NORTH-HOLLAND

Evaluating Predictive Quality Models Derived from Software Measures: Lessons Learned

Filippo Lanubile
Department of Computer Science, University of Maryland, College Park, Maryland

Giuseppe Visaggio
Dipartimento di Informatica, University of Bari, Bari, Italy

This paper describes an empirical comparison of several modeling techniques for predicting the quality of software components early in the software life cycle. Using software product measures, we built models that classify components as high-risk, i.e., likely to contain faults, or low-risk, i.e., likely to be free of faults. The modeling techniques evaluated in this study include principal component analysis, discriminant analysis, logistic regression, logical classification models, layered neural networks, and holographic networks. These techniques provide a good coverage of the main problem-solving paradigms: statistical analysis, machine learning, and neural networks. Using the results of independent testing, we determined the absolute worth of the predictive models and compare their performance in terms of misclassification errors, achieved quality, and verification cost. Data came from 27 software systems, developed and tested during three years of project-intensive academic courses. A surprising result is that no model was able to effectively discriminate between components with faults and components without faults. © 1997 Elsevier Science Inc.

1. INTRODUCTION

The construction of predictive systems is one of the main purposes of software measurement. Predictive systems have been built from product metrics by applying different kinds of modeling techniques. Multiple linear regression analysis has been used to predict the number of corrective changes (Khoshgoftaar et al., 1992; 1993). Discriminant analysis has been applied to detect fault-prone modules (Munson and Khoshgoftaar, 1992; Khoshgoftaar et al., 1996). Logistic regression has been used for modeling to identify high-risk components (Briand et al., 1993a; 1993b). Principal component analysis has often been used to improve the accuracy of discriminant models (Munson and Khoshgoftaar, 1992; Khoshgoftaar et al., 1996) or regression models (Briand et al., 1993a; 1993b; Khoshgoftaar et al., 1993). Logical classification models have been used extensively to identify high-risk modules (Selby and Porter 1988; Porter and Selby, 1990; Briand et al., 1993a; 1993b; Porter 1993) and reusable software components (Esteva and Reynolds, 1991). Layered neural networks have already been applied to building reliability growth models (Karunanithi et al., 1992a; 1992b), to predicting the gross change (Khoshgoftaar and Szabo, 1994), and the degree of reuse (Boetticher et al., 1993). Holographic networks, a nonconnectionist type of neural network, have been proposed for predicting software quality (Lanubile and Visaggio, 1994). Empirical investigations have not yet been performed in the software engineering field but have in other areas, such as financing (Soucek et al., 1994) and manufacturing (Jensen, 1994).

Many of the past studies have focused on predicting the presence of faults early in the software life cycle. Being able to know just after the coding phase, or even design phase, which parts are more subject to fail, allows software managers to focus their resources on inspecting or testing those error-

Filippo Lanubile is on sabbatical from University of Bari.
Address correspondence to Filippo Lanubile, Department of Computer Science, A.V. Williams Building, University of Maryland, College Park, MD 20742.

J. SYSTEMS SOFTWARE 1997; 38:225–234

655 Avenue of the Americas, New York, NY 10010

0164-1212/97/$17.00
PII S0164-1212(96)00153-7

prone components. The expected benefit is to achieve a more reliable product at a lower cost. However, all the studies have applied a very few candidate techniques (usually two or three). Furthermore, we cannot directly compare the results across the studies because of the lack of common evaluation criteria.

Theories and models become accepted by the scientific communities when different researchers obtain the same results running independent empirical studies. Thus, we began this study with the goal of externally replicating these past studies, and thus, to understand which modeling technique, if any, is better in predicting the fault-proneness of software components. Our replication is characterized by the following features:

- Use of product measures in predicting the fault-proneness of software components.
  Software product metrics are very popular as predictor variables of quality models. Unfortunately, quality is such a general concept that it needs to be decomposed in terms of other attributes which are directly measurable. Among the existing product quality attributes, we have focused on fault-proneness because most of the development time and cost is spent on detecting and fixing faults. We measured the fault-proneness of software components in terms of the number of faults found during testing. To predict fault-proneness, most studies measure both design and code attributes. However, there is no unique set of product metrics that all the studies use. Our predictor variables measure the following product attributes: coupling, size, control-flow structure, data structure, and documentation. Detailed definitions of the metrics are found in Table 1. They are essentially the same as those used by Munson and Khoshgoftaar (1992) to construct their predictive models.
- Reduction of the prediction problem to a classification problem.
  A major problem in predicting software quality using the number of component faults as a direct metric is the highly skewed distribution of faults because the majority of components have no faults or very few faults. Instead of estimating the number of potential faults in a software component, we determine whether a component is likely to be fault-prone or not. In this case, the direct metric of software quality is the class to which the software component belongs (high risk or low risk), and the prediction model is reduced to a classification model.
- Broader coverage of the modeling techniques already used in practice for classification.
  Our study compares the following modeling techniques: discriminant analysis, logistic regression, logical classification models, layered neural networks, and holographic networks. Principal component analysis has also been included as an optional preprocessing step before applying dis-

