PAPER *Special Section on Knowledge-Based Software Engineering*

A Model for Detecting Cost-Prone Classes Based on Mahalanobis-Taguchi Method

Hirohisa AMAN[†a)], ***Member***, **Naomi MOCHIDUKI**[†*], ***Nonmember***, ***and*** **Hiroyuki YAMADA**[†], ***Member***

SUMMARY In software development, comprehensive software reviews and testings are important activities to preserve high quality and to control maintenance cost. However it would be actually difficult to perform comprehensive software reviews and testings because of a lot of components, a lack of manpower and other realistic restrictions. To improve performances of reviews and testings in object-oriented software, this paper proposes a novel model for detecting cost-prone classes; the model is based on Mahalanobis-Taguchi method——an extended statistical discriminant method merging with a pattern recognition approach. Experimental results using a lot of Java software are provided to statistically demonstrate that the proposed model has a high ability for detecting cost-prone classes.

key words: *metrics, cost-proneness, prediction, discriminant analysis, Mahalanobis-Taguchi method*

1. Introduction

Comprehensive software reviews and testings are important activities to preserve high quality of software products and to control maintenance cost in software development [1]–[3]. In actual software development, however, it would be difficult to achieve comprehensive software reviews and testings because of a lot of components, a large size, a complex design, too short development duration, a lack of manpower, and other realistic restrictions. It results in an increased maintenance cost to ensure an acceptable quality level in a software product. There have been proposed some models and metrics for predicting maintenance cost and cost-related attributes in software development projects [4]–[7], such as maintenance time and manpower requirements. These models and metrics are practical but not ultimate solutions since they are not approaches to obviate rising maintenance cost.

A more effective solution to control maintenance cost in software projects is to detect cost-prone components (which would require higher cost than others in their maintenance activities) and/or fault-prone components (which would include one or more faults), and perform careful reviews and testings for such cost/fault-prone components. There have been studied some prediction models based on linear or nonlinear regression analysis: linear regression models for predicting lines of codes (LOC) to be changed in an object class through a version-upgrade using some object-oriented software metrics [8]–[10], a logistic regression model and a multivariate adaptive regression splines (MARS) model for evaluating a fault-proneness of class [11], [12], etc. While the regression model approach is an effective prediction method, it will be difficult to fit various software attributes with the regression line (or curve), especially outliers have harmful effects on the model construction, so that pattern recognition methods (*e.g.* Bayesian classifier) [13] and statistical classification methods (*e.g.* discriminant analysis) [14]–[17] would be promising approaches to predict cost/fault-prone classes. This paper proposes a novel model for predictively detecting cost-prone classes in object-oriented software, based on Mahalanobis-Taguchi (MT) method [18] that is an extended statistical discriminant method merging with a pattern recognition approach.

This paper is organized as follows: Section 2 discusses the theory of MT method along with brief descriptions of the existing discriminant analysis methods. Section 3 proposes a novel discriminant model for detecting cost-prone classes, based on MT method. Section 4 presents empirical studies using actual data collected from three large open-source software projects——Eclipse, Azureus, and jEdit. Section 5 draws our conclusion and future work.

2. Mahalanobis-Taguchi (MT) Method

An object class is a basic component in object-oriented software quality control. The contribution of this paper is to develop an effective model for predictively detecting classes whose maintenance activities would require higher cost than others; such classes are hereinafter called "cost-prone" classes. The model to be developed in this paper will discriminate cost-prone classes from the others using some software metrics, *i.e.* quantified software characteristics. This section presents brief discussions about some statistical methods underlying our discriminant model, and the key method, Mahalanobis-Taguchi (MT) method.

2.1 Discriminant Analysis Method

Consider an entity which has some observable properties. The entity can be expressed as a vector whose elements describe the observable properties, and then the entity vector can be plotted on a scatter diagram. Now let our entities be fallen into several groups. Discriminant analysis [15], [16]

Manuscript received June 29, 2005.
Manuscript revised October 6, 2005.

[†]The authors are with the Faculty of Engineering, Ehime University, Matsuyama-shi, 790–8577 Japan.
[*]Presently, with the FUJITSU Broad Solution & Consulting, Inc.
a) E-mail: aman@cs.ehime-u.ac.jp

DOI: 10.1093/ietisy/e89–d.4.1347

is a basic statistical method for drawing boundaries between those groups on the scatter diagram. For the sake of convenience, the remains of this section will discuss the case that we have two entity-groups in a two-dimensional space.

A simple method for separating two groups is the perpendicular bisector between the mean points of two groups. Figure 1 shows two entity-groups A and B, and the perpendicular bisector between their mean points: the entities of group A and B are marked by circles ("$\circ$") and crosses ("$\times$"), respectively; the mean points of group A and B are denoted by $\bar{\boldsymbol{x}}_{\boldsymbol{A}}$ and $\bar{\boldsymbol{x}}_{\boldsymbol{B}}$, respectively. In Fig. 1, however, three entities of group B ($\times$'s) are fallen into group A side. In order to avoid such inappropriate separation, we should take into account the dispersion of data as well.

The separation discussed above is based on Euclidean distances from the mean points of groups. We can introduce another distance with consideration for dispersion of data: Mahalanobis distance [15], [16]. For each entity group, Mahalanobis distance can be a group-specific distance that is based on the dispersion of data in the group. Mahalanobis distance is useful for describing a closeness of entity to a group on the scatter diagram, and the distance has been applied to the discriminant analysis as a common method for determining the closest group of entities. See Appendix A for the details of Mahalanobis distance.

For an entity vector $\boldsymbol{x}$, let $D_{(A)}(\boldsymbol{x})$ be Mahalanobis distance between $\boldsymbol{x}$ and $\bar{\boldsymbol{x}}_{\boldsymbol{A}}$; similarly, let $D_{(B)}(\boldsymbol{x})$ be Mahalanobis distance from $\bar{\boldsymbol{x}}_{\boldsymbol{B}}$. The locus of the following equation has been a boundary between the two groups:

$$|D_{(A)}(\boldsymbol{x})|^2 = |D_{(B)}(\boldsymbol{x})|^2 \ .$$

Figure 2 shows an example of the boundary based on Mahalanobis distance.

The perpendicular bisector such as shown in Fig. 1 is a particular case of Mahalanobis distance-based boundary, in which the two entity-groups have the same dispersion. If two entity-groups have the same or approximately the same dispersions (*i.e.* the same or approximately the same variance-covariance matrices), the perpendicular bisector is a useful boundary between two groups; otherwise, Mahalanobis distance-based boundary would be a better boundary between groups. Needless to say, we can not make a clear separation for the two groups that overlaps with each other on the scatter diagram, even if we use the Mahalanobis distance-based boundary.

