A Pattern Recognition Approach for Software Engineering Data Analysis

Lionel C. Briand, Victor R. Basili, *Fellow, IEEE,* and William M. Thomas, *Member, IEEE*

***Abstract*—In order to plan, control, and evaluate the software development process, one needs to collect and analyze data in a meaningful way. Classical techniques for such analysis are not always well suited to software engineering data. In this paper we describe a pattern recognition approach for analyzing software engineering data, called optimized set reduction (OSR), that addresses many of the problems associated with the usual approaches. Methods are discussed for using the technique for prediction, risk management, and quality evaluation. Experimental results are provided to demonstrate the effectiveness of the technique for the particular application of software cost estimation.**

***Index Terms*— Classification, data analysis, empirical modeling, machine learning, pattern recognition, quality evaluation, risk assessment, software development cost prediction, stochastic modeling.**

I. Introduction

MANAGING a large scale software development requires the use of quantitative models to provide insight and support control based upon historical data from similar projects. Basili has introduced a paradigm of measurement based, improvement-oriented software development, called the improvement paradigm [2], [3]. This paradigm provides an experimental view of the software activities with a focus on learning and improvement, implying the need for quantitative approaches for the following uses:

- building predictive models of the software process, product, and other forms of experience (e.g., effort, schedule, and reliability) based upon common characteristics;
- recognizing and quantifying the influential factors (e.g., personnel capability, storage constraints) on various issues of interest (e.g., productivity improvement, effort estimation) for the purpose of understanding and controlling the development;
- evaluating software products and processes from different perspectives (e.g., productivity, fault rate) by comparing them to projects with similar characteristics.

Classical techniques for data analysis have limitations when used on software engineering data. In this paper we present a new data analysis technique, based on both machine learning principles and statistics, designed to overcome some of these limitations.

The paper is organized as follows. In Section II we discuss the needs and constraints in building effective models for the software development environment. Section III discusses some of the more common model construction approaches. In Section IV we present our technique for analyzing software engineering data, called optimized set reduction (OSR). Section V explains how this approach may ultimately be used not only for prediction, but also for risk analysis and quality evaluation. In Section VI, experimental results are provided to demonstrate the effectiveness of the approach for the particular application of cost estimation modeling.

Manuscript received October 1, 1991; revised August 1, 1992. This work was supported in part by NASA Grant NSG 5123, and by AFOSR 90-0031. Recommended by R. Selby and K. Torii.

The authors are with the Institute for Advanced Computer Studies and Department of Computer Science, University of Maryland, College Park, MD 20742.

IEEE Log Number 9203765.

II. Requirements for an Effective Modeling Process

Based upon the constraints associated with the data and the analysis procedures, we generate a set of requirements for model building approaches. In the text that follows, we refer to the variable to be assessed as the "Dependent Variable" (e.g. productivity, fault rate) and the variables explaining the phenomenon as "Independent Variables" or "explanatory variables" (e.g. personnel skills, data base size).

2.1. Constraints Related to Software Engineering Data

Model building to support software engineering can be difficult due to the following inherent constraints.

- C1: It is very difficult to make valid assumptions about the form of the functional relationships between variables and the probability distributions of variables on their ranges. Therefore, the capabilities of classical statistical approaches seem limited.
- C2: In the field of software engineering, we are often faced with data sets that contain both continuous and discrete explanatory variables (e.g., lines of code, team experience, application domain). There are several statistical modeling techniques that deal with these different types of variables (e.g., least-square regression versus ANOVA) [1], [11]. However, building models (e.g., cost models) requires the use of both types of variables.
- C3: Because of the lack of precision in the data collection process and because of unexpected events (e.g., unstable requirements) in the development process, extreme/atypical explanatory/dependent variable values occur. In software engineering, it is usually the case that a large number of factors (which vary widely from one environment to another) can affect the dependent vari-

ables. Therefore, information that could help to validate and understand atypical vectors is not always available. Also, the fact that we are working with a large number of explanatory variables makes it difficult to distinguish between those vectors that are atypical (i.e., outliers in statistics) and those that actually represent the main trends of the data set.

- C4: The interdependencies of explanatory variables can affect the understandability of models, but are not always harmful to their accuracy (e.g., regression models [11]). On the other hand, very complex interdependencies may exist. For example, the structural complexity of a piece of software can be a very significant factor of productivity if the programmer is inexperienced with the application domain and programming language. However, complexity has a milder impact on productivity if the programmer is experienced. Thus, the impact of complexity on productivity is dependent on the ordinal explanatory variable programmer experience.
- C5: An independent variable may be a much stronger factor on a particular part of its range/value domain, a phenomenon known in statistics as "heteroscedasticity." It is easy to see that the accuracy of a model that does not consider such issues may be significantly affected and the model may not provide pieces of information very important to decision making.
- C6: Missing information is a common problem in software measurement. There are several causes of this: limited budget for data collection, collecting data is time consuming, collecting some of the data is technically impossible (e.g., no tool) or not humanly desirable (e.g., engineer's work evaluation), and our lack of understanding of the problem, due to the newness of the software measurement field and the wide variability from one development environment to another.All of the above can generate incompleteness in the data collection process. The last issue has been partially addressed by [3], [4]. For example, suppose we wish to predict project productivity according to collected physical features of the system and predefined quality requirements. Also, suppose we do not have any information about team experience related to the programming environment and the application domain. This information might be somewhat irrelevant (i.e., the variance of the prediction is small) if the structural complexity of the software and the required reliability are low. However, if high reliability on a complex software system is expected, then low experienced people are likely to generate large schedule and/or budget slippages and make any prediction based exclusively on other criteria meaningless.

2.2. Requirements to Alleviate These Constraints

Matching the constraints, we can define requirements for effective data analysis or empirical modeling procedures as follows.

- R1 [matches C1]: The data analysis procedure should avoid assumptions about the relationships between the variables and the probability density distribution on the independent and dependent variable ranges.
- R2 [C2]: The modeling process needs to capture the impact of, and be effective in, integrating all explanatory variables regardless of their type, i.e., discrete, continuous. Also, the constructed models need to provide a consistent way of interpreting each variable's effects on the dependent variable.
- R3 [C3]: It is preferable to use modeling techniques that are robust to outliers, i.e., a small number of data vectors cannot change dramatically the characteristics of the model.
- R4 [C4]: We need a modeling technique that accounts for interdependencies among the explanatory variables, i.e., that produces, despite interdependencies, a readable and interpretable model. The modeling technique must address the issue of interdependencies by providing the context within which each parameter of the model appears to be a relevant and significant piece of information.
- R5 [C5]: Heteroscedasticity should be addressed by determining on which part of its range/value domain an independent variable strongly affects the dependent variable of interest.
- R6 [C6]: Missing information obviously reduces our ability to predict and learn. We need to better understand whether or not the lack of a piece of data is an obstacle to assessment. This means that we need a model that not only generates predictions but provides some insight into the reliability of each individual prediction, rather than a global reliability of the entire model.

