Empirical Software Engineering, 8, 117–138, 2003.

Fault-Threshold Prediction with Linear Programming Methodologies

MAURIZIO PIGHIN
Department of Mathematics and Computer Science, University of Udine, Italy

VILI PODGORELEC
Laboratory for System Design, University of Maribor, Slovenia

PETER KOKOL
Laboratory for System Design, University of Maribor, Slovenia

Editor: Lionel Briand

Abstract. This paper presents a new experimental methodology that operates on a series of programs structural parameters. We calculated some simple metrics on these parameters and then we applied linear programming techniques on them. It was therefore possible to define a model that can predict the risk level of a program, namely how prone it is to containing faults.

The new system represents the software files as points on an n-dimensional space (every dimension is one of the structural attributes for each file). Starting from this model the problem to find out the more dangerous files is brought back to the problem to separate two sets in $\Re^n$. A solution to this linear programming problem was achieved by using the MSM-T method (multisurface method tree), a greedy algorithm, which iterative divides the space in polyhedral regions till it reaches an empty set. The classification procedure is divided in two steps: the learning phase, which is used to tune the model on the specified environment and the effective selection. It is, therefore, possible to divide the n-dimensional space and find out the risk-regions of the space, which represent the dangerous files.

All the process was tested in an industrial application, to validate the soundness of the methodology experimentally. A comparison between linear programming and other risk definition techniques was provided.

Keywords: Predictive metric, linear programming, risk evaluation.

1. Introduction

Reliability is one of the most important aspects of software systems of any kind. The size and complexity of software has grown dramatically during the last decades and especially during the last few years. When the requirements for and dependencies of computers increase, chances of crises from failures also increase. The impact of these failures ranges from inconvenience to economic damages to loss of lives—therefore it is clear that software reliability is becoming a major concern for software engineers

and computer scientists (Bush, 1994; Fenton, 1997; IEEE, 1992; Konrand, 1992; Roche, 1994).

Software development is a complex process in which software faults are inserted into the code during the development process or during maintenance. The literature on this subject shows that the pattern of faults insertion phenomena is related to measurable attributes of the software objects (Coleman, 1994; Kokol, 1999; Montanari, 1994; Paulish, 1994; Pighin, 1997).

During the last 20 years, hundreds of metrics have been proposed for software assessment. The measurements carried out during initial phases of a development cycle can be used for purposes that are not specifically related to assessment. This kind of measurements are known as predictive metrics, a term that indicates a predictive element that can be used as an example of the dangerous programs prediction identified by the number of faults remaining in software (Fioravanti, 2001; Grady, 1993; Munson, 1992; Pighin, 1998, 1999; Thomson, 1994).

Due to the large number of predicting software metrics and various attributes produced by them, it is very difficult for a practicing software engineer, while assessing or predicting the reliability of a software system/program, to answer some of the following questions:

- Will proposed metric/attributes predict the reliability accurately?
- What will the maximal accuracy of the prediction be?
- Does proposed metric best fit my environment?
- Is the metric easy to use and to understand?

In this paper we will present a new methodology that applies linear programming techniques to elementary counting of structural parameters, to define a model which can predict the risk level of a program, namely how prone it is to containing faults. Usually, risk reflects not only the number of faults encountered but also the consequences of those faults (crash of the system, failures of different severity, etc.). In our environment, we associate the risk only to the number of faults remaining in software in order to simplify the predicting model.

Finally we present the results of this methodology in a particular industrial software environment.

In this paper the terms program and file are used as synonymous.

2. The Predictive Methodology

2.1. The Rationale

The basic idea is to compare the number of faults detected in a code to certain critical values found in the same code for some of the measured parameters. We

aimed at linking the degree of reliability of a program to its particular structural features. In general, the target of the methodology is to identify, in a specified environment, parameters that explain the presence of faults in many programs, considering the values measured for them. The identification of these parameters, and of files that have a high risk to contain faults, can be used to pre-process these files during the releasing and testing session.

2.2. The Definition of the Set of Attributes

The starting point of our analysis was the definition and measurement of a set of attributes connected to the structure of software products after the code phase. Such parameters may be, for example, the total number of lines of code and of lines of comment, the occurrence of various types of instruction, the operators and the types of data used, etc.

The choice of parameters depends on the application, and on a whole range of characteristics relating to the programming environment, such as the language used, the available tools, the rules governing software development and internal testing, the types of problems faced, etc. In any case, only aspects related to the structural and lexicon-syntactic features of the adopted programming language were considered, not those connected to the semantic content.

Then, for each program we need to consider the fault signals since the measurement started. These values are necessary to tune the engine for the following measurements correctly.

First, we considered whether the chosen set of parameters would be sufficiently large to identify the structure of a program. In our methodology, we started with a very large set of structural parameters. These parameters were usually affected by the persistence of multicollinearity (for example, lines of code, lines of comment and total lines). It was necessary to reduce the total number to a smaller set of independent parameters. This was achieved by statistical procedures eliminating parameters that were heavily dependent on other parameters, or that were completely irrelevant in the context (for example the *goto* statement was used only twice in the studied environment). With this method it is possible to define a subset of structural parameters which the statistical analysis identified as being reasonably free from multicollinearity, plus the dependent variable, the number of faults (Munson, 1992; Pighin, 1997).

In order to apply the proposed linear programming methodology correctly, it is necessary to further reduce the number of parameters to an optimal subset that still represents the real structure of the files. This can be reached by applying a feature selection procedure (see forward for details), which reduces and optimizes this first attribute set, and on which the mathematical method can efficiently build a model representing the structure of code related to the faults found on files.

2.3. The Theoretic Model

At this point we consider the problem of defining the predicting system, which, starting from the measurements of selected parameters, is able to discriminate between programs containing faults and programs that are substantially correct.

