An Investigation on the Use of Machine Learned Models for Estimating Software Correctability

Mauricio A. de Almeida[1] **and Hakim Lounis**

Centre de Recherche Informatique de Montréal
550, Sherbrooke O., #100
Montréal, H3A 1B9 Qc, Canada
{mdealmei, hlounis}@crim.ca
phone: (1) (514) 840-1234

Walcelio L. Melo

Oracle Brazil and
Universidade Católica de Brasília
SCN Qd. 02 - Bl. A- Salas 604
Brasilia, DF Brazil 70712-900
wmelo@acm.org
phone: (55) (61) 327-5151

1. Guest researcher at CRIM and assistant professor at Faculdade de Tecnologia de Sao Paulo, Sao Paulo , Brazil.

ABSTRACT

In this paper we present the results of an empirical study in which we have investigated Machine Learning (ML) algorithms with regard to their capabilities to accurately assess the correctability of faulty software components. Three different families algorithms have been analyzed: Top Down Induction Decision Tree, covering, and Inductive Logic Programming (ILP). We have used (1) fault data collected on corrective maintenance activities for the Generalized Support Software reuse asset library located at the Flight Dynamics Division of NASA's GSFC and (2) product measures extracted directly from the faulty components of this library. In our data set, the software quality models generated by the inductive logic programming (ILP) presented the best results from the point of view model accuracy. In addition, the models generated by ILP algorithms, in form of first-order logic rules, can be used as coding guidelines.

Keywords

Software correctability, machine learning algorithms, predictive software quality model building.

1.0 INTRODUCTION

Software maintenance consumes most of the resources in many software organizations. We must be able to better characterize, assess, and improve the maintainability of software products in order to decrease maintenance costs. Maintenance involves activities such as correcting errors, migrating software to new technologies, and adapting software to deal with new environment requirements.

Corrective maintenance is the part of software maintenance devoted to correcting errors. Mostly, when software maintainers have to correct a faulty software component, they rely almost exclusively on their previous experience in order to estimate the effort they will spend to do it. Even though highly experienced software maintainers may make accurate predictions, the estimation process remain informal, error-prone, and poorly documented, making it difficult to replicate and spread throughout the organization. In general, software maintenance organizations tend to assign corrective maintenance activities to young software engineers who do not know a great deal about software systems they have to maintain.

In order to improve corrective maintenance, we must be able to provide models which help software maintainers better assess the maintainability of software products and estimate corrective maintenance effort. The benefits of having such models for software maintenance are numerous. For instance, estimation models can help us optimize the allocation of resources to corrective maintenance activities. Evaluation models can help us made decisions about when to re-structure or re-engineer a software component in order to make it more maintainable. Understanding models can help us know better the underlying reasons about the difficulty of correcting specific kinds of errors.

Many different approaches have been proposed to build corrective maintenance estimation/evaluation models (see Section 2.0). In this paper, we show the results of an empirical study in which we have investigated different ML algorithms with regard to their capabilities to generate accurate correctability models. To do so, we have studied four very-well known, public-domain ML algorithms belonging to three different families of ML techniques. We have compared these algorithms with regard to their capabilities to assess the difficulty of correct Ada faulty components. To do so, we have used (1) data collected on corrective maintenance activities for the Generalized Support Software reuse asset library located at the Flight

Dynamics Division of NASA's GSFC and (2) internal product measures extracted directly from the faulty components of this library. The results show that ML algorithms are able to generate statistically significant prediction models. These prediction models, also, demonstrated to be more accurate than those built with logistic regression combined with principal component analysis.

The paper is organized as follows: Section 2.0 describes related work. Section 3.0 briefly presents the ML algorithms we have studied. Section 4.0 presents the data used in this comparative study and the analysis method. Section 5.0 provides the experimental results. Finally, conclusions and directions for future research are outlined.

2.0 RELATED WORK

As far as we know, Selby and Porter [24] have been the first to use a ML classification algorithm to automatically construct software quality models. They have used ID3, a ML classification algorithm, to identify those product measures that are the best predictors of interface errors likely to be encountered during maintenance.

After Selby & Porter many others, e.g., [3], [15], have used ML classification algorithms to construct software quality predictive models. In the arena of building up corrective maintenance cost models via ML classification algorithms, as far as we know, the works we present in Table 1 are the most relevant. Table 1 provides a schematic comparison of these works with regard to the following criteria: ML classification algorithm(s), data set, software measures, where ASAP and AMADEUS stands for the Ada products measures provided by the ASAP tool and the Amadeus system, respectively.

TABLE 1. Related work

		Data Set		
Related Work	**Classification Technique**	**Application domain**	**Environment**	**Product Measures**
Briand & al., 1993 [7]	OSR & Logistic Regression	Flight Dynamic Application	NASA GFSC FDD	ASAP
Jor-gensen [15]	OSR & Neural Net-works	Manage-ment Information Systems	Distinct Norvegian Software Companies	Lines of Code
Basili & al., 1997 [3]	C4.5	GSS reuse asset library	NASA GFSC FDD	Ama-deus
Our work	C4.5, NewID, CN2, and FOIL	GSS reuse asset library	NASA GFSC FDD	ASAP

Briand & al. [7] have constructed cost models for error isolation and error correction. To do so, they have used OSR [8], an approach which combines statistical and ML classification principles. Like us, the predictor variables for the model are product measures extracted via ASAP from a set of Ada components from the NASA SEL. They also benchmark OSR against a traditional machine learning algorithm (ID3/NewID) and logistic regression. The correctness of the classification model generated with OSR was considered superior than both the logistic regression and ID3/NewID models.

Jorgensen [15] has constructed a predictive model of the cost of corrective maintenance using OSR for generating a logical classification model. The predictor variable for the model is the size measured in lines of code. The data used came from distinct software organizations and application domains. He has compared OSR to neural networks and least-square regression. The logical classification model was deemed more accurate than the least-square regression and neural network classification model.

More recently, Basili & al. [3] have constructed a corrective maintenance cost model using the C4.5 system [28] for generating a logical classification model. The predictor variables for the model are measures of internal software product attributes extracted with Amadeus [1]. The model demonstrates good prediction accuracy. Our work is, in fact, a continuation of Basili et al.'s work: the data set we have used comprises the one used by them. We have also used the same version of C4.5, i.e., version 8.

The next section presents the ML algorithms we have used in this study. The data we have used in our study are available under request allowing thus other researchers to compare our results with their own classification techniques.

3.0 MACHINE LEARNING ALGORITHMS

Most of the work done in machine learning has focused on supervised machine learning algorithms. Starting from the description of classified examples, these algorithms produce definitions for each class. In general, they use an attribute-value representation language that allows the use of statistical properties on the learning set. Nevertheless, others use the first order logic language. It has better expressive capabilities than the attribute-value language. It permits the expression of relations between objects. An important consequence is the diminution of the learning data-set size. Both are helpful for constructing efficient software quality models, and section 5 will present some results obtained with them. This section gives an overview of machine learning algorithms we have used in this work. Two algorithm methods emerge in the attribute-value-based family: the divide and conquer method and the covering one. The following table summarizes the four ML algorithms we have used.