However not all pairs of non-overlapped groups are properly separated using Mahalanobis distance-based boundary. Figure 3 shows an example of such a case that the Mahalanobis distance-based boundary could not separate two groups A ($\circ$'s) and B ($\times$'s). In Fig. 3, the two groups are not overlapped; group B gathers in a ring around group A. $\bar{\boldsymbol{x}}_{\boldsymbol{A}}$ and $\bar{\boldsymbol{x}}_{\boldsymbol{B}}$, the mean points of group A and B, stay in close to each other. Because of the closeness between $\bar{\boldsymbol{x}}_{\boldsymbol{A}}$ and $\bar{\boldsymbol{x}}_{\boldsymbol{B}}$, no boundaries drawn between $\bar{\boldsymbol{x}}_{\boldsymbol{A}}$ and $\bar{\boldsymbol{x}}_{\boldsymbol{B}}$ can separate the two groups properly in Fig. 3. Mahalanobis distance-based

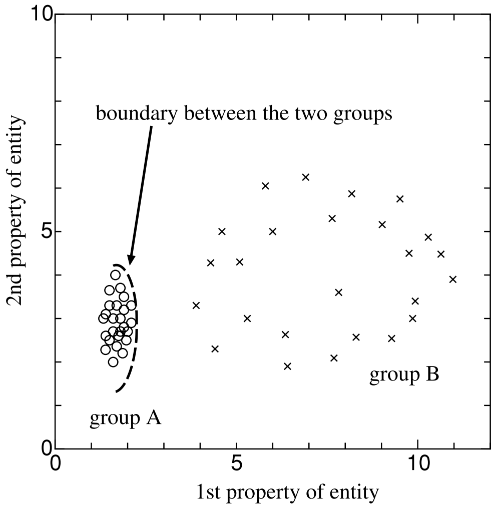


Fig. 2 Mahalanobis distance-based boundary.

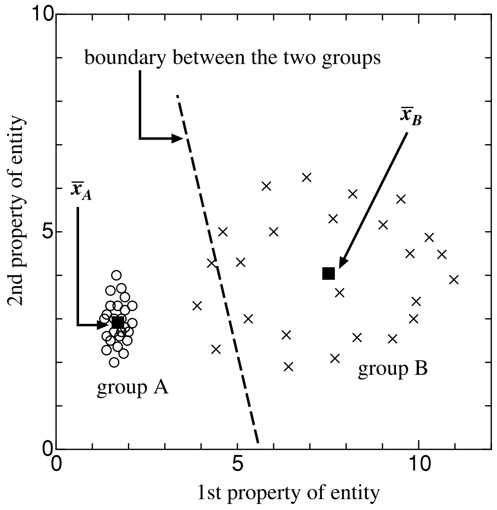


Fig. 1 Perpendicular bisector between two groups.

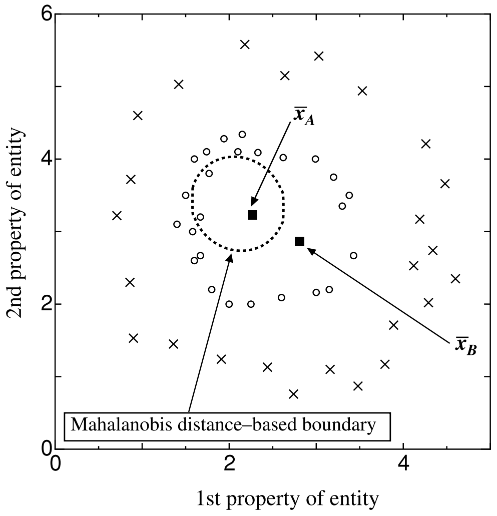


Fig. 3 A case that the mean points of two groups are close.

boundary would fail to discriminate data in such cases that the mean points of groups stay in close each other.

N. Mochiduki *et al.* [17] have applied the above Mahalanobis distance-based discriminant analysis to an object-oriented software maintainability prediction. In their empirical work, 927 version-upgrade cases of object classes written in Java were collected from some open-source software development sites, and the older classes of the version-upgrade cases were measured by the following five metrics [19], [20]: (1) Stmts, the number of executable/declaration statements; (2) NCM, the number of class methods; (3) NMA, the number of new methods neither inherited nor overriding; (4) NMO, the number of methods overriding that of an ancestor class; (5) PIM, the number of public instance method. Those object classes are plotted into a five-dimensional space using their metric values; the five-dimensional space corresponds to a scatter diagram discussed previously. The LOC modified in version-upgrade (see Appendix B) is also counted for each version-upgrade case. Then the largest 5% cases of modified LOC were considered to be high-cost cases, and their older classes were regarded as high-cost classes. N. Mochiduki *et al.* have tried to separate those high-cost classes from the others using Mahalanobis distance-based boundary in the space where classes are plotted using the above metrics. Their empirical results showed that the Mahalanobis distance-based discriminant model were only modestly beneficial because of a closeness between the mean points of two groups (high-cost classes and others) as the issue discussed previously.

We carried out a check experiment for the empirical work performed by N. Mochiduki *et al.* using the same metrics suite. Table 1 shows a summary of our empirical data measured for jEdit [27] version 4.0; jEdit is an open-source text editor written in Java, and that has been used in Mochiduki's experiment as a part of empirical data. According to their definition of "high-cost" class, we considered high-cost classes to be ones whose modified LOC were of the largest 5% in the version-upgrade "ver.4.0 → 4.1." In Table 1, $\boldsymbol{\mu}_{\{1,2\}}$ and $\boldsymbol{\sigma}_1$ denote the means and the standard deviations of the metric values, respectively. We could confirm the issue that the mean points of the two groups stay in close, since we can see $\boldsymbol{\mu}_2 \in [\boldsymbol{\mu}_1 - \boldsymbol{\sigma}_1, \boldsymbol{\mu}_1 + \boldsymbol{\sigma}_1]$, *i.e.* the distance between the mean points of groups is less than the standard deviation of the high-cost classes for each metric. Therefore the existing discriminant analysis methods discussed above (see Figs. 1 and 2) will not be useful in discriminating high-cost classes and the others for that software. Notice that the above discussion does not mean the distribution of metric value is similar to Fig. 3. Figure 3 is an example to explain that the existing discriminant analysis methods could not separate data properly when the mean points of groups stay in close each other.

The issue on a closeness between the mean points of two groups leads us to require more powerful and flexible discriminant method.

2.2 Mahalanobis-Taguchi (MT) Method

We now introduce another Mahalanobis distance-based method applicable to the above problematic case that the mean points of two groups are close to each other: the method is called Mahalanobis-Taguchi (MT) method [18].

