Empirical Software Engineering, 9, 229–257, 2004.
© 2004 Kluwer Academic Publishers. Manufactured in The Netherlands.

Comparative Assessment of Software Quality Classification Techniques: An Empirical Case Study

TAGHI M. KHOSHGOFTAAR taghi@cse.fau.edu
Empirical Software Engineering Laboratory, Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431 USA

NAEEM SELIYA
Empirical Software Engineering Laboratory, Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431 USA

Editor: Audris Mockus

Abstract. Software metrics-based quality classification models predict a software module as either fault-prone (*fp*) or not fault-prone (*nfp*). Timely application of such models can assist in directing quality improvement efforts to modules that are likely to be *fp* during operations, thereby cost-effectively utilizing the software quality testing and enhancement resources. Since several classification techniques are available, a relative comparative study of some commonly used classification techniques can be useful to practitioners.

We present a comprehensive evaluation of the relative performances of seven classification techniques and/or tools. These include logistic regression, case-based reasoning, classification and regression trees (CART), tree-based classification with S-PLUS, and the Sprint-Sliq, C4.5, and Treedisc algorithms. The use of expected cost of misclassification (ECM), is introduced as a singular unified measure to compare the performances of different software quality classification models. A function of the costs of the Type I (a *nfp* module misclassified as *fp*) and Type II (a *fp* module misclassified as *nfp*) misclassifications, ECM is computed for different cost ratios. Evaluating software quality classification models in the presence of varying cost ratios is important, because the usefulness of a model is dependent on the system-specific costs of misclassifications. Moreover, models should be compared and preferred for cost ratios that fall within the range of interest for the given system and project domain. Software metrics were collected from four successive releases of a large legacy telecommunications system. A two-way ANOVA randomized-complete block design modeling approach is used, in which the system release is treated as a block, while the modeling method is treated as a factor. It is observed that predictive performances of the models is significantly different across the system releases, implying that in the software engineering domain prediction models are influenced by the characteristics of the data and the system being modeled. Multiple-pairwise comparisons are performed to evaluate the relative performances of the seven models for the cost ratios of interest to the case study. In addition, the performance of the seven classification techniques is also compared with a classification based on lines of code. The comparative approach presented in this paper can also be applied to other software systems.

Keywords: Software quality classification, decision trees, case-based reasoning, logistic regression, expected cost of misclassification, analysis of variance.

1. Introduction

The use of software in high-assurance and mission-critical systems increases the need to develop and quantify measures of software quality. Subsequently, software metrics are useful in the timely prediction of high-risk components during the software development process (Basili et al., 1996; Briand et al., 2002; Ping et al., 2002). Such a prediction enables software managers to target quality improvement efforts to the needed areas. For example, prior to the system test, identifying the components that are likely to be faulty during operations can improve the effectiveness of testing efforts. Various software quality modeling techniques have been developed and used in real-life software quality predictions.

Classification-based modeling for software quality estimation is a proven technique in achieving better software quality control (Ebert, 1996; Ohlsson et al., 1996, 1998; Schneidewind, 1995). Some classification techniques used for software quality estimation include optimal set reduction (Briand et al., 1993), logistic regression (Khoshgoftaar and Allen, 1999; Schneidewind, 2001), decision trees (Khoshgoftaar et al., 2000; Suarez and Lutsko, 1999; Takahashi et al., 1997), neural networks (Khoshgoftaar et al., 1997; Paul, 1992; Pizzi et al., 2002), and case-based reasoning (Ross, 2001). Software metrics-based quality classification models classify software modules into groups. A two-group classification model is the most commonly used, in which program modules are grouped into classes such as, fault-prone (*fp*) and not fault-prone (*nfp*).[1] Software analysts interested in predicting whether a software component will be *fp* or *nfp*, can utilize classification models to obtain the needed prediction. When associating with such models, two types of misclassification errors are encountered: Type I error, that is, a *nfp* component is predicted as *fp*, and Type II error, i.e., a *fp* component is misclassified as *nfp*. Practically speaking, Type II errors are more severe in terms of software development costs and the organization's reputation. They may involve inspection and correction to components after they have been deployed for operations.

In our previous empirical studies related to software quality classification modeling, we have investigated several classification techniques, including classification and regression trees (CART) (Khoshgoftaar et al., 2000), tree-based classification with S-PLUS (Khoshgoftaar et al., 2002), the Treedisc algorithm (Khoshgoftaar and Allen, 2001), the C4.5 algorithm (Ponnuswamy, 2001; Quinlan, 1993), the Sprint-Sliq algorithm (Khoshgoftaar and Seliya, 2002), logistic regression (Khoshgoftaar and Allen, 1999), and case-based reasoning (Ross, 2001). Classification models were calibrated using case studies of different large-scale software systems, including the one presented in this paper. The models are calibrated to classify software modules as either *fp* or *nfp*, as defined by the system being modeled.

Software quality classification models are usually compared based on how accurately they can classify (predict) observations. It has been observed that when comparing two-group software quality classification models, their Type I and Type II misclassification error rates are used (Ebert, 1996; Schneidewind, 1995). As an attempt to utilize a singular comparative measure, some researchers have used the overall misclassification error rate of models. Though the Type I, Type II, and overall misclassification error rates have been commonly used as accuracy measures, they may not necessarily be the most appropriate and effective measures. We now explain in the following paragraphs, why this is so.

When performing empirical studies with different classification methods (Khoshgoftaar and Allen, 1999, 2001; Khoshgoftaar et al., 2000; Khoshgoftaar and Seliya, 2002; Ponnuswamy, 2001; Ross, 2001), our research group has observed difficulty in comparing different models based solely on the misclassification error rates. For example, consider classification models built by two competing methods ("A" and "B") using a given data set for the software system under consideration. The difficulty of using only the models' error rates for comparison occurs when method "A" (as compared to method "B") has a lower Type I error rate and a higher Type II error rate. Moreover, the difficulty in comparing competing methods based solely on their error rates increases when multiple methods are to be compared. In addition, the problem is compounded when multiple data sets (or system releases) are used for evaluating the relative performances of two or more classification methods. We can see that the use of two measures (Type I and Type II error rates) makes it difficult to compare the performances of competing methods. However, if method "A" always performs better (has lower Type I and Type II error rates) than method "B" across all data sets, then we can undoubtedly infer that method "A" is better.

A quick solution to using the two error rates as performance measures would be to use the overall misclassification error rate. However, such an approach assumes that the corrective costs of the two misclassifications are equal, i.e., cost ratio of one. Since from a software engineering point of view, it is obvious that Type II errors are more costly than Type I errors, the use of the overall misclassification error rate as an accuracy measure may lead to misleading results. An improved solution would be to utilize an appropriate singular measure to compare different classification models in the context of the software quality estimation problem. Since the inspection (and corrective) costs and efforts involved with the Type I and Type II errors are (practically speaking) different, a unified measure that incorporates the two types of costs is warranted.

This paper introduces the use of the expected cost of misclassification (ECM) (Johnson and Wichern, 1992), as a singular measure to compare the performances of different software quality classification models. Though it is not claimed that the ECM values of competing models may not fluctuate across the system releases, we note that the comparison difficulty is reduced because only one performance measure is used to compare the competing techniques. In a previous study, we investigated the use of ECM to evaluate overfitting tendencies of a decision tree-based model (Khoshgoftaar and Allen, 2001). Since ECM integrates the project- and system-specific costs of the Type I and Type II errors into a singular measure, it provides a practical measure to compare the accuracies and performances of software quality classification models. To our knowledge, this is the first study to investigate the use of ECM as a practical performance measure for evaluating usefulness of software quality classification models.

Very few studies that compare different software quality classification modeling methods, have been performed. Ebert (1996) presents a comparative study of five software quality classification methods, which include: Pareto classification, crisp classification trees, factor-based discriminant analysis, neural networks, and fuzzy classification. The study uses two small-scale case studies, consisting of 67 (10 software metrics) and 451 (six software metrics) modules, respectively. Based on these two case studies, it was suggested that among the five methods compared, fuzzy classification is a better method. The different modeling methods were compared using the Type I and Type II misclassification error rates, the use of which (as discussed earlier) can lead to dubious results.

This paper presents a comparative analysis of seven different classification modeling methods for a case study of a large-scale legacy telecommunications (high-assurance) system. Software metrics and fault data were collected over four successive system releases. The 28 software metrics that we used consist of software product and execution metrics. Other case studies of the legacy system were also performed, however, they are not presented due to similarity of results and conclusions. When building classification models with the different techniques, a common model building and validation approach was followed.

The model-selection strategy adopted for the case study consisted of obtaining the preferred balance (see Sections 4.1 and 4.3) of equality between the two error rates, with Type II being as low as possible. This implies that among the different (by changing respective parameters) models obtained for a given classification technique, the model-selection strategy is such that priority is given to models that yield an approximately equal balance between the two error rates, and among such models a model that has the lowest Type II error is selected as the final model. Such an approach was followed directly based upon our discussions with, and inputs provided by, the project management team of the legacy system. The relative comparison of the classification methods is based on their predictive performances (ECM values) in the context of varying cost ratios.