III. Current Approaches to Empirical Modeling

Two main approaches have been used in software engineering to form stochastic multivariate models: multivariate regression analysis and classification trees. In this section we present a review of these techniques outlining their strengths and weaknesses. This discussion will be used as a basis for introducing and justifying the new approach that we present in this paper.

3.1. Regression Analysis

Regression analysis can be very effective for prediction (least-square regression) or classification (logistic regression) when some prerequisite conditions are met: 1) the explanatory variables are independent, 2) the functional form of the regression equation is well approximated, 3) most of the explanatory variables are continuous (interval, ratio level), and 4) the error term is constant in the space defined by the dependent variable and the explanatory variables (i.e., homoscedasticity [11]). These conditions are rarely met in software engineering data sets, making the use of these models somewhat difficult (in terms of prediction and interpretation). However, regression analysis is well known and numerous tools are available to support this technique and facilitate the interpretation of the models. Several problems related to how regression analysis deals with the issues mentioned in Section II are presented below.

- The regression techniques use "dummy" variables in order to deal with discrete variables. Each possible value of the variable becomes a parameter whose the value is set to 1 (or another constant above 0) when true or 0 otherwise. Thus, the number of parameters that can potentially be integrated in the model may increase very rapidly.
- It has been shown that outliers can strongly affect the regression modeling process and thereby the resulting regression equation. Identification and exclusion of these overly influential data points is a complex process in an N-dimension sample space (i.e., multivariate context).
- A linear regression model would generate a unique coefficient for each explanatory variable (more complex equations including combinations of parameters appear difficult to use in practice). Because of the dependence of the complexity-productivity model on programmer experience, a single coefficient cannot sufficiently represent the influence of complexity on productivity. Also, using more complex functional forms would be difficult since we usually have a poor understanding of the phenomena we are studying.
- A model level value of goodness of fit like the coefficient of determination in least-squares regression analysis is not suitable to deal with partial information because it fails to yield an individual reliability measure for each prediction. Therefore, in our context it appears difficult for regression analysis to deal with partial information since we cannot differentiate cases where the available information is sufficient to get accurate estimations.
- Regression analysis can be very effective when the user is dealing with a small number of independent explanatory variables. However, in software engineering data sets this is usually not the case. Therefore, the model creation process becomes unstable, i.e., various variable selection heuristics (e.g., backward, forward) can yield very different models [11], and the removal of a variable can dramatically change the resulting equation. Also, the interpretation of such models based on regression equations and correlation matrices appears complex in practice.

3.2. *Classification Trees*

More recently, the use of classification tree techniques [6], [17] has appeared in the software engineering literature [16], [18]. These techniques generate partition trees based on a historical data set describing past experiences of interest (e.g., characteristics/attributes of past software developments). They produce interpretable classification models to help software engineers and managers take remedial actions based upon quantitative models.

In order to analyze the improvements offered by classification trees over multiple regression techniques, consider the simple partition tree example presented in Fig. 1. In this tree, the historical data set is represented by Node 0. This node is successively partitioned (according to a heuristic process described in [17]) into exclusive subsets until they

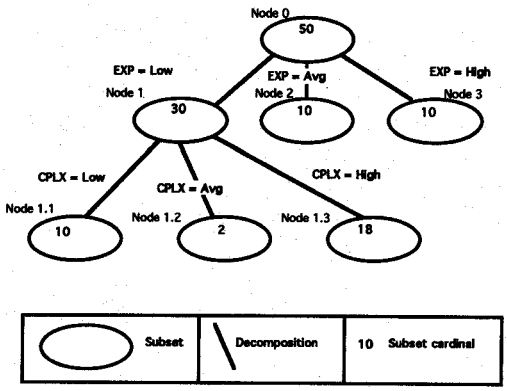


Fig. 1. Example of classification tree.

reach a predefined termination criterion (usually a certain distribution on the dependent variable range [18]) which stops the partition process. Each leaf of the tree contains a set of similar historical experiences which are described by a set of similar characteristics (e.g., explanatory variables such as team experience (EXP), design complexity (CPLX) in Fig. 1). To assess an unknown characteristic (e.g., productivity) of a future event (e.g., a new software development), we find the leaf of the tree which also characterizes the new event. By examining the distribution on the productivity range within the leaf, we can estimate the productivity of the new development. For instance, to assess the productivity of a project characterized by low team experience and low project complexity, the distribution on the productivity range of all past developments within Node 1.1 will be analyzed. Then, the expected value of the distribution may be used as a prediction.

Classification trees represent, in our field of application, a major improvement over regression techniques based upon the following advantages.

- They deal with discrete variables in a straightforward way (e.g., experience (EXP) in Fig. 1). This addresses the issue outlined in requirement R3.
- The modeling process can be effectively automated (see [18]). This is particularly important in our application domain where exploratory analysis is more common than confirmatory analysis.
- The tree structure is a very intuitive and easy-to-interpret way of representing data analysis results. It fulfils our need to understand in order to take sound corrective actions.
- Interdependencies between explanatory variables are taken into account to some extent, i.e., each partition is formed in a certain context defined by the set on which the partition is performed. In Fig. 1, complexity (CPLX) seems to be a relevant variable in the context where EXP is low. However, CPLX is not selected whenever EXP is average or high; therefore, another characteristic can be selected. This addresses our need to understand in context the impact of the explanatory variables on the dependent variable.

However, several issues still need to be considered.

- Continuous variable ranges need to be divided into in-

tervals (e.g., the CPLX range in three intervals: Low, Average, High). This can be done using cluster analysis techniques [11] or heuristics to optimize the interval homogeneity [12], [15] .Despite interesting preliminary results, the issue of class definition needs further investigation.

- The model assumes that given a particular explanatory variable X, the model variance on the dependent variable range is about constant for all values of X (i.e., X is an equally good or bad predictor regardless of its corresponding value or category). Let us assume that Node 1.3 in Fig. 1 shows a very homogeneous distribution of low productivity projects (i.e., the variance of the population on the productivity range is significantly lower than in Node 1 as a whole). However, assume Node 1.1 encompasses a project population of variance equal to that of Node 1. This classification tree strongly implies that low team experience and high project complexity lead to low productivity regardless of the other productivity factors. However, when complexity is low, productivity is strongly affected by factors not present in the data set. This is one inherent problem associated with tree-structured models: at each level of partition, they select the most relevant variable, which assumes the variable (e.g., CPLX) to be equally relevant regardless of its value (e.g., Low or High). In our example, CPLX is relevant (i.e., significantly lowers the population variance/entropy/heterogeneity) whenever its value is High. However, a value Low (Node 1.1) does not seem to improve the homogeneity of Node 1's project population. (This is the heteroscedasticity issue.) The fact that CPLX was selected as an attribute under Node 1 means that CPLX showed the best average/weighted homogeneity over all of Node 1's generated subset populations. This may be easily explained considering that the population in Node 1.3 is much larger than in Node 1.1. In fact, another explanatory variable should have been selected in the cases where CPLX was low in order to converge as fast as possible (this is particularly important for small data sets) to an optimal subset homogeneity. As we see here, this is not possible in the context of a classification tree. In summary, the partition tree structure forces the model to perform nonrelevant subset partitions and thereby 1) slows down the convergence toward an optimal subset homogeneity, 2) includes nonrelevant pieces of information in the model.
- A tree structure may force the modeling process to ignore some variables that could be useful for some predictions. In Fig. 1, Node 1.2 only contains 2 elements (e.g., projects). Therefore, in a situation where EXP is low and CPLX is average, the tree is unusable for prediction (Node 1.2's population is not statistically significant). In this particular case, it is necessary to use another variable yielding significant population subsets. On the other hand, we have seen that the variable CPLX yields an optimal homogeneity whenever the complexity is high. Therefore, removing CPLX from the model is ignoring an important piece of information for a subset of the projects. This may even affect the accuracy of the model. Unfortunately, this dilemma is difficult to solve within the context of a classification tree.