MT method evaluates a closeness of an entity to particular entity-group using Mahalanobis distance. For example, let us consider a product quality control. Using some observable properties of product, we try to detect low quality products in a set of products unexamined. We get started by collecting some products that have good standard quality, and express them as the vectors of observable properties. The set of vectors corresponds to a group of standard products. Then, for a product unexamined, we calculate the Mahalanobis distance from the standard product group (see Appendix A for the calculation). The Mahalanobis distance describes a lack of closeness of the product to the standard product group[†]. In other words, the greater distance from the standard product group express the less quality of the product. In the field of quality engineering, MT method is often used as a useful method for discriminating product quality as discussed above.

We briefly mention the difference between the discriminant analysis described in Sect. 2.1 and MT method. The goal of discriminant analysis is to draw a boundary between entity-groups in a scatter diagram. MT method, however, is a method for quantifying a closeness of entity to particular entity-group. In that sense, MT method seems to be a pattern recognition method rather than discriminant method. It is fundamentally different that MT method does not draw any boundaries "between the mean points" of entity-groups.

MT method can properly discriminate entities even in the above problematic case that the mean points of entity-groups are close to each other in the scatter diagram (*e.g.* Fig. 3). In the case of Fig. 3, we can consider group A to be a standard entity-group discussed above, and regard the entities of group B as unexamined ones. Then we can evaluate closenesses of those unexamined entities (group B) to the standard entity-group (group A), using Mahalanobis distance. It is likely that an entity of group B has a larger Mahalanobis distance than the entities of group A, and the entities of group B are properly judged as the ones "other than group A" by MT method; we will show the numerical example in Sect. 3 (see Table 2).

Table 1 Empirical data measured for jEdit ver.4.0.

group	μ, σ	Stmts	NCM	NMA	NMO	PIM
high-cost classes	μ_1	1267.14	17.14	72.71	2.14	185.00
	σ_1	1239.46	39.65	77.33	2.73	194.01
others	μ_2	106.79	2.97	5.75	1.42	160.78

[†]More precisely, the vectors of standard product group have to be normalized such that the mean of Mahalanobis distance within the group is equal to one.

Table 2 Coordinate data and Mahalanobis distances of group A and B shown in Fig. 3.

Group A

i	$\boldsymbol{x}_i^T$	$D(\boldsymbol{x}_i)$
1	(2.99, 4.00)	1.05
2	(1.77, 3.80)	0.69
3	(2.62, 4.02)	0.81
4	(2.15, 4.34)	0.97
5	(1.94, 4.28)	0.95
6	(2.25, 2.00)	1.08
7	(3.20, 3.75)	1.11
8	(3.15, 2.20)	1.23
9	(3.30, 3.35)	1.09
10	(3.38, 3.50)	1.21
11	(1.60, 4.00)	0.93
12	(2.60, 2.09)	1.02
13	(2.00, 2.00)	1.13
14	(1.67, 3.20)	0.63
15	(3.00, 2.16)	1.16
16	(1.67, 2.67)	0.83
17	(1.80, 2.20)	1.06
18	(3.43, 2.67)	1.27
19	(1.58, 3.00)	0.77
20	(2.33, 4.09)	0.76
21	(1.50, 3.50)	0.82
22	(1.60, 2.60)	0.93
23	(2.10, 4.10)	0.77
24	(1.74, 4.10)	0.90
25	(1.40, 3.10)	0.93

Group B

j	$\boldsymbol{x}_j^T$	$D(\boldsymbol{x}_j)$
1	(2.18, 5.58)	2.05
2	(1.36, 1.45)	1.89
3	(1.42, 5.03)	1.74
4	(2.64, 5.15)	1.75
5	(0.87, 3.72)	1.50
6	(3.89, 1.71)	2.06
7	(2.74, 0.76)	2.17
8	(3.48, 0.87)	2.33
9	(3.16, 1.10)	2.01
10	(3.03, 5.42)	2.14
11	(0.86, 2.30)	1.75
12	(4.60, 2.35)	2.50
13	(0.90, 1.53)	2.15
14	(4.26, 4.21)	2.32
15	(4.48, 3.66)	2.38
16	(1.91, 1.24)	1.81
17	(4.34, 2.74)	2.18
18	(3.53, 4.94)	2.08
19	(4.29, 2.02)	2.23
20	(0.95, 4.60)	1.75
21	(0.71, 3.22)	1.64
22	(4.19, 3.17)	2.01
23	(2.44, 1.13)	1.83
24	(3.79, 1.17)	2.30
25	(4.12, 2.53)	1.98

3. Discriminant Model Based on MT Method

We now apply MT method to object-oriented software quality control. Our goal is to predictively discriminate cost-prone classes from the others using some metrics. The following is the algorithm for constructing our discriminant model.

Algorithm 1 (Discriminant model construction):
Entities to be examined are object classes, and observable quality-properties of those classes are measured by p metrics, m_j (for $j = 1, \ldots, p$); to avoid the potential for multicollinearity, no pairs of the metrics should have a middle or strong correlation in their measurements. We consider that all pairs of metrics used in the following algorithm must have correlation coefficients less than 0.4 and grater than -0.4, *i.e.* their absolute values are less than 0.4 [22].

I. Determination of calculation parameters

(i) Collect n sample classes, c_i (for $i = 1, \ldots, n$), to form a standard class set in which each class has a good standard quality in terms of maintenance cost. MT method empirically requires that the number of samples is grater than three-times the number of metrics, *i.e.* $n > 3p$.

(ii) Measure sample class c_i using metric m_j (for $i = 1, \ldots, n$; $j = 1, \ldots, p$); let x_{ij} be the metric value. The vector of metric values, $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{ip})^T$, expresses observable quality-properties of c_i. The vector is hereinafter called "metric vector."

(iii) For each metric m_j, calculate the mean of metric values, $\overline{x}_j$, and the standard deviation, σ_j.

(iv) Normalize $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{ip})^T$ with $\overline{x}_j$ and σ_j:

$$z_i = \left(\frac{x_{i1} - \overline{x}_1}{\sigma_1}, \ \ldots, \ \frac{x_{ip} - \overline{x}_p}{\sigma_p} \right)^T,$$

and compute the correlation matrix R with respect to the p metrics in $\{z_i\}$. Notice that the mean point of metric vectors is in the origin of coordinates when we use the above normalized vectors.