Table 1. Predictor Variables

Symbol	Name/Description
Size	
LOC	Number of lines of code
NCLOC	Number of noncomment lines of code
N	Halstead program length, where N = N1 + N2 and N1 is the total number of operators
V	Halstead volume, where $V = N * \log_2 n$ and n = n1 + n2 is the program vocabulary
Control Flow Structure	
VG	McCabe cyclomatic complexity, where $VG = e - n + 2$ for a flowchart with e edges and n nodes
Data Structure	
n2	Halstead number of unique operands
N2	Halstead total number of operands
Coupling	
fanin	Henry & Kafura fanin, where the fanin of a module M is the number of local flows that terminate at M, plus the number of data structures from which information is retrieved by M
fanout	Henry & Kafura fanout, where the fanout of a module M is the number of local flows that emanate from M, plus the number of data structures that are updated by M
IF	Henry & Kafura information flow, where $IF = (fanin * fanout)^2$
Documentation	
DC	Density of comments, where DC = CLOC/LOC and CLOC is the number of comment lines of program text

criminant analysis and logistic regression. Although this list of techniques cannot be considered exhaustive, it includes the main general approaches used to solve classification problems: statistical analysis, machine learning, and neural networks. Statistical techniques, like discriminant analysis and logistic regression, usually try to find an explicit numerical formula which determines a classification completely. Machine learning methods, like logical classification trees, try to deduce exact if-then-else rules that can be used in the classification process. The neural network approach, including layered neural networks and holographic networks, trains a neural network to reproduce a given set of correct classification examples without providing formulas, rules, or any insight in how learning and predictions are accomplished.

The next section characterizes the software environment and the data used in the empirical study. The third section describes how the modeling techniques were used to build the predictive models. The intent of this section is that an independent researcher should be able to review our implementation choices and perform external replications of this study. The fourth section shows the criteria that we used to validate and compare the models. The fifth and sixth sections report, respectively, the results of testing our predictive models against the evaluation criteria, and the results from other similar studies. Finally, the last section summarizes the lessons we learned from this study.

2. DATA DESCRIPTION

The data for this study was collected from projects performed by 27 teams of three students, during three years of a software engineering course at the University of Bari, Italy. Each team developed a business application based on the same requirements specification but independently designed and coded over a period of 4–10 months. The resulting software systems range in size from 1100 to 9400 lines of Pascal source code.

From each system, we randomly selected a group of 4 to 5 components, ranging in size from 60 to 530 lines of code, for a total of 118 components. Here, the term software component refers to a functional abstraction of code such as a procedure, function, or main program. Each group of 4 to 5 components was tested by a different student team from another software engineering course. Faults found during testing were attributed to individual components.

The distribution of faults discovered during the independent unit testing, shown in Figure 1, was heavily skewed in favor of components with no faults or only one fault. To build unbiased classification models, we decided to have an approximately equal number of components in the classes of reliability. Thus, we defined as high risk any software component where faults were detected during testing, and low risk any component with no faults discovered. The same criterion has been used by Porter (1993) and Briand et al. (1993b) for distinguishing between high-risk and low-risk components.

Besides the fault data, 11 software product metrics were used as predictor variables to construct the predictive models. Table 1 shows the product metrics we used in this study. The metrics have been selected to measure both the implementation and the design attributes of the components, such as size, control flow structure, data structure, and coupling; one documentation metric is also measured.

Our prediction models were based on software developed and tested by small student teams. Although this can be considered as a threat to the