IV. A Pattern Recognition Approach for Analyzing Data

Based on the specific needs described above and the weaknesses of currently used techniques, our goal has been to combine the expressiveness of classification trees with the rigor of a statistical basis. To do so, we have developed an approach called OSR. Rather than generating a partition tree, this process generates a set of patterns relevant to the object to be predicted, or, if one is interested in understanding the general trends, relevant to an entire data set.

4.1. Basic Principles

Assume we want to assess a particular characteristic of an object (e.g., the fault density of a component). We refer to this characteristic as the dependent variable (Y). The object is represented by a set of explanatory variables which describe the software component (called X's). These variables can be either continuous or discrete. For example, a software component may be described by two X's, its cyclomatic complexity (continuous) and the type of its function (discrete). Also, assume we have a historical data set containing a set of pattern vectors that contain the previously cited X's plus an associated actual Y value. We will call the X's portion of the pattern vector a measurement vector.

The goal of the OSR algorithm is to determine which subsets of experiences (i.e., pattern vectors) from the historical data set provide the best characterizations of the object to be assessed. That is, we try to determine which subsets of the data set yield the "best" probability distributions on the Y range. A good probability distribution on the Y value domain is one that a concentrates a large number of pattern vectors in either a small part of the range (Y is continuous) or a small number of dependent variable categories (Y is discrete). One of the commonly used probability distribution evaluation functions is the information theory entropy H. Alternative probability distribution evaluation functions are discussed in [14], [17], and [18]. Each of the subsets yielding "optimal" distributions, referred to as optimal subsets, are characterized by a set of conditions, or predicates that are true for all pattern vectors in that subset. Each set of predicates characterizing a subset is called a pattern. Fig. 2 shows an example of a pattern and its associated probability distribution in the data set. The pattern is composed of three predicates where the dependent variable to be assessed is "development productivity." Fig. 2 shows that if these predicates (i.e., ComPLeXity = Nominal, RELiabilitY=Low, DATA base size = High) are true for a project, then its productivity is most likely to be in the second productivity class.

4.2. Formal Definition of the OSR Process

We want to identify optimal subsets in the historical data set. We can formalize the process using set theory and predicate calculus by defining the function Opt. Let us assume we

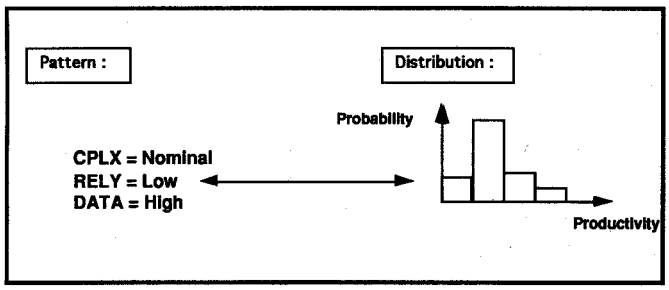


Fig. 2. Example of a pattern and its associated probability distribution.

have a set of m explanatory variables $\{x_1, x_2, ...x_m\}$and a corresponding set of explanatory variable value domains $\{EV_1, EV_2, ..., EV_m\}$. Let us define the measurement vector domain to be $MV = \times\ EV_i,\ i \in (1...m)$. The dependent variable value domain (DV) may be seen as a set of classes that can be either intervals or categories. Therefore, the value domain of the pattern vectors in the data set can be represented as $PV = DV \times MV$. Let PVS be a set of pattern vectors representing the historical data set $PVS \subseteq PV$. A predicate is defined as an X_i and its corresponding explanatory variable value.

Definition 1: Let PSS be a subset of PVS and let the measurement vector mv describe the object to be assessed. VALID(PSS, mv) is true if mv contains at least one predicate that is true for all the pattern vectors in the set PSS. This indicates that there is at least one predicate that is true for mv and for all pattern vectors in PSS, and thus PSS is a valid candidate subset of PVS in an OSR decomposition for mv.

$$PSS \subseteq PVS \wedge mv \in MV \wedge \exists i\ \in \{1...m\}$$
$$\text{such that } \forall\ pv \in PSS(mv(i){=}pv(i)) \Rightarrow \text{VALID}(PSS, mv)$$

Definition 2: $TC(PSS, PVS)$ is true if the two data sets PVS and PSS do not show a statistically significant difference in distribution on the DV range. This is may be evaluated by performing statistical inference tests for comparing distributions. We currently use a binomial test for proportions since it does not have any requirement to be applicable (e.g., minimum expected frequencies like the Chi-square test of independence) [9]. For each dependent variable class, the probability that proportions in PSS and PVS differ by chance is calculated. If for at least one of the classes, this probability is below a level of significance TC defined by the user, then we reject the hypothesis that the two distributions are identical. TC stands for *termination criterion* because the OSR process will be terminated if the condition defined by TC is true.

Definition 3: $EMIN(PSS_1, PVS)$ is true if PSS_1 is one of the subsets of PVS yielding a minimal normalized entropy H upon all statistically significant subsets of pattern vectors (e.g., a one-vector subset has a minimal entropy but it is not a statistically significant subset and therefore is not relevant here).

$$\Big(PSS_1 \subseteq PVS \wedge \neg TC(PSS_1, PVS)\Big)$$
$$\wedge \Big(\forall PSS_2 \subseteq PVS\big(\neg TC(PSS_2, PVS) \wedge H(PSS_1)$$
$$\leq H(PSS_2)\big)\Big) \Rightarrow EMIN(PSS_1)$$

where

$$H(PSS) = \sum_{d \in DV} -p(PSS, d)\log_{|\mathrm{DV}|}p(PSS, d)$$

where $p(PSS, d)$ is the *a priori* probability that a vector that is an element of PSS has a dependent variable value belonging to the dependent variable class d.

Definition 4: Opt(PVS, mv) is a function yielding a set of optimal pattern vector subsets. The subsets are optimal in the sense that they are the subsets with the lowest entropies in the collection of valid subsets.

$$\text{Opt}(PVS, ms) = \{PSS \subseteq PVS \mid \text{VALID}(PSS, mv) \wedge EMIN(PSS, PVS)\}.$$

However, the function Opt as defined cannot be used as an algorithm to extract the optimal subsets. The most important reasons follow.