TABLE 2. ML algorithms used in the study

ML algorithms	Algorithm family	Description language	Induced knowledge
NewID [5]	Divide & conquer family: Top Down Induction Decision Tree -TDIDT-	Attribute-value	Decision tree
CN2 [11]	Covering family	Attribute-value	Rules
C4.5 [28]	Divide & conquer family: -TDIDT-	Attribute-value	Decision tree & rules
FOIL [29]	Inductive Logic Programming -ILP-	First order logic	Clauses

3.1 The divide and conquer family

In this family, the induced knowledge is generally represented by a decision tree. It is the case of algorithms like ID3 [26] [27], CART [6], ASSISTANT [10]. The principle of this approach could be summarized by this algorithm:

If all the examples are of the same class

Then - create a leaf labelled by the class name;

Else - select a test based on one attribute;

- divide the training set into subsets, each associated to one of the possible values of the tested attribute;

- apply the same procedure to each subset;

Endif.

The key step of the algorithm above is the selection of the "best" attribute to obtain compact trees with high predictive accuracy. Information-based heuristics have provided effective guidance for the division process.

NewID and C4.5 are two algorithms of this family. They induce Classification Models, also called Decision Trees, from data. They are derived from the well known ID3 algorithm. ID3 works with a set of examples where each example has the same structure, consisting of a number of attribute/value pairs. One of these attributes represents the class of the example. The problem is to determine a decision tree that on the basis of answers to questions about the non-class attributes, correctly predicts the value of the class attribute. Usually the class attribute takes only the values {true, false}, or {success, failure}, or something equivalent.

A decision tree is important not because it summarizes what we know, i.e. the training set, but because we hope it will classify correctly new cases. Thus when building classification models one should have both training data to build the model and test data to verify how well it actually works.

A measure of entropy is used to measure how informative a node is. In general, if we are given a probability distribution $P=(p_1, p_2, ..., p_n)$, then the Information conveyed by this distribution, also called the Entropy of P, is:

$$I(P) = -(p_1*log(p_1) + p_2*log(p_2) + .. + p_n*log(p_n))$$

This notion is exploited to rank attributes and to build decision trees where at each node is located the attribute with greatest gain among the attributes not yet considered in the path from the root.

One of the weaknesses of the original ID3 is that it does not directly deal with attributes that have continuous ranges or unknown values. So below we examine how it can be extended to do so.

- NEWID and C4.5 Extensions:

C4.5 and NewID introduce a number of extensions of the original ID3 algorithm. C4.5 is an extension of ID3 that accounts for unavailable values, continuous attribute value ranges, pruning of decision trees, rule derivation, and so on. Like C4.5, NewID accounts for continuous attribute value ranges, unknown and don't-care values, and post-pruning of decision trees. Let us explain some of these extensions.

In building a decision tree, we can deal with training sets that have examples with unknown attribute values by evaluating the gain for an attribute by considering only the examples where that attribute is defined. In using a decision tree, examples that have unknown attribute values are classified by estimating the probability of the various possible results.

To deal with the case of attributes with continuous ranges, we can proceed as follows. Say that attribute A_i has a continuous range. We examine the values for this attribute in the training set. Say they are, in increasing order, V_1, V_2, ..., V_m. Then for each value V_j, j=1..m, we partition the examples into those that have A_i values up to and including V_j, and those that have values greater than V_j. For each of these partitions we compute the gain, and choose the partition that maximizes the gain.

The decision tree built using the training set, because of the way it was built, deals correctly with most of the examples in the training set. In fact, in order to do so, it may become quite complex. Pruning of the decision tree is done by replacing a whole subtree by a leaf node. The replacement takes place if a decision rule establishes that the expected error rate in the subtree is greater than in the single leaf.

Finally, it is easy to derive a rule set from a decision tree by writing a rule for each path in the decision tree from the root to a leaf. In that rule the left-hand side is easily built from the label of the nodes and the labels of the edges. The resulting rules set can be simplified: let LHS be the left-hand side of a rule, and let LHS' be obtained from LHS by eliminating some of its conditions. We can certainly replace

LHS by LHS' in this rule if the subsets of the training set that satisfy respectively LHS and LHS' are equal.

3.2 The covering family

The covering family represents classification knowledge as a disjunctive logical expression defining each class. It could be summarized by the following algorithm:

- find a conjunction that is satisfied by some examples of the target class, but no examples from another class;

- append this conjunction as one disjunct of the logical expression being developed;

- remove all examples that satisfy this conjunction and, if there are still some remaining examples of the target class, repeat the process.

CN2 [11] belongs to the covering family. It is considered as a descendant of AQ [30]. It shares the previous algorithm but uses a divide and conquer method to construct a suitable conjunction; for this reason, it is also considered as a descendant of ID3. It induces classification rules expressed in an attribute-value formalism (propositional calculus with typed variables). In this context, an example is a conjunction $\&_i$ $attribute_i=val$ $(i=1, n)$, where n is the number of attributes in the language. The aim of CN2 is to induce a complete and coherent description (with a polynomial complexity) using a version of the star algorithm:

Repeat

- start with the general rule: "everything --> <class>";

- specialize the rule;

- retain the more significant disjunctive term;

Until no more rules to find.

CN2, NewID, and C4.5 are attribute-value systems. Indeed, their description language is the propositional one, and specifically, the attribute-value language. Let us enumerate some problems encountered with this type of description language.

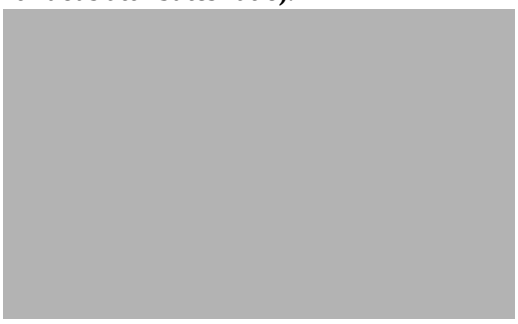
- Some problems with attribute-value systems:

- Performances of both CN2 and NEWID decrease when the learning set is corrupted with noise.

- Don't-care and unknown values produce the drop of classification performances; the correctness ratio of the two algorithms decreases and the size of the induced knowledge (e.g., the decision tree) takes large proportion.

- In presence of attributes with continue value ranges, the predictive accuracy of the algorithms becomes worse than when they deal only with nominal attributes. The following figure [20] illustrates this fact.

FIGURE 1. Correctness in terms of continuous attributes ratio (% of correctness in terms of continuous attributes ratio).



On the other hand, this leads to an increasing size of the learned knowledge. For example, the decision tree induced by NewID becomes very large in terms of nodes, leaves and path lengths.

- Languages used by these algorithms have limited expressive capabilities. Indeed, they can not express relations between objects.

FOIL, the next algorithm we present has a better expressive language; it allows the learning of interesting rules with relations between objects.

3.3 Inductive Logic Programming Family: moving to a better expressive language

First order logic language has better expressive capabilities than the attribute-value language. It permits the expression of relations between objects. An important consequence is the diminution of the learning data-set size.

FOIL represents the family of machine learning algorithms called ILP family. Contrary to its ancestors and because the propositional attribute-value formalism is limited, the expression language of FOIL is function-free Horn clause logic. It learns a logical definition represented by clauses starting from the extensional definition of the target relation. A learned clause defines the target relation in terms of itself and the other relations; so, it permits recursivity.

The operation of FOIL can be summarized as follows:

- Establish the learning set consisting of constant tuples, some labelled + and some -

- Repeat

Find a clause defining part of the target relation

Remove all tuples that satisfy the right-hand side of this clause from the learning set

Until there are no + tuples left in the learning set.

Like ID3, FOIL exploits a heuristic by computing an information-based estimate for assessing the usefulness of a literal as the next component of the right-hand side of a clause.