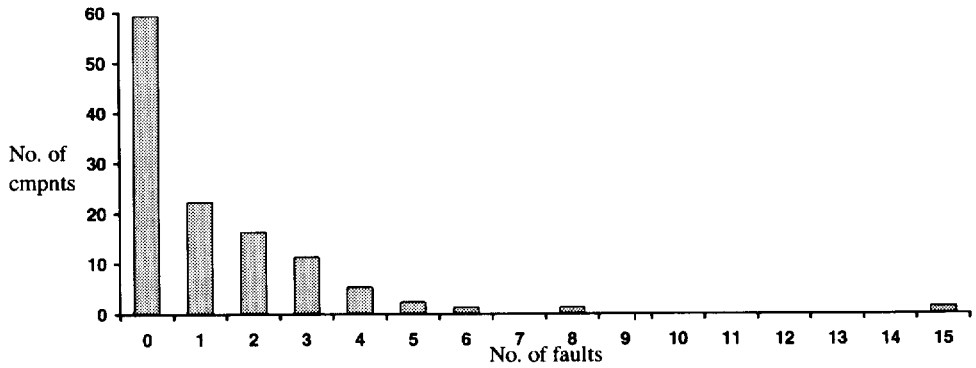

Figure 1. Distribution of faults per software component.

external validity of the study, there is no theory which restricts the use of product metrics as predictors of fault-proneness to some specific domain, environment, or engineering experience. On the contrary, in the early results of the application of the Personal Software Process (PSP), there is only a weak relationship between defect rates and experience, before learning PSP, and no relationship at the completion of the training (Humphrey, 1996).

3. BUILDING THE PREDICTIVE MODELS

For each of the 118 components, we had 11 product metrics and the risk class resulting from testing. We divided the data set into two groups. Two-thirds of the components (79 observations) were randomly selected to create and tune the predictive models. The remaining third of the components (39 observations) provided the data to test the models. From now on, the first group of observations will be called the training set and the second one the testing set.

There are many ways to build a predictive model using a given modeling technique. We describe our implementation choices to make possible the replication of the experiment in other environments as well as improvement in the application of the techniques.

3.1 Principal Component Analysis

Linear modeling applications, such as regression and discriminant analysis. can produce unstable models when the independent variables are strongly related. In this case, principal component analysis is applied to reduce the dimensions of the metric space and obtain a smaller number of orthogonal domain metrics (Dillon and Goldstein, 1984).

In our study, we used the FACTOR procedure in the SAS statistical package (SAS, 1989) to extract the principal components, rotate them, and compute the scoring coefficients. As input parameters to the procedure, we set all prior communalities to 1.0 for the eleven product metrics, and defined the minimum eigenvalue criterion as 0.9. As a result, three distinct domain metrics were retained. An orthogonal rotation was then applied, using the varimax method. In Table 2, each column shows the degree of relationship between the eleven metrics and the three orthogonal domains. Values in bold print indicate which domain dominates a metric. Domain 1 includes the metrics measuring implementation attributes; domain 2 contains those metrics related to design attributes; and domain 3 consists of the only metric that was intended to capture the documentation characteristics. The three principal components account for 85% of the variability in the eleven metrics. For each software component of our data set, the values of the three domain metrics were derived and used as input to discriminant analysis and logistic regression.

Table 2. Rotated Factor Pattern

Metric	Domain 1	Domain 2	Domain 3
V	**0.98338**	0.07621	−0.04700
N	**0.98209**	0.09208	−0.04874
LOC	**0.97486**	0.07603	−0.02989
NCLOC	**0.97448**	0.06976	−0.05521
N_2	**0.95392**	0.11957	−0.06178
$v(G)$	**0.87488**	0.19214	−0.01642
η_2	**0.73870**	0.01342	−0.00998
fanout	0.16845	**0.88696**	0.01091
IF	−0.01610	**0.85161**	0.02215
fanin	0.12539	**0.82472**	−0.12569
DC	−0.07395	−0.06408	**0.99215**
Eigenvalues before rotation	6.30601	2.10209	0.98032
Eigenvalues after rotation	6.10241	2.27254	1.01348
% Variance	55.47642	20.65942	9.21344
Cumulative % Variance	55.47642	76.13584	85.34928

3.2 Discriminant Analysis

Discriminant analysis develops a discriminant function or classification criterion to place each observation into one of a set of mutually exclusive groups (Dillon and Goldstein, 1984). It requires that there exists a prior knowledge of the classes, in our case, low-risk and high-risk components. To develop the classification criterion, we used the DISCRIM procedure in the SAS statistical package (SAS, 1989). The DISCRIM procedure applies a parametric method that uses a measure of generalized square distance. The procedure was set to compute the measure of generalized square distance on the basis of the pooled covariance matrix. The generalized square distance from an observation $\mathbf{x}$ to a class j is given by

$$D_j^2(\mathbf{x}) = \left(\mathbf{x} - \overline{\mathbf{x}}_j\right)^T \mathbf{S}^{-1} \left(\mathbf{x} - \overline{\mathbf{x}}_j\right)$$

where $\mathbf{x}$ is the vector containing the variables of the observation, $\overline{\mathbf{x}}_j$ is the vector containing means of the variables in the group j, and $\mathbf{S}$ is the pooled covariance matrix.