- The number of possible predicate combinations makes the search execution time prohibitive.
- We want the patterns to contain a minimal set of predicates, i.e., we want all the predicates in the pattern to have a significant impact on the resulting pattern entropy.
- We lose some information about the relative impact of the various predicates in the entropy reduction process.
- The contexts in which the various predicates appear relevant are undetermined.

Therefore, we implement a "greedy" algorithm (OSR) using the function Opt to address the above issues. The OSR algorithm can be roughly described as a three step recursive algorithm.

- Step 1: If the dependent variable is continuous, its range is divided into a set of classes according to two main factors: the required model accuracy and the size of the data set. Then, the ranges / categories of the explanatory variables are divided / clustered into classes (e.g., $Class_{i1} \ldots Class_{ij}$ for the explanatory variable X_i, such as low, medium, high for complexity) based on meaningful class creation techniques. Numerous techniques can be used to create meaningful classes (e.g., cluster analysis) [11]. However, this issue will not be addressed in this paper.
- Step 2: Select all the pattern vectors in the data set having a value for the explanatory variable X_i belonging to $Class_{ik}$, where the X_i for the object to be assessed belongs to the same class, and the subset characterized by the predicate $X_i \in Class_{ik}$ yields the minimum *statistically significant* value for H. Several subsets (characterized by different predicates) yielding "similar" minimal entropies (i.e., the similarity criterion has to be defined by the user of the algorithm) can be extracted at once. Let us call PSS_j the extracted subsets of pattern vectors.

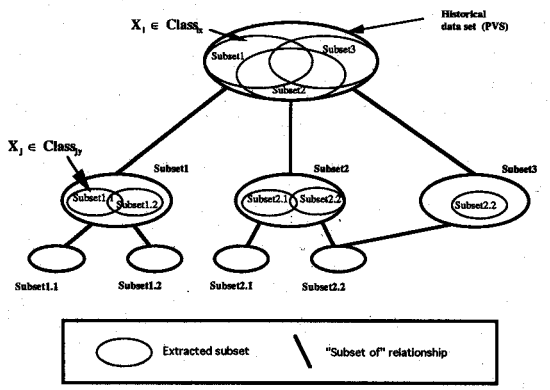


Fig. 3. OSR hierarchy.

- Step 3: Step 2 is repeated in a recursive manner on each subset PSS_j and each successive subset until a predefined termination criteria is reached. For example, the extraction process may be stopped when no selection can *significantly* improve the entropy.

This OSR algorithm can be formally specified as a two parameter recursive function where PVS is the historical data set and mv the vector describing the object to be assessed:

$$\text{OSR(PVS, mv)} = \begin{cases} \bigcup_{PSS \in Opt(PVS,mv)} (OSR(PSS, mv)) & if\ Opt(\text{PVS, mv}) \neq \emptyset \\ \text{PVS,} & \text{otherwise.} \end{cases}$$

The whole subset extraction process can be represented as a hierarchy (see Fig. 3). Note that this representation should not be confused with a partition tree since: 1) the extracted subsets are not exclusive, and 2) a subset can have several parent subsets. Each path of the hierarchy represents a pattern (e.g., Fig. 3: $X_i \in Class_{ix}$ AND $X_j \in Class_{jy}$) that is relevant to the particular prediction to be performed. As shown in Fig. 3, two patterns may yield exactly the same subset.

The extracted subsets (i.e., leaves of the hierarchy) form various probability distributions across the dependent variable range, and may show different trends. For each extracted subset, a dependent variable prediction is performed by considering the subset extracted as past experience representative of our current problem, using the probability distributions of the extracted subset. The prediction rules are based on *bayesian probability theory* and are more completely described in Section 5.1. Each pattern prediction (i.e., hierarchy leaf) is used to make a final global prediction based on predefined decision rules.

To make decisions effectively, we must evaluate the accuracy of the identified patterns. To do this, we use the entropy measure that characterizes each of the extracted subsets: the larger the entropy, the larger the uncertainty about the prediction. This approach will be illustrated by the experiments presented in Section VI.

V. Prediction, Risk Management and Quality Evaluation within the OSR Framework

Prediction, quality evaluation and risk assessment are all based on a similar quantitative approach even though they have three different purposes. The following sections demonstrate the use of OSR to address these three issues.

5.1. Prediction

For prediction, we are interested in estimating the value of one dependent variable based on the leaves of the hierarchy (extracted subsets). The dependent variable is a measurable object characteristic that is not known or accurately assessable at the time it is needed. For example, one may wish to predict the error rate expected for a particular project based on other characteristics (independent variables) that may be measured, evaluated subjectively with a reasonable accuracy, or estimated through other models. The approach to prediction varies based upon whether the dependent variable is discrete or continuous.

If the dependent variable is defined on a discrete range, then prediction becomes a classification problem, i.e., given the set of conditional probabilities associated with each dependent variable class C_i as calculated for the measurement_vector$_j$, the decision maker will most likely choose the class with the highest probability. Thus, letting $P(C_i|$ measurement_vector$_j)$ represent the probability that measurement vector measurement_vector$_j$ comes from the pattern class C_i, $P(C_i|$ measurement_vector$_j)$ can be estimated by the ratio of the number of pattern vectors falling into class C_i to the total number of pattern vectors.

But, one may not always want to choose the class with highest probability. One may choose the class based upon the loss associated with an incorrect classification. This is a Bayesian approach. In this case, a loss matrix L has to be defined by the decision maker where L_{ik} represents the loss of having chosen the strategy appropriate for C_k when the dependent variable class is actually C_i. A Bayesian classifier [19] will try to minimize the conditional average risk or loss R_k(measurement_vector$j$) $(k = 1 \ldots m)$ considering the m defined dependent variable classes.

$$R_k(\text{measurement\_vector}_j) = \sum_{i=1}^{m} L_{ik} \times P(C_i \mid \text{measurement\_vector}_j).$$

The Bayesian classifier assigns measurement_vector$_j$ to the class k with the lowest R value.

If the dependent variable is defined on a continuous range, then the notion of distance between two values on the range is meaningful. In this case, OSR provides an approximation of the actual density function P(Dependent Variable | measurement_vector$_j$) by assuming it to be uniform in each class C_i. $P(C_i \mid$ measurement_vector$_j)$ can be calculated as sum of the distances between the subset pattern vectors and the class mean for each of each of the C_i classes. Call this total distance TD_n, where n represents the class index. Note that TD_n is inversely proportional to the concentration of pattern vectors around the class mean for class n. Then calculate the probability that measurement_vector$_j$ falls in class C_n as

follows:

$$P(C_n|\text{measurement\_vector}_j) = 1-\left(TD_n/\sum_{i=1}^{m} TD_i\right)/m-1$$

where m is the number of C_i classes.

This formula assumes that the probability is inversely related to the total distance (TD_n) of the pattern vectors to the class mean. Compared to the "ratio" approach, this refines the probability calculation since it takes into account the distances between the subset of pattern vectors and the class means, not just their membership in a specific class.