As we have said above, and contrary to attribute-value-based algorithms, FOIL allows us to discover rules that consider relations between objects. It has better expressive capabilities than the attribute-value language used in pre-cited algorithms. For example, running it with a set of data concerning the maintenance of a library of reusable components, we can obtain such a rule:

high(A):-executable(A,B),
maximum_statement_nesting_depth(A,C),
lines_of_comments(A,D),commentsdivsize(A,E),
N1(A,F),N_2(A,G),less_or_equal(E,F),
~less_or_equal(B,G),C<>4,C<>43, less_or_equal(C,D)

This rule can be read as:

*"a faulty component has a high corrective maintenance cost **if** the comments density (#commentsLines / # source lines of code) is less or equal to number of Operators, **and** executable statements is greater than number of operands, **and** maximum_statement_nesting_depth is less or equal to the number of lines of comments, **and** the maximum statement nesting depth is different from 4 and 43"*.

It is an example of the expression language allowed by first-order logic (or a subset of it); indeed, these relations between attributes (here, software metrics) could never be induced by attribute-value-based algorithms. However, FOIL involves a great amount of work on learning data before it runs, because it is sensitive to the way initial relations have been stated.

4.0 STUDY OVERVIEW

4.1 The studied environment

In this study, we have used data from the maintenance of a library of reusable components. This library, known as the Generalized Support Software (GSS) reuse asset library, is located at the Flight Dynamics Division (FDD) of NASA's Goddard Space Flight Center (GSFC). Component development began in 1993. Subsequent efforts focused on generating new components to populate the library and on implementing specification changes to satisfy mission requirements. The first application using this library was developed in early 1995. The asset library currently consists of 1K Ada83 components totalling approximately 515 KSLOC.

4.2 Data Collection

In this study, we collected error and fault data about this library. An *error* is represented by a single software Change Request Form (CRF) [17] filled by developers and configurers to institute and document a change to one or more components. A *fault* pertains to a single component and is evidenced by the physical change of that component in response to a particular error CRF. In this study, we have only used those components representing Ada 83 files. A *faulty* component version becomes a *fixed* component version after it is corrected. We are only interested in the Ada faulty component versions.

For each CRF, we have collected data on: (1) error identification and error correction, including the names and version numbers of the Ada source code components that had faults in them, (2) the effort expended to repair all faults associated with the error, (3) source code metrics characterizing these particular components. The ASAP tool [13] was used to extract source code metrics from the Ada faulty component versions.

Given the fact that we are only concerned with building models for assessing the cost of corrective maintenance, we have only analyzed faulty components, i.e., only components which have been modified for correcting errors. Based on the data we have, a study combining faulty and non faulty components is difficult to be undertaken, since we do not have data about the effort spend by the software engineers on analyzing non faulty components during corrective maintenance activities. Although, we believe that the software engineers spend time on non-faulty components to fix errors and thus repair faulty components. NASA SEL's change request form, which is the source of the process data used in this study, does not provide any information about effort spend on non-faulty components.

ASAP has been already used in the industry and academia in similar benchmark studies [7]. Therefore, the use of ASAP can help us compare our results with these studies. ASAP extracts a set of measures similar, e.g., size metrics, cyclomatic complexity, Hastead's metrics, etc., to other traditional tools, e.g. Amadeus [1], Matrix, Logiscop, and it is free available and well documented. ASAP shares the same weekness of these tools: most of the measures are highly correlated to size and they do not capture high level internal software product attributes, e.g. modularity. The use of higher sophisticate measures could indeed improve the accuracy of the prediction models we have generated. Unfortunately, tools able to extract higher level coding measures from Ada programs, e.g. cohesion and coupling [9] [21], either were not available when this study took place or are too expensive to be used in an academic work as this one. The construction of the tools to extract these measures in the framework of this current study would also be cost-prohibitive. Finally, it is important to point out that the definition and/or validation of software measures is beyond the scope of this paper.

4.3 Dependent variables

In our study, the dependent variable is the total effort spent to isolate and correct a faulty component. Isolation and correction effort at NASA SEL is measured on a 4-point ordinal scale: 1 hour, from 1 hour to 1 day, from 1 to 3 days, and more than 3 days. Once an error is found during configuration and testing, the maintainer finds the cause of the error, locates where the modifications are to be made, and determines that all effects of the change are accounted for. Then the maintainer modifies the design (if necessary), code, and documentation to correct the error. Once the maintainer fixes the error, the maintainer provides the names of the components changed (in our case the faulty components).

Figure 2 and Figure 3 show the distribution of the isolation effort across the 4 NASA SEL effort categories, respectively. 1HR stands for 1 hours or less, 1DAY for more than 1 hours and less than 1 working day, 3DAY for more than an one working day and less than 3 days, and, finally, NDAY for more than 3 days. As we can see in Table 3, most of the faults have been considered easy to be isolated (53% spent 1 hour or less and only 6% spent more than 1 day to be isolated) and corrected (64% spent less than 1 day to be corrected, only one spent more than 3 days).

TABLE 3. Isolation and correction effort frequencies

	Isolation Effort		Correction Effort	
Category	**Count**	**%**	**Count**	**%**
1HR	86	52.44	27	16.46
1DAY	69	42.07	78	47.56
3DAY	9	5.49	58	35.37
NDAY	0	0	1	.61

FIGURE 2. Histogram of isolation effort by faulty componentsf

FIGURE 3. Histogram of correction effort by faulty components

In general, most of the work on this area has been concerned with building dichotomous classification models about external software properties (e.g, high/low software correctability), based on product (e.g., number of lines of code), and process (e.g., change effort) metrics [3] [7] [12] [19]. To build the classification model, we have also dichotomized the corrective maintenance cost into two categories: low and high correct maintenance cost. By doing so, we facilitate the comparison of our results with other works [3] [8][12] [15] [19].

NOTE TO MAURICIO: Please include here comments about the fact ML alghoritms work better on dichotomous dependent variables and references to the study which this is commented.

To dichotomize the corrective maintenance effort in two categories, we used the following approaches.

- Regarding isolation effort, we have dichotomized the isolation effort categories in only two categories: 1 hour or less (i.e., low cost) and more than 1 hour (i.e., high cost). In this way, we have 52% of the data points classified as low cost and 48% as high cost.
- Regarding correction effort, we have dichotomized the correction effort categories also in two categories: 1 working day or less (i.e., low cost) and more than 1 day (i.e., high cost). In this way, we have now 64% of the data points classified as low cost and 36% as high cost.
- And, finally, we converted the four effort categories into average values following [4]. We assumed an 8 hour day, and took the average value for each of the categories of corrective maintenance effort. Therefore, the category of “1 Hour” was changed to 0.5 hours, the category of “1 hour to 1 Day” was changed to 4.5 hours, the category of “from 1 to 3 Days” was changed to 16 hours, and the category of “more than 3 Days” was changed to 32 hours. We then summed up these values for isolation and correction costs. This gives us an average overall corrective maintenance cost. The cost of

corrective maintenance is measured as the *total* effort taken to isolate and correct an error. We used the median of total corrective maintenance cost as the cutoff point for dichotomization. By doing so, the 164 Ada faulty components were classified in the following way: 85 components as having "high" cost and 79 as having "low" cost.

In Section 5.0, we will show how each algorithm behaves for each one of the dichotomizations described above.

4.4 Independent Variables

In this study, the independent variables are the ASAP product measures extracted from the faulty components. Other information about the CRF, such as characteristics of the error (i.e., initialization, interface, etc.) were not used, since they are not available before error isolation and, therefore, they should not be used to develop a model for predicting total corrective maintenance effort. Table 20 shows the descriptive statistics of measures. Table 21 shows the rank correlations coefficients expressed in percentage among 19 measures used in this study. The figures in bold represent the pairs of measures which presents a high correlation with each other. For instance, the pair # of operators (N1) (M18) and # of operands (N2) (M19) presents a very high correlation (Spearman square rho - r^2_s = 96%). In other words, in our data set, these two measures are extremely related to each other.