We are interested in the implementation of a procedure that can divide the input file set in two distinct groups. The first one represents the class of files with a number of faults less or equal to a defined threshold (FT, fault threshold) ($C1$ subset). The second represents the class of files with more faults ($C2$ subset). We accept that the procedure does not define the classification of a small subset of files that has a high degree of structural uncertainty. Files whose representation is close to the class borders are not taken in consideration, because their classification is uncertain.

The proposed measure of the software reliability represents each code file with an n-dimensional point, based on the n attributes (the structural parameters) measured on the file.

The problem of discriminating between files of class $C1$ and of class $C2$ can be brought back to the problem of separating two disjoint sets in $\Re^n$ (i.e., the set of tuples of n values of parameters related to files in class $C1$ and the set of tuples of n values of parameters related to files in class $C2$).

If A and B are two disjoint sets on the n-dimensional real space, containing respectively **m** and **k** vectors, the two sets are separable provided there exists a plane defined by the equation $\underline{w} \cdot \underline{x} = \theta$, called separation plane, which determines two hemi-spaces

$$H_- = \{\underline{x} \in \Re^n | \underline{w} \cdot \underline{x} \leq \theta\}$$
$$H_+ = \{\underline{x} \in \Re^n | \underline{w} \cdot \underline{x} > \theta\}$$

and

$$A \in H_- \quad B \in H_+ \text{ (or viceversa)}$$

Denote $A \in \Re^{m \times n}$ and $B \in \Re^{k \times n}$ to be the matrices containing vectors in sets A and B, the separability of the two sets implies the inequality

$$\min_{1 \leq i \leq \mathbf{m}} A_i \underline{v} > \max_{1 \leq j \leq \mathbf{k}} B_j \underline{v} \text{ for some } \underline{v} \in \Re^n \tag{1}$$

where A_i indicates the i^{th} row of matrix A.

The condition (1) is equivalent to the following:

$$A\ \underline{w} \geq \underline{e} \cdot \theta + \underline{e}, \quad B\ \underline{w} \leq \underline{e'} \cdot \theta - \underline{e'} \tag{2}$$

for some $\underline{w} \in \Re^n, \theta \in \Re$ with $\underline{e} \in \Re^{\mathbf{m}}$ and $\underline{e'} \in \Re^{\mathbf{k}}$ of unitary elements.

The rule (2) $\Rightarrow$ (1) is clear, while the opposite can be demonstrate by fixing

$$\varepsilon = \min_{1 \leq i \leq \mathbf{m}} A_i \underline{v} - \max_{1 \leq j \leq \mathbf{k}} B_j \underline{v} > 0, \quad \underline{w} = \frac{2 \cdot \underline{v}}{\varepsilon} \quad \text{and} \quad \theta = \min_{1 \leq i \leq \mathbf{m}} \frac{A_i \underline{v}}{\varepsilon} + \max_{1 \leq j \leq \mathbf{k}} \frac{B_j \underline{v}}{\varepsilon}$$

If the convex envelopes of the sets A and B are disjoint, the sets are separable and it is possible to determine $\underline{w} \in \Re^n$ and $\theta \in \Re$ which satisfy the inequality (2).

If they are not disjointed, it is possible to satisfy only a weaker version of condition (2):

$$A\ \underline{w} + \underline{y} \geq \underline{e} \cdot \theta + \underline{e}, \quad B\ \underline{w} - \underline{z} \leq \underline{e'} \cdot \theta - \underline{e'}$$
$$\text{for some} \quad \underline{w} \in \Re^n,\ \theta \in \Re,\ \underline{y} \in \Re^m_+,\ \underline{z} \in \Re^k_+ \tag{3}$$

Therefore, it is possible to determine a plane that approximately satisfies the condition (2), minimizing the mean of their violations, solving the following linear programming problem (4):

$$\min_{\underline{w},\theta,\underline{y},\underline{z}} \frac{e^T \cdot \underline{y}}{\mathbf{m}} + \frac{e'^T \cdot \underline{z}}{\mathbf{k}}$$
$$A \cdot \underline{w} + \underline{y} \geq \underline{e} \cdot \theta + \underline{e}$$
$$B \cdot \underline{w} - \underline{z} \leq \underline{e'} \cdot \theta - \underline{e'}$$
$$\underline{y}, \underline{z} \geq \underline{0} \tag{4}$$

The solution to this problem gives:

- A separation plane for sets A and B, if they are separable: the optimum value is zero which is obtained by relating the violation $\underline{y} = \underline{0}$ and $\underline{z} = \underline{0}$; in this case it is not possible to obtain $\underline{w} = \underline{0}$, because both conditions $-1 \geq \theta$ and $\theta \geq 1$ will be true.

- An optimal plane which approximately divide the A and B sets, if they are not separable: it is possible to demonstrate that the solution of the problem above is always a plane $\underline{w} \cdot \underline{x} = \theta$ with $\underline{w} \neq 0$ (Ben-Bassat, 1992).

The number of variables of the linear programming problem is $n + m + k + 1$, while the number of constraints is $2 * (m + k)$.

2.4. The MSM-T Method

The solution to the linear programming problem was obtained by using multi surface method-tree (Ben-Bassat, 1992). MSM-T is a greedy algorithm, which, having two sets of disjoint points of $\Re^n$ as input, iteratively divides the n-dimensional space in polyhedral regions, each containing the points of one set only, or it is empty (or contains elements from both sets but the total number of elements is less than a fixed value). The procedure implies that there is a first (learning) phase in which the system is tuned to known data and the following (matching) phase in which the system can produce real measurements (Mangasarian, 1992). The theoretical accuracy of the

whole systems is 100%: in the worst case a polyhedral region is defined for each point of the learning set (Mangasarian, 1994).

This methodology was early applied to other fields. For instance in cancer diagnosis, a similar model was built and learnt and is now running for the risk of breast cancer prediction (Mangasarian, 1994).