The following expected value can be calculated on C_i.

$$\mu i = \frac{\text{lower\_boundary\_}C_i + \text{upper\_boundary\_}C_i}{2}.$$

In other words, the actual density function is approximated by a histogram, where each column represents the conditional probability of a particular pattern vector x that lies in a particular dependent variable class C_i. No assumption has been made with respect to the form of this probability density function. The expected value on the total dependent variable range can be approximated as follows.

$$E[\text{dependent\_variable}/\text{measurement\_vector}] = \sum_{i=1}^{m} P(C_i|\text{measurement\_vector}_j) \times \mu i.$$

This expected value can be used as an estimate of the dependent variable. The average error interval expected may be estimated by using the correlation of accuracy to entropy. This correlation will be confirmed by the experiments described in Section VI.

5.2. Risk Management

Software development organizations are interested in assessing the risk associated with management and technical decisions in order to guide and improve the development processes. Referencing [10], the risk associated with an action (e.g., software development) may be described through three dimensions:

- D1: the various possible outcomes;
- D2: the potential loss associated with them;
- D3: the chance of occurrence for each outcome.

One encounters multiple types of interdependent risks during software development (e.g., technical, schedule, cost) and this makes risk management and modeling complex. Also, the notion of risk is subjective because the associated loss strongly depends upon one's point of view. Charette [10] writes: "One individual may view a situation in one context, and another may view the exact same situation from a completely different one." According to one's goals and responsibilities, risk will be defined in different ways, in the form of various models.

To make the link between this description of risk and OSR, the following straightforward associations may be established:

- outcomes (i.e., dimension D1) and dependent variable classes;
- potential loss (i.e., dimension D2) and distance on the dependent variable range between the class mean and the planned value;
- chance of occurrence (i.e., dimension D3) and conditional probability for each dependent variable class.

To analyze risk during software development, we calculate the expected difference (distance on the range) between planned and actual values for each dependent variable representing a potential risk (e.g., schedule, effort,). Call these distances *dependent variable expected deviations*. From a decision-maker's perspective, the potential loss resulting from a decision is intrinsically a function of several dependent variable expected deviations that may be seen as a specific, subjective risk model. Therefore, a risk analysis model may be defined as a function that combines several dependent variable expected deviations, parameters (e.g., reflecting management constraints) and constants (e.g., weights). The calculation details are illustrated in an example.

Assume a budget and schedule have been imposed on a project manager by upper management requiring a specified productivity Pr be achieved to reach the management goals. From the point of view of the project manager, the risk of failure may be represented as a simple function calculating the productivity expected deviation (PED) from the conditional probabilities and dependent variable class means:

$$\text{PED} = \sum_{i=1}^{m} P(Ci|\text{measurement\_vector}_j) \times (Pr - \mu i).$$

Based on the result of this estimation, the project manager will be able assess the feasibility of the job. Some analysis can be performed by the manager to see how the risk evolves according to controllable project parameters (i.e., some of the independent variables). Also, based on such a model, a suitable risk/effort trade-off can be made to improve competitiveness on a commercial proposal. One's perspective of risk may be more complex than the previously defined function. For example, assume that a contractor wishes to define risk of financial loss if the system is delivered late and/or there are effort overruns. One can define the schedule expected deviation (SED) as the expected delay, i.e., the difference between the planned and predicted schedule and the effort expected deviation (EED) as the expected effort overrun, i.e., the difference between the planned and predicted effort expenditures. Then

$$\text{SED} = \text{Estimated\_Size}/(\text{PED} \times \text{Avg\_Team\_Size})$$

$$\text{EED} = \text{Estimated\_Size}/\text{PED}$$

where Estimated_Size is either a parameter, like Avg_Team_Siz (i.e., provided as an input by the manager), or another dependent variable (i.e., result of an OSR using some Function Point-like metrics, for example, as independent variables). So the financial loss function can be defined as a function of both variables SED and EED.

Now suppose the cost of delay on a particular contract is exponential to the delay itself. This exponential assumption is based upon predictions with respect to the delay of other projects dependent upon the completion of this project and the

resulting compensations to be given to the customer. Thus, the SED needs to be weighted by some cost per delay unit (CDU) that is an exponential function of SED. Also suppose that CEU is the average cost per effort unit, i.e., the average cost per staff hour for the specific project development team. Then we can define:

$$\text{Financial\_loss} = \text{SED} \times \text{CDU} + \text{EED} \times \text{CEU}.$$

5.3. Quality Evaluation

For any quality model, we need a baseline to be able to make sensible comparisons. For example, assume that the quality perspectives of interest (i.e., quality drivers) are productivity and fault_rate and that OSR yields clear patterns (i.e., low entropy) for both variables in the available data set. These patterns represent the expected distributions in the current development environment for the project under study.

The relationship between the actual quality factor (e.g., productivity) for some project and the expected value based on the recognized patterns provides a basis for quality evaluation. For example, suppose the actual productivity for the project falls far below the expected value based on the predicted patterns. This implies that the quality of the project with respect to productivity is low. Using the pattern as a basis of comparison, we may ask where the difference comes from. Several causes may be investigated: incomplete or inadequate data collection, some possible new variables affecting the development process, or the process quality (e.g., conformance to the process model) is quite low. Quality for a particular factor could be defined as the distance between the actual and the predicted quality value (what could have been reasonably expected) based on the recognized pattern(s):

$$\text{Quality\_deviation} = AQ - \sum_{i=1}^{m} P(Ci|\text{measurement\_vector}_j) \times \mu i$$

with AQ the actual value of the quality factor.

We can see global quality as an inherently subjective function of deviations on several quality factor ranges. These deviations can be combined to provide a global assessment of quality specific to a particular user. This idea is illustrated in the following example. If we try to include in the quality model both the fault_rate and productivity quality drivers and assume an approach similar to the productivity_deviation evaluation for calculating a Fault_deviation, then a global quality evaluation may be formalized by the following quality model. Let us define NFD as fault_deviation (i.e., fault rate deviation) normalized by the fault rate standard deviation in the available data set and NPD as the equivalent variable for Prod_deviation. Based upon these unitless deviations, we define the following quality model.

- If NFD < 0, NPD > 0, the larger |NFD * NPD| is, the better the quality.
- If NFD > 0, NPD < 0, the larger |NFD * NPD| is, the worse the quality.
- If both NFD and NPD are negative, the larger NFD/NPD is, the better the quality.
- If both NFD and NPD are positive, the smaller NFD/NPD is, the worse the quality.
- If both NFD and NPD have the same sign and NFD/NPD has a value close to 1, then quality may be assessed as average or nominal.

This particular quality model takes into account two dependent variables and illustrates that a quality model may be a subjective function of several distances on the respective dependent variable ranges. This model might be modified, according to the user perspective of quality, to change the weighting of the various dependent variables or factors, e.g., doubling the effect of fault rate in the evaluation of quality.

VI. Experimental Results

In this section we demonstrate the effectiveness of the approach by showing the results of applying OSR to the problem of cost estimation and illustrating that it can extract meaningful patterns from the available data sets. Given a collection of software projects, OSR was used to estimate productivity for each project based upon the COCOMO cost drivers. A set of generated patterns was then analyzed to determine the most influential factors affecting productivity in that data set. Finally, the OSR predictions are compared to predictions from two other effort estimation techniques to provide a basis for evaluation.