4.4.1 Evaluating Prediction Accuracy

In order to evaluate the model, we need formal measures for evaluating the classification performance of the estimation models produced by the different ML algorithms. Evaluating model accuracy tells us how good is the model expected to be as a predictor. Two criteria for evaluating the accuracy of predictions are the measures of correctness and completeness.

Correctness (high, resp. low) is defined as the percentage of components that have deemed as having a high (resp. low) corrective maintenance and were really with a high (resp. low) corrective maintenance cost. Completeness (high, resp. low) is defined as the percentage of those components that were judged as having a high (resp. low) corrective maintenance cost. All the measures above are expected to be as high as possible, because when they are low, it will lead to a wrong allocation of resources to maintain the faulty components. Table 4 summarizes the formal measures of the learned model classification performance.

TABLE 4. Formal measures of classification performance [25]

		Predicted cost: High cost	Predicted cost: Low cost	Completeness
Real cost	High cost	n_{11}	n_{12}	$n_{11}/(n_{11}+n_{12})$
Real cost	Low cost	n_{21}	n_{22}	$n_{22}/(n_{21}+n_{22})$
	Correctness	$n_{11}/(n_{11}+n_{21})$	$n_{22}/(n_{12}+n_{22})$	

The model accuracy measures how correct is the model. It is given by the formula in Equation 1. In addition, we have used a measure of prediction validity as it was presented in [19] and also used in [3]. It means that if the statistical significant coefficient *p-value* of the computed value of the Chi^2 test is less than 0.05 then we can say that the generated model has predictive validity.

(EQ 1)

$$Accuracy = \frac{\sum_{i=1}^{2} n_{ii}}{\sum_{i,j=1}^{2} n_{ij}}$$

In order to calculate the values of the formal measures of classification performance as described in Table 4, we used a V-fold cross-validation procedure [6]. For each observation X in the sample, a model is developed based on the remaining observations (sample - X). This model is then used to predict whether observation X will be classified as either costly or not costly. This validation procedure is commonly used when data sets are small.

5.0 RESULTS

5.1 Statistical Analysis

Before performing the benchmark of the 5 ML algorithms presented in Section 3.0, we will first build a predictive model using a statistical procedure. As suggested in [23], we perform a principal component analysis (PCA) [14] to determine the actual underlying dimensions of our dataset. Indeed, despite differences in their definitions, many of measures used in this study capture similar underlying dimensions, i.e., they are highly correlated with each other (see Table 21). A small number of dimensions can be used instead of the measures as potential explanatory variables, thus simplifying the subsequent analysis. Using multivariate logistic regression [18], we then analyze the relationships between correctablity and the rotated factors generated by the PCA in order to build a classification model for software correctability.

PCA yields 4 principal components whose Eigenvalue is

above one, an usual criterion in PCA to select principal components (PCs) [14]. The four selected principal components (PC) represent four orthogonal dimensions in the sample space formed by all the measures. Table 22 shows, for these four principal components, what are the weightings of each measure in the linear expression forming each rotated PC. Rotated PCs capture the same information as non-rotated ones and are sometimes referred to as factors. The larger the absolute weight associated with a measure, the larger the impact of this measure on the principal component.

In addition, Table 23 shows the Eigenvalue of each PC, the percentage of variance of the standardized variables that is explained by the rotated PC, the cumulative variance explained from top to bottom, and the cumulative eigenvalue also explained from top to bottom. The first factor accounts to 61% of the variance, factor 2 for 13%, and so on.

Based on the results obtained in the PC analysis, we can focus on the construction of the multivariate model for the purpose of classification using the univariate/multivariate logistic regression for each dichotomization approach defined in the previous section. Logistic regression has shown to have better properties than discriminant analysis, e.g., no distributional assumptions [22].

5.1.1 Isolation effort

Table 28 shows the results obtained from the standard multivariate logistic regression. Table 29 shows the contingency table obtained from the application of this multivariate logistic regression classification model. This model yields an accuracy of 58% and it has predictive validity. It means that the statistical significant coefficient *p-value* of the computed value of the Chi^2 test is less than 0.05, in this case p-value=0.0492. In Table 30, we have showed the results from the univariate logistic regression model for each factor. As we can see, only the univariate logistic regression model which uses factor 4 as independent variable is statistically significant, however, its classification model has no predictive validity (see Table 31).

5.1.2 Correction Effort

Table 32 shows the results obtained from the standard multivariate logistic regression. Table 33 shows the contingency table obtained from the application of this multivariate logistic regression classification model. Although, this model yields a good overall accuracy (65%), it has not predictive validity (p-value > 0.05). Again, the classification models generated from the univariate logistic regression of each one of the 4 factors are not statistically significant (see Table 34). Although, the classification model generated from the univariate logistic regression analysis of Factor 4 yields an accuracy of 64% and a very high completeness (low) (see Table 35), this model has not predictive validity (p-value > 0.05).

5.1.3 Average effort

Table 24 and Table 26 show the results from the standard multivariate logistic regression and the univariate logistic regression for each factor, respectively. Table 25 shows the contingency table obtained from the application of the multivariate logistic regression classification model on our data set. Also, Table 27 presents the contingency tables derived from the application of the univariate logistic regression classification for Factor 4, 3, 2, and 1, respectively. The classification models generated by the univariate logistic regression with factor 1, factor 2, and factor 3 are not statistically significant. It means that the statistical significant coefficient *p-value* of the computed value of the Chi^2 test is less than 0.05, therefore these models have not predictive validity. The classification model generated by the multivariate logistic regression yields an overall acurracy of 61% (see Table 25) and it has predictive validity ($Chi^2(1)=7.70$, p-value=0.005).

Concluding, the approach we have used to convert the four effort categories (i.e., 1 hour, from 1 hour to 1 day, from 1 to 3 days, and more than 3 days) into average values has demonstrated to be adequate. The classification models built with multivariate logistic regression using the average effort as the dependent variable and the principal components as independent variables proved to be more accurate than those using the isolation and correction effort as dependent variables. However, as we will see in the next section, the results obtained with C4.5 and Foil are better than those presented in this section.

5.2 Quantitative comparison

5.2.1 Data preparation

The ML algorithms used for this study are: C4.5, C4.5rules, CN2, NewID, and FOIL. All except FOIL use attribute-values as input representation language description and therefore are easy to use with minor input format adjustments.

The preparation of the data for FOIL is more complicated. FOIL exploits relations as input representation language instead of attribute-value. Than the first question is how to change the attribute-value data we have into a relational form. The solution we choose is to enumerate each component and associate to each component the values of each attribute by a relation named with the attribute name.

FOIL does not deal with real numbers but only with constants. This is also an important problem once part of the values in the data-set are real numbers. This was solved by declaring all the numbers that appear in the input as constants.

The final problem is then the comparison between the constants which represent the numbers. Once they are string constants, the only possible comparisons would be equality and its negation. We expect, however, that the resulting rules could have comparative relations like “less or equal” and “greater than”. To make it possible all the possible relations “less or equal” were added to the data

5.2.2 Isolation effort

Tables 5, 6, 7, 8, and 9 present the quantitative results of the study for NewID, CN2, C4.5 tree, C4.5 rules, and Foil, respectively, with the isolation effort dependant variable. All the models are statistically significant. The overal accuracy of all the model variates from 60% (NewID) to 66% (CN2, C4.5 rules and trees). C4.5 rules presents, however, the best predictive validaty. This result is, thus, better than the results shown in Section 5.1.1 which have been obtained with logistic regression.