Classification models built using CART (Khoshgoftaar et al., 2000), S-PLUS (Khoshgoftaar et al., 2002), Treedisc algorithm (Khoshgoftaar and Allen, 2001), C4.5 algorithm (Ponnuswamy, 2001), Sprint-Sliq algorithm (Khoshgoftaar and Seliya, 2002), logistic regression (Khoshgoftaar and Allen, 1999), and case-based reasoning (Ross, 2001) are compared in this paper. The first five are the tree-based classification techniques that were investigated by our research team. Since the costs of misclassifications may be influenced by the nature and required-reliability of software systems, we compare the seven methods at different cost ratios, C_{II}/C_I: ratio of costs of the Type II error rate (C_{II}) and the Type I error rate (C_I).

Comparing classification models at different cost ratios will facilitate organizations to choose an appropriate modeling method, that performs best at a cost ratio which suits the software system being modeled. Based on our discussions with the project management team of the legacy telecommunications system, a range of 20 to 100 was identified as the likely cost ratio values for the high-assurance system. Therefore, our comparative evaluation is based on selected cost ratios that cover the possible range for the legacy system. The ECM values of the classification models are compared for cost ratios of 20, 25, 50, and 100.

Performance comparison of different classification modeling methods is performed by building two-way ANOVA randomized complete block design models followed by one-tailed multiple-pairwise comparisons of the seven modeling techniques (Berenson et al., 1983). In the ANOVA models, we use system release (Releases 2, 3, and 4) as the "blocks" and the classification modeling methods as the "factor". This is done to observe if the respective system releases and classification models are significantly different from their respective counterparts. Release 1 was not used as a block since it was used to build or train the classification models. The comparative technique adopted in our study is not limited to only seven modeling methods. It can be extended to compare fewer than, or more than, the number of methods compared in this paper. However, data from multiple releases or multiple projects is needed to effectively utilize ANOVA design models for performance comparisons of software quality classification models. In the case when software metrics data is not available from multiple releases of a given system, similar performance comparisons can be done by means of one-way ANOVA models.

In addition to relatively comparing the seven classification techniques, we also present a comparison of their classification performances with a software quality classification based on lines of code. This was done to investigate how the seven classification techniques perform with respect to classification based on a simple method.

The layout of the rest of the paper is as follows. In Section 2, a brief description of the different classification modeling methods is presented. In Section 3, the case study used in this paper is described. Section 4 discusses the modeling objective, methodology, and techniques employed in comparing the different classification models. Sections 5 and 6 present the results and conclusions of our comparative study.

2. Software Quality Classification Techniques

This section presents a brief description of the classification methods compared in this paper. The aim of this section is to give a brief overview of the classification technique, and not (due to lack of space) to present an extensive algorithmic detail. Our research group has performed extensive empirical research in software quality classification modeling using all of the methods discussed. A generalized classification rule for model-building was adopted for all methods discussed.

2.1. CART

Classification and regression trees is a widely used decision tree system with many applications to data mining, predictive modeling, and data preprocessing (Breiman et al., 1984; Steinberg and Colla, 1995). CART is a statistical tool (Salford Systems Inc.) that automatically sifts large complex databases, searching for, discovering, and isolating significant patterns and relationships in the data. Our previous research related to software quality classification include the use of CART (Khoshgoftaar et al., 2000).

Table 1. Notations.

Symbol	Description
ECM	Expected cost of misclassification
NECM	Normalized expected cost of misclassification
fp	A fault-prone or high risk module
nfp	A not fault-prone or low risk module
C_I	Cost of Type I misclassification error
C_{II}	Cost of Type II misclassification error
π_{fp}	Prior probability of *fp* modules
π_{nfp}	Prior probability of *nfp* modules
p	The p-value for hypothesis testing
α	The significance level for hypothesis testing
CBR	Case-based reasoning (Leake, 1996; Ross, 2001)
LOG	Logistic regression (Khoshgoftaar and Allen, 1999)
CART	Classification and regression trees (Breiman et al., 1984)
SPT	The Sprint-Sliq classification tree algorithm (Khoshgoftaar and Seliya, 2002; Shafer et al., 1996)
TD	The Treedisc classification tree algorithm (Khoshgoftaar and Allen, 2001)
S-PLUS	The regression tree algorithm of S-PLUS (Khoshgaftaar et al., 2002)
C4.5	The C4.5 classification tree algorithm (Ponnuswamy, 2001; Quinlan, 1993)

The algorithms of CART search for questions that split nodes into relatively homogeneous child nodes, such as a group consisting largely of responders, or high-risk components. As the tree evolves, the nodes become increasingly more homogeneous, identifying important segments. The set of predictor variables used to split the nodes into segments, read directly off the tree and summarized in the variable importance tables, are the key drivers of the response variable. The methodology solves a number of performance, accuracy, and operational problems that still plague many current decision-tree methods. Some of innovations of CART include: solving the tree size problem using strictly two-way splitting, incorporating automatic testing and validation, and providing a completely new method for handling missing values in the data set.

2.2. S-PLUS

A solution for advanced data analysis, data mining, and statistical modeling (Clark and Pregibon, 1992), the S-PLUS (Mathsoft Inc.) tool combines an intuitive graphical user interface with an extensive data analysis environment to offer ease of use and flexibility. Among other data mining functions, S-PLUS includes regression tree-based models.

At the core of the S-PLUS system is *S*, a language designed specifically for data visualization and exploration, statistical modeling and programming with data. *S* provides a rich, object-oriented environment designed for interactive data discovery. With a huge library of functions for all aspects of computing with data, *S* offers good extensibility. Our recent research related to software quality classification included using the regression tree algorithm of S-PLUS (Khoshgoftaar et al., 2002). In-depth mathematical details of the S-PLUS regression tree algorithm are presented in Clark and Pregibon (1992). The predictors are software metrics treated by S-PLUS as ordinal measures which are used to build regression trees to predict the response variable. Hereon, we use the notation S-PLUS to indicate the tool's regression tree-based modeling algorithm.

2.3. C4.5 Algorithm

The C4.5 algorithm is an inductive supervised learning system which employs decision trees to represent a quality model. C4.5 is a descendent of another induction program, ID3 (Quinlan, 1993), and it consists of four principal programs: decision tree generator, production rule generator, decision tree interpreter, and production rule interpreter. The algorithm uses these four programs when constructing and evaluating classification tree models. Different tree models were built by varying parameters: minimum node size before splitting and pruning percentage (Ponnuswamy, 2001).

The C4.5 algorithm commands certain pre-processing of data in order for it to build decision tree models. Some of these include attribute value description type, predefined discrete classes, and sufficient number of observations for supervised learning. The classification tree is initially empty and the algorithm begins adding decision and leaf nodes, starting with the root node.

2.4. Treedisc Algorithm

The Treedisc algorithm is a SAS macro implementation of the modified CHi-square Automatic Interaction Detection algorithm (Khoshgoftaar and Allen, 2001). It constructs a regression tree from an input data set, that predicts a specified categorical response variable based on one or more predictors. The predictor variable is selected to be the variable that is most significantly associated with the dependent variable according to a chi-squared test of independence in the contingency table.

Regression tree-based models are built by varying model parameters in order to achieve the preferred balance between the misclassification error rates, and to avoid overfitting of classification trees. A generalized classification rule is used to label each leaf node after the regression tree is built. This classification rule is very similar to the approach followed, when using S-PLUS regression trees as classification trees (Khoshgoftaar et al., 2002).

2.5. *Sprint-Sliq algorithm*

Sprint-Sliq is an abbreviated version of Scalable PaRallelizable INduction of decision Trees-Supervised Learning In Quest (Mehta et al., 1996; Shafer et al., 1996). The algorithm can be used to build classification tree models that can analyze both numeric and categorical attributes. It is a modified version of the classification tree algorithm of CART, and uses a different pruning technique based on the minimum description length principle (Mehta et al., 1995). The algorithm has excellent scalability and analysis speed. Classification tree modeling using Sprint-Sliq is accomplished in two phases: a tree building phase and a tree pruning phase. The building phase recursively partitions the training data until each partition is either "pure" or meets the stop-splitting rules set by the user.

The IBM Intelligent Data Miner tool, which implements the Sprint-Sliq algorithm, was used by our research group to build classification trees (Khoshgoftaar and Seliya, 2002). Sprint-Sliq uses the Gini Index to evaluate the goodness of split of all the possible splits (Shafer et al., 1996). A class assignment rule is needed to classify modules as *fp* and *nfp*.

2.6. *Logistic Regression*

Logistic regression is a statistical modeling technique that offers good model interpretation. Independent variables in logistic regression may be categorical, discrete or continuous. However, the categorical variables need to be encoded (e.g., 0, 1) to facilitate classification modeling. Our research group has used logistic regression to build software quality classification models (Khoshgoftaar and Allen, 1999).