The core of the methodology is the generation of a sequence of planes solving linear problem (4) with particular values: initially the problem is applied to A and B sets, then to the subsets of a polyhedral region determined by the intersection of complementary subsets defined by planes previously generated. More precisely:

- a plane is generated by dividing A and B sets, defining two couples of subsets: the couple of A and B subset in the upper hemi-space and the couple in the lower hemi-space;
- the procedure is applied iteratively creating a plane which divides each non-void couple;
- the procedure stops when all the couples are void or after a fixed number of iterations.

The following example is the result of application of MSM-T Method dividing two sets in $\Re^2$ that are not linearly separable: in each polyhedral region there are elements of one set only.

The decision regarding the maximum number of iterations of the MSM-T method during the learning phase is equivalent to defining the generalization capacity of the system. A too high number of planes causes an over-training, specializing the system on the learning data instead of only on the structure that they represent. A too low

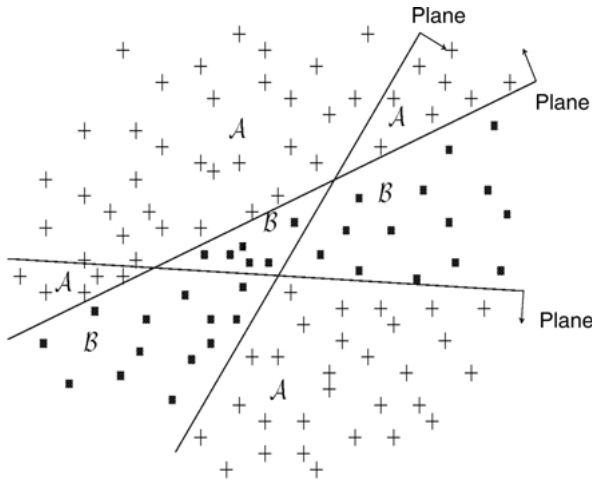

Figure 1. MSM-T method dividing two sets in $\Re^2$.

number of iterations causes an under-trained system, which cannot evaluate the detail of the structure of the training set. In the matching phase there are no iterations: the planes defined in the learning phase are used to discriminate the two classes.

2.5. The Feature Selection Procedure

The classification methodology needs the application of a feature selection procedure (Langley, 1994), to the parameters definition. This procedure is used with the following targets:

- the reduction of the dimension of data used by the classification algorithm;
- the reduction of execution time, both in learning and in recognition phase;
- the increment of the precision of the classification algorithm, due to the elimination of irrelevant data.

By given n attributes, the basic idea of the selection techniques consists of searching for an optimal (with respect to a defined evaluation function) subset among 2^n subsets. An exhaustive search is usually too expensive, so empirical selection criteria are used. The feature selection activity is divided in three phases:

- definition of next subset to be examined;
- evaluation of the subset according to a predefined evaluation function;
- checking for exit-criteria, to avoid exhaustive search.

We can imagine a generic selection criteria as a function F: $\Re^n \rightarrow \Re^p$ which transforms the input n-space to a p-space.

3. The Experiment

3.1. The Target of the Experiment

We defined an experiment to assess if the proposed methodology was able to discriminate between programs containing faults and programs that were substantially correct.

The validation is effectively obtained by submitting a new set, not analyzed in the training phase but of known results, to the classification procedure, and matching the real results with the ones predicted by the model. The effective validity of the prediction system depends on the adequacy with which the partition of $\Re^n$ obtained

with the model represents the $C1$ and $C2$ classes, as defined at the beginning of Section 2.3.

3.2. The Methodology of the Experiment

The experiment is divided in two phases:

- In the first (learning) phase, the system is fed with a random subset of the real set of files (training sample), to build the abstract model of the two classes of files.
- In the second (recognition) phase, we apply the remaining subset (testing sample) to the system, measuring the correctness of classification.

To augment the precision of the classified set, it is admitted a tolerance ρ that permits to avoid evaluation of points too close to the separation planes. If we call x_i the tuple of p values associated to a file f_j of the testing set, the classification criteria is defined by the following rules:

- if $\underline{w} \cdot \underline{x_j} \leq \theta - \rho$ the file f_j is classified in the set associated to hemi-space H_-;
- if $\underline{w} \cdot \underline{x_j} > \theta + \rho$ the file f_j is classified in the set associated to hemi-space H_+;
- elsewhere f_j is not classified.

The classification procedure has to be able to identify those files whose characteristics would suggest a high degree of risk. In this regard, the predictive decision-making procedure might make mistakes in two distinct ways: by making a so-called Type I error, whereby a program is presumed to contain a number of faults exceeding the threshold, when opposite is true; or by making a Type II error, in which a program is believed to contain the number of faults not exceeding the threshold, when the threshold is exceeded. We evaluate the percentage of total of errors on the test set: the accuracy of the prediction is the difference of this value to 100% (often called classification correctness, Fioravanti, 2001).

3.3. The Working Environment

The working environment chosen for the experiment was a software house that employs about twenty programmers, who produce management software, written in C, on Informix Relational Database, in the Unix Operating System.

The analysis was done on a particular management application, constituted of 372 files amounting to a total of more than 300,000 lines of code. Moreover, for each program, we considered the fault signals up to the moment when measurement

started. By faults we mean all the malfunctions encountered during the internal test phase and after the release of the software.

These programs had been produced over a period of seven years and for each of them, the following data relating to fault calls and corrections were available:

- the date the file was created;
- the date of the last modification to the code;
- the number of faults found up to the time of the analysis.

3.4. The Operative Steps of the Experiment

We started our analysis with a large set of structural parameters (about 220). The first measurement was done automatically on the entire set of programs by a parser, which counted the occurrences of various structural parameters. Some values were slightly elaborated (for instance, a parameter is the ''mean number of parameters in function calls in a file'').

The second step of the method was the definition of the training and testing sets. We selected 200 files (about 54% of total) as a training set at random, while the remaining 172 files were used as the testing set.

We applied two methods of reduction, both operating on the training set.