II. Definition of Mahalanobis distance function

For any class whose metric vector is $\boldsymbol{x} = (x_1, \ldots, x_p)^T$, the following equation gives the Mahalanobis distance[†] from the mean point of standard class set, $D(\boldsymbol{x})$:

$$D(\boldsymbol{x}) = \sqrt{\frac{z^T R^{-1} z}{p}} \ , \tag{1}$$

where

$$z = \left(\frac{x_1 - \overline{x}_1}{\sigma_1}, \ \ldots, \ \frac{x_p - \overline{x}_p}{\sigma_p} \right)^T .$$

In order to evaluate a uniformity of the standard class set, compute the standard deviation of $D(\boldsymbol{x}_i)$'s, σ_D. (The mean of them is expected to be 1 since Eq. (1) uses a normalized vector z and a normalizing factor $1/p$.) MT method requires that the standard class set has a high uniformity in terms of metric values; hence, remove the class c_i from the standard class set when $D(\boldsymbol{x}_i) > 1 + 2\sigma_D$. ("1" is the mean of $D(\boldsymbol{x}_i)$'s.) If one or more classes are removed from the standard set, go back to I-(ii). Notice that we should keep $n > 3p$; if the above removal operation makes $n \leq 3p$, halt the performance of algorithm, and review another samples.

III. Formulation of discriminant rule

Given a class c unexamined, whose metric vector is $\boldsymbol{x} = (x_1, \ldots, x_p)^T$. Let τ be a threshold value of Mahalanobis distance for discriminating classes. If $D(\boldsymbol{x}) \geq \tau$, we consider that c will be other than the standard class, *i.e.* c is a cost-prone class; otherwise, we consider that c will have an acceptable quality in terms of maintenance cost.

While the decision of τ depends on circumstances of analysis, we propose an algorithm for deciding τ later. (see Algorithm 2) □

We can apply the above algorithm to the case of Fig. 3 as follows: Consider the data shown in Fig. 3 to be two-dimensional metric vector observed in classes; Fig. 3 describes 50 classes where the groups A and B have 25 classes, respectively.

[†]Equation (1) uses the correlation matrix instead of variance-covariance matrix (see Appendix A) since the vectors are normalized in the computation.

I-(i). Consider group A to be the standard class set, and regard each data of group A as class c_i (for $i = 1, \ldots, 25$).

I-(ii). For each c_i, consider the coordinate data in Fig. 3 to be the elements of metric vector $\boldsymbol{x}_i = (x_{i1}, x_{i2})^T$. Table 2 shows the elements of these vectors.

I-(iii). The means and the standard deviations of metric values are $\overline{x}_1 = 2.27$, $\overline{x}_2 = 3.23$, $\sigma_1 = 0.677$ and $\sigma_2 = 0.812$.

I-(iv). The correlation matrix in the normalized vectors $\{z_i\}$ is

$$R = \begin{pmatrix} 1.00 & -0.0859 \\ -0.0859 & 1.00 \end{pmatrix}.$$

II. For any class whose metric vector is $\boldsymbol{x} = (x_1, x_2)^T$, the Mahalanobis distance from the mean point of standard class set, $D(\boldsymbol{x})$, is given by Eq. (1). Table 2 shows $D(\boldsymbol{x}_i)$ (for $i = 1, \ldots, 25$). The standard deviation of Mahalanobis distances σ_D is expressed as $\sigma_D = 0.175$. Then no classes have Mahalanobis distances grater than $1 + 2\sigma_D = 1.35$.

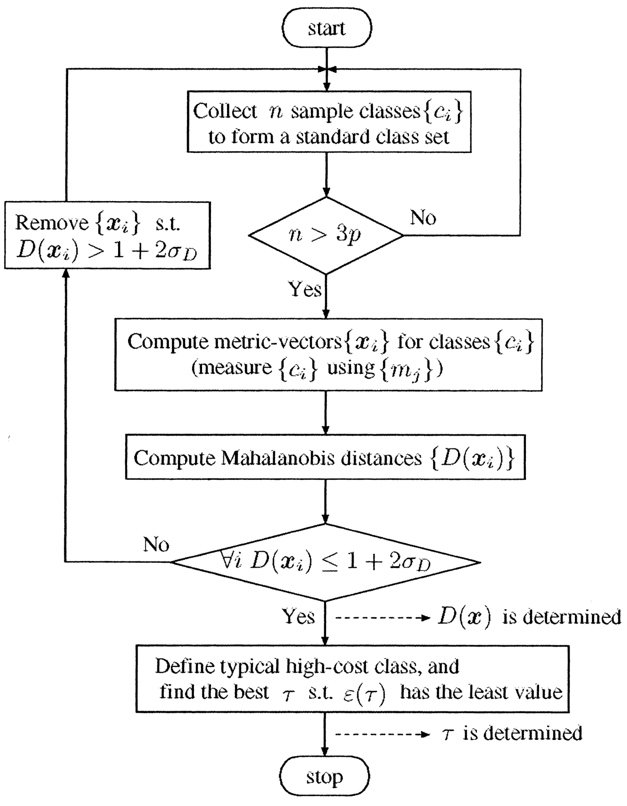
III. Suppose $\tau = 1 + 3\sigma_D = 1.52$, and examine the classes of group B with τ. Table 2 shows the Mahalanobis distances computed for group B, $D(\boldsymbol{x}_j)$ (for $j = 1, \ldots, 25$). Using $\tau = 1.52$ we can discriminate 24 classes (96%) of group B from group A properly. Needless to say, all (25) classes of group A have Mahalanobis distances less than τ, so that the model separates 49 classes (98%) properly. The model proposed has a high ability for discriminating entities even in the above problematic case shown in Fig. 3.
Notice that Fig. 3 is an example in which the existing discriminant analysis methods (discussed in Sect. 2.1) could not separate data properly. This example says MT method would be a better and promising discriminant method than the existing ones, but not MT method is intended to be used for only the case of Fig. 3. □

While the above calculation has used "$1 + 3\sigma_D$" as the threshold value τ, it has been one example among many available threshold values. "$1 + 3\sigma_D$" seems to be a reasonable threshold since the standard class set (group A) consists of the classes whose Mahalanobis distances are less than or equal to "$1 + 2\sigma_D$." However we could not find a sound ground to believe that the gap between "$2\sigma_D$" and "$3\sigma_D$" is sufficient for discriminating the cost-prone classes from the others. To decide an objective and valid threshold value for predicting cost-prone classes, we now propose a decision algorithm using version-upgrade data in object-oriented software.