Let x_j be the *j*th independent variable, and let $\mathbf{x}_i$ be the vector of the *i*th module's independent variable values. A module being *fp* is designated as an "event". Let q be the probability of an event, and thus $q/(1-q)$ is the odds of an event. The logistic regression model has the form,

$$\log\left(\frac{q}{1-q}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_j x_j + \beta_m x_m \tag{1}$$

where, log means the natural logarithm, β_j is the regression coefficient associated with independent variable x_j, and m is the number of independent variables.

Logistic regression suits software quality modeling because most software engineering measures do have a monotonic relationship with faults that is inherent in the underlying processes. Given a list of candidate independent variables and a significance level, α, some of the estimated coefficients may not be significantly different from zero. Such variables should not be included in the final model.

2.7. Case-Based Reasoning

Case-based reasoning (CBR) (Kolodner, 1993; Leake, 1996), is a technique that aims to find solutions to new problems based on past experiences, which are represented by "cases" in a "case library". The case library and the associated retrieval and decision rules constitute a CBR model. In the context of a classification problem, each case in the case library has known attributes and class membership. The working hypothesis of CBR for software quality classification modeling is that a module currently under development is probably *fp* if a module with similar attributes in an earlier release (or similar project) was *fp*.

A CBR system can take advantage of availability of new or revised information by adding new cases or by removing obsolete cases from the case library. Its good scalability provides fast retrieval even as the size of the case library scales up. CBR systems can be designed to alert users when a new case is outside the bounds of current experience. Our research group was the first to use CBR in the context of building and evaluating software metrics-based quality classification models (Ross, 2001). A generalized classification rule, similar to the ones used in Khoshgoftaar and Allen (2000), is utilized to classify modules as *fp* or *nfp*.

3. Case Study Description

The case study data was collected over four successive releases, from a very large legacy telecommunications system (LLTS). The software system is an embedded-computer application that included finite-state machines. Using the procedural development paradigm the software was written in PROTEL (a high-level language) and was maintained by professional programrs in a large organization. A software module was considered as a set of related source-code files. Fault data was collected at the module-level by the problem reporting system, and consisted of post-release faults discovered by customers during system operations. Faults were recorded only if their discovery resulted in changes to the source code of the respective module. Preventing discovery of faults after deployment was a high priority for the developers, because visits to customer sites involved extensive consumption of monetary and other resources.

Configuration management data analysis identified software modules that were unchanged from the prior release. Fault data collected from the problem reporting system were tabulated into problem reports and anomalies were resolved. Due to the nature of the system being modeled, that is, a high-assurance system, the number of modules associated with post-release faults were very few as compared to modules with no faults. Two clusters of modules were identified: unchanged and updated. The updated modules consisted of those that were either new or had at least one update to their source code since the prior release. Among the unchanged modules, almost all of them had no faults, and therefore, were not considered for modeling purposes.

We selected updated modules with no missing data in relevant variables. These updated modules had several millions lines of code, with a few thousand of these

Table 2. Distribution of faults discovered by customers.

	Percentage of updated modules			
Faults	Release 1 (%)	Release 2 (%)	Release 3 (%)	Release 4 (%)
0	93.7	95.3	98.7	97.7
1	5.1	3.9	1.0	2.1
2	0.7	0.7	0.2	0.2
3	0.3	0.1	0.1	0.1
4	0.1	*		
6	*			
9	*			

*One module.

modules in each system release. The number of updated modules (that remained after unchanged modules or those with missing data were removed) that were considered for each of the four release are: 3649 for Release 1, 3981 for Release 2, 3541 for Release 3, and 3978 for Release 4. A module was considered as *nfp* if it had no post-release faults, and *fp* otherwise.

The distribution of the post-release faults for each release is summarized in Table 2. The proportion of modules with no faults among the updated modules of Release 1 was $\pi_{nfp} = 0.937$, and the proportion with at least one fault was $\pi_{fp} = 0.063$. We observe that the percentage of modules with no faults generally increases across successive releases. This is to be expected for a legacy system in an environment similar to that of LLTS, because the development process for such a high-assurance software system generally tends to improve with time. However, this may not hold for another software system because other factors such as new development staff or changes in system functionality could increase the number of faults and introduce additional *fp* modules.

The set of available software metrics is usually determined by pragmatic considerations. A data mining approach is preferred in exploiting software metrics data (Fayyad, 1996), by which a broad set of metrics are analyzed rather than limiting data collection according to predetermined research questions. Data collection for LLTS involved extracting source code from the configuration management system. Measurements were recorded using the EMERALD software metrics analysis tool (Khoshgoftaar et al., 2000). Preliminary data analysis selected metrics that were appropriate for our modeling purposes. Software metrics collected included 24 product metrics, 14 process metrics and four execution metrics. The 14 process metrics were not used in our empirical evaluation, because this study explores prediction of *fp* (and *nfp*) modules for software quality modeling after the coding (implementation) phase and prior to system tests. The case study, consists of 28 independent variables (Tables 3 and 4) that were used to predict the response variable: Class which identifies a software module either as *fp* or *nfp*.

The software product metrics in Table 3 are based on call graph, control flow graph, and statement metrics. An example of call graph metrics is number of distinct procedure calls. A module's control flow graph consists of nodes and arcs depicting

Table 3. Software product metrics.

Symbol	Description
Call graph metrics	
CALUNQ	Number of distinct procedure calls to others.
CAL2	Number of second and following calls to others. $CAL2 = CAL - CALUNQ$ where CAL is the total number of calls.
Control flow graph metrics	
CNDNOT	Number of arcs that are not conditional arcs.
IFTH	Number of non-loop conditional arcs, i.e., if-then constructs.
LOP	Number of loop constructs.
CNDSPNSM	Total span of branches of conditional arcs. The unit of measure is arcs.
CNDSPNMX	Maximum span of branches of conditional arcs.
CTRNSTMX	Maximum control structure nesting.
KNT	Number of knots. A "knot" in a control flow graph is where arcs cross due to a violation of structured programming principles:
NDSINT	Number of internal nodes (i.e., not an entry, exit, or pending node).
NDSENT	Number of entry nodes.
NDSEXT	Number of exit nodes.
NDSPND	Number of pending nodes, i.e., dead code segments.
LGPATH	Base 2 logarithm of the number of independent paths.
Statement metrics	
FILINCUQ	Number of distinct include files.
LOC	Number of lines of code.
STMCTL	Number of control statements.
STMDEC	Number of declarative statements.
STMEXE	Number of executable statements.
VARGLBUS	Number of global variables used.
VARSPNSM	Total span of variables.
VARSPNMX	Maximum span of variables.
VARUSDUQ	Number of distinct variables used.
VARUSD2	Number of second and following uses of variables. $VARUSD2 = VARUSD - VARUSDUQ$ where $VARUSD$ is the total number of variable uses.

Table 4. Software execution metrics.

Symbol	Description
USAGE	Deployment percentage of the module.
RESCPU	Execution time (microseconds) of an average transaction on a system serving consumers.
BUSCPU	Execution time (microseconds) of an average transaction on a system serving businesses.
TANCPU	Execution time (microseconds) of an average transaction on a tandem system.

the flow of control of the program. Statement metrics are measurements of the program statements without implying the meaning or logistics of the statements. The problem reporting system maintained records on past problems. The proportion of installations that had a module, *USAGE*, was approximated by deployment data on a prior system release. Execution times in Table 4 were measured in a laboratory setting with different simulated workloads.

4. Modeling Methodology

In this section, we present a brief discussion of the approach adopted in comparing the different software quality classification modeling methods. We note that the details of classification modeling with the individual techniques have been omitted in this paper due to paper-size concerns. Moreover, this has been done because the focus of this study is to compare the various classification techniques we have investigated in our previous studies. Please refer to the appropriate references for the details regarding the individual techniques (Khoshgoftaar and Allen, 1999, 2001; Khoshgoftaar et al., 2002, 2000; Khoshgoftaar and Seliya, 2002; Ponnuswamy, 2001; Ross, 2001).

4.1. Objective for Classification Models

In the case of high-assurance and mission-critical systems, due to the large disparity between the costs of misclassifications, it is highly desirable to detect and rectify almost all modules that are likely to be faulty during operations. As mentioned earlier, for a system similar to LLTS, the cost ratio is likely (based on practical experiences of similar projects) to fall within a range of 20 to 100. On the other hand, for a non-critical business application the cost ratio is likely to fall within the range 10 to 25. The quality improvement needs of software projects are dependent on the application domain and the nature of the system being developed. Hence, when evaluating techniques for calibrating classification models it is important that the practical needs and objectives of the system being modeled are considered.