The first reduction was done by eliminating parameters which were not relevant (with less than three instances in all programs, like, for instance, *goto* parameter) or which were strongly correlated to other parameters (for example number of words and number of lines in programs). The threshold of correlation was 0.90 and the first variable was left in the model. This first step brought to a subset of 149 parameters (for instance, number of lines of code, number of operators, number of function calls, number of preprocessing instructions, etc.). This first reduction method is very general and can be used on every set of data.

The second was a reduction by a feature selection. The main reason for this reduction is the utility to operate with a smaller set of variables while solving the linear programming problem. The exhaustive evaluation of the optimal subset for analysis will require $2^{149} \cong 7.14 \cdot 10^{44}$ steps, which is not realistic. We adopted a feature selection heuristic method to find out a more manageable subset of parameters.

We analyzed several procedures. The best results were obtained with two methodologies: using a sequential forward selection based on the Kendall correlation between the values of risk-parameters in the files of the training set and using the number of faults found in the same files.

In the first method we started our analysis with an empty set of parameters and a correlation index of -1. At each step we inserted the parameter that guarantees the maximum increment of the correlation between two series of values in the list of active attributes. The first series of values were obtained as a particular ''risk

function'' (see Section 5.1.4) of the active parameters in each file and the second series of values were formed with the number of signaled faults in the same file. The exit-criterion was obtained when entering new parameters we had no increment of the correlation. The final accuracy result with this feature selection is slightly worse (about 1–1.5%) than that with the following selection.

In the second method (that was chosen), we calculated for each parameter the value of Kendall correlation between two series of values, the first obtained with the value of each parameter in each file and the second with number of signaled faults in the same file. We selected all the parameters whose value of correlation was over the 80% of the maximum value of correlation. The procedure stops when all the parameters are analyzed.

In the other methods, we selected the parameters with different procedures: different intervals of values obtained with the above mentioned ''risk function'', different intervals of correlations of values of each parameters and errors, preliminary semantic division of parameters and then selection in each group of higher values of ''risk function'' or higher correlation with errors.

The selected parameters are influenced by the random definition of training set, so we decided to repeat the training set selection 150 times with 150 different random selections from the entire set. In the definition of final subset of the parameters we considered all the parameters which appeared in the each selection.

In the second method above described, with this procedure we limited the analysis to 29 structural parameters which determine the structure of the classes in this particular environment. They can be grouped in the following principal classes (in parenthesis an example of the class):

1. Comments (number of lines of comments).

2. Lines of code (number of lines of code).

3. Control structures (number of control instructions).

4. Preprocessing instructions (number of inclusions).

5. Function calls (mean number of parameters in function calls in a file).

6. Variable/constant definitions (number of array declaration).

At this step we had a subset of parameters, but not a predictor itself. We analyzed this subset with elementary statistics, with low level accuracy results: the core of the prediction is the linear programming model.

We chose five faults as value of FT (Fault threshold). This choice is due to the fact that the mean fault number of the whole starting set is 5.35 and that this value was used in other experiments with other methodologies which are compared with this experiment (see Section 5). With other values near to this, the results obtained with linear programming method are very similar: this choice is not essential to final accuracy.

We started the learning phase. The standard engine used to run the MSM-T algorithm was CPLEX (version 3 running in UNIX environment). CPLEX is one of the most used MILP (mixed integer linear programming) Solver (Cplex_1, Cplex_2).

The number of iterations of the algorithm was calculated by trying out different situations. The problem of over-training appeared with three or more iterations: the results were very good on the training set (an accuracy over 98% with 10 iterations), but worse on test set. This tuning of the number of iterations was carried out during a previous section of experimentation, with a different random selection of training and testing sets.

The engine worked with 230 variables and 400 constraints.

At the end of the learning phase, we validated the model with the test set. We repeated this validation procedure with three different configurations to reach a classification respectively of 100%, about 90% ($\rho = 0.34$) and about 80% ($\rho = 0.85$) of the test sample: in the last two steps we avoided defining the classification of elements near the planes of division of the polyhedral regions.

We repeated the analysis for each of the 150 learning and test sets, and for each of the three configurations we calculated the medium and standard deviation values. The results obtained are illustrated in Table 1.

We repeated the analysis and calculated the total accuracy results with MSM-T methodology also with the other different selection criteria that brought to different types and numbers of parameters selected. The mean values of the total errors obtained with the various selection methods differ by 4–6 percent from the final results of total errors of Table 1 (the mean percentage of total error in the worst case, with 100% of files classified was about 22%). At last we calculated the final accuracy results with MSM-T methodology without feature selection. The percentages of total errors were almost double that of the final results of total errors of Table 1. We can conclude that the results are affected by the method of selection used, as described in Section 2.5, but the general validity of the partitioning depends substantially on the linear programming method used which can works with lower accuracy also without any feature selection.

Table 1. Results of the experiment.

Files classified (%)		Type I err (%)		Type II err (%)		Total err (%)		Accuracy (%)
Mean	Std	Mean	Std	Mean	Std	Mean	Std	
100.00	0.00	32.59	5.71	8.42	2.37	**16.03**	2.52	83.97
90.23	2.41	28.52	6.16	6.45	2.21	**13.08**	2.65	86.92
80.18	4.03	25.32	6.53	4.87	2.26	**10.92**	2.85	89.08

4. The Assessment of Linear Programming Risk Definition Analysis

The purpose of this analysis was to build a model with specific predicting capabilities. We produced a predictive model able to identify files with a high risk of faults and separate them from those that are basically sound.

The results of the experimental analysis verified that about 89% of files were correctly classified in the best case (excluding about 20% of files of uncertain structure).

Another aspect is that the percentage of Type II errors is very low. Since we want to be able to use the results in order to fine-tune testing procedures on programs, Type I errors are usually less dangerous since, at most, one further session of more detailed testing or a better restructuring will be required (and may indeed be superfluous) on a defect-free program. An high Type II error means that dangerous programs will not be signaled and so no particular action will be taken for them. This implies that an early identification or correction of potential defects will be more difficult. It is worth pointing out that in particular contexts, especially when testing resources are scarce and all programs identified as dangerous can not be tested, is useful to reach lower values of Type I errors than of Type II errors (Mockus, 2000).