6.1. Experiment Design

The data set for the experiment comes from two sources: the COCOMO database of 63 projects [5], which is used as a learning sample, and the fifteen projects used by Kemerer in the evaluation of a collection of software cost models [12]. The COCOMO projects are a mix of business, system, control, high level interface, and scientific applications. A significant percentage of these projects have been developed in FORTRAN (38%) and a very small number in Cobol (8%). The other projects involve a variety of data processing applications primarily developed in Cobol (87%).

In what follows, we will use the term data set to refer to the COCOMO and Kemerer data sets, the test sample refers to the Kemerer data set (the sample that is used to assess the OSR model), and the learning sample refers to the data set minus the project that is being predicted. Thus, 15 optimized set reductions will be performed, one for each of the test sample pattern vectors. Each time, the pattern vector to be assessed will be removed from the whole data set to form the learning sample (77 projects). The 78-pattern vector data set is small enough to assess the capability of the approach to deal with relatively small samples.

This is similar to a situation where an organization has data on 77 projects and wants to assess a new one. However, one must consider that the 78 projects were developed in various environments, at different times, and with data collected by different people according to different procedures, e.g., project productivities lie over a very large range (i.e., from 20 to 2491 LOC/MM). Ideally we would like to have all the projects from a single environment.

TABLE I
RESULTS OF COMMERCIAL MODELS

Model	MRE
SLIM	772%
Intermediate COCOMO	583%
Function Points	103%
ESTIMACS	85%

To give a general basis for comparison, we first reference the results obtained by Kemerer with a variety of models [12]. Although the models are very inaccurate, they are uncalibrated results. The difficulties in tailoring the COCOMO cost drivers to various environments causes a loss of consistency in the data collection regardless of the analysis technique and tailoring the models to the local environment should provide a substantial improvement in accuracy.

6.2. *Predicting Development Costs Using OSR*

OSR was used to model development productivity (i.e., size/effort) using the COCOMO cost drivers as the independent variables. The size metric used is the "Adjusted Delivered Source Instruction" as defined in [5], and the effort unit is staff-months. Effort was estimated as size divided by estimated productivity. The ranges of the independent variables have been decomposed into two intervals, with the boundary being located either just above or below nominal, depending on the distribution. The productivity range was divided into five intervals containing, to the extent possible, an equivalent number of pattern vectors. The termination criterion was implemented in a very simple way: the reduction would stop if either no subset showed an improvement in entropy, or if no subset contained a minimal acceptable number of projects (which was set at eight) to ensure significance of the calculated entropy. Only one subset was extracted at each set reduction.

The results for each of the 15 data points of the test sample are shown in Table II. The seven columns contain the project number, the actual productivity, the predicted productivity, the actual effort, the predicted effort, the magnitude of relative error, and the entropy yielded by OSR. It should be noted that for calculating the predicted productivities, these 15 projects were weighted more heavily (three times) than the COCOMO projects because they are more representative of the predicted projects, COBOL programs for business applications.

Table III shows both the generated patterns and extracted subsets for each of the 15 predictions. The second column contains the selected predicates of each pattern, where L and H denote the lower and higher parts of the range, respectively. The third column contains the projects included in the extracted subsets, i.e., C01 to C63 for the COCOMO data set and K01 to K15 for the fifteen new projects.

The two data points with highest productivity (projects 4 and 9 in Table II) yield large effort overestimation. However, these two projects have a productivity far above the other 76 projects

TABLE II
OSR PREDICTIONS

Project	Actual Prod	Pred Prod	Actual Effort	Pred Effort	MRE	Entropy
1	884	299	287	846	1.94	0.63
2	491	935	82	44	0.46	0.24
3	580	674	1107	668	0.40	0.45
4	2467	643	87	333	2.83	0.06
5	1338	952	336	473	0.41	0.27
6	595	1196	84	42	0.50	0.47
7	1853	1016	23	42	0.83	0.47
8	1535	1006	130	199	0.53	0.52
9	2491	431	116	670	4.78	0.56
10	542	1028	72	38	0.47	0.06
11	983	1028	258	247	0.04	0.06
12	557	1025	231	125	0.46	0.06
13	1028	1035	157	155	0.01	0.06
14	667	1070	247	154	0.38	0.27
15	881	964	70	62	0.11	0.06

TABLE III
OSR PATTERNS AND EXTRACTED SUBSETS

Project	Pattern	Extracted Subset
1	RELY=H VEXP=H DATA=L	C10,C11,C12,C16,C17,C19,C22,C23,C27,C33,C34,C58, K11
2	STOR=L PCAP=H AEXP=H	C03,C05,C39,C41,C47,C49,C61,C63, K04,K10,K12,K13,K15
3	VIRT=L MODP=H RELY=L	C03,C07,C21,C38,C39,C41,C42,C61, K14
4	STOR=L ACAP=H PCAP=H	C03,C39,C47,C49,C61,C63, K10,K11,K12,K15
5	STOR=L ACAP=H VIRT=L	C03,C38,C39,C55,C61, K04,K10,K12,K14
6	DATA=H TURN=L	C03,C21,C25, K07,K08,K09,K12,K13,K14
7	DATA=H TURN=L	C03,C21,C25, K06,K08,K09,K12,K13,K14
8	STOR=L TURN=L TIME=L CPLX=L	C03,C07,C36,C41,C49,C55, K04,K05,K12,K14,K15
9	VIRT=L STOR=H	C19,C21,C42,C43,C44,C45,C46, K01,K03
10	STOR=L ACAP=H PCAP=H	C03,C39,C47,C49,C61,C63, K04,K11,K12,K15
11	STOR=L ACAP=H PCAP=H	C03,C39,C47,C49,C61,C63, K04,K10,K12,K15
12	STOR=L ACAP=H PCAP=H	C03,C39,C47,C49,C61,C63, K04,K10,K11,K15
13	STOR=L PCAP=H TURN=L	C03,C41,C47,C49,C61,C63, K04,K12,K15
14	STOR=L ACAP=H VIRT=L	C03,C38,C39,C55,C61 K04,K05,K10,K12
15	STOR=L ACAP=H PCAP=H	C03,C39,C47,C49,C61,C63, K04,K10,K11,K12

of the data set. These productivity values might be the result of several problems: the size evaluation has not been performed according to the rules described in [5], or something occurred that was not captured by the 15 COCOMO cost drivers. More information on these particular projects is needed to make a more thorough analysis. To keep these two projects from introducing noise in our results, we have performed analyses with and without these projects.

We see a clear association between MRE and entropy. There is a correlation of 0.71 between MRE and entropy (with projects 4 and 9 omitted), and 0.80 between MRE and the square of entropy. If we test the hypothesis that the MRE's are smaller in classes with low entropy, we get satisfactory levels of significance. For example, if we define two entropy intervals containing predictions of entropies below and above 0.40, we obtain a level of significance of 0.025,

TABLE IV
INDEPENDENTVARIABLE OCCURRENCES

Number of Occurrences	Cost Drivers
0	TOOL, SCED, LEXP
1	CPLX, TIME, AEXP, VEXP, MODP
2	RELY
3	DATA
4	VIRT, TURN
7	ACAP, PCAP
11	STOR

using a nonparametric Mann-Whitney U test [9]. So, the entropy of the pattern on which the productivity prediction is based is calculable and provides an assessment of the accuracy of the estimate, i.e., the lower the entropy values, the more precise the estimates. For example, if the entropy is around 0.06, then the expected accuracy should be around 22%. Finally, encouraging results (average MRE = 35%) were achieved in 85% of the cases (with projects 4 and 9 excluded).