While it is beneficial to detect prior to operations as many *fp* (actual) modules as possible, the usefulness of a classification model is affected by the percentage of actual *nfp* modules that are predicted as *fp* by the model, that is, a model's Type I error rate. For example, despite capturing many of the actual *fp* modules, a model with a very low Type II error rate and a very high Type I error rate is not practically useful. This is because, given the large number of modules predicted as *fp* (many of them are actually *nfp*), deploying the limited quality improvement resources will pose a difficulty.

Therefore, a more useful model is one that can obtain a preferred balance between the error rates, according to the needs of the system being modeled. In the context of two-group classification modeling, an inverse relationship is observed between the Type I and Type II error rates (Khoshgoftaar and Allen, 1999, 2001; Khoshgoftaar et al., 2002, 2000; Khoshgoftaar and Seliya, 2002). More specifically, as Type I increases, Type II decreases and as Type I decreases, Type II increases. Based on this property, a classification model with the preferred balance between the error rates can be obtained. A preferred balance is one that yields a classification model that is useful to the software quality improvement needs of the development team. For example, some software projects may consider a model with a very low Type II error (regardless of the Type I error rate) as the preferred classification model. A different application may consider another preferred balance between the error rates.

The model selection strategy adopted for the case study consists of finding the preferred balance of equality between the two error rates, with Type II being as low as possible. Such a strategy was used based upon the recommendation of the software quality engineers and the project management team of the system being modeled. In addition to keeping the number of ineffective inspections and testing of the predicted *fp* modules at acceptable limits, keeping the Type II error rate low ensures the detection of as many *fp* modules as possible. It should be noted that this model-selection strategy is appropriate only for a high-assurance system such as LLTS, and we do not advocate that it is appropriate for all software systems. Naturally, for other software systems (e.g. business applications, medical devices, etc.) different model-selection strategies may be deemed more appropriate.

To summarize, the classification modeling objective for this case study was: (1) selecting the appropriate model based on the system-specific preferred balance between the two error rates, and (2) evaluating the predictive performance with respect to the ECM values computed for different cost ratios when the selected model is applied to the test data sets. A detailed description of how a classification model with the preferred balance (for a given technique) was obtained for the case study is presented in Section 4.3.

4.2. Calibrating Classification Models

To justify an unbiased comparative study of the seven classification techniques, we strived to use a common model-selection and model-evaluation approach which were discussed in the previous section. The following modeling steps were employed to build, select, and validate classification models for the legacy telecommunications system.

1. *Preprocessing and Formatting Data:* In some modeling tools, the *fit* and *test* data sets need to be converted to a format acceptable by the tool. For example when using CART, data sets have to be converted to the SYSTAT file format. In the case of CBR, standardization or normalization of the data may be required when certain similarity functions are used. Further details regarding preprocessing data for different methods are presented in Khoshgoftaar and Allen (1999, 2001), Khashgoftaar et al. (2002, 2000), Khoshgoftaar and Seliya (2002), Ponnuswamy (2001), and Ross (2001).

2. *Building Models:* The modules of Release 1 were used as the fit data set to build and select the respective classification models. Certain parameters specific to the modeling tool, are varied to build different classification models. For example, when using S-PLUS (Khoshgoftaar et al., 2002), *mindev* and *minsize* are varied to build different regression tree models. A generalized classification rule similar to the one presented in Khoshgoftaar and Allen (2000) was used for all methods to classify software modules as either *fp* or *nfp*.

3. *Selecting Models:* In order to validate our comparative study, it was important that model-selection for all the modeling techniques be based on the same data set. For the case study, model-selection was based on the Release 2, i.e., based on the misclassification error rates computed when Release 2 is used as the *test* data set.

   In the case of CART and CBR, model-selection was based on the misclassification error rates obtained from applying a cross-validation (Khoshgoftaar et al., 2000; Ross, 2001) technique on the *fit* (Release 1) data set. Cross-validation was not explicitly available for the other five modeling techniques. Since model-selection for two of the seven techniques was based on Release 1, we were concerned with the model-selection validity of this comparative study. Upon a close inspection of the Release 2 misclassification error rates for the different CART and CBR models, we observed that the same model would have been selected if Release 2 was used for model-selection purposes. Therefore, the model-selection validity of our comparative study is assured because the selected models for the seven techniques remained unchanged when Release 2 is used for model-selection.

4. *Evaluating Models:* Releases 2, 3, and 4 are used as test data sets to evaluate the (predictive) classification accuracy of the different modeling techniques. The computed Type I and Type II misclassification error rates (for the test data sets) are used to determine the ECM values for the different cost ratios that were considered. Only cost ratios that are likely for a system similar to LLTS were considered in our evaluation. The cost ratios considered are 20, 25, 50, and 100.

4.3. Selecting a Preferred Model

In this section, we illustrate the description for obtaining the preferred balance with the logistic regression classification technique (Khoshgoftaar and Allen, 1999). Similar descriptions for the other classification techniques can be obtained from the respective references. In the case of classification modeling with logistic regression, a module being *fp* is designated as an "event".[2] The maximum likelihood estimates of the model parameters (b_j) are computed using the iteratively re-weighted least squares algorithm (Myers, 1990), where b_j is the estimated value of β_j (see Section 2.6). The estimated logistic regression model takes the following form:

$$\log_e\left(\frac{\hat{q}}{1-\hat{q}}\right) = b_0 + b_1 x_1 + \cdots + b_j x_j + \cdots + b_m x_m \tag{2}$$

where $\hat{q}$ is the estimate of the probability of a module being *fp* and $1-\hat{q}$ is the estimate of the probability of a module being *nfp*. The following procedure illustrates how a logistic regression model can be used to classify program modules as either *fp* or *nfp*.

1. Compute $\hat{q}/(1-\hat{q})$ using Equation (2).

2. Assign a module's class, that is, Class($\mathbf{x}_i$), by using the classification rule,

$$\text{Class}(\mathbf{x}_i) = \begin{cases} nfp & \text{if } \left(\frac{1-\hat{q}}{\hat{q}}\right) > \zeta \\ fp & \text{otherwise} \end{cases} \tag{3}$$

   where, ζ is a modeling parameter which can be empirically varied to obtain the preferred classification model (Khoshgoftaar and Allen, 1999). In our case study, we varied the value of the parameter ζ to obtain the preferred balance between the error rates.

The misclassification rates for the logistic regression models based on the different values of ζ are presented in Table 5. The table only presents the error rates for Releases 1 and 2. The inverse relationship between the error rates is clearly observed with respect to ζ. A very high value of ζ yielded a very low Type II error and a very high Type I error. On the other hand, a very low value of ζ yielded a very high Type II error and a very low Type I error. For example, when $\zeta = 50$ the corresponding fitted (Release 1) model yielded a Type I error rate of 67% and a Type II error rate of

Table 5. Preferred balance for logistic regression.

	Release 1		Release 2	
ζ	Type I (%)	Type II (%)	Type I (%)	Type II (%)
50	67.00	3.50	64.79	4.23
30	49.80	8.70	46.70	11.64
25	42.90	13.50	39.90	14.81
20	34.90	19.70	31.59	21.69
18	31.00	21.00	28.11	24.87
16	**27.00**	**24.90**	**25.00**	**29.60**
10	15.50	38.90	14.61	41.80
5	6.20	60.30	6.20	67.20
1	0.40	91.70	0.40	91.53
0.067	0.00	100.00	0.00	100.00

3.5%. Moreover, when $\zeta = 1$, the corresponding model yielded a Type I error rate of 0.4% and a Type II error rate of 91.7%.

According to the model-selection strategy described in Section 4.1, the preferred balance (based on the fit data set, i.e., Release 1) was obtained when $\zeta = 16$. We observe that for $\zeta = 16$ the two error rates are approximately equal with the Type II error rate being low. Other values of ζ were also considered at the time of modeling, however, since they did not yield different empirical results, their corresponding models are not presented in the table.

4.4. Expected Cost of Misclassification

Recall, the cost of a Type I misclassification, C_I, is the effort wasted on inspecting or testing a *nfp* module. On the other hand, the cost of a Type II misclassification, C_{II}, is the rectification cost incurred due to the lost opportunity to correct faults prior to operations. When the proportions of each class are approximately equal and the costs of the misclassification for the system being modeled are approximately equal, the overall misclassification error rate can be a satisfactory measure of a model's predictive accuracy. However, we have seen that in software applications of high-assurance and mission-critical systems, the proportion of *fp* modules is often very small in comparison to the proportion of *nfp* modules, and C_{II} is usually several magnitudes greater than C_I. Therefore, a model that yields a low ECM is preferred.