Ultimately we considered that the method is relatively stable regarding the results obtained with the 150 different selections. We calculated the confidence interval of the mean values of last row of Table 1. With a probability of 99% the mean values of total errors is 10.92 ± 0.60, of Type I errors is 25.32 ± 1.37 and of Type II errors is 4.87 ± 0.48.

Nevertheless, it is important to note that this methodology is sensitive to good feature selection mechanism: using different selections the results can be different (4–6 percent of total accuracy). Without feature selection the results can significantly change, as described in Section 3.4.

The proposed methodology is also sensitive to over-training, which can describe perfectly the training set, but cannot describe the whole system correctly.

It is useful to conduct a good tuning session in which all these aspects (feature selection method, number of iterations of the algorithm, other minor parameters of the system as fault threshold, or level of uncertainty to be eliminated) are optimized for the particular environment.

The first benefit is that this analysis can be used as a filter before the testing phase and the release of the software, thus enabling specific analytical testing techniques or restructuring to be performed upon those programs regarded as being more dangerous.

The second interesting aspect is that the feature selection procedure extracts a subset of parameters to focus on the analysis and the testing effort.

The third positive aspect is that, according to its very nature, the model is not abstract, but adapts to different environmental characteristics. Then, the model can be continuously revised while the environment changes. By incrementing the database of programs and of signaled faults it is possible to recalculate the model, following the modification of the environment: this predicting system is continuously able to be trained, adding other points whose class ($C1$ or $C2$) is known to the learning set or modifying the number or type of parameters. The cited system predicting the risk of breast cancer is continuously fed with new cases that augment its predicting precision (Langley, 1994).

A limitation to this kind of analysis is that, being closely correlated to the

environment and especially to the language used, it is necessary to start with a large database of statistical data (programs and related signaled faults). It is difficult to calculate the model in new projects with a short history, or in staff who significantly change developing methodologies. The proposed approach can be very useful in mature projects, but it is less useful in new projects or with new staff. In these situations it needs to be continuously revised alongside the growing and evolving of system data.

Another negative aspects is that this metric is relatively simple to understand in its general aspect, but the algorithms are quite complex: a software engineer could not clearly understand the motivations of calculated results and the system could appear as a black box which indicates dangerous files and risk parameters (but not their ranking).

5. Comparison With Other Models

In literature it is possible to analyze other risk-definition methodologies. These methodologies were based on a combination of existing metrics or on new kinds of measurements.

In previous studies we analyzed the methodologies which applied two types of criteria:

- *Statistical methods*, using discriminant analysis, factorization, multivariate regression, mean values, etc.

- *Decision trees methods*, using inductive inference, which is the process of moving from concrete examples to general models, where the goal is to learn how to classify objects by analyzing a set of instances (already solved cases) whose classes are known.

For these methodologies we evaluated the general results found in literature and we proposed an experiment on the same set of programs, of parameters, and of signaled faults used for the linear programming risk-threshold definition, applying the same general criteria: we divided the starting set with a random procedure in a learning set used to build the model, and in a (remaining) testing set used to evaluate the calculated model.

In literature there are also other different predicting methodologies, we shall compare in future to our results (Aljahdali, 2001; El Aman, 2001; Fenton, 2000; Graves, 2000; Lanubile, 1995; Telin, 1999; Zhiwei, 2000).

It is now possible to have for statistical and decision tree methods an idea of the basic principles, some bibliographic references, and some results in terms of accuracy (total and referred only to Type I error and Type II error) on a homogeneous test set. In the final paragraph some other features that can be reached with different techniques, and that have an operative utility, are classified:

1. Identification of risk-parameters.

2. Ranking of risk-parameters.

3. Possibility of reduction of the set of analysis eliminating uncertain files.

A deeper evaluation should include other parameters, like effort required, sensitivity to context, applicability/robustness/stability of results, which could be planned in future works.

For the discriminant analysis method, we used a general purpose statistical package (MINITAB). For all other methods the software for analysis was built for the experiments.

5.1. Statistical Methods

The fundamental hypothesis of statistical techniques (details of these methods can be found in Basili, 1996; Briand, 1998, 1999; Bucci, 1998; Khoshgoftaar, 1994; Fioravanti, 1998; Munson, 1992; Nesi, 1998) is similar to that proposed in this paper. For a given set of multiple and diverse observations there exists a classification into two or more mutually exclusive groups. The observations correspond, more specifically, to programs; usually they are combined in two groups (or populations) only. The first contains files with a fairly small number of faults, the second contains a fairly high number of faults. It was thus to be expected that the characteristics of the programs would be as similar as possible within a group and, at the same time, markedly different between programs belonging to different groups.

Results obtained in literature (Munson, 1992; Fioravanti, 2001; Briand, 1998) reach an accuracy of about 80–90% depending the methodology and the percentage of classified files. In one experiment (Fioravanti, 2001), combining 42 different metrics, the result of accuracy reached the 97%. Diminishing the number of metrics to a reduced and more manageable set of 12 metrics, the values lowered to 86%.

Using our set of programs and related data, we tested some of different methodologies that are summarized in following paragraphs. The first was a general method of discriminant analysis, while the remaining three methods used a threshold of particular (and different) measurements as reference level for risk value (we called them threshold methods).

5.1.1. Discriminant Analysis

We used two methodologies to build the model (details of experiment in Pighin, 1997).

1. Discriminant analysis using the whole set of 220 parameters, as they stand, without any factorization.

2. Preliminary factorization of the 220 parameters, by using the main components technique; this way we would be able to reduce significantly the number of distinct discriminant variables, and above all with a correlation equal to zero (perpendicular). The aim was to group the variables on the basis of correlation values so as to explain the variability of the observed data by means of a more concise description of the structure of dependence (in the experiment the number of factors used was 7 and 10).

