Science, vol. 3173. Berlin, Heidelberger: Springer-Verlag, 2004, pp. 616–621.
[20] V. N. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer-Verlag, 1995.
[21] T. Joachims, "Transductive inference for text classification using support vector machines," in *Proceedings of the 16th International Conference on Machine Learning*, (ICML'99), Bled, Slovenia, 1999, pp. 200–209.
[22] T. Joachims, Ed., *Learning to Classify Text using Support Vector Machines: Methods, Theory, and Algorithms*. Kluwer, may 2002.
[23] S. Aeberhard, D. Coomans, and V. O. de, "Comparative analysis of statistical pattern recognition methods in high dimensional settings," *Pattern Recognition*, vol. 27, no. 8, pp. 1065–1077, 1994.
[24] X. Wang, F. Xing, and P. Guo, "Comparison of discriminant analysis methods applied to stellar data classification," in *Third International Symposium on Multispectral Image Processing and Pattern Recognition* (MIPPR'03), ser. Proceedings of SPIE, vol. 5286. Bellingham, WA: The International Society for Optical Engineering, 2003, pp. 758–763.
[25] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, 2nd ed. Boston: Academic Press, 1990.
[26] T. M. Khoshgoftaar and E. B. Allen, "Predicting fault-prone software modules in embedded systems with classification trees," in *Proceedings of Fourth International Symposium on High-Assurance Systems Engineering*, Washington, DC USA, 1999, pp. 105–112.
[27] A. P. Worth and M. T. D. Cronin, "The use of discriminant analysis, logistic regression and classification tree analysis in the development of classification models for human health effects," *Journal of Molecular Structure(Theochem)*, vol.622, pp. 97–111, 2003.
[28] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. California: Wadsworth, Inc., 1984.
[29] L. W. Y. and S. Y. S., "Split selection methods for classification trees," *Statistica Sinica*, vol. 7, pp. 815–840, 1997.
[30] M. R. Lyu, Ed., *Handbook of Software Reliability Engineering*. IEEE Computer Society Press and McGraw-Hill Book Company, 1996.
[31] V. R. Basili and D. H. Hutchens, "An empirical study of a syntactic complexity family," *IEEE Transactions on Software Engineering*, vol. SE-9, no. 6, pp. 664–672, nov 1983.
[32] R. K. Lind, "An experimental study of software metrics and their relationship to software error," Master's thesis, University of Wisconsin-Milwaukee, Milwaukee, Dec. 1986.
[33] M. H. Halstead, *Elements of Software Science*. New York: Elsevier, 1977.
[34] H. Jensen and K. Vairavan, "An experimental study of software metrics for real-time software," *IEEE Transactions on Software Engineering*, vol. SE-11, no. 2, pp. 231–234, feb 1985.
[35] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, 1976.
[36] B. Efron and R. J. Tibshirani, *An introduction to the Bootstrap*, Chapman and Hall, 1993.
[37] I. Jolliffe, *Principal Component Analysis*. New York: Springer-Verlag, 1986.
[38] J. C. Munson and T. M. Khoshgoftaar, "The dimensionality of program complexity," in *Proc. 11th International Conference on Software Engineering*, Pittsburgh, 1989, pp. 245–253.


1071-9458/05 $20.00 © 2005 **IEEE**

IEEE COMPUTER SOCIETY