The ECM measure (Equation (4)), takes prior probabilities of the two classes and the costs of misclassifications into account (Johnson and Wichern, 1992). Since in many organizations, it is not practical to quantify the individual costs of misclassifications, we normalize ECM[3] with respect to C_I (Equation (5)), facilitating the use of the cost ratio instead of individual misclassification costs.

$$\text{ECM} = C_I Pr(fp \mid nfp)\pi_{nfp} + C_{II} Pr(nfp \mid fp)\pi_{fp} \tag{4}$$

$$\text{NECM} = \frac{\text{ECM}}{C_I} = Pr(fp \mid nfp)\pi_{nfp} + \frac{C_{II}}{C_I} Pr(nfp \mid fp)\pi_{fp} \tag{5}$$

The prior probabilities of the *fp* and *nfp* classes are given by, π_{fp} and π_{nfp} respectively. $Pr(fp \mid nfp)$ is the proportion of the *nfp* modules incorrectly classified as *fp* and conversely, $Pr(nfp \mid fp)$ is the proportion of the *fp* modules incorrectly classified as *nfp*. The prior probabilities, that is, π_{fp} and π_{nfp}, are estimated as the respective proportions in the given data set. We compared the classification models of the different modeling methods at different cost ratios, that is, by varying C_{II}/C_I in Equation (5). Evaluating models across a range of cost ratios is more practical since the actual costs of misclassifications are unknown at the time of modeling. Moreover, the sensitivity (robustness) of a classification model in light of the possible costs and effort values, can be observed.

It can be argued that advocating the use of ECM as a practical model-evaluation measure can be extended for model-selection purposes. However, one has to be careful in doing so, because the selected model may not serve the practical quality improvement objectives of the project management team, as discussed in Section 4.1. For example, for a cost ratio of 100 (empirical upper bound for high-assurance systems) if a classification model demonstrates a very low Type II error rate and a high Type I error rate, then its ECM value is likely to be very low, leading to conclusion that it be selected as the preferred model.

However, such a model is not useful for practical quality improvement purposes, because a high Type I error rate indicates that many of the predicted *fp* modules are actually *nfp*. As discussed in Section 4.1, applying the limited resources allocated for software quality improvement to all of the (a relatively large number) predicted *fp* modules becomes infeasible. Inspecting a large number of modules according to the given organization's software inspection and testing process may not be practically possible. Therefore, the first criterion for model-selection should be to respect the needs of the software quality assurance team for the system being modeled. In our study, model selection was done as per the recommendation provided by the software quality engineers of the legacy system.

We note that the use of ECM as a unified singular performance measure may overlook some underlying predictive behavior of a classification model. For example, the prediction of a selected model may not yield a good preferred balance but may yield a low ECM value. However, tracking the stability and robustness of the model performances (with respect to Type I and Type II errors) across the system releases is out of scope for this paper, and can be considered in future works.

4.5. Two-Way ANOVA: Randomized Complete Block Design

ANOVA, abbreviated for analysis of variance, is a commonly used statistical technique when comparing differences between the means of three or more independent groups or populations. In our study, we employ the two-way ANOVA: Randomized complete block design modeling approach (Berenson et al., 1983; Neter et al., 1996), in which n heterogeneous subjects are classified into b homogeneous groups, called blocks so that the subjects in each block can then be randomly assigned, one each, to the levels of the factor of interest prior to the performance of a two-tailed F test, to determine the existence of significant factor effects.

Selecting the appropriate experimental design approach depends on the level of reduction in experimental error required. Since the primary objective for selecting a particular experimental design is to reduce experimental error (variability within data), a better design could be obtained if subject variability is separated from the experimental error (Neter et al., 1996). A two-way ANOVA randomized complete block design is a restricted randomization design in which the experimental units are first sorted into homogeneous groups, i.e., blocks, and the treatments are then assigned randomly within the blocks.

We are interested in observing if the different modeling methods and the different system releases are significantly different from their respective counterparts. The observed data for each release constitutes a replication. Since within each release, the observed data is not affected by releases, blocking by release will reduce the experimental error variability and will make the experiment more powerful (Berenson et al., 1983). We employ NECM predicted by different modeling methods for different releases as the response variable in our experimental design models. Since the analysis of variance models are based on some underlying assumptions such as normality of data and randomness of the variable, we note that any significant deviations from these assumptions were not observed.

Experimental design models are built using NECM values computed for different cost ratios. Two-way ANOVA models for our comparative study involved seven factor treatments (seven classification methods) and three blocks (system releases 2, 3, and 4). The p-values (for different cost ratios) in the ANOVA design models (Table 7, later), indicate the significance of the difference between the various modeling methods as well as between the different system releases. To develop the ANOVA procedure for a randomized complete block design, Y_{ij}, the observation in the ith block of $B(i = 1, 2, \ldots, b)$ under the jth level of factor $A(j = 1, 2, \ldots, a)$, can be represented by the model,

$$Y_{ij} = \mu + A_j + B_i + \varepsilon_{ij} \tag{6}$$

where $\mu =$ overall effect or mean common to all observations; $A_j = \mu \cdot_j - \mu$, a treatment effect peculiar to the jth level of factor A (method); $B_i = \mu \cdot_i - \mu$, a block effect (system release) peculiar to the ith block of B; $\varepsilon_{ij} =$ random variation or experimental error associated with the observation in the ith block of B under the jth level of factor A; $\mu \cdot_j =$ true mean for the jth level of factor A; $\mu \cdot_i =$ true mean for the ith block of B; Y_{ij} is an NECM value in the context of this paper.

4.6. Hypothesis Testing: A **p***-value Approach*

Hypothesis testing is concerned with the testing of certain specified (i.e., hypothesized) values for those population parameters. Statisticians and software analysts alike, often perform hypothesis tests (Berenson et al., 1983) when comparing different models. A null hypothesis, H_0, is tested against its compliment, the alternate hypothesis, H_A. Hypotheses are usually set up to determine if the data supports a belief as specified by H_A. These tests indicate the significance (α) of difference between two methods or populations.

The selection of the pre-determined significance level α, may depend on the analyst and the project involved. In some cases the selection of α, may be too ambiguous or difficult (Beaumont, 1996). In such situations, it may be preferred to perform hypothesis testing without setting a value for α. This may be achieved by employing the p-value approach to hypothesis testing (Beaumont, 1996; Berenson et al., 1983). This approach involves finding a value p, such that a given H_0 will not be accepted for any $\alpha \geq p$. Otherwise, H_0 will not be rejected, that is, $\alpha < p$. If this probability (p-value) is very high, H_0 is not rejected, while if this likelihood is very small (traditionally lower than 0.05 or 0.1), H_0 is rejected. Hypotheses tests may be one-tailed or two-tailed, depending on the alternative hypothesis, H_A, of interest to the researcher (Beaumont, 1996; Berenson et al., 1983).

We use the Minitab software tool (Beaumont, 1996), which has a provision for statistical comparative analysis. We compute the p-values to determine if a method is significantly better than another method. These p-values are used in deciding on the performance order of the different tree-based classification methods. In making decisions regarding the rejection of H_0, the appropriate test statistic would be compared against the critical values for the particular sampling distribution of interest. For our comparative study, we use the F statistic (Berenson et al., 1983). If the F test statistic is distributed as $F(v_1, v_2)$, then p-value is given by,

$$p = Pr\{F(p; v_1, v_2) \leq F(v_1, v_2)\} \tag{7}$$

where, v_1 and v_2 are the degrees of freedom for the F distribution, $F(p; v_1, v_2)$ is the entry in the F-table (Beaumont, 1996), and $F(v_1, v_2)$ is the computed statistic for the hypothesis test.

4.7. Multiple-Pairwise Comparison

The ANOVA block design models do not specify or indicate which means differ from which of the other means. Multiple comparison methods facilitate a more detailed information about the differences of these means. Specifically they provide a statistical technique to compare two methods (e.g. methods A and B) at a time. A variety of multiple comparison methods are available, and for our study we employ Bonferroni's multiple comparison equation (Beaumont, 1996; Berenson et al., 1983). Hypothesis testing using the p-value approach (see Section 4.6) is performed, yielding the p-value which indicates the level of difference between the two methods being compared. The null and alternate hypotheses used for the multiple-pairwise comparison study are given by,

$$H_0 : \text{NECM}_A \geq \text{NECM}_B \tag{8}$$

$$H_A : \text{NECM}_A < \text{NECM}_B \tag{9}$$

4.8. Comparison with a Simple Method

A software quality assurance team of a given software project is often interested in knowing how well a given software quality model performs as compared to obtaining a model based on simple software metrics, such as software size. One of the most simple software metric is lines-of-code, which has often been used as a rule of thumb to detect problematic software modules, i.e., the more the lines-of-code (LOC), the greater is the likelihood of having software faults. In addition to performing a relative comparison of the seven classification techniques, we provide the classification results obtained by classifying modules as *fp* or *nfp* according to their LOC.

The modules in the Release 1 (fit) data set are ordered in a descending order of their LOC, that is, the module at the top of the list has the most LOC and is therefore the most *fp*, while the module at the bottom of the list has the least LOC and is therefore the most *nfp*. Subsequently, starting from the top, the number of modules considered as *fp* according to LOC is varied until the two error rates satisfy the model selection strategy described in Section 4.1.