TABLE 5. NewID results for isolation effort

		Predicted cost: Low cost	Predicted cost: High cost	Completeness
Real cost	Low cost	52	34	60%
	High cost	31	47	60%
	Correctness	63%	58%	Accuracy=60%
	$Chi^2(1)=7.03$		p-value=0.008	

TABLE 6. CN2 results for isolation effort

		Predicted cost: Low cost	Predicted cost: High cost	Completeness
Real cost	Low cost	60	26	70%
	High cost	30	48	62%
	Correctness	67%	65%	Accuracy=66%
	$Chi^2(1)=16.19$		p-value=0.0001	

TABLE 7. C4.5 tree results for isolation effort

		Predicted cost: Low cost	Predicted cost: High cost	Completeness
Real cost	Low cost	60	26	70%
	High cost	29	49	63%
	Correctness	67%	65%	Accuracy=66%
	$Chi^2(1)=17.50$		p-value<0.0001	

TABLE 8. C4.5 rules results for isolation effor

		Predicted cost: Low cost	Predicted cost: High cost	Completeness
Real cost	Low cost	52	34	60%
	High cost	21	57	73%
	Correctness	71%	63%	Accuracy=66%
	$Chi^2(1)=18.63$		p-value<0.0001	

TABLE 9. FOIL results for isolation effort

		Predicted cost: High cost	Predicted cost: Low cost	Completeness
Real cost	High cost	35	43	44%
	Low cost	20	66	76%
	Correctness	63%	60%	Accuracy=62%
	$Chi^2(1)=8.57$		p-value=0.0034	

5.2.3 Correction effort

Tables 10, 11, 12, 13, and 14 present the quantitative results of the study for NewID, CN2, C4.5 tree, C4.5 rules, and Foil, respectively, having the correction effort as the dependant variable. The overall accuracy of all the models constructed by all the algorithms is not very high (C4.5 rules and trees with 59%). The models generated by CN2, NewID and Foil have not predictive validity (p-value > 0.05). Although, the estimation models for correction effort constructed with the studied ML algorithms are not ver high, they behavior better than the model built with logistic regression. As showed in Section 5.1.2, all the correction effort estimation models built with logistic regression have no predictive validity.

TABLE 10. NewID results for correction effort

		Predicted cost: High cost	Predicted cost: Low cost	Completeness
Real cost	High cost	74	31	70%
	Low cost	33	26	44%
	Correctness	69%	45%	Accuracy=61%
	$Chi^2(1)=3.52$		p-value=0.0605	

TABLE 11. CN2 results for correction effort

		Predicted cost: High cost	Predicted cost: Low cost	Completeness
Real cost	High cost	60	46	56%
	Low cost	29	30	50%
	Correctness	67%	39%	Accuracy=55%
	$Chi^2(1)=0.85$		p-value=0.35	

TABLE 12. C4.5 tree results for correction effort

		Predicted cost: Low cost	Predicted cost: High cost	Completeness
Real cost	Low cost	96	9	91%
	High cost	59	0	0%
	Correctness	62%	0%	Accuracy=59%
	$Chi^2(1)=5.35$		p-value=0.02	

TABLE 13. C4.5 rules results for correction effort

		Predicted cost: Low cost	Predicted cost: High cost	Completeness
Real cost	Low cost	96	9	91%
	High cost	59	0	0%

TABLE 13. C4.5 rules results for correction effort

Correctness	62%	0%	Accuracy=59%
$Chi^2(1)$=5.35		p-value=0.02	

TABLE 14. FOIL results for correction effort

		Predicted cost		
		High cost	Low cost	Completeness
Real cost	High cost	11	48	18%
	Low cost	30	75	71%
	Correctness	26%	60%	Accuracy=52%
	$Chi^2(1)$=1.99		p-value=0.16	

5.2.4 Average effort

Tables 15, 16, 17, 18, and 19 present the quantitative results of the study for NewID, CN2, C4.5 tree, C4.5 rules, and Foil, respectively, in which average effort is the dependent variable.

TABLE 15. NewID results

		Predicted cost		
		High cost	Low cost	Completeness
Real cost	High cost	47	38	55%
	Low cost	41	38	48%
	Correctness	53%	50%	Accuracy=52%
	$Chi^2(1)$=0.19		p-value=0.66	

TABLE 16. CN2 results

		Predicted cost		
		High cost	Low cost	Completeness
Real cost	High cost	49	36	58%
	Low cost	39	40	51%
	Correctness	56%	53%	Accuracy=54%
	$Chi^2(1)$=1.13		p-value=0.29	

TABLE 17. C4.5 tree results

		Predicted cost		
		High cost	Low cost	Completeness
Real cost	High cost	50	35	59%
	Low cost	21	58	73%
	Correctness	70%	62%	Accuracy=66%
	$Chi^2(1)$=17.34		p-value=0.0001	

TABLE 18. C4.5 rules results

		Predicted cost		
		High cost	Low cost	Completeness
Real cost	High cost	50	35	59%
	Low cost	18	61	77%
	Correctness	74%	64%	Accuracy=68%
	$Chi^2(1)$=21.91		p-value<0.0000	

TABLE 19. Foil results

		Predicted cost		
		High cost	Low cost	Completeness
Real cost	High cost	51	33	61%
	Low cost	13	61	82%
	Correctness	80%	65%	Accuracy=71%
	$Chi^2(1)$=30.39		p-value<0.0000	

From the point of view of prediction validity measures, the models generated by NewID and CN2 are not statistically significant (Chi^2 test p-value>0.05). All the other generated models are, from this point of view, statistically significant, since the p-values are less than 0.001 (far away from the threshold of 0.05).

FOIL presents the best results in our experiment. This study confirms another one, made with artificial data sets [20]. The correctness and completeness are pretty high (around 80%) and the overall accuracy is 5% higher than the second best algorithm (C4.5 rules). This means that in some cases one can allocate resources to corrective maintenance of faulty components with a 82% of confidence.

It is also important to point out that the approach we have used to convert the four effort categories (i.e., 1 hour, from 1 hour to 1 day, from 1 to 3 days, and more than 3 days) into average values has demonstrated, again, to be adequate. For instance, the classification models generated with C4.5 rules and trees yield an overall accuracy of 68% and 66%, respectively.

Another important result is that rule-based predictive models (e.g., C4.5_rules) obtain better results than decision tree-based ones (e.g., C4.5 trees and NewID). It is interesting to notice that the pairs <NewID, CN2> and <C4.5, C4.5_rules> present the same behavior where better results are obtained with rules description than with decision trees. This seems to agree with the results obtained by [20]. In fact, the model induced by C4.5_rules is a generalization of the one induced by C4.5. It seems that an over-specialization of the model does not improve the predictive abilities of the system. Further studies are needed to provide a full explanation about such a pattern

In a recent study [3] where another set of metrics have been used, Basili and his colleagues obtained similar results using C4.5 rules (correctness 76% and overall accurary 73%). Although the results are difficult to compare, since Basili and his colleagues [3] have used a different data set and independent variables (i.e., software metrics), the results of our study demonstrated again that C4.5 rules have worked better than C4.5 decision trees. In addition, the results obtained with C4.5 rules on both studies are quite close (around 75%).