The posterior probability of an observation $\mathbf{x}$ belonging to class j is

$$p_j(\mathbf{x}) = e^{-0.5D_j^2(\mathbf{x})} \Big/ \sum_i \left(e^{-0.5D_i^2(\mathbf{x})}\right)$$

where i and j, in the case of two classes such as low and high risk, are, respectively, 1 and 2. An observation is classified in the class with the largest posterior probability value.

We built two different discriminant models: the first one, applying discriminant analysis directly on the original eleven product metrics, and the second one, using as input the three domain metrics obtained from the principal component analysis.

3.3 Logistic Regression

Logistic regression is a special type of regression analysis which models the response variable by calculating a function of the response probabilities to fit a linear model (Agresti, 1990). The standard response function computes the probability of class membership according to the following equation:

$$\log\left(\frac{p}{1-p}\right) = c_0 + \sum_{i=1}^{n} c_i x_i$$

where p can be interpreted as the probability that a software component is high risk, while the predictor variables x_i are the product metrics.

Unlike discriminant methods, logistic regression is not based on normality assumptions and thus is preferable to discriminant analysis when the variables do not have multivariate normal distributions within classes (Press and Wilson, 1978).

In our study, we used the CATMOD procedure in the SAS statistical package (SAS, 1989) to perform logistic regression. The regression coefficients c_i were computed through a maximum-likelihood estimation.

As for the discriminant analysis, two regression models were built: the first one is based on the eleven product measures, while the second one uses the three domain metrics that have been generated from the principal component analysis.

3.4 Logical Classification Models

Logical classification models are classifiers that can be expressed as decision trees or sets of production rules. They are generated through a recursive algorithm that, at each step, selects the attribute that best discriminates between components within a target class and those outside it. To build the classification model, we used the C4.5 system (Quinlan, 1993), an extension of the ID3 system (Quinlan, 1986). The C4.5 system partitions continuous attributes, in our case, the product metrics, finding the best threshold among the set of training cases. We used the gain criterion to select the best attribute to branch on at each step of the tree building. The recursive partition method continues to split the training set until each subset in the partition contains cases of a single class, or until no split gives a gain in information. If a subtree is found to misclassify at least as many items as does replacing the subtree with a leaf, then the subtree is replaced with the leaf. The decision tree built using the training set had 12 levels composed of 11 decision nodes and 12 leaf nodes. The decision tree was then converted to a set of production rules by forming a rule corresponding to each path from the root of the tree to each of its leaves. To simplify the collection of rules, C4.5 dropped conditions using a pruning heuristic based on pessimistic error estimation. We set the pruning confidence level to 0.05. As a final result, C4.5 produced a rule classifier made up of 12 rules ranging from one to five compound conditions.

3.5 Layered Neural Networks

We used a typical feed-forward neural network (Rumelhart et al., 1986), characterized in our experiment by one input layer of 11 neurons, each connected to a product metric, one output layer of only one neuron that provides the predicted risk, and one layer of 50 hidden neurons. Among the supervised algorithms, we chose the most popular one, the back-propagation algorithm, which adjusts network weights by iteration until a user-defined error tolerance is achieved, or a maximum number of iterations has been completed. For the neural network simulation software, we used a freeware program developed at University of Bari. The network weights were initially set to random values between -1.0 and 1.0 using a sigmoid distribution. We trained the network with a value of 0.1 for the error tolerance, 1 for the learning rate, and 0.7 for the momentum rate. After many trials over more than twenty hours, the neural network could not converge to an optimal solution including all the 79 observations of the training set. Thus, we stopped the training process after 9000 iterations with a trained state accounting for 78 observations.

Since the network's input and output are bounded between 0 and 1, we reduced the input data using a direct scaling. When testing the network, we increased the error tolerance to 0.5 so that low-risk components correspond to observations with an output value in the first half of $[0, 1]$ and high-risk components to observations with an output value in the second half.

3.6 Holographic Networks

With holographic networks, information is encoded inside holographic neurons rather than in the connection weights between neurons (Sutherland, 1990). A holographic neuron holds a correlation matrix that enables memorizing stimulus-response associations. Individual associations are learned deterministically in one noniterative transformation. Holographic neurons internally work with complex numbers in polar notation so that the magnitude (from 0.0 to 1.0) is interpreted as the confidence level of data, and the phase (from 0 to 2π) serves as the actual data value.