Once the final model is selected according to classification based on LOC, the number of modules considered as *fp* is recorded as is denoted by FP_{LOC}. Subsequently, the modules in the test data sets, i.e., Releases 2, 3, and 4, are ordered according to their LOC and the top FP_{LOC} modules are considered as *fp* and the Type I, Type II, and overall misclassification error rates are computed. Moreover, the respective expected cost of misclassification are also computed for the four cost ratios mentioned earlier.

5. Results and Analysis

The results of our comparative analysis is based on observing the performances of the seven[4] classification modeling techniques for the test data sets, that is, Releases 2, 3, and 4. Release 1 was not included in the comparison, because it was used to train the individual models. Initially we wanted to consider evaluating the performances of the seven techniques based solely on their Type I and Type II error rates. However, comparisons based on such an approach proved difficult and inappropriate, as discussed in Section 1.

The predictive (on test data sets) performances of the seven models including the Type I, Type II, and overall misclassification error rates and the NECM values for the different cost ratios are presented in Table 6. These cost ratios represent the likely range for the legacy telecommunications system. For each of the techniques, the preferred model was selected based on the practical model-selection strategy adopted for the legacy system. Our comparison goal is to observe (NECM values) the relative performances of the seven techniques, and if possible, suggest a relative rank-order of their models. The comparative analysis and conclusions for each of the cost ratios, i.e., 20, 25, 50, and 100, is discussed at a significance level of 10%. This holds true for the analysis of variance models and the multiple pairwise comparisons.

The two-way ANOVA block design results for the seven techniques are presented in Table 7. The NECM values computed for the test data sets, are used as the response variable for the ANOVA models. The notations used in Table 7 are as follows: DF—degrees of freedom, SS—sums of squares, MS—mean squares, and F—the F statistic. The table indicates the respective *p*-values of whether the seven models perform significantly different than each other, and whether the three releases yield significantly different NECM values with respect to each other.

We observe that for all the cost ratios, the NECM values across the three releases (Releases 2, 3, and 4) are significantly different: *p*-values of less than or equal to 0.1%. The difference in the performance across the system releases may be reflective of the software engineering domain. Shepperd and Kododa (2001) recently discussed

Table 6. Model performances on test data sets.

		Misclassification rates (%)			NECM for each C_{II}/C_I			
Model	Release	Type I	Type II	Overall	20	25	50	100
SPT	2	24.47	26.84	24.58	0.567	0.652	1.075	1.920
	3	25.38	20.83	25.32	0.500	0.566	0.894	1.550
	4	28.48	26.88	28.44	0.606	0.690	1.114	1.960
CART	2	31.67	23.28	31.27	0.590	0.663	1.030	1.763
	3	30.30	14.89	30.10	0.472	0.518	0.753	1.222
	4	35.64	22.82	35.34	0.621	0.693	1.053	1.772
SPL	2	25.08	26.46	25.15	0.568	0.652	1.068	1.902
	3	26.67	21.28	26.60	0.518	0.585	0.920	1.591
	4	32.24	21.74	31.99	0.576	0.644	0.987	1.672
TD	2	25.08	29.10	25.27	0.602	0.693	1.152	2.068
	3	28.91	25.53	28.87	0.593	0.673	1.075	1.879
	4	28.33	28.26	28.33	0.622	0.711	1.156	2.046
C4.5	2	22.94	29.10	23.23	0.582	0.673	1.132	2.048
	3	25.44	21.27	25.39	0.506	0.573	0.908	1.578
	4	31.34	28.26	31.27	0.650	0.739	1.184	2.074
CBR	2	25.34	30.16	25.57	0.617	0.712	1.187	2.138
	3	27.22	27.66	27.23	0.604	0.691	1.126	1.998
	4	31.63	29.35	31.58	0.666	0.759	1.221	2.145
LOG	2	25.00	29.60	25.22	0.607	0.700	1.167	2.099
	3	27.90	12.80	27.70	0.423	0.463	0.665	1.068
	4	36.70	19.60	36.31	0.591	0.653	0.961	1.579

Table 7. Two-way ANOVA models.

C_{II}/C_I	Source	DF	SS	MS	F	p-value
20	Method	6	0.0180	0.0030	2.430	0.090
	Release	2	0.0391	0.0196	15.84	0.000
	Error	12	0.0148	0.0012		
	Total	20	0.0719			
25	Method	6	0.0306	0.0051	2.770	0.063
	Release	2	0.0548	0.0274	14.89	0.001
	Error	12	0.0221	0.0018		
	Total	20	0.1075			
50	Method	6	0.1540	0.0257	3.630	0.027
	Release	2	0.1883	0.0942	13.31	0.001
	Error	12	0.0849	0.0071		
	Total	20	0.4272			
100	Method	6	0.7019	0.1170	4.090	0.018
	Release	2	0.7322	0.3661	12.80	0.001
	Error	12	0.3433	0.0286		
	Total	20	1.7774			

the important relationship between the performance of a prediction technique and the characteristics of the software metrics data. A recent work by Ohlsson and Runeson (2002) demonstrated that replicating empirical studies on classification models in a context other than the original, can yield different results depending on how different parameters are chosen. In the case of the legacy system studied in this paper, it was observed that the software development process improved with time, i.e., the number of *fp* modules reduced over the different system releases.

It is also seen in Table 7 that for all the cost ratios, the seven classification models are different from each other at a 10% significance level. This was an important step in our comparative study, because if the ANOVA models had revealed insignificant *p*-values for the "method" factor, then there would be no need to perform multiple pairwise comparisons to evaluate the relative differences of the seven models. However, since the *p*-values are significant, we proceeded with the pairwise comparisons for all the seven methods. A given model among the seven is compared with the other six models using a one-tailed pairwise comparison. For example, CART is compared with SPL, SPT, TD, C4.5, LOG, and CBR individually. Thus, for each pair of methods (say A and B), we have two comparisons: is A better than B? and is B better than A? The null and alternate hypotheses for the multiple pairwise comparisons are presented in Section 4.7.

The *p*-values obtained from the multiple pairwise comparisons are presented in Table 8. For a given cost ratio, the table can be viewed as a 7×7 matrix, i.e., each pair of two methods forms a comparison. This implies that methods in the second column are compared with the methods listed in the headings of the tables. Example comparisons would be, {SPT vs. CART, CART vs. SPT}, {SPT vs. SPL, SPL vs. SPT}, and so on. A * in the table indicates the fact that a given model is not compared to itself. Since we have seven modeling methods, there are 42 comparisons for each cost ratio. The *p*-values indicate the significance level of difference in the NECM values between two methods for a particular cost ratio.

We are interested in determining (if possible) the performance order of these seven models. To do so, we present our discussion based on the comparisons for one of the cost ratios shown in Table 8. Subsequently, similar analysis and discussions can be made for the other cost ratios. Let us consider the comparisons for $C_{II}/C_I = 25$. A quick look at the *p*-values for this case indicates that many of the pairwise comparisons are not statistically significant at a 10% significance level: shown by the many 1s. However, *p*-values of some of the comparisons are significant at 10%. In the next few paragraphs we analyze the *p*-values for this cost ratio, and comment on the significance of their relative performances.

When LOG is compared to CART, SPL, and SPT we observe that there are no statistical differences (*p*-values of 1.000) in their relative performances, that is, all four models essentially yielded similar NECM values for a cost ratio of 25. We also observe that when compared to C4.5, the LOG model is not better ($p = 0.399$) at the 10% significance level. However, when compared to CBR ($p = 0.019$) and TD ($p = 0.086$), its performance is significantly better. Observing the pairwise comparisons between TD, C4.5, and CBR it is indicated that neither is significantly better than the other two.

Table 8. Multiple pairwise comparison: p-values.