5.3 FOIL rules

As we have commented in Section 3.0, FOIL is able to built predictive software models via rules expressed in first order logic. Here, we show one of the rules generated by FOIL taken arbitrarily to exemplify the rule's interpretation.

high(A):-executable(A,B),
maximum_statement_nesting_depth(A,C),
lines_of_comments(A,D),commentsdivsize(A,E),
$N1(A,F),N_2(A,G)$,less_or_equal(E,F),
~less_or_equal(B,G),C<>4,C<>43, less_or_equal(C,D)

This rule can be read as:

> *"a faulty component has a high corrective maintenance cost* ***if*** *the comments density (#commentsLines / # source lines of code) is less or equal to number of Operators,* ***and*** *executable statements is greater than number of operands,* ***and*** *maximum_statement_nesting_depth is less or equal to the number of lines of comments,* ***and*** *the maximum statement nesting depth is different from 4 and 43".*

Let us analyze the first part of the left hand side rule *"the comments density is less or equal N_1"*. It is a comparison between two different unities.

Comments density stands for comments divided by size, where 'comments' is the number of comment lines, and 'size' is the size of the program in lines. So 'comment density' is an adimensional constant. N_1 is the number of operators. In a first look it seems that we can not compare these two metrics because they are different. The number of operators in a component is also one of many possible indirect measures of the complexity of that component. The number of comment lines and consequently the comments density of a component are direct consequences of the programming style, but as the programming style is fixed it is acceptable to consider that, as the complexity of the component grows, the number of comment lines and the comments density grows together. So, in this sense the number of comment lines and the comment density can also be indirect measures of the complexity of the component with regard to our dataset. Taking both 'number of operators' and 'comment density' as complexity measures, we think that they can be compared. The meaning of this part of the rule may be that one condition for an Ada faulty component to have a high corrective maintenance cost is when it is not well commented in comparison with the complexity revealed by the number of operators.

6.0 CONCLUSIONS

In this paper, we have empirically investigated different machine learning techniques with regard to their capabilities to generate accurate correctability models. Four very well known, public-domain machine learning (ML) algorithms have been studied. We have compared these algorithms with regard to their capabilities to assess the difficulty of correct Ada faulty components from the Generalized Support Software reuse asset library located at the Flight Dynamics Division of NASA's GSFC. The results show that the inductive logic programming algorithms are superior to the top-down induction decision tree, top-down induction attribute value rules, and covering algorithms, i.e. the overall accuracy of the model build using FOIL was higher than the other algorithms. Also, we have compared the accuracy of ML algorithms to multivariate logistic regression combined with principal components analysis. The results showed that the classification model generated by Foil and C4.5 (trees and rules) proved to be more accurate to those generated by the logistic regression technique.

The model that we developed identifies Ada fault component versions that are associated with costly corrective maintenance rather than trying to predict the exact effort for corrective maintenance a component version. We therefore use the characteristics of a faulty component version as input into the model, and the total corrective maintenance effort for the error as the output of the model. Given that the model we developed is a classification model, it classifies an Ada faulty component version into one of two cost categories: Low Cost and High Cost. This allows the model to predict whether a component version is associated with a costly, or otherwise.

A prediction identifying component versions that are going to be associated with costly errors can help managers allocate resources for the maintenance activities. It should be noted, however, that the model does not identify which component versions in the asset library are likely to have faults, only which of the faulty versions should be more or less expensive to isolate and correct. Application of such predictions assumes that the manager knows beforehand which components are likely to contain a fault. Models for the prediction of fault-prone Ada components in the SEL environment have been developed in the past [9]. Once a component version has been identified as potentially fault-prone, then it is possible to predict the cost of rework category when fixing an error that leads to faults in that version. Using this additional information, a manager can better improve the resource allocation for maintenance.

The work we have presented addresses two different but complementary domains: software engineering and machine learning domains. Our plans for the future, therefore, include new work in these two domains. With regard to software engineering, we intend to do the following work:

- The generation of other quality models, such as reliability, error-proneness, etc.
- The use of other set of measures which enriches the measures provided by ASAP. Although very well know, the set of measures we have used are not able to capture higher level product attributes, e.g., modularity.
- Replicate the study using other data sets.

- Provide guidelines which help software managers to take preventive action early in the process life-cycle. The rules generated by the ML algorithms can be a rich source of information to software engineers and managers to understand the causes of problems (in our case, costly corrective maintenance). By generating models which can be used at the early product-life cycle, i.e., design phase, we can be able to take corrective action early, and thus saving effort and resources early as well.

With regard to machine learning domain, we intend to:

- Make a deeper analysis to understand why ML algorithms that deal with rules seems to behave better than the ones which deal with trees.
- Propose an extension to FOIL in order to allow it to deal directly with numerical descriptors.
- Provide application-domain knowledge to the ML algorithms.

ACKNOWLEDGEMENTS

The authors wish to thank Vic Basili from University of Maryland -SEL- and Steven Condon from CSC for providing the data used in the paper. We are also grateful to R. Tesoriero and P. Mackenzie for their feedback on the early versions of this paper. During this work, W Melo was in part, supported by the Software Quality Group of Bell Canada, and by NSERC operation grant #OGP 0197275.

7.0 REFERENCES

[1] Amadeus Software Research Inc. *"Getting Started with Amadeus"*. Amadeus Measurement System. 1994.

[2] V. Basili, L. Briand, W. L. Melo. *""How reuse influences productivity in object-oriented systems"*, Communications of ACM, 39(10):104-116, October 1996.

[3] V. Basili, Condon, K. El Emam, R. B. Hendrick, W. L. Melo. *"Characterizing and Modeling the Cost of Rework in a Library of Reusable Software Components"*. In Proc. of the IEEE 19th Int'l. Conf. on S/W Eng., Boston, MA, May 1997.

[4] V. Basili and B. Perricone. *"Software Errors and Complexity: An Empirical Investigation"*. In CACM, 27(1):42-52, January 1984.

[5] R. Boswell. *"Manual for NewID"*. The Turing Institute, January 1990.

[6] L. Breiman, J. Friedman, R. Olshen and C. Stone. *"Classification and Regression Trees"*. Published by Wadsworth, 1984.

[7] L. Briand and V. Basili. *"A Classification Procedure for the Effective Management of Changes during the Maintenance Process"*. In Proc. of the Int'l Conf. on S/W Maintenance, pages 328-336, 1992.

[8] L. Briand, V. Basili, C. Hetmanski. *"A Pattern Recognition Approach for Software Engineering Data Analysis"*. In IEEE TSE, 18(1), Nov. 1992.

[9] L. Briand, S. Morasca, V. Basili. *"Defining and Validating High-Level Design Metrics"*. Technical Report , University of Maryland, Dep. of Computer Science, College Park, MD, 20742, 1994. CS-TR-3301.

[10] B. Cestnik, I. Bratko, I. Kononenko. *"ASSISTANT 86: a knowledge elicitation tool for sophisticated users"*. Progress in machine learning, Sigma Press, 1987.

[11] P. Clark & T. Niblet. *"The CN2 induction algorithm"*. In Machine Learning Journal, 3, p 261-283.

[12] W. W. Cohen & P. Devanbu. *"A Comparative Study of Inductive Logic Programming Methods for Software Fault Prediction"*. Technical Report AT&T Labs-Research, 1996.

[13] D. Doubleday. *"ASAP: An Ada Static Source Code Analyzer Program"*. CS-TR-1895, Computer Science Department, University of Maryland, College Park, MD. August, 1995.

[14] Dunteman. *"Principal Component Analysis"*. SAGE publications, 1989.

[15] M. Jorgensen. *"Experience with the Accuracy of Software Maintenance Task Effort Prediction Models"*. In IEEE TSE, 21(8):674-681, August 1995.

[16] M. Halstead. *"Elements of Software Science"*. North-Holland, Amsterdam, 1977.