Table 2. Discriminant analysis.

Factorization level	Classified (%)	Type I err (%)	Type II err (%)	Total err (%)	Accuracy (%)
No factorization	91.3	7.8	26.1	**12.4**	**87.6**
No factorization	84.7	7.2	21.7	**10.6**	**89.4**
7	84.0	7.7	25.0	**11.5**	**88.5**
7	72.3	7.5	18.6	**9.7**	**90.3**
10	87.6	7.0	16.1	**9.1**	**90.9**
10	81.6	6.3	14.3	**8.2**	**91.8**

5.1.2. McCabe Metric

The idea is to use the McCabe complexity metric as risk predictor (details of this method can be found in Pighin, 1998). The first step was the calculation of the mean value of the McCabe cyclomatic number in each group of the training set. Then the reference value was calculated as the mean value of the results: this is the prediction threshold using the McCabe cyclomatic value.

Table 3. McCabe metric.

Classified (%)	Type I err (%)	Type II err (%)	Total err (%)	Accuracy (%)
100.0	19.7	34.3	**24.2**	**75.8**
78.8	17.1	29.2	**21.1**	**78.9**

5.1.3. Alpha Metric

In this methodology alpha metric is used as a risk measurement. Alpha metric calculates the information density of a software file by transforming a file into the Brownian random walk. Brownian walk is then further analyzed using average displacement function and regression techniques (further details can be found in Kokol, 1999).

Table 4. Alpha metric.

Classified (%)	Type I err (%)	Type II err (%)	Total err (%)	Accuracy (%)
100.0	18.1	35.8	**30.9**	**69.1**
81.3	16.5	30.8	**24.7**	**75.3**

5.1.4. RPSM Metric

In this methodology RPSM (risk predictive structural metric) metric is used as a risk predictor. For each of the 220 parameters, we analyzed the training set of files. We divided the interval between the maximum and minimum value of the parameter in eleven equal sub-intervals. We evaluated the sub-interval with more occurrences of the parameter (MI). Then for each file in the training set we calculated a distance from the interval in which the parameter is present for the selected file and MI, weighting this distance with the number of faults in the file. Summing up in the training set and normalizing the values for all the parameters, we calculated a parameter risk value. Summing up and normalizing for the files we calculated a file risk value. Then we calculated the mean of file risk values in each group of the training set. The reference value was the mean of the results: this is the prediction threshold using RPSM metric (details of this method can be found in Pighin, 1999).

Table 5. RPSM metric.

Classified (%)	Type I err (%)	Type II err (%)	Total err (%)	Accuracy (%)
100.0	31.5	14.9	**18.6**	**81.4**
90.0	27.6	11.7	**14.9**	**85.1**
80.5	24.5	8.4	**11.7**	**88.3**

5.2. Decision Trees

Inductive inference is the process of moving from concrete examples to general models, where the goal is to learn how to classify objects by analyzing a set of instances (already solved cases) whose classes are known. Instances are typically represented as attribute-value vectors. Learning input consists of a set of such vectors, each belonging to a known class, and the output consists of a mapping from attribute values to classes. This mapping should accurately classify both the given instances and other unseen instances. A decision tree (Quinlan, 1993) is a formalism that expresses such mappings and consists of tests or attribute nodes linked to two or more sub-trees and leaves or decision nodes labeled with a class which means the decision.

Decision trees methods don't allow the reduction of the data set, so all the experiments are performed on 100% of cases. We evaluated the potential of this approach using two different methodologies.

5.2.1. Classical Decision Trees

A test node computes some outcome based on the attribute values of an instance, where each possible outcome is associated with one of the sub-trees. An instance is classified by starting at the root node of the tree. If this node is a test, the outcome for the instance is determined and the process continues using the appropriate sub-tree. When a leaf is eventually encountered, its label gives the predicted class of the instance (details of this methodology can be found in Podgorelec, 1999).

Table 6. Classical decision trees.

Classified (%)	Type I err (%)	Type II err (%)	Total err (%)	Accuracy (%)
100.0	13.7	55.6	**25.0**	**75.0**

5.2.2. Evolutionary Decision Trees

Evolutionary algorithms are adaptive heuristic search methods that may be used to solve all kinds of complex search and optimization problems. They are based on the evolutionary ideas of natural selection and genetic processes of biological organisms. As the natural populations evolve according to the principles of natural selection and "survival of the fittest", so by simulating this process, evolutionary algorithms are able to evolve solutions to real-world problems, if they have been suitably encoded. They are often capable of finding optimal solutions even in the most complex of search spaces or at least they offer significant benefits over other search and optimization techniques.

Opposite to the classical induction of decision trees, in the evolutionary approach instead of using a heuristic function, genetic operators are used. Selection forces the evolution towards better solutions (tournament selection is used in our case). Crossover produces a new tree from two parents that are cut on a randomly chosen path through the trees. Mutation consists of four parts: exchange of an attribute (mutation rate, mr = 0.03), exchange of a split threshold (mr = 0.17), exchange of a test node with decision (mr = 0.01), and exchange of a decision node with a test (mr = 0.09). A fitness function determines the quality of a single evolved decision tree:

$$LFF = \sum_{i=1}^{K} w_i \cdot (1 - acc_i) + \sum_{i=1}^{N} c(t_i) + w_u \cdot n_u$$

where K is the number of decision classes, N is the number of attribute nodes in a tree, acc_i is the accuracy of classification of objects of a specific decision class d_i, w_i is the importance weight for classifying the objects of a decision class d_i, n_u is number of unused decision (leaf) nodes, i.e., where no object from the training set fall into, and w_u is the weight of the presence of unused decision nodes in a tree. The cost $c(t_i)$ represents the computational expense of obtaining the attribute's value. In our example it has not null values only for the few parameters which were slightly elaborated from simple counting (details can be found in Podgorelec, 2002; Sprogar, 2000).