C_{II}/C_I	Model	SPT	CART	SPL	TD	C4.5	CBR	LOG
20	SPT	*	1.000	1.000	0.371	1.000	**0.086**	1.000
	CART	1.000	*	1.000	0.448	1.000	**0.106**	1.000
	SPL	1.000	1.000	*	0.301	1.000	**0.068**	1.000
	TD	1.000	1.000	1.000	*	1.000	1.000	1.000
	C4.5	1.000	1.000	1.000	1.000	*	0.324	1.000
	CBR	1.000	1.000	1.000	1.000	1.000	*	1.000
	LOG	1.000	1.000	1.000	0.128	0.598	**0.028**	*
25	SPT	*	1.000	1.000	0.402	1.000	**0.098**	1.000
	CART	1.000	*	1.000	0.237	0.944	**0.055**	1.000
	SPL	1.000	1.000	*	0.262	1.000	**0.061**	1.000
	TD	1.000	1.000	1.000	*	1.000	1.000	1.000
	C4.5	1.000	1.000	1.000	1.000	*	0.357	1.000
	CBR	1.000	1.000	1.000	1.000	1.000	*	1.000
	LOG	1.000	1.000	1.000	**0.086**	0.399	**0.019**	*
50	SPT	*	1.000	1.000	0.513	1.000	0.145	1.000
	CART	0.764	*	1.000	**0.063**	0.252	**0.016**	1.000
	SPL	1.000	1.000	*	0.215	0.754	**0.056**	1.000
	TD	1.000	1.000	1.000	*	1.000	1.000	1.000
	C4.5	1.000	1.000	1.000	1.000	*	0.471	1.000
	CBR	1.000	1.000	1.000	1.000	1.000	*	1.000
	LOG	0.555	1.000	1.000	**0.043**	0.175	**0.011**	*
100	SPT	*	1.000	1.000	0.598	1.000	0.188	1.000
	CART	0.387	*	1.000	**0.034**	0.126	**0.010**	1.000
	SPL	1.000	1.000	*	0.205	0.6592	**0.059**	1.000
	TD	1.000	1.000	1.000	*	1.000	1.000	1.000
	C4.5	1.000	1.000	1.000	1.000	*	0.561	1.000
	CBR	1.000	1.000	1.000	1.000	1.000	*	1.000
	LOG	0.372	1.000	0.997	**0.033**	0.119	**0.009**	*

If we denote LOG, CART, SPL, and SPT into one group (say A), and C4.5, TD, and CBR into another group (say B), we note that each of the models in a given group perform similar to the other models in that group. So practically speaking, if one were to pick a model from each of the groups, then any of the respective models can be selected without significantly sacrificing performance. It is indicated that all the models of group A are significantly better (at 10%) than the CBR model of group B, as indicated by the respective p-values. All the models of group A, except LOG, do not perform better (or are similar) than the C4.5 and TD models of group B at the 10% significance level. We observe an overlap between the two groups, i.e., the CART, SPL, and SPT models of group A and the C4.5, and TD models of group B performed similar. Therefore, among these five techniques any one can be picked without significantly sacrificing performance.

Therefore, for the cost ratio of 25 we can identify two performance-based clusters, i.e., {LOG, CART, SPL, SPT} and {C4.5, TD, CBR}. The grouping of models into two clusters does not indicate that all models of group A are better than all models of group B. It simply indicates that all models within their respective groups showed

Table 9. Classification performances based on LOC.

	Misclassification rates (%)			NECM for each C_{II}/C_I			
Release	Type I	Type II	Overall	20	25	50	100
1	33.25	32.31	33.19	0.719	0.820	1.329	2.347
2	30.25	23.28	29.92	0.577	0.650	1.017	1.750
3	36.00	27.66	35.89	0.686	0.773	1.209	2.080
4	31.47	25.00	31.20	0.610	0.689	1.082	1.870

similar performances with respect to their ECM values. As per the discussion in the previous two paragraphs, there is clearly an overlap between the two clusters. A pure or strict ranking (at 10% significance) of all the seven models is not feasible for this case.

The classification results obtained by ordering the modules of Release 1 and determining the appropriate FP_{LOC} value as discussed in Section 4.8 are shown in Table 9. It was determined that the top 1292 modules of Release 1 would be considered *fp* because the corresponding Type I and Type II error rates (shown in Table 9) were the most appropriate according to the model selection strategy. When comparing the NECM values (for all cost ratios) of Table 9 with those of the seven classification models shown in Table 6, we observe that for Release 3 all the seven classification models perform better than the LOC method. However, for Release 2 the latter generally performed better than or similar to the seven models.

Among the models of group A, we observe that the LOG and SPL models perform better than the LOC method for Releases 3 and 4. Among the models of group B, we observe that the NECM values of LOC method for Releases 2 and 4 are lower than those of models in group B. Overall, it is observed that classification based on LOC yielded reasonably comparable performance. It did not show absolutely inferior performance nor did it yield absolutely superior performance. However, the reasonably comparable performance of LOC is limited only to this case study and its environment, and may be reflective of the high degree of maturity of the high-assurance legacy system and its software development process. In the case of a different software system and/or different environment, classification based on a simple method such as LOC may not yield similar performance as shown for the LLTS software system.

A comparative analysis, similar to the one illustrated above, of the p-values shown in Table 8 can be obtained for the other cost ratios. Though, the cost ratios considered in our case study reflect a high-assurance systems such as LLTS, this comparative work can be performed for other software systems, such as business applications. However, since the needs of another system may be different, the modeling objective and model-selection strategy may differ than those discussed for the legacy telecommunication system. The observations and conclusions stated in this study cannot be generalized for another software system, because it may have different quality improvement objectives. In addition, the cost ratio for another software system may also be different. However, the comparative methodology can certainly be applied to other systems for selecting a preferred model among respective competitors.

In an empirical software engineering effort, threats to internal validity are unaccounted influences that may affect case study results. In the context of this study, poor estimates can be caused by a wide variety of factors, including measurement errors while collecting and recording software metrics; modeling errors due to the unskilled use of software applications; errors in model-selection during the modeling process; and the presence of outliers and noise in the training data set. Measurement errors are inherent to the data collection effort, which has been discussed earlier. In our comparative study, a common model-building and model-selection approach have been adopted. Moreover, the statistical analysis (ANOVA models, pairwise comparison, etc.) was performed by only one skilled person in order to keep modeling errors to a minimum.

Threats to external validity are conditions that limit generalization of case study results. To be credible, the software engineering community demands that the subject of an empirical study be a system with the following characteristics (Votta and Porter, 1995): (1) developed by a group, rather than an individual; (2) developed by professionals, rather than students; (3) developed in an industrial environment, rather than an artificial setting; and (4) large enough to be comparable to real industry projects. The software system investigated in this study meets all these requirements.

6. Summary

Software quality estimation models can effectively minimize software failures, improving the operational-reliability of software-based systems. Many software applications involve the use of software in high-assurance systems, which creates the need to develop and quantify effective software quality prediction models. The timely prediction of *fp* components during the software development process can enable the software quality assurance team to target inspection and testing efforts to those components. Software quality classification modeling techniques facilitate such a timely estimation. Software metrics-based quality classification models, classify software modules into classes.

In this study, we compared the predictive performances of seven classification methods which were used to build two-group classification models that classified modules as either *fp* or *nfp*. The modeling methods that were compared included CART, S-PLUS, the Sprint-Sliq algorithm, the C4.5 algorithm, the Treedisc algorithm, case-based reasoning, and logistic regression. The comparative study is presented through a case study of a legacy telecommunications system. ANOVA block design models are designed to study whether the classification methods were significantly different than each other with respect to their ECM values. The block design was also used to investigate whether the system releases were significantly different than each other with respect to a given model's predictive performance.

Expected cost of misclassification is introduced as a unified singular performance measure for comparing different software quality classification models. The use of ECM was warranted because of the misleading tendencies and ambiguities associated when Type I and Type II errors are used to compare different classification models. It incorporates the importance of misclassification costs on the usefulness of classification models. The relative performances of the seven models was investigated by performing one-tailed multiple pairwise comparisons with respect to their ECM values. Models were evaluated for a set of cost ratio values that were considered appropriate for the legacy telecommunications system, that is, 20 to 100.

We note that our comparative evaluation for the case study is based on comparing only the ECM values of the different techniques. Other criteria, such as simplicity in model-calibration, complexity of model-interpretation, and stability of models across releases may decide which modeling techniques are better than others. We also reiterate that the results of this paper cannot be generalized because the comparative study is based on a single software system. Moreover, due to the lack of intuition as to why some classification techniques sometimes behaved differently than others, it is not possible to draw a definitive conclusion about which method(s) will perform the best. We also note that the success/failure of a particular prediction method has a strong relationship with the characteristics of the estimation problem, specifically the characteristics of the data set(s) and the system domain (Shepperd and Kadoda, 2001).

Future work may include, investigating a similar comparative study as the one presented in this study, with software metrics and fault data from a software system other than a telecommunications system.

Acknowledgments

We thank the three anonymous reviewers for their insightful comments. We also thank the coordinating editor for providing useful suggestions. We thank John P. Hudepohl, Wendell D. Jones and the EMERALD team for collecting the necessary case-study data. We also thank Meihui Mao for her assistance with data analysis and statistical modeling. This work was supported in part by Cooperative Agreement NCC 2-1141 from NASA Ames Research Center, Software Technology Division, and Center Software Initiative for the NASA software IV&V Facility at Fairmont, West Virginia.

Notes

1. Some of the notations used in this paper are presented in Table 1.
2. This section only presents a discussion on how to obtain the preferred balance with logistic regression, and is not meant to provide a complete classification procedure of the technique.

3. In the paper, the notation ECM refers to the normalized expected cost of misclassification (NECM), and the two are used interchangeably.
4. The notations used for the different methods/techniques/tools are shown in Table 1.

References

Basili, V. R., Briand, L. C., and Melo, W. L. 1996. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering* 22(10): 751–761, October.

Beaumont, G. P. 1996. *Statistical Tests: An Introduction with Minitab Commentary*. Prentice Hall.