[17] G. Heller, J. Valett and M. Wild. *"Data Collection Procedure for the Software Engineering Laboratory (SEL) Database"*. Technical Report SEL-92-002, Software Engineering Laboratory, 1992.

[18] D. Hosmer and S. Lemeshow. *"Applied Logistic Regression"*. Wiley-Interscience. 1989.

[19] F. Lanubile and G. Visaggio. *"Evaluating Predictive Quality Models Derived from Software Measures: Lessons Learned"*. Technical Report ISERN-96-03, International Software Engineering Research Network, 1996.

[20] H. Lounis & G. Bisson. *"Evaluation of learning systems: an artificial data-based approach"*. In Lecture Notes in Artificial Intelligence, p 463-481, March 1991.

[21] H. Lounis & W. L. Melo. *"Identifying and measuring coupling on modular systems"*. In Proc. of the 7th Int'l Conf. on Software Technology, Curitiba, Brazil, June 9-13, 1997, pages 23-40. Organized by Centro Internacional de Tecnologia do Software (CTIS), Curitiba, Parana, Brazil.

[22] S. Menard. "Applied Logistic Regression Analysis", SAGE publications, 1995.

[23] J. Munson and K. Khoshgoftaar. *"The Detection of Fault-Prone Programs"*. IEEE Trans. Software Eng., SE-18 (5):423-433, 1992.

[24] A. Porter and R. Selby. *"Empirically-guided software development using metric-based*

classification trees". IEEE Software, 7(2):46-54, March 1990.

[25] S. M. Weiss, C. A. Kulikowski. "*Computer Systems That Learn"*. Morgan Kaufmann Publishers, Inc. Sao Francisco, CA. 1991.

[26] J.R. Quinlan. "*Discovering rules from large collections of examples: a case study"*. In E.S in the micro-electronic age, D.Michie (Ed), Edinburgh university press, 1979.

[27] J.R. Quinlan. "*Induction of decision tree"*. Machine Learning journal 1, p 81-106, 1986.

[28] J. R. Quinlan. "*C4.5: Programs for Machine Learning"*. Morgan Kaufmann Publishers, Sao Mateo, CA, 1993.

[29] J.R. Quinlan. "*Learning Logical Definitions from Relations"*. In machine learning journal, vol 5, n°3, p 239-266, August 1990.

[30] R.S. Michalski, I. Mozetic, J. Hong, & N. Lavrac. "*The AQ15 inductive learning system: an overview and experiments*". Technical report UIUCDCS-R-86-1260, dpt of computer science, university of illinois at Urbana-Champaign.

TABLE 20. Descriptive statistics of the measures used in this study

Measure ID	Measures	Mean	Median	Minimum	Maximum	Std.Dev.
M1	**cyclomatic complexity**	65.02	51	0	272	67.77
M2	**statements**	196.15	157	3	1085	173.92
M3	**executabl**	112.24	87	0	752	136.96
M4	**declarative**	83.91	72.5	3	333	58.00
M5	**total source lines**	683.38	568.5	7	3318	547.28
M6	**Ada language statements**	196.15	157	3	1085	173.92
M7	**lines of code**	526.67	457	4	2273	386.24
M8	**maximum statement nesting dep**	13.04	4	1	96	19.78
M9	**lines of comment**	57.85	38	0	1076	111.45
M10	**Comments/size**	0.07	0.06	0	0.42	0.07
M11	**total statement nesting dep**	446.73	281.5	2	2723	515.62
M12	**inline comments**	11.02	0	0	84	16.45
M13	**average statement nesting_dep**	7.29	2.35	0.67	49.39	10.87
M14	**blank_lines**	98.86	54	1	797	121.83
M15	**blank_lines/size**	0.13	0.14	0.01	0.43	0.09
M16	**# of distint operators**	155.33	85	5	1091	186.52
M17	**# of distinct operands (n2)**	194.53	151	7	1419	214.50
M18	**# of operators(N1)**	1428.28	1210	10	6789	1151.22
M19	**# of operands (N2)**	969.90	773.5	8	5035	853.87

TABLE 21. Rank correlation (Spearman square rho - r^2_s) in percentage betweem the 19 used measures

Metrics	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15	M16	M17	M18	M19
M1	75	**92**	5	68	75	65	43	32	2	**85**	29	43	54	6	27	14	58	57
M2	100	**79**	31	**89**	**100**	**89**	38	38	1	**92**	26	40	55	2	42	35	**87**	**88**
M3		100	3	67	**79**	60	55	35	2	**82**	28	55	60	11	37	23	54	57
M4			100	34	31	42	0	6	1	21	5	0	5	10	4	15	50	45
M5				100	**89**	**95**	34	47	2	**84**	36	34	63	3	43	36	**88**	**87**
M6					100	**89**	38	38	1	92	26	40	55	2	42	35	**87**	**88**
M7						100	25	38	0	**83**	25	27	47	0	35	33	**96**	**92**
M8							100	20	3	42	37	**96**	65	26	68	56	22	31
M9								100	56	35	19	17	30	1	27	17	27	28
M10									100	1	2	2	2	1	3	0	0	0
M11										100	30	44	59	3	33	27	**79**	**81**
M12											100	32	60	24	31	29	24	27
M13												100	61	21	67	60	24	34
M14													100	43	54	43	42	48
M15														100	11	6	0	0
M16															100	**71**	33	39
M17																100	33	44
M18																	100	**96**
M19																		100

TABLE 22. Rotate Principal Components

Measure ID	Measures	Factor 1	Factor 2	Factor 3	Factor 4
M1	cyclomatic complexity	**.851505**	.116251	.051423	.238098
M2	statements	**.898579**	.392422	.136547	-.011359
M3	executabl	**.817164**	.447950	.137578	.165239
M4	declarative	**.764910**	.118957	.084584	-.424269
M5	total source lines	**.870993**	.258735	.356466	.144572
M6	Ada language statements	**.898579**	.392422	.136547	-.011359
M7	lines of code	**.938445**	.266313	.115973	.003508
M8	maximum statement nesting dep	.252627	**.944985**	.025164	.166035
M9	lines of comment	.393874	.052507	**.829323**	.152014
M10	Comments/size	-.120269	.016633	**.913011**	-.059099
M11	total statement nesting dep	**.908311**	.165106	.011069	.081385
M12	inline comments	.405575	-.015189	.571979	.435208
M13	average statement nesting_dep	.204997	**.951678**	.016295	.163836
M14	blank_lines	.577129	.269940	.474944	.499251
M15	blank_lines/size	-.062748	.232034	.098130	**.900337**
M16	# of distint operators	.336491	**.928074**	.058272	.071271
M17	# of distinct operands (n2)	.529859	**.814802**	.047109	-.043180
M18	# of operators(N1)	**.944013**	.253529	.124841	-.006661
M19	# of operands (N2)	**.922169**	.316058	.131894	.006233

TABLE 23. Eigen Values

	Eigenvalue	% total Variance	Cumul. Eigenval	Cumulative %
Factor 1	11.63319	61.22734	11.63319	61.22734
Factor2	2.40756	12.67136	14.04075	73.89870
Factor3	2.17650	11.45524	16.21725	85.35394
Factor4	1.03084	5.42545	17.24808	90.77939

TABLE 24. Average effort:
Standard multivariate logistic regression results

	Const.B0	FACTOR1	FACTOR2	FACTOR3	FACTOR4
Estimate	.07896	.2607	.18372	-.0400	.3634
Standard Error	.1609	.1656	.1682	.1606	.1631
Wald's Chi-square	.2407	2.4769	1.1932	.06208	4.9649
p-level	.623647	.11553	.27467	.80323	.02587