Table 7. Evolutionary decision trees.

Classified (%)	Type I err (%)	Type II err (%)	Total err (%)	Accuracy (%)
100.0	15.1	22.2	**17.0**	**83.0**

6. Conclusions

From Tables 1 to 7 we can extract the following conclusions:

1. In general statistical and mathematical methods work better with respect to the accuracy.

2. The operative constraint in accuracy seems to be (at least with our data) near 80–85% if 100% of cases have been classified or 85–90% if only about 80% of files have been classified. These constraints are probably due to noise present in any population of software files. In our case the noise can be the consequence of the fact that not all the faults have been detected during the analysis of files, different styles of programming and testing, different "age" of files, etc. Probably this noise can not be eliminated using current techniques.

3. The remaining methods are close to this accuracy, especially considering the case when 100% of files have been classified.

4. Some methods can embed the reduction of the set eliminating the uncertain classifications. The considerable reduction does not exceed 20–25%. In this case the accuracy rises of 8–10%. If these methods are used as a filter before testing phase, we have no information on eliminated files which should be prudentially tested as they would be dangerous.

5. Some methods work as black box (especially statistical methods with factorizations), others as white box defining clearly the risk parameters, and possibly the ranking of risk. This is important to support the test or maintenance engineer.

All these aspects of different approaches are summarized in Table 8.

Table 8. Summary of methods by operative features.

Method	Best total accuracy	Reduction of analyzed files	Definition of risk parameters	Ranking of risk parameters
Linear programming	✓	✓	✓	
Statistical (general)	✓	✓		
Statistical (threshold)		✓	✓	✓
Decision trees			✓	✓

About linear programming techniques, the experimental validation and the comparison with other methodologies demonstrate that these methodologies can produce sound models of predicting metrics, comparable to other methods, with or without feature selection. The best results are obtained using good feature selection methods which must be trimmed during system tuning.

Future trends of work with linear programming criteria will be:

- Compare the MSM-T method with other predicting methods applying exactly the same feature selection procedure.
- Test the procedure on different databases of programs, to use more complex and sophisticated minimization algorithms based on different elements.
- Evaluate different algorithms which consider the different severity of faults.

The difficulties of these approaches is that the increment of accuracy (and consequently of computational complexity) is balanced by the heterogeneity of elements of the model (files of code) which could limit the results of the added effort (Kokol, 2001).

Acknowledgments

This research is financed by SALADIN Project (Software Architectures and Languages to Coordinate Distributed Mobile Components), Minister of University and Scientific Research, Italy; New Software Complexity Metrics for Real World Applications, Bilateral Project Italy–Slovenia, 1999–2000; MERCOS Project (Measuring Reusability by Complexity Similarity), Bilateral Project Italy–Slovenia, 2001–2005.

References

Aljahdali, S. H., Sheta, A., and Rine, D. 2001. Prediction of software reliability: a comparison between regression and neural network non-parametric models. *Proceedings ACS/IEEE International Conference on Computer Systems and Applications*.

Basili, V. R., Briand, L. C., and Melo, W. L. 1996. A validation of object oriented design metrics as quality indicators. *IEEE Transaction on Software Engineering* 751–761.

Ben-Bassat, M. 1992. Decision tree construction via linear programming. *Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Science Society Conference* 97–101.

Briand, L. C., Daly, J. W., and Wust, J. K. 1999. A unified framework for coupling measures in object oriented systems. *IEEE Transaction on Software Engineering* 25(1): 91–120.

Briand, L. C., Daly, J. W., Wust, J. K., and Porter, D. V. 1998. Exploring the relationship between design measures and software quality in object oriented systems. *Journal of Systems and Software*.

Bucci, G., Fioravanti, F., Nesi, P., and Perlini, S. 1998. Metrics and tools for system assessment. *Proceedings of IEEE Conference on Complex Computer Systems*. USA: IEEE Publ., 36–46.

Bush, M., and Fenton, M. 1990. Software measurement: A conceptual framework. *Journal of System & Software* 12: 223–231.

Bush, M. 1994. The changing perception of software metrics. *Software Quality Management*, edited by Ross M. et al., CMP, 1: 417–429.

CPLEX_1, www.cplex.com, http://www.ilog.com/products/cplex

CPLEX_2, www.ilog.com

Coleman, M. 1994. Using metrics to evaluate software system maintainability. *IEEE Computer* 27(8): 44–49.

El Emam, K., Benlarbi, S., Goel, N., Rai, S. N. 2001. Comparing case-based reasoning classifiers for predicting high risk software components. *Journal of Systems and Software* 55.

Fenton, N. E. 1197. *Software Metrics – A Rigorous Approach*. Chapman & Hall, 2nd ed.

Fenton, N. E., and Ohlsson, N. 2000. Quantitative analysis of faults and failures in a complex software system. *IEEE Transactions on Software Engineering* 26.

Fioravanti, F., Nesi, P., and Perlini, S. 1998. Assessment of system evolution through characterization. *Proceedings of the IEEE Conference on Software Engineering*. USA: IEEE Publ., 456–459.

Fioravanti, F., and Nesi, P. 2001. Prediction model for software fault correction effort. *Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*. CMSR2001, USA: IEEE Publ., 114–120.

Grady, R. B. 1993. Practical results from measuring software quality. *Communications of the ACM* 36(11): 62–68.

Graves, T. L., Karr, A. F., Marron, J. S., and Siy, H. 2000. Predicting fault incidence using software change history. *IEEE Transactions on Software Engineering* 26(7).

IEEE Transactions on Sofware Engineering, 1992, Special Issue on Software Measurement Principles, Techniques, and Environments, November.

Khoshgoftaar, T. M., and Oman, P. 1994. Software metrics: charting the course. *IEEE Computer* 27(9): 13–15.