In our study, we used HNet Discovery System, a single cell forward system capable of mapping up to 1000 stimulus inputs to 100 response outputs. Input data were converted to the range $[0, 2\pi]$ using a sigmoid function and interpreted as phase orientation of complex values with a unity magnitude. On the other hand, the response was converted using a linear interpolation. These conversion methods provided the maximum symmetry in the distribution of data. We trained the network to obtain a maximum error of 0.1 for each observation.

4. EVALUATION CRITERIA

To evaluate the predictive models, we used a set of criteria that are based on the analysis of categorical data. In our study, we have two variables, real risk and predicted risk, that can assume only two discrete values, low and high, in a nominal scale. Thus, the data can be represented by a two-dimensional contingency table, shown in Table 3, with one row for each level of the variable real risk and one column for each level of the variable predicted risk. The intersections of rows and columns contain the frequency of observations (n_{ij}) corresponding to the combination of variables. Row totals ($n_{i.}$) and column totals ($n_{.j}$) correspond to the frequency of observations for each of the variables. In our context, the first row contains low-risk components, i.e., with no faults, while the second row contains high-risk components, including at least one fault. The first column contains components that the models classify as low risk, while the second column contains components classified as high risk.

Table 3. Two-Dimensional Contingency Table

	Predicted Risk		
Real Risk	Low	High	
low	n_{11}	n_{12}	$n_{1.}$
high	n_{21}	n_{22}	$n_{2.}$
	$n_{.1}$	$n_{.2}$	n

The evaluation criteria are predictive validity, misclassification rate, achieved quality, and verification cost. We use the criterion of predictive validity for assessment since we determine the absolute worth of a predictive model by looking at its statistical significance. A model that does not meet the criterion of predictive validity should be rejected. The remaining criteria can be used to perform a cost/benefit analysis on the models which have passed the predictive validity criterion. Depending on the project priorities, a software engineering manager can compare the accepted models and make different choices. If software quality is a critical requirement, he might choose the predictive model that identifies most of the high-risk components, even if a great part of the verification effort is wasted because of wrong predictions. On the other hand, if the effort must be minimized, he could choose the predictive model that requires the lowest verification effort, even if the quality achieved at the end of the verification is lower than for other models.

4.1 Predictive Validity

Predictive validity is the capability of the model to predict the future component behavior from present and past behavior. The present and past behavior are represented by data in the training set while the future behavior of components is described by data in the testing set. Having data represented by a contingency table, we apply the predictive validity by testing the null hypothesis of no association between the row variable (real risk) and the column variable (predicted risk), i.e., the predictive model is not able to discriminate low-risk components from high-risk components. The alternative hypothesis is one of general association. A chi-square (χ^2) statistic (Conover, 1971) with a distribution of one degree of freedom is applied to test the null hypothesis.

4.2 Misclassification Rate

For our predictive models, which classify components as either low risk or high risk, two misclassification errors are possible. A Type 1 error is made when a high-risk component is classified as low risk, while a Type 2 when a low-risk component is classified as high risk. It is desirable to have both types of error small. However, since the two types of errors are not independent, software engineering managers should consider their different implications. As a

result of a Type 1 error, an actual high-risk component could pass quality control. This would cause the release of a lower quality product and more fix effort when a failure happens. As a result of a Type 2 error, an actual low-risk component will receive more testing and inspection effort than needed.

In the contingency table, the number of Type 1 and Type 2 errors is given, respectively, by n_{21} and n_{12}. We use the following measures of misclassification (Schneidewind, 1994):

- Proportion of Type 1: $P_1 = n_{21}/n$;
- Proportion of Type 2: $P_2 = n_{12}/n$;
- Proportion of Type 1 + Type 2: $P_{12} = (n_{21} + n_{12})/n$.

4.3 Quality Achieved

We are interested in measuring how effective the predictive models are in terms of the quality achieved after the components classified as high risk have undergone an extra verification activity. We suppose that the verification will be so exhaustive as to find all the faults in the components that are actually high risk. So if all the high-risk components are properly classified, all defects will be removed by the extra verification, and perfect quality will be achieved. However, quality will be degraded with each high-risk component that is not identified.

We measure the criterion of achieved quality using the completeness measure (Briand et al., 1993a) which is the percentage of faulty components that have been actually classified as such by the model.

- Completeness: $C = n_{22}/n_2$.

4.4 Verification Cost

Quality is achieved by increasing the cost of verification due to an extra effort in inspection and testing for the components that have been flagged as high-risk. We measure the verification cost by using two indicators. The former inspection (Schneidewind, 1994), measures the overall cost by considering the percentage of components that should be verified. The latter wasted inspection is the percentage of components that do not contain faults but have been verified because they have been incorrectly classified.