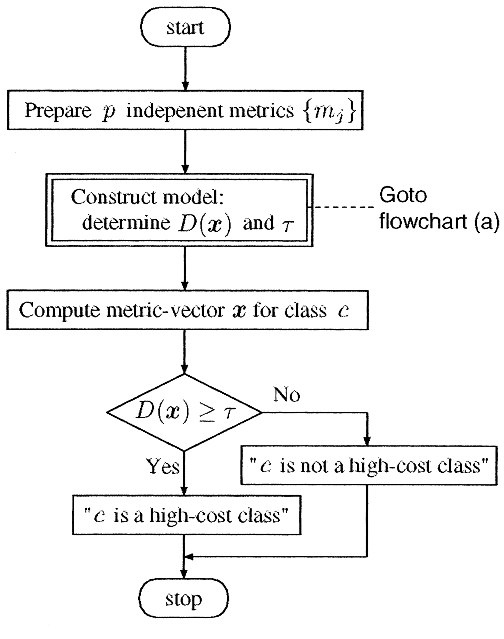
Table 3 Contingency table for classes.

	$D(x)$		total
	$< \tau$	$\geq \tau$	
typical high-cost classes	e_1	s_1	k_1
others	s_2	e_2	k_2

Algorithm 2 (Decision of threshold value τ):
Given an object-oriented software, including k classes, to be analyzed with the above discriminant model. Let $\{D_i\}_{i=1}^{k}$ be the sequence of Mahalanobis distances computed for the


start
Collect n sample classes $\{c_i\}$ to form a standard class set
$n > 3p$
No
Yes
Remove $\{\boldsymbol{x}_i\}$ s.t. $D(\boldsymbol{x}_i) > 1 + 2\sigma_D$
Compute metric-vectors $\{\boldsymbol{x}_i\}$ for classes $\{c_i\}$ (measure $\{c_i\}$ using $\{m_j\}$)
Compute Mahalanobis distances $\{D(\boldsymbol{x}_i)\}$
No
$\forall i\ D(\boldsymbol{x}_i) \leq 1 + 2\sigma_D$
Yes
$D(\boldsymbol{x})$ is determined
Define typical high-cost class, and find the best τ s.t. $\varepsilon(\tau)$ has the least value
τ is determined
stop

(a) model construction flow.


start
Prepare p indepenent metrics $\{m_j\}$
Construct model: determine $D(\boldsymbol{x})$ and τ
Goto flowchart (a)
Compute metric-vector $\boldsymbol{x}$ for class c
$D(\boldsymbol{x}) \geq \tau$
No
Yes
"c is not a high-cost class"
"c is a high-cost class"
stop

(b) model operation flow.

Fig. 4 Flowcharts of model operation and construction.

k classes (see Algorithm 1). Now categorize the k classes into the two groups (1) "typical" high-cost classes, and (2) the others, with a particular guideline (*e.g.* "whether a class has hundreds of LOC modified in the version-upgrade or not"). For any threshold value τ, the k classes are classified into four categories as the contingency table [23] shown in Table 3, where $k_1 + k_2 = k$.

Define $\tau = D_i$ such that $\forall j\ [\varepsilon(D_i) \leq \varepsilon(D_j)]$, *i.e.* $\varepsilon(\tau)$ has the least value when $\tau = D_i$, where $\varepsilon(\cdot)$ is the sum of error rates given by the following equation[†]:

$$\varepsilon(\tau) = \frac{e_1}{k_1} + \frac{e_2}{k_2}\ . \tag{2}$$

If $\forall j\ [\varepsilon(D_i) \leq \varepsilon(D_j)]$ holds for two or more D_i's, adopt the least D_i as τ. □

Algorithm 2 is based on the idea that the best threshold value τ has the least error in the discrimination of data. That is a pattern recognition algorithm separating "high-cost class" patterns from the others, and the decision of threshold value corresponds to a pattern learning. The algorithm uses only "typical" high-cost class patterns since a variety of patterns would provide a noise to the pattern learning.

In this section, we have proposed a discriminant model for predicting cost-prone classes. Figure 4 summarizes the model construction flow and the operation flow.

The related work includes some regression models for predicting maintenance costs, LOC to be changed in a class through a version-upgrade, using some object-oriented software metrics [8]–[10]. While the regression model approach is an effective prediction method, outliers have harmful effects on the model construction, and would make a restrictive prediction model. Our model focuses on predictively discriminating cost-prone classes, rather than estimating maintenance costs of classes with a regression equation. Our model overcomes a failing of the existing statistical discriminant methods (see Sect. 2), and has also a robustness against outliers in the model construction since the model adopts a pattern recognition approach.

4. Empirical Study

This section provides empirical validations of the discriminant model proposed in Sect. 2.

The following empirical studies focus on version-upgrades in object-oriented software. We consider LOC modified in a version-upgrade of a class to be a basic measure of maintenance cost that went into the class [8], [9], [14]. The more modified LOC corresponds to the higher maintenance cost. A modified LOC is composed of added LOC, removed LOC, and changed LOC, where comment statements and empty (only white space(s)) lines are excluded. See Appendix B for the details.

The goal of the proposed model is to discriminate "cost-prone" classes from the others; cost-prone classes would have more modified LOC than the other classes in further version-upgrades. We collected 5,760 classes written in Java from three open-source software projects, and constructed our models with Algorithms 1 and 2. Then we performed statistical tests to demonstrate an effectiveness of the proposed model.

4.1 Eclipse

Eclipse [24] is a universal tool platform, and a well-known and widely used open-source object-oriented software.

We collected 2,593 classes from the "`jdt.core`" component in Eclipse version 2.0, 2.1, and 3.0. This empirical study tries to predictively detect cost-prone classes in Eclipse ver.2.1, using

- the metric data collected from the classes in ver.2.0, and
- the version-upgrade data in ver.2.0 → 2.1.

The cost-proneness of a class in ver.2.1 is evaluated using the version-upgrade data in ver.2.1 → 3.0.

Now we perform our empirical study on Eclipse as follows: (see also Fig. 5)

1. [Measurement of LOC modified in version-upgrade]:
   Count LOC modified through the version-upgrades "ver.2.0 → 2.1" and "ver.2.1 → 3.0."

2. [Metric data collection in Eclipse ver.2.0]:
   Measure software quality attributes of classes in Eclipse ver.2.0 using the metrics described later.

3. [Model construction]:
   Construct our discriminant model using

   - the metric data measured in ver.2.0, and
   - the modified LOC in ver.2.0 → 2.1.

   (see Algorithm 1)

   Decide our threshold value τ using the modified LOC in ver.2.0 → 2.1. (see Algorithm 2)

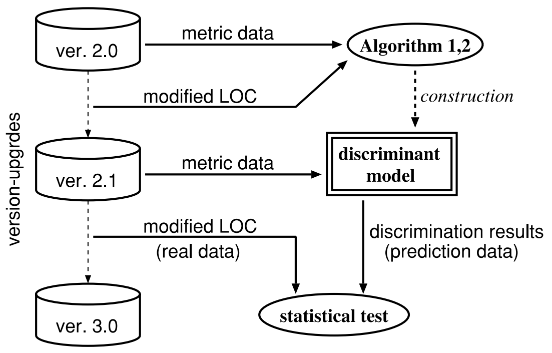


Fig. 5 Data flow in empirical study of Eclipse.

[†]In Table 3, e_1 and e_2 correspond to the numbers of classes failed in the discrimination with τ. Equation (2) uses normalized impact level of the errors e_1 and e_2.

4. [Prediction of cost-prone classes in Eclipse ver.2.1]:
   Measure software quality attributes of classes in Eclipse ver.2.1 using the metrics, and predict cost-prone classes in the next version-upgrade 2.1 → 3.0 using the metric data and the constructed model.

5. [Statistical test]:
   Perform a statistical test to validate whether the classes predicted as cost-prone have actually more modified LOC than the others in the version-upgrade ver.2.1 → 3.0. □

4.1.1 Measurement of LOC Modified in Version-Upgrade

We measured LOC modified in the version-upgrades ver.2.0 → 2.1 and ver.2.1 → 3.0. Table 4 and Fig. 6 shows a summary of statistics concerning the modified LOC.

4.1.2 Metric Data Collection in Eclipse Ver.2.0

To collect metric data of Java classes, we developed a suite of metric collection tools. Table 5 shows the 16 metrics that are available in our tool suite. See the literatures [19]–[21] for more details of these metrics.

Using our tool suite, we performed metric data collection for the 596 classes included in Eclipse ver.2.0. We omit the list of their metric data for lack of space.

4.1.3 Model Construction

Construct our discriminant model according to Algorithms 1 and 2, using the modified LOC in ver.2.0 → 2.1 (that are counted in Sect. 4.1.1) and the metric data of ver.2.0 (that are collected in Sect. 4.1.2).

Now we have 16-dimensional metric vectors in which each element corresponds to each metric shown in Table 5. In order to avoid the potential for multicollinearity in Mahalanobis distance computations, we have to use only the metrics that have high independency in their measurements. In this empirical work, the following five metrics are selected as our independent metrics since the other 11 metrics had middle or strong correlations (absolute value of their correlation coefficients are grater than or equal to 0.4 [22]) with one of the five metrics:

- DIT: the depth of a class in the inheritance tree. The deeper (the higher depth level) classes would be more likely to be overridden their methods and/or to be appended new methods. DIT value is related to a testability of the class.
- NAI: the number of attribute in a class. The more NAI value means that the class has more information to be managed, and the class has more roles. The less NAI value would indicate the higher reusability of the class.
- NCM: the number of class methods in a class. NCM denotes how many operations are common in the all instances of the class. It is a rare case that a class has many class methods, so that the larger NCM value may mean a poor class design.
- NCV: the number of class variables in a class. This is the number of variables shared in the all instances of the class. As in the case of NCM, it is a rare design to have many class variables, and the larger NCV value may indicate a poor class design.
- NM: the total number of methods in a class. NM shows how many operations can be performed in the class. This is a basic measure concerning the size and func-

Table 4 Statistics of modified LOC in Eclipse version-upgrades.

number of data	mean (μ)	standard deviation (σ)
883	91.9	235

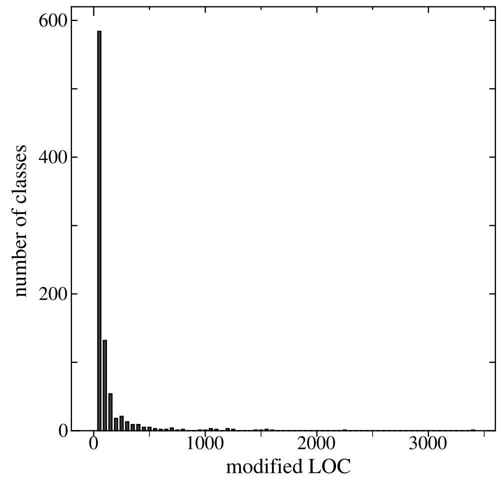


Fig. 6 Histogram of modified LOC in Eclipse version-upgrades.

Table 5 Metrics used in data collections.

metric	description
DIT	depth of inheritance tree; the depth level of the class in the class inheritance tree.
NAI	number of attributes in the class. (excluding inherited ones)
NCM	number of class methods in the class.
NCV	number of class variables in the class.
NIM	number of instance method in the class.
NIV	number of instance variables in the class.
NM	number of all methods in the class.
NMA	number of methods newly added in the class. (not inherited, and not overridden)
NMI	number of methods inherited from the ancestors and not overridden in the class.
NMO	number of methods overridden in the class.
NMNpub	number of non-public methods implemented in the class.
NMpub	number of public methods implemented in the class.
NumPara	number of parameters; the sum of number of method parameters implemented in the class.
PIM	number of public instance methods implemented in the class.
SIX	specialization index; the metric value is computed as NMO × DIT / (NMO + NMA + NMI).
Stmts	number of executable/declaration statements in the class.

tion points of the class. The more NM value denotes that the class would have more efforts of design, implementation, and maintenance for the class.

The selected five metrics seem to be a reasonable measure suite for predicting cost-prone classes, since these metrics are concerning a class testability, class design, and development efforts.

In Algorithm 1-I-(i), we determine n sample classes to form a standard class set in which the classes have good standard quality. In the context of this empirical work, those n sample classes should have low modified LOC in the version-upgrade. The distribution of modified LOC is remarkably concentrated in lower values (see Fig. 6), and it is difficult to statistically determine lower outliers using the mean (μ) and standard deviation (σ) since the frequently used criteria, such as $\mu - \sigma$ and $\mu - 2\sigma$, make no sense in this case (they have negative values; see Table 4). We now empirically define a standard class set such that a class of the standard set has a modified LOC less than 10. Then we obtain the standard class set containing 203 classes.

In Algorithm 1-I-(ii),(iii) and -II, we got the Mahalanobis distances $D(\boldsymbol{x}_i)$ (for $i = 1, \ldots, 203$) and the standard deviation $\sigma_D = 0.685$. Then we found six classes whose Mahalanobis distances are grater than $1 + 2\sigma_D = 2.37$, and removed those six classes from the standard class set. We iterated Algorithm 1-I-(ii),(iii) and -II, until all classes' Mahalanobis distances are less than or equal to $1 + 2\sigma_D$. After 16 iterations, we got a sophisticated class set composing 135 classes.

In Algorithm 2, we determine the threshold value τ. Now we have to define "typical" high-cost classes with a particular guideline; in the context of this empirical work, we have to determine "typical" large modified LOC in the version-upgrade. We consider higher outliers in the distribution of modified LOC to be those typical large ones, and use the guideline that typical large modified LOC is grater than $\mu + 2\sigma$. We selected 8 classes whose modified LOC are grater than $\mu + 2\sigma = 561.9$ (see Table 4) as typical high-cost classes, and the remaining 396 classes as the others. Then we found that $\varepsilon(\tau)$ has the least value when $\tau = 3.10$ (see Fig. 7), *i.e.* our threshold value of Mahalanobis distance is $\tau = 3.10$.

Now our construction of discriminant model has been completed. The followings are the details of that model: Given a class whose metric vector is $\boldsymbol{x} = (x_1, x_2, x_3, x_4, x_5)^T$.

- Mahalanobis distance function:

$$D(\boldsymbol{x}) = \sqrt{\frac{z^T R^{-1} z}{5}} \quad ,$$

where

$$z = \left(\frac{x_1 - 2.96}{1.53}, \frac{x_2 - 1.02}{1.29}, \frac{x_3 - 1.93}{1.44}, \frac{x_4 - 72.8}{37.0}, \frac{x_5 - 37.0}{13.9} \right)^T \quad ,$$

and

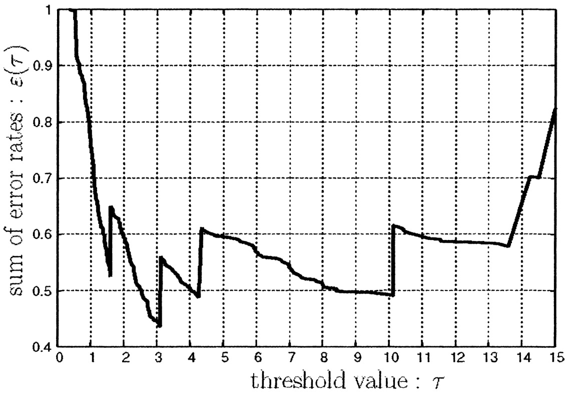


Fig. 7 Sum of error rates $\varepsilon(\tau)$.

Table 6 Means and standard deviations of metric values in Eclipse.

group	μ, σ	DIT	NAI	NCM	NCV	NM
high-cost classes	μ_1	2.00	8.13	2.13	47.25	122.13
	σ_1	1.07	9.78	3.27	109.73	98.04
others	μ_2	2.79	3.75	2.69	77.45	47.60

$$R^{-1} = \begin{array}{c} \begin{array}{ccccc} \text{DIT} & \text{NAI} & \text{NCM} & \text{NCV} & \text{NM} \end{array} \\ \begin{pmatrix} 8.06 & 0.359 & 2.32 & -6.19 & -3.75 \\ 0.359 & 1.57 & 0.545 & -0.578 & 0.513 \\ 2.32 & 0.545 & 14.0 & -1.22 & -14.0 \\ -6.19 & -0.578 & -1.22 & 6.77 & 0.766 \\ -3.75 & 0.513 & -14.0 & 0.766 & 17.1 \end{pmatrix} \end{array}.$$

- Discriminant rule:
  if $D(\boldsymbol{x}) \geq \tau = 3.10$ then we consider the class is a cost-prone class; otherwise, the class is not cost-prone. □

Table 6 shows the means and the standard deviations of the metric values in Eclipse. Since we can see $\boldsymbol{\mu}_2 \in [\boldsymbol{\mu}_1 - \boldsymbol{\sigma}_1, \boldsymbol{\mu}_1 + \boldsymbol{\sigma}_1]$, *i.e.* the mean points of the two groups stay in close, our MT-based model would be better in discriminating cost-prone classes than the existing statistical discriminant models discussed in Sect. 2.1.

4.1.4 Prediction of Cost-Prone Classes in Eclipse Ver.2.1

We have constructed a discriminant model using data of Eclipse ver.2.0: the metric data of ver.2.0 and the version-upgrade data in ver.2.0 → 2.1. Now we apply the model to the metric data of ver.2.1 to predict cost-prone classes in the next version-upgrade ver.2.1 → 3.0.

We had 618 classes in Eclipse ver.2.1, and inputed their metric data into the model constructed. Notice that the 618 classes include not only upgraded classes (modified LOC > 0) but also non-modified ones (modified LOC = 0).

The model predicted that 210 classes are cost-prone, and 408 classes are the others. Table 7 shows statistics of their actual modified LOC in the version-upgrade ver.2.1 → 3.0 where μ and σ denote the mean and the standard deviation of modified LOC, respectively.

Table 7 Statistics of modified LOC in classes predicted.

prediction	number of class	μ	σ
cost-prone	210	181	397
others	408	42.3	70.9

4.1.5 Statistical Test

Finally we performed a statistical test in order to validate the above predictions made by our model.

In this empirical work, we have considered that cost-prone classes would have large modified LOC in further version-upgrades. Thus the set of cost-prone classes predicted by our model should have a statistically significant difference from the set of other classes in their modified LOC, *i.e.* it should be statistically showed that our cost-prone classes have larger modified LOC than the others.

We make the following hypotheses:

H_0 **(null hypothesis)**
The mean of modified LOC in classes predicted as cost-prone (μ_1) is equal to the mean of that in the other classes (μ_0): $\mu_0 = \mu_1$.

H_1 **(alternative hypothesis-1)**
The mean of modified LOC in classes predicted as cost-prone is grater than the mean of that in the other classes: $\mu_1 > \mu_0$.

H_1' **(alternative hypothesis-2)**
The mean of modified LOC in classes predicted as cost-prone is "exactly" grater than the mean of that in the other classes: $\mu_1 > 2\mu_0$. □

The test can be performed with the normal probability distribution. Notice that this case corresponds to a test in large size samples; the sizes of two groups are 210 and 408. If our sample size is small (< 30), we should perform Student's t-test instead [25].

Let $\overline{X}_1$ and $\overline{X}_0$ be the sample means of modified LOC in the classes predicted as cost-prone and as the others, respectively. Similarly let s_1 and s_0 denote the sample standard deviations of modified LOC in the classes predicted as cost-prone and as the others, respectively. Assume the null hypothesis H_0 is true. Then the distribution of $\overline{X}_1 - \overline{X}_0$ is the normal distribution $N(0, \sigma_s^2)$, where

$$\sigma_s = \sqrt{\frac{s_1^2}{n_{s_1}} + \frac{s_0^2}{n_{s_0}}},$$

and n_{s_1} and n_{s_0} are the numbers of classes predicted as cost-prone and as the others, respectively. From Table 7, we have $\overline{X}_1 - \overline{X}_0 = 138.7$, and $\sigma_s = 27.7$. In this case, the p-value is $2.72{\times}10^{-07}$, and the null hypothesis (H_0) can be rejected; we can accept the alternative hypothesis-1 (H_1) with the significance level 0.00001%. When we replace $\overline{X}_0$ with $2\overline{X}_0$, the p-value is 2.50×10^{-4}. We can also accept the alternative hypothesis-2 (H_1') with the significance level 0.1%.

Therefore we could validate that our model has a high ability for predictively discriminating cost-prone classes in Eclipse `jdt.core` component.

4.2 Other Software: Azureus and jEdit

We also performed our model constructions and statistical validations to other open-source software: Azureus [26], and jEdit [27]. Azureus is a cross-platform Java BitTorrent client, and that is a heavily-downloaded software in SourceForge.net [28] (it is No.1 on the download-count ranking at the time of writing this paper). jEdit is a programmer's text editor written in Java, and that has been maintained for 5 years at SourceForge.net.

We collected 2,549 classes and 618 classes from Azureus (ver.2.0.8.2, 2.1.0.0 and 2.2.0.0) and jEdit (ver.4.0, 4.1 and 4.2), respectively. We present only their empirical results for lack of space.

- Azureus
  In measurements of classes included in Azureus, the following six metrics were independent: DIT, NAI, NCM, NCV, NMO, and PIM; hence we used those six metrics for our model construction. Note that the selection procedure of metrics is the same as the case of Eclipse, *i.e.* we selected independent metrics such that the absolute values of correlation coefficients in all metrics pairs are less than 0.4[†] (see Algorithm 1).
  The model predicted that 23 classes are cost-prone ones, and the remaining 595 classes are the others. Since this case corresponds to a small sample size, we performed Student's t-test with the degree of freedom $616 (= 23 + 595 - 2)$. The test results said the alternative hypothesis-1 (H_1) could be accepted with the significance level 0.001% (p-value = 4.87×10^{-5}), and the alternative hypothesis-2 (H_1') could be also accepted with the significance level 1% (p-value = 0.00575).
- jEdit
  We constructed our discriminant model using the following four metrics[††] that are independent in the measurements of classes included in jEdit: DIT, NAI, NCM, and NMO.
  The model predicted that 7 classes are cost-prone ones, and the remaining 162 classes are the others. We also performed Student's t-test with the degree of freedom $167 (= 7 + 162 - 2)$ in this case. As the results, we could accept the alternative hypothesis-1 (H_1) (p-value is approximately 0; it was too small to evaluate in our computer environment), and we could also accept the alternative hypothesis-2 (H_1') with the significance level 0.000000000001% (p-value = 3.22×10^{-15}).

We summarize our empirical results in Table 8. These empirical results statistically show our discriminant model is a useful model for predictively detecting cost-prone classes.

[†]The threshold value is determined according to the literature [22].

[††]The selection procedure of metrics is also the same as the cases of Eclipse and Azureus.

Table 8 Summary of empirical results.

software	metrics used in the model	p-value ($H'_1 : \mu_1 > 2\mu_0$)
Eclipse	DIT, NAI, NCM, NCV, NM	2.50×10^{-4}
Azureus	DIT, NAI, NCM, NCV, NMO, PIM	5.75×10^{-3}
jEdit	DIT, NAI, NCM, NMO	3.22×10^{-15}

We have used different metrics suites for different software, Eclipse, Azureus, and jEdit. It is natural that different software have different quality characteristics, and require different metrics suites for representing their quality attributes. Our model could flexibly accept the differences in the metrics for all software. A software requiring more metrics may have more complex distribution of metric data, then may be harder for discriminating cost-prone classes from the others, and vice versa. That might be one of interpretations of the differences in the empirical results shown in Table 8; the more metrics used in the model leads to the higher (the more wrong) p-value.

5. Conclusion and Future Work

We have proposed a model for predictively discriminating "cost-prone" classes, based on Mahalanobis-Taguchi (MT) method. MT method is a powerful method to detect low quality products using some observable properties, and we have applied the idea of MT method to object-oriented software quality control. The model proposed in this paper has been designed for discriminating cost-prone classes from the others using object-oriented software metrics. This model overcomes failings of the existing statistical discriminant methods [15]–[17] and regression models [8]–[10].

In our empirical work, we collected 5,760 classes written in Java from three open-source software projects: Eclipse, Azureus, and jEdit. For each software, we constructed the discriminant models, and performed statistical tests to validate the models. The empirical results show our model can predictively discriminate cost-prone classes from the others, with the significance level 1% (p-values are 2.50×10^{-4}, 5.75×10^{-3} and 3.22×10^{-15} in Eclipse, Azureus and jEdit, respectively.) The proposed model will be useful in predictively detecting cost-prone classes written in Java, and would aid effective reviews and testings of object-oriented software.

The proposed model could be a general framework for predicting software quality attributes using software metrics, since the model has no restrictions on metrics to be used (excepting independency), and on quality attributes to be predicted (not only "cost-proneness"). The application of this model into other quality attributes will be one of our future work. The followings will also be our future work: (1) investigation of automatically generated and/or reused code (including code clones [29]) effects on our model, (2) study of the impact of comment statements, excluded in our empirical work, on version-upgrades, and (3) analysis of source code changing histories [30], [31] and failure reports [32], [33] in order to find other useful factors for enhancement of our model.

Acknowledgments

The authors would like to thank the developers of open-source software used in the empirical studies. The authors also wish to thank the anonymous reviewers for their thoughtful and helpful comments.

This research was partially supported by the Inamori Foundation, and the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Young Scientists (B), 2004, 16700037.

References

[1] S.L. Pfleeger, Software Engineering, 2nd ed., Prentice Hall, London, 2001.
[2] C. Kaner, J. Falk, and H.Q. Nguyen, Testing Computer Software, 2nd ed., John Wiley & Sons, New York, 1999.
[3] G. Sabaliauskaite, S. Kusumoto, and K. Inoue, "Extended metrics to evaluate cost effectiveness of software inspection," IEICE Trans. Inf. & Syst., vol.E87-D, no.2, pp.475–480, Feb. 2004.
[4] R.K. Bandi, V.K. Vaishnavi, and D.E. Turk, "Predicting maintenance performance using object-oriented design complexity metrics," IEEE Trans. Softw. Eng., vol.29, no.1, pp.77–87, Jan. 2003.
[5] K. Pillai and V.S.S. Nair, "A model for software development effort and cost estimation," IEEE Trans. Softw. Eng., vol.23, no.8, pp.485–497, Aug. 1997.
[6] F. Fioravanti and P. Nesi, "Estimation and prediction metrics for adaptive maintenance effort of object-oriented systems," IEEE Trans. Softw. Eng., vol