Kokol, P., Podgorelec, V., Zorman, M., and Pighin, M. 1999. Alpha—a generic software complexity metric. *Proceedings of European Software Control and Metrics Conference*, ESCOM-SCOPE 99, Maastrich NL: Shaker Publishing BV, 397–405.

Kokol, P., Podgorelec, V., and Pighin, M. 2001. The operative constraints of software reliability prediction methods. *Proceedings of Systemic, Cybernetics and Informatics Conference*, SCI-2001, Orlando USA: IIIS Publication, Skokie-Illinois (USA), 229–234.

Konrad, M. S. 1992. Application of measurement theory to software metrics. *ACM SIGPLAN Notices* 27(12): 13–19.

Langley, P. 1994. Selection of relevant features in machine learning. *Proceedings of the AAAI Fall Symposium on Relevance*, 1–5.

Lanubile, F., Lonigro, A., and Visaggio, G. 1995. Comparing models for identifying fault-prone software components. *Proceedings of the 7th International Conference on Software Engineering and Knowledge Engineering*. Washington, D.C., 312–319.

Mockus, A., and Weiss, D. M. 2000. Predicting risk of software changes. *Bell Labs technical Journal* 5(2): 169–180.

Mangasarian, O. L., and Bennet, K. P. 1992. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, 1–24.

Mangasarian, O. L., Street, W. N., and Wolberg, W. H. 1994. Breast cancer diagnosis and prognosis via linear programming, *Tech. Rep. 94–10*, Madison: University of Winsconsis.

Montanari, A., and Pighin, M., 1994. Identification of experimental parameters for automatic software evaluation and testing. *Proceedings of SQM'94*, Edinburgh, 335–348.

Munson, J. C., and Khoshgoftaar, T. M. 1992. The detection of fault-prone programs. *IEEE Transactions on Software Engineering* 18(5): 423–433.

Nesi, P., and Querci, T. 1998. Effort estimation and prediction of object oriented systems. *The Journal of Systems and Software* 42: 89–102.

Paulish, D. J., and Carleton, A. D. 1994. Case studies of software process improvement measurement. *IEEE Computer* 27(9): 50–57.

Pighin, M. 1995. An experimental managerial metric to improve software quality. *Proceedings of ICQS'95*, Maribor, Slovenija, 91–96.

Pighin, M., and Zamolo, R. 1997. Predictive metric based on discriminant statistical analysis. *Proceedings of ICSE'97*. Boston, USA, 262–270.

Pighin, M. 1998. Dangerous complexity thresholds: an experimental definition. *Proceedings of the European Software Measurement Conference*. FESMA'98, Antwerp (BEL): Technological Institute Publications, 247–255.

Pighin, M., and Kokol, P. 1999. RPSM: A risk-predictive structural experimental metric. *Proceedings of the European Software Measurement Conference*. FESMA'99, Antwerp (BEL): Technological Institute Publications, 459–464.

Podgorelec, V., and Kokol, P. 2002. Evolutionary induced decision trees for dangerous software modules prediction. *Information Processing Letters* 82(1): 31–38.

Quinlan, J. R. 1993. *Programs for Machine Learning*. Morgan Kaufmann.

Roche, J. M. 1994. Software metrics end measurement principles. *ACM SIGSOFT SE Notes* 19(1): 2–18.

Šprogar, M., Kokol, P., Hleb Babiè, S., Podgorelec, V., and Zorman, M. 2000. Vector decision trees. *Intelligent Data Analysis*. IOS Press, 4(3–4): 305–321.

Thelin, T., and Runeson, P. 2000. Robust estimations of fault content with capture-recapture and detection profile estimators. *Journal of Systems and Software, special issue on EASE'99, 3rd Conference on Empirical Assessment and Software Evaluation* 52.

Thomson, H. E., and Mayhew, P. 1994. A practical approach for software process improvement. *Software Quality Management* edited by Ross M. et al., CMP, 1: 149–164.

Zhiwei Xu, Khoshgoftaar, T. M., and Allen, E. B. 2000. Prediction of software faults using fuzzy nonlinear regression modeling. *Proceedings of Fifth IEEE International Symposium on High Assurance Systems Engineering*, HASE 2000.

Maurizio Pighin received his Dottore degree in Electronic Engineering in 1980 from the University of Padova, Italy. Since 1981 he has been with the Department of Mathematics and Computer Science of the University of Udine, Italy, where, since 1986, he is a researcher and assistant professor of Computer Science. He currently teaches advanced courses of Software Engineering and Information Systems. He is

actually the Delegate for Information Systems of the University of Udine. His major research interests are in the area of Software Engineering and particularly in Metrics, Reuse and Human Resources Organization. His research accomplishments include the study of models of predictive code metric based on statistical analysis of project data and on the implementation of reuse and maintenance methodologies based on Information Retrieval engines. He has been involved in the organization of some important events in the fields of Software Engineering

Vili Podgorelec received his MSc degree in 1999 and PhD degree in Computer Science in 2001 from the University of Maribor, Slovenia. He is currently an assistant professor with the Faculty for Electrical Engineering and Computer Science, University of Maribor. His main research interests include Intelligent Systems, Software Engineering, Medical Informatics, and Computational Intelligence. He has been participating in many national and international research projects, is author of several journal papers on computational intelligence, medical informatics and software engineering and has received several international awards and grants for his research activities, including IEEE Region 8 Best Student Paper award in 1999 for his work on automatic medical diagnostic decision support model.

Peter Kokol was born in Celje, Slovenia. He earned a PhD in Computer Science from the University of Maribor, Slovenia in 1992. His research interests are Software Enginering, System Theory, Complexity, Software Metrics, Intelligent Systems and Medical Informatics. He has published more then 300 technical papers, among them about 50 in recognized international journals. He chaired CBMS and MIE conferences. He is a member of IEEE and ACM, chair of the IEEE TC on Computational Medicine and consultant for World Bank projects. He is also a lead person of several European and international projects.