- Inspection: $I = n_{.2}/n$;
- Wasted Inspection: $WI = n_{12}/n_{.2}$.

5. RESULTS

We applied the evaluation criteria on the testing set and analyzed the resulting data.

Table 4 shows the associations of the predictions and the real behavior of the components. The rightmost two columns show the chi-square values and the probabilities of incorrectly rejecting the null hypothesis which is incorrectly saying that there is a significant association. The most popular probability value used as a threshold to establish significance is 0.05. If p is less than 0.05, there is a significant association and it is correct to reject the null hypothesis. Since all the probability values are much higher than 0.05, we must accept the null hypothesis of no association between predicted risk and real risk.

Table 5 shows the results of comparing the predictive models to each other with respect to the remaining criteria. All the data are represented as percent-

Table 4. Assessment of Predictive Models

Modeling Techniques	χ^2	p^*
Discriminant analysis	0.244	0.621
Principal components + Discriminant analysis	0.685	0.408
Logistic regression	0.648	0.421
Principal components + Logistic regression	1.761	0.184
Logical classification model	0.215	0.643
Layered neural network	0.648	0.421
Holographic network	0.227	0.634

*p is the probability of incorrectly rejecting the null hypothesis of no association between predicted and real risk.

Table 5. Comparison of Predictive Models

	Misclassification rate			Achvd quality	Verification cost	
Modeling Techniques	P_1	P_2	P_{12}	C	I	WI
Discriminant analysis	28.21	25.64	53.85	42.11	46.15	55.56
Principal comp.+ Discriminant analysis	15.38	41.03	56.41	68.42	74.36	55.17
Logistic regression	28.21	28.21	56.41	42.11	48.72	57.89
Principal comp.+ Logistic regression	12.82	46.15	58.97	73.68	82.05	56.25
Logical classification model	25.64	20.51	46.15	47.37	43.59	47.06
Layered neural network	28.21	28.21	56.41	42.11	48.72	57.89
Holographic network	25.64	28.21	53.85	47.37	51.28	55.00

ages. The first three columns of data show the misclassification rates. Recall that a random prediction should have a proportion of Type 1 + Type 2 errors of 50%, and proportions of Type 1 and Type 2 errors of 25% each. In this study, the proportions of Type 1 + Type 2 errors range between 46 and 59%. Discriminant analysis and logistic regression, when applied in conjunction with principal component analysis, have high proportions of Type 2 error (respectively, 41 and 46%) in comparison with the proportions of Type 1 error (respectively, 15 and 13%). On the other hand, the other models have balanced values of Type 1 and Type 2 error, ranging between 20 and 28%.

Looking at the achieved quality and verification cost results, it is possible to better interpret the misclassification results. The highest values of quality correspond to the models built with principal component analysis followed by either discriminant analysis or logistic regression (completeness is, respectively, 68 and 74%). However, these high values of achieved quality are obtained by inspecting the great majority of components (inspection is, respectively, 74 and 82%), thus wasting more than one half of the verification effort (wasted inspection is, respectively, 55 and 56%). None of the other models discovers even half of the high-risk components and waste nearly half or more of the verification effort.

6. RELATED WORK

Some empirical studies, relevant to this work, are summarized in the following.

Briand et al. (1993b) presented an experiment for predicting high-risk components using two logical classification models (Optimized Set Reduction and classification tree) and two logistic regression models (with and without principal components). Design and code metrics were collected from 146 components of a 260 KLOC system. OSR classifications were found to be the most complete (96%) and correct (92%), where correctness is the complement of our wasted inspection. The classification tree was more complete (82%) and correct (83%) than logistic regression models. The use of principal components improved the accuracy of logistic regression, from 67 to 71% completeness and from 77 to 80% correctness.

Porter (1993) presented an application of classification trees to data collected from 1400 components of six FORTRAN projects in a NASA environment. For each component, 19 attributes were measured, capturing information spanning from design specifications to implementation. He measured the mean accuracy across all tree applications according to completeness (82%) and to the percentage of components whose target class membership is correctly identified (72%), that is, the complement of the Proportion of Type 1 and Type 2 error.

Munson and Khoshgoftaar (1992) detected faulty components by applying principal component analysis and discriminant analysis to discriminate between programs with less than five faults and programs having five or more faults. The data set included 327 program modules from two distinct Ada projects of a command and control communication system. They collected 14 metrics, including Halstead's metrics together with other code metrics. Applying discriminant analysis with principal components resulted in correctly recognizing 79% of the modules with a total misclassification rate of 5%.