TABLE 25. Average effort:
Classification results from the multivariate logistic regression model

		Predicted cost		
		High cost	Low cost	Completeness
Real cost	High cost	59	26	69%
	Low cost	38	41	52%
	Correctness	61%	61%	Accuracy=61%
	$Chi^2(1)=7.70$		p-value=0.005	

TABLE 26. Average effort:
Results from univariate logistic regression for each factor

	Const.B0	FACTOR4	Const.B0	FACTOR3	Const.B0	FACTOR2	Const.B0	FACTOR1
Estimate	0.075	0.362	0.073	-0.034	0.075	0.176	0.076	0.253
Standard Error	0.159	0.161	0.156	0.155	0.157	0.159	0.158	0.162
Wald's Chi-square	0.225	5.064	0.640	0.825	0.226	1.214	0.235	2.449
p-level	0.635	0.024	0.219	0.049	0.634	0.270	0.628	0.118

TABLE 27. Average effort:
Classification results from the univariate logistic regression models

Factor 4

		Predicted cost: High cost	Predicted cost: Low cost	Completeness
Real cost	High cost	52	33	61%
	Low cost	38	41	52%
	Correctness	58%	55%	Accuracy=57%

$Chi^2(1)=2.83$ p-value=0.09

Factor 3

		Predicted cost: High cost	Predicted cost: Low cost	Completeness
Real cost	High cost	79	6	93%
	Low cost	77	2	3%
	Correctness	51%	25%	Accuracy=49%

$Chi^2(1)=1.81$ p-value=0.18

Factor 2

		Predicted cost: High cost	Predicted cost: Low cost	Completeness
Real cost	High cost	53	32	62%
	Low cost	54	25	32%
	Correctness	50%	44%	Accuracy=48%

$Chi^2(1)=0.65$ p-value=0.42

Factor 1

		Predicted cost: High cost	Predicted cost: Low cost	Completeness
Real cost	High cost	43	42	51%
	Low cost	34	45	57%
	Correctness	56%	52%	Accuracy=54%

$Chi^2(1)=0.94$ p-value=0.33

TABLE 28. Isolation effort:
Standard multivariate logistic regression results for isolation effort

$Chi^2(4)=5.1497$ p=.27230

	Const.B0	FACTOR1	FACTOR2	FACTOR3	FACTOR4
Estimate	-0.101	-0.051	0.150	0.078	-0.317
Standard Error	0.159	0.163	0.162	0.160	0.161
Wald's Chi-square	0.400	0.099	0.853	0.240	3.864
p-level	0.527	0.753	0.356	0.624	0.049

TABLE 29. Isolation effort:
Classification results from standard multivariate logistic regression results

		Predicted cost: High cost	Predicted cost: Low cost	Completeness
Real cost	High cost	40	38	51%
	Low cost	31	55	64%
	Correctness	56%	59%	Accuracy=58%

$Chi^2(1)=3.87$ p-value=0.0492

TABLE 30. Isolation effort:
Results from univariate logistic regression for each factor

Model	Chi²(1)=3.9271, p=0.047		Chi²(1)=.2334, p=0.62896		Chi²(1)=0.845, p=0.358		Chi²(1)=0.11228, p=0.7376	
	Const.B0	FACTOR4	Const.B0	FACTOR3	Const.B0	FACTOR2	Const.B0	FACTOR1
Estimate	-0.100	-0.314	-0.098	0.076	-0.098	0.145	-0.098	-0.053
Standard Error	0.158	0.160	0.156	0.158	0.157	0.158	0.156	0.157
Wald's Chi-square	0.398	3.839	0.390	0.232	0.388	0.834	0.390	0.112
p-level	0.528	0.050	0.533	0.630	0.533	0.361	0.532	0.738

TABLE 31. Isolation effort:
Classification results from the univariate logistic regression models

Factor 4

		Predicted cost: High cost	Predicted cost: Low cost	Completeness
Real cost	High cost	40	38	51%
	Low cost	34	52	60%
	Correctness	54%	58%	Accuracy=56%
	$Chi^2(1)=2.28$		p-value=0.1311	

Factor 3

		Predicted cost: High cost	Predicted cost: Low cost	Completeness
Real cost	High cost	4	74	5%
	Low cost	4	82	95%
	Correctness	50%	53%	52%
	$Chi^2(1)=0.02$		p-value=0.8824	

Factor 2

		Predicted cost: High cost	Predicted cost: Low cost	Completeness
Real cost	High cost	18	60	23%
	Low cost	17	69	80%
	Correctness	51%	53%	53%
	$Chi^2(1)=0.27$		p-value=0.60	

Factor 1

		Predicted cost: High cost	Predicted cost: Low cost	Completeness
Real cost	High cost	0	78	0%
	Low cost	0	86	100%
	Correctness	0%	52%	52%
	$Chi^2(1)=0$		p-value=0.94	

TABLE 32. Correction effort:
Standard multivariate logistic regression results

Chi²(4)=5.3433 p=0.25388

	Const.B0	FACTOR1	FACTOR2	FACTOR3	FACTOR4
Estimate	-0.597	0.016	-0.056	-0.117	0.363
Standard Error	0.166	0.171	0.171	0.167	0.170
Wald's Chi-square	12.877	0.008	0.107	0.489	4.572
p-level	0.000	0.927	0.743	0.484	0.033

TABLE 33. Correction effort:
Classification results from standard multivariate logistic regression

		Predicted cost		
		High cost	Low cost	Completeness
Real cost	High cost	5	54	8%
	Low cost	3	102	97%
	Correctness	63%	65%	Accuracy=65%
	$Chi^2(1)=2.57$		p-value=0.1090	

TABLE 34. Correction effort:
Univariate logistic regression

Model	Chi²(1)=4.723, p=0.0298		Chi²(1)=0.50717, p=0.477		Chi²(1)=0.10064 p=0.75107		Chi²(1)=0.01372 p=0.9067	
	Const.B0	FACTOR4	Const.B0	FACTOR3	Const.B0	FACTOR2	Const.B0	FACTOR1
Estimate	-0.595	0.361	-0.579	-0.122	-0.577	-0.052	-0.576	0.019
Standard Error	0.166	0.169	0.163	0.176	0.163	0.166	0.163	0.163
Wald's Chi-square	12.838	4.571	12.592	0.478	12.557	0.099	12.552	0.014
p-level	0.000	0.033	0.000	0.490	0.000	0.752	0.000	0.907

TABLE 35. Correction effort:
Classification results from univariate logistic regression results

Factor 4

		Predicted cost		
		High cost	Low cost	Completeness
Real cost	High cost	2	57	3%
	Low cost	2	103	98%
	Correctness	50%	64%	Accuracy=64%
	$Chi^2(1)=0.35$		p-value=0.5541	

Factor 3

		Predicted cost		
		High cost	Low cost	Completeness
Real cost	High cost	0	59	0%
	Low cost	0	105	100%
	Correctness	0%	64%	64%
	$Chi^2(1)=0.16$		p-value=0.68	

Factor 2

		Predicted cost		
		High cost	Low cost	Completeness
Real cost	High cost	0	59	0%
	Low cost	0	105	100%
	Correctness	0%	64%	64%
	$Chi^2(1)=0.35$		p-value=0.5541	

Factor 1

		Predicted cost		
		High cost	Low cost	Completeness
Real cost	High cost	0	59	0%
	Low cost	0	105	100%
	Correctness	0%	64%	64%
	$Chi^2(1)=0.35$		p-value=0.5541	