6.3. Understanding Development Cost Using OSR

Historical data can be analyzed to provide intuition about the development environment (e.g., which variables affect productivity). This would help an organization improve its development processes and management techniques by focusing attention on influential factors in that organization.

To obtain such intuition, OSR can be run for each of the pattern vectors in the data set, using all other pattern vectors as the learning sample. Counts of the occurrences of each independent variable in the extracted subsets provides an indication of the impact of that independent variable in differentiating among the dependent variable classes. Table IV provides the number of occurrences of each independent variable used to create some decomposition for the given test sample. When a factor yields a low number of occurrences, several causes may be considered.

- The factor is quite constant in the learning sample and therefore will not help in differentiating projects.
- The factor doen't have much influence on the dependent variable studied.
- There is an interdependence between the independent variable and some more influential independent variables.

Table IV has 2 columns: the first one gives nonweighted counts of occurrences and the second shows the cost drivers matching them. Two distinct cost driver subsets may be observed that yield very different counts of occurrences (< 5, > 6). Considering the test sample size, we may only say that there is clearly a set of three very influential cost drivers for which the data collection must be as accurate and consistent as possible (ACAP, PCAP, STOR).

Based upon our analysis, having sufficient storage appears to be a very significant consideration. This makes sense for systems focusing on data processing and dealing with very large amounts of data. The data set shows that most of these fifteen projects fall in the category "high" or "very high" with respect to the cost driver DATA. The cost driver STOR appeared at the top level of decomposition in ten of the 15 OSR's performed. (In all these cases STOR was low.) The average entropy for the ten projects is much lower (i.e., 0.17) than the average entropy for the other projects (i.e., 0.52). This argues further that STOR is very significant for the predictive ability of the OSR model for this particular test sample. When STOR was high, it was not a good predictor, i.e., high storage constraints make productivity difficult to predict for this particular data set. This is an example that justifies our concern about heteroscedasticity (constraint C5).

It has been shown [5] that staff capabilities (Analyst, Programmers) play a crucial role in achieving optimal productivity. When STOR, ACAP, and PCAP were all included in the decompositions, the best entropy value was obtained (i.e., 0.06). Therefore, ACAP and PCAP seem to significantly improve the average entropy of the recognized patterns (i.e., 0.06 instead of 0.17). Moreover, all the projects where ACAP and PCAP were used had high capability teams. This shows that productivity predictability is more accurate at the higher ranges of capabilities. Table IV shows that the reliability and complexity factors have a weaker influence on productivity than expected (based on results published in [5]). However, the fact that most of the 15 projects fall in the "nominal" category and there are no extreme complexities or reliabilities present may explain the lack of significance of these parameters for this particular test sample. With respect to CPLX and RELY, results are more difficult to interpret because of the lack of variation in the test sample. However, their low significance might be explained as a consequence of the nominal reliability and complexity of the projects included. Since most of these projects are business data processing applications, this result seems to make sense. According to the counts of occurrences in Table IV, the three most influential cost drivers (i.e., ACAP, PCAP, STOR) belong to the category of the seven most influential cost drivers in the COCOMO cost driver ranking, despite the different nature of these 15 projects.

6.4. A Comparative Evaluation of the OSR Technique

To evaluate OSR in predicting productivity and effort, a comparison with more conventional techniques is provided. For the first technique, a tailored intermediate COCOMO model was built by recalculating the nominal equations based on both the Kemerer and COCOMO data sets, as described in [5]. The second technique was to estimate productivity using the COCOMO cost drivers in a regression model built with a stepwise selection process. As with the OSR experiment, each technique was applied once for each project in the test sample, using the remaining projects as a learning sample. Again, the 15 Kemerer data points were weighted three times the influence of the COCOMO projects for building the regression models.

Table V summarize the results by giving, for three entropy intervals, the average Magnitude of Relative Error (MRE) of

TABLE V
COMPARISON OF MODEL RESULTS

Entropy Class	MRE-OSR	MRE-CC	MRE-SR
$H \leq 0.06$	0.22	0.33	0.48
$0.06 < H \leq 0.47$	0.49	1.71	0.80
$H > 0.47$	1.24	0.77	1.03

the effort estimation in the respective intervals for each modeling technique, (columns MRE-OSR, MRE-CC for calibrated COCOMO and MRE-SR for stepwise regression).This provides some insight into the correlation between the accuracy of the effort estimation and the entropy.

Comparing the results of the OSR and regression-based techniques leads to several observations. First, for this data set, the OSR technique provides more accurate predictions than either the tailored COCOMO model or the stepwise regression model. For ten of the 15 projects, the prediction of the OSR model was more accurate than that of both these regression models. If outliers are not removed, the two regression based models had an average MRE of 207% and 116%, respectively, while the OSR model had an average MRE of 94%. If projects 4 and 9 (which had extremely high productivities) are not considered, the MRE for the regression models becomes 104% and 72% respectively, while the OSR model is 50%.

The results for OSR are significantly better than for the calibrated COCOMO model in the lower entropy range (i.e., below 0.50 where patterns have actually been recognized). In fact, a Wilcoxon T test [9] comparing pair by pair the MRE of the two techniques shows a significant difference in MRE: $p = 0.03$. The difference between the MRE's yielded by the OSR and stepwise regression models show less significance but can be considered satisfactory considering the size of the test sample: $p = 0.09$.

Among the higher entropies, all techniques produce relatively poor predictions with no statistically significant differences. For OSR, this result was expected, since poor entropy implies that no significant pattern has been found, resulting in a prediction based on a wide probability distribution, which is likely to yield a wide range of results.

VII. CONCLUSIONS

The OSR was developed to address issues related to building multivariate stochastic models in order to better plan, control, and evaluate the software development process. The procedure has so far exhibited the following positive characteristics.

- It makes no assumptions with respect to probability density functions on the dependent and independent variable ranges. It does not attempt to fit data to predefined distributions; rather, it uses the data to approximate the actual distribution (i.e., patterns). Also, no particular mathematical relationship between the dependent variable and independent variables needs to be assumed. Thus OSR seems to fulfill R1.
- It handles discrete and continuous independent variables in a consistent way and therefore meets R2.
- All pattern vectors have the same weight when calculating a subset entropy. Therefore, no vector or small subset of vectors can have a dramatic impact on the structure of the patterns generated. The technique fulfills requirement R3.
- During the optimized set reduction process, each independent variable is evaluated in the context defined by the previously selected predicates. Thus, to some extent, interdependencies among explanatory variables are taken into account when building the patterns (requirement R4)
- Since a tree structure is not imposed, homoscedasticity is not assumed when extracting the patterns, i.e., an independent variable may be a significant predictor only within a certain part of its range/value domain. Requirement R5 is therefore addressed.
- It allows an estimation of accuracy for each prediction so we can answer the question: Is the prediction usable? This fulfills R6. When relevant independent variables are not available at the time of the prediction, OSR still allows a prediction to be made; however, OSR will provide a warning if the prediction is expected to be poor. Other techniques provide model-level warnings (such as a low R-square for regression techniques), rather than individual prediction evaluations, like OSR.