Khoshgoftaar et al. (1996) again applied principal component analysis and discriminant analysis to identify fault-prone modules (modules with five or more faults) in a large telecommunications system. They used 1980 modules consisting of 194 new, 917 reused but modified, and 869 reused without modification. For product metrics, they used three call-graph-based metrics and six control-flow-graph-based metrics. They also used reuse information as additional categorical predictor variables. They classified 38.0% of the modules as fault prone when using product metrics only, and 31.4% when including the reuse variables too. The real percentage of faulty modules was 12.1%. The Proportion of Type 1 error (Type II misclassification rate in their study) was 21.25% with product metrics only, and 13.75% with also reuse variables. The Proportion of Type 2 error (Type I misclassification rate in their study) was, respectively, 32.4% and 23.8%. Finally, the Proportion of Type 1 and Type 2 error combined was 31.1% using only product metrics and 22.6%, including the reuse variables.

7. LESSONS LEARNED

This empirical investigation of the modeling techniques for identifying high-risk components has taught us three main lessons:

- Principal component analysis does not always produce a better input for predictive models.
  In our study, we built two classification models for both discriminant analysis and logistic regression. The first pair of models was based on the eleven original product measures, while the second pair used the three orthogonal domain metrics that had been generated from the principal component

analysis. An unexpected result of using the principal component analysis as a preprocessing step is that the improved quality was exclusively the result of classifying most of components to be high risk.

- It is not always possible to successfully predict the future behavior of software products.
  Despite the variegated selection of modeling techniques, no model satisfied the criterion of predictive validity; that is, no model was able to discriminate between components with faults and components without faults. This result is in contrast with the software measurement literature which always reports successful results in recognizing fault-prone components from product measures. The previous section provides some examples.

- Predictive modeling techniques are only as good as the data they are based on.
  The relationship between software product measures and the presence of faults cannot be considered an assumption that holds for any data set and project. An assumption is a statement that is postulated to be true without the need to be verified. Past positive findings at showing correlation between product measures and number of faults have built a confidence that this relationship is a general property. However, the underlying phenomena continue to be poorly understood, and we do not really know what findings can be reused across environments and projects. Whereas the research underlying the validation of software product measures as internal attributes of software quality is not novel, it is only within the past few years that researchers have begun to worry about a rigorous and local validation (Schneidewind, 1992; Fenton, 1994; Briand et al., 1995; Kitchenham et al, 1995; Pfleeger, 1995). Predictive models are very attractive to build, but they can be a waste of time if we rely on false assumptions instead of building a local process for selecting valid predictors.

ACKNOWLEDGMENTS

This work was partially supported by NASA under grant 01-5-26775 and the Italian MURST under the 40% project "V & V in software engineering."

We would like to thank the students from the University of Bari for providing the fault data used in this study, Aurora Lonigro and Giulia Festino for their support in processing and analyzing the data, Carolyn Seaman, and the anonymous reviewers for their suggestions and comments on a first draft of this paper. This work has also benefited from the encouragement of the participants at the 20th Annual Software Engineering Workshop, Goddard Space Flight Center.

REFERENCES

Agresti, A., *Categorical Data Analysis*, John Wiley & Sons, New York, 1990.

Boetticher, G., Srinivas, K., and Eichmann, D., A neural net-based approach to software metrics. In: *Proc. 5th Int. Conf. Software Eng. and Knowledge Eng.*, 1993, pp. 271–274.

Briand, L. C., Thomas, W. M., and Hetmanski, C. J., Modeling and managing risk early in software development. In: *Proc. 15th Int. Conf Software Eng.*, 1993a, pp. 55–65.

Briand, L. C., Basili, V. R., and Hetmanski, C. J., Developing Interpretable Models with Optimized Set Reduction for Identifying High-Risk Software Components. *IEEE Trans. Software Eng.*, 19 (11), 1028–1044 (November 1993b).

Briand, L., El Eman, K., and Morasca, S., Theoretical and empirical validation of software product measures, ISERN-95-03, International Software Engineering Research Network, 1995.

Conover, W. J., *Practical Nonparametric Statistics*, Wiley, New York, 1971.

Dillon, W. R., and Goldstein, M., *Multivariate Analysis: Methods and Applications*, John Wiley & Sons, New York, 1984.

Esteva, J. C., and Reynolds, R. G., Identifying Reusable Software Components by Induction, *Int. J. Software Eng. and Knowledge Eng.*, 1 (3), 271–292 (1991).

Fenton, N. E., Software Measurement: A Necessary Scientific Basis. *IEEE Trans. Software Eng.*, 20 (3), 199–206 (March 1994).