Berenson, M. L., Levine, D. M., and Goldstein, M. 1983. *Intermediate Statistical Methods and Applications: A Computer Package Approach*. Englewood Cliffs, NJ: Prentice Hall, USA.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. 1984. *Classification And Regression Trees*. Belmont, California, USA: Wadsworth International Group, 2nd edition.

Briand, L. C., Basili, V. R., and Hetmanski, C. J. 1993. Developing interpretable models with optimized set reduction for identifying high-risk software components. *IEEE Transactions on Software Engineering* 19(11): 1028–1044, November.

Briand, L. C., Melo, W. L., and Wust, J. 2002. Assessing the applicability of fault-proneness models across object-oriented software projects. *IEEE Transactions on Software Engineering* 28(7): 706–720.

Clark, L. A., and Pregibon, D. 1992. Tree-based models. In J. M. Chambers and T. J. Hastie, (eds), *Statistical Models in S*. Wadsworth International Group, Pacific Grove, California, pp. 377–419.

Ebert, C. 1996. Classification techniques for metric-based software development. *Software Quality Journal* 5(4): 255–272.

Fayyad, U. M. 1996. Data mining and knowledge discovery: Making sense out of data. *IEEE Expert* 11(4): 20–25.

Johnson, R. A., and Wichern, D. W. 1992. *Applied Multivariate Statistical Analysis*. Englewood Cliffs, NJ: Prentice Hall, USA, 2nd edition.

Khoshgoftaar, T. M., and Allen, E. B. 1999. Logistic regression modeling of software quality. *International Journal of Reliability, Quality and Safety Engineering* 6(4): 303–317.

Khoshgoftaar, T. M., and Allen, E. B. 2000. A practical classification rule for software quality models. *IEEE Transactions on Reliability* 49(2): 209–216.

Khoshgoftaar, T. M., and Allen, E. B. 2001. Controlling overfitting in classification-tree models of software quality. *Empirical Software Engineering* 6(1): 59–79.

Khoshgoftaar, T. M., Allen, E. B., and Deng, J. 2002. Using regression trees to classify fault-prone software modules. *IEEE Transactions on Reliability* 51(4): 455–462.

Khoshgoftaar, T. M., Allen, E. B., Hudepohl, J. P., and Aud, S. J. 1997. Applications of neural networks to software quality modeling of a very large telecommunications system. *IEEE Transactions on Neural Networks* 8(4): 902–909.

Khoshgoftaar, T. M., Allen, E. B., Jones, W. D., and Hudepohl, J. P. 2000. Classification tree models of software-quality over multiple releases. *IEEE Transactions on Reliability* 49(1): 4–11.

Khoshgoftaar, T. M., and Seliya, N. 2002. Software quality classification modeling using the SPRINT decision tree algorithm. In *Proceedings: 14th International Conference on Tools with Artificial Intelligence*. Washington, DC, USA: IEEE Computer Society, November, pp. 365–374.

Kolodner, J. 1993. *Case-Based Reasoning*. San Mateo, California USA: Morgan Kaufmann Publishers, Inc.

Leake, D. B. (ed.) 1996. *Case-Based Reasoning: Experience, Lessons, and Future Directions*. Cambridge, MA USA: MIT Press.

Mehta, M., Agarwal, R., and Rissanen, J. 1996. SLIQ: A fast scalable classifier for data mining. IBM White Paper Series: Available at www.almaden.ibm.com/cs/quest, March.

Mehta, M., Rissanen, J., and Agarwal, R. 1995. MDL-based decision tree pruning. IBM White Paper Series: Available at www.almaden.ibm.com/cs/quest, August.

Myers, R. H. 1990. *Classical and Modern Regression with Applications*. PWS-KENT, Boston, MA, USA.

Neter, J., Kutner, M. H., Nachtsheim, C. J., and Wasserman, W. 1996. *Applied Linear Statistical Models*. 4th edition. Chicago, IL, USA: Irwin McGraw-Hill.

Ohlsson, M. C., Helander, M., and Wohlin, C. 1996. Quality improvement by identification of fault-prone modules using software design metrics. In *Proceedings: International Conference on Software Quality*, Ottawa, Ontario, Canada, pp. 1–13.

Ohlsson, M. C., and Runeson, P. 2002. Experience from replicating empirical studies on prediction models. In *Proceedings: 8th International Software Metrics Symposium*. Ottawa, Ontario, Canada: IEEE Computer Society, June, pp. 217–226.

Ohlsson, N., Zhao, M., and Helander, M. 1998. Application of multivariate analysis for software fault prediction. *Software Quality Journal* 7(1): 51–66.

Paul, R. 1992. Metric-based neural network classification tool for analyzing large-scale software. In *Proceedings: International Conference on Tools with Artificial Intelligence*. Arlington, Virginia, USA: IEEE Computer Society, November, pp. 108–113.

Ping, Y., Systa, T., and Muller, H. 2002. Predicting fault-proneness using OO metrics, an industrial case study. In T. Gyimothy and F. B. Abreu (eds), *Proceedings: 6th European Conference on Software Maintenance and Reengineering*. Budapest, Hungary, March, pp. 99–107.

Pizzi, N. J., Summers, A. R., and Pedrycz, W. 2002. Software quality prediction using median-adjusted class labels. In *Proceedings: International Joint Conference on Neural Networks*, volume 3. Honolulu, Hawaii, USA: IEEE Computer Society, May, pp. 2405–2409.

Ponnuswamy, V. 2001. Classification of software quality with tree modeling using C4.5 algorithm. Master's thesis, Florida Atlantic University, Boca Raton, Florida USA, December. Advised by Taghi M. Khoshgoftaar.

Quinlan, J. R. 1993. Machine Learning, *C4.5: Programs For Machine Learning*. San Mateo, California: Morgan Kaufmann.

Ross, F. D. 2001. An empirical study of analogy based software quality classification models. Master's thesis, Florida Atlantic University, Boca Raton, FL USA, August. Advised by T. M. Khoshgoftaar.

Schneidewind, N. F. 1995. Software metrics validation: Space shuttle flight software example. *Annals of Software Engineering* 1: 287–309.

Schneidewind, N. F. 2001. Investigation of logistic regression as a discriminant of software quality. In *Proceedings: 7th International Software Metrics Symposium*. London UK: IEEE Computer Society, April, pp. 328–337.

Shafer, J., Agarwal, R., and Mehta, M. 1996. SPRINT: A scalable parallel classifier for data mining. IBM White Paper Series: Available at www.almaden.ibm.com/cs/quest, September.

Shepperd, M., and Kadoda, G. 2001. Comparing software prediction techniques using simulation. *IEEE Transactions on Software Engineering* 27(11): 1014–1022.

Steinberg, D., and Colla, P. 1995. *CART: A supplementary module for SYSTAT*. Salford Systems, San Diego, California.

Suarez, A., and Lutsko, J. F. 1999. Globally optimal fuzzy decision trees for classification and regression. *Pattern Analysis and Machine Intelligence* 21(12): 1297–1311, IEEE Computer Society.

Takahashi, R., Muraoka, Y., and Nakamura, Y. 1997. Building software quality classification trees: Approach, experimentation, evaluation. In *Proceedings: 8th International Symposium on Software Reliability Engineering*. Albuquerque, NM, USA: IEEE Computer Society: November, pp. 222–233.

Votta, L. G., and Porter, A. A. 1995. Experimental software engineering: A report on the state of the art. In *Proceedings of the 17th. International Conference on Software Engineering*. Seattle, WA USA: IEEE Computer Society, April, pp. 277–279.

Taghi M. Khoshgoftaar is a professor of the Department of Computer Science and Engineering, Florida Atlantic University and the Director of the Empirical Software Engineering Laboratory. His research interests are in software engineering, software complexity metrics and measurements, software reliability and quality engineering, computational intelligence, computer performance evaluation, multimedia systems, and statistical modeling. He has published more than 150 refereed papers in these areas. He has been a principal investigator and project leader in a number of projects with industry, government, and other research-sponsoring agencies. He is a member of the Association for Computing Machinery, the IEEE Computer Society, and IEEE Reliability Society. He served as the general chair of the 1999 IEEE International Symposium on Software Reliability Engineering (ISSRE'99). He was the recipient of the Researcher of the Year Award at Florida Atlantic University for the year 2000. He is the general chair of the 2001 IEEE International Symposium on High-Assurane Systems Engineering HASE'01). He has served on technical program committees of various international conferences, symposia, and workshops. He has served as North American editor of the *Software Quality Journal*, and was on the editorial board of the *Journal of Multimedia Tools and Applications*.

Naeem Seliya recieved the MS degree in Computer Science from Florida Atlantic University, Boca Raton, FL, USA, in 2001. He is currently a Ph.D. candidate in the Department of Computer Science and Engineering at Florida Atlantic University. His research interests include software engineering with computational intelligence, software metrics, software architecture, data mining, software quality and reliability, object oriented design, and computer data security. He is student member of the IEEE Computer Society and the Association for Computing Machinery.