Our current research tackles the issue of providing easily interpretable patterns so that management decisions/corrective actions can be taken during software development based on empirical quantitative models. Mechanisms for simplifying patterns and a framework for supporting interpretation have been developed. Techniques for dealing more effectively with partial information are being investigated. OSR is currently being applied to a large variety of modeling problems and thereby being evaluated on other data sets [7], [8]. A tool supporting the OSR approach is being developed at the University of Maryland as a part of the TAME project [3].

VIII. APPENDIX: ACRONYMS

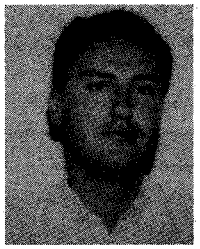
- MRE: Magnitude of Relative Error
- OSR: Optimized Set Reduction
- COCOMO cost drivers:
  - ACAP: Analyst Capability
  - AEXP: Application Domain Experience
  - CPLX: Complexity
  - DATA: Data Base Volume
  - LEXP: Programming Language Experience
  - PCAP: Programmer Capability
  - MODP: Use of Modern Programming Practices
  - RELY: Reliability
  - SCED: Schedule Constraints
  - STOR: Storage Constraints
  - TIME: Timing Constraints
  - TOOL: Use of Software Tools
  - TURN: Turnaround Time
  - VEXP: Virtual Machine Experience
  - VIRT: Virtual Machine Volatility

ACKNOWLEDGMENT

The authors thank G. Caldiera, S. Morasca, and the referees for their valuable comments. Also, they especially want to thank Chris Kemerer for providing them with the data set related to the 15 new projects .

REFERENCES

[1] A. Agresti, *Categorical Data Analysis.* New York: Wiley, 1990.
[2] V. Basili, "Quantitative evaluation of software methodology," in *Proc. First Pan Pacific Computer Conf.*, July 1985.
[3] V. Basili and H. Rombach, "The TAME project: Toward improvement-oriented software environments," *IEEE Trans. Software Eng.*, vol. SE-14, June 1988.
[4] V. Basili and D. Weiss, "A methodology for collecting valid software engineering data," *IEEE Trans. Software Eng.*, Nov. 1984.
[5] B. Boehm, *Software Engineering Economics.* Englewood Cliffs, NJ: Prentice-Hall, 1981.
[6] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees* Monterey, CA: Wadsworth & Brooks/Cole (advanced books and software), 1984.
[7] L. Briand, V. Basili, and C. Hetmanski, "Providing an empirical basis for optimizing the verification and testing phases of software development," presented at the IEEE Int. Symp. Software Reliability Engineering, Oct. 1992.
[8] L. Briand and A. Porter, "An alternative modeling approach for predicting error profiles in Ada systems," EUROMETRICS '92, European Conference on Quantitative Evaluation of Software and Systems, Apr. 1992.
[9] J. Capon, *Elementary Statistics for the Social Sciences.* Belmont, CA: Wadsworth, 1988.
[10] R. Charette, *Software Engineering Risk Analysis and Management.* New York: McGraw-Hill, 1989.
[11] W. R. Dillon, *Multivariate Analysis: Methods and Applications.* New York: Wiley, 1984.
[12] U. Fayyad and K. Irani, "On the handling of continuous-valued attributes in decision tree generation," *Machine Learning*, vol. 8, pp. 87–102, Mar. 1992.
[13] C. Kemerer, "An empirical validation of software cost estimation models," *Commun. ACM,* vol. 30, May 1987.
[14] J. Mingers, "An empirical comparison of selection measures for decision-tree induction," *Machine Learning*, vol. 3, Mar. 1989.
[15] A. Porter and R. Selby, "Evaluating techniques for generating metric-based classification trees," *J. Syst. Software,* vol. 12, pp. 209–218, July 1990.
[16] H. Potier, J. Albin, R. Ferreol, and A. Bilodeau, "Experiments with computer software complexity and reliability," in *Proc. Sixth Int. Conf. Software Engineering*, 1982.
[17] J. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, 1986.
[18] R. Selby and A. Porter, "Learning from examples: Generation and evaluation of decision trees for software resource analysis," *IEEE Trans. Software Eng.*, 1988.
[19] J. Tou and R. Gonzalez, *Pattern Recognition Principles.* Reading, MA: Addison-Wesley, 1974.



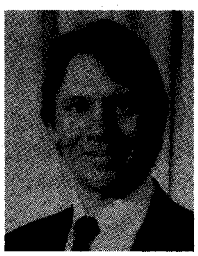
Lionel C. Briand received the Bachelor's degree in geophysics and the Masters degree in computer science from the Universite Pierre et Marie Curie (Paris 6). He is currently a Ph.D. candidate at the Laboratoire de Recherche en Informatique, Universit Paris-sud.

He has been a software engineer at CISI Ingenierie Co. (Paris, France) working on databases, expert systems, and software design methodologies. He is presently a faculty research assistant in the Department of Computer Science at the University of Maryland, College Park. His research interests include software measurement, quality control, evaluation, and risk management for large scale software development.

Victor R. Basili (M'83–SM'84–F'90) is a Professor in the Institute for Advanced Computer Studies and the Computer Science Department at the University of Maryland, College Park, MD, where he served as chairman for six years.

He was involved in the design and development of several software projects, including the SIMPL family of programming languages. He is a founder and principal of the Software Engineering Laboratory, a joint venture between NASA Goddard Space Flight Center, the University of Maryland and Computer Sciences Corporation, established in 1976. He has been working on the development of quantitative approaches for software management, engineering, and quality assurance by developing models and metrics for improving the software development process and product. He is currently measuring and evaluating software development in industrial and government settings and has consulted with many agencies and organizations, including IBM, GE, CSC, GTE, MCC, AT&T, Motorola, HP, Boeing, NRL, NSWC, and NASA. He has authored more than 100 papers. He recently completed a four-year term as Editor of the IEEE TRANSACTIONS ON SOFTWARE ENGINEERING and serves on the editorial board of the *Journal of Systems and Software.*

Dr. Basili received a NASA Group Achievement Award for the GRO Ada experiment in 1989, and in 1982 he received the Outstanding Paper Award from the TRANSACTIONS ON SOFTWARE ENGINEERING for his paper on the evaluation methodologies.



William M. Thomas (S'89–M'91) received the B.S. degree in mathematics from Bucknell University and the M.S. degree in computer science from the University of Maryland. He is a Ph.D. degree candidate in the Department of Computer Science at the University of Maryland, College Park.

He is currently a member of the technical staff at the Software Engineering Center of the MITRE Corporation in McLean, VA. His research interests include modeling, software measurement, software quality, and software reuse.