Humphrey, W. S., Using a Defined and Measured Personal Software Process. *IEEE Software*, 13 (3), 77–88 (May 1996).

Jensen, G., Quality control in manufacturing based on fuzzy classification. In: *Frontier Decision Support Concepts* (V. L. Plantamura, B. Soucek, G. Visaggio, eds.), John Wiley & Sons, New York, 107–118, 1994.

Karunanithi, N., Whitley, D., and Malaiya, Y. K., Prediction of software reliability using connectionists models, *IEEE Trans. Software Eng.*, 18 (7), 563–573 (July 1992).

Karunanithi, N., Whitley, D., and Malaiya, Y. K., Using Neural Networks in Reliability Prediction. *IEEE Software*, 53–59 (July 1992).

Khoshgoftaar, T. M., Munson, J. C., Bhattacharya, B. B., and Richardson G. D., Predictive Modeling Techniques of Software Quality from Software Measures. *IEEE Trans. Software Eng.*, 18 (11), 979–987 (November 1992).

Khoshgoftaar, T. M., Lanning, D. L., and Munson, J. C., A comparative study of predictive models for program changes during system testing and maintenance. In: *Proc. Conf. Software Maintenance* 1993, pp. 72–79.

Khoshgoftaar, T. M., Allen, E. B., Kalaichelvan, K. S., and Goel, N., Early Quality Prediction: A Case Study in

Telecommunications. *IEEE Software*, 65–71 (January 1996).

Khoshgoftaar, T. M., and Szabo, R. M., Improving code churn prediction during the system test and maintenance phases. In: *Proc. of the Int. Conf. Software Maintenance* 1994, pp. 58–67.

Kitchenham, B., Pfleeger, S. L., and Fenton, N., Towards a Framework for Software Measurement Validation *IEEE Trans. Software Eng.*, 21 (12), 929–943 (December 1995).

Lanubile, F., and Visaggio, G., Quality evaluation on software reengineering based on fuzzy classification. In: *Frontier Decision Support Concepts* (V. L. Plantamura, B. Soucek, G. Visaggio, eds.), John Wiley & Sons, New York, 119–134, 1994.

Munson, J. C., and Khoshgoftaar, T. M., The Detection of Fault-Prone Programs. *IEEE Trans. Software Eng.*, 18 (5), 423–433 (May 1992).

Pfleeger, S. L., Maturity, Models, and Goals: How to Build a Metric Plan. *J. Syst. Software*, 31, 143–155 (1995).

Porter, A. A., Developing and analyzing classification rules for predicting faulty software components. In: *Proc. 5th Int. Conf. Software Eng. and Knowledge Eng.* 1993, 453–461.

Porter, A. A., and Selby, R. W., Empirically Guided Software Development Using Metric-Based Classification Trees. *IEEE Software*, 46–54 (March 1990).

Press, S. J., and Wilson, S., Choosing Between Logistic Regression and Discriminant Analysis. *J. of the American Statistical Association*, 73, 699–705 (1978).

Quinlan, J. R., Induction of decision trees. *Machine Learning*, 1 (1), 81–106 (1986).

Quinlan, J. R., *C4.5: Programs for Machine Learning*, Morgan Kauffman Publishers, San Mateo, CA, 1993.

Rumelhart, D., Hinton, G., and Williams, R., Learning internal representations by error propagation. In: *Parallel Distributed Processing*, Vol. I, MIT Press, Cambridge, MA, 318–362, 1986.

SAS Institute Inc., *SAS/STAT User's Guide*, Version 6, Fourth Edition, 2 vols., Cary, NC: SAS Institute Inc., 1989.

Schneidewind, N. F., Methodology for Validating Software Metrics. *IEEE Trans. Software Eng.*, 18 (5), 410–422 (May 1992).

Schneidewind, N. F., Validating Metrics for Ensuring Space Shuttle Flight Software Quality. *Computer*, 50–57 (August 1994).

Selby, R. W., and Porter, A. A., Learning from Examples: Generation and Evaluation of Decision Trees for Software Resource Analysis, *IEEE Trans. Software Eng.*, 14 (12), 1743–1757, (December 1988).

Soucek, B., Sutherland, J., and Visaggio, G., Holographic decision support system: credit scoring based on quality metrics. In: *Frontier Decision Support Concepts* (V. L. Plantamura, B. Soucek, G. Visaggio, eds.), John Wiley & Sons, New York, 171–182, 1994.

Sutherland, J., A Holographic Model of Memory, Learning and Expression. *Int. J. Neural Syst.*, 1 (3), 259–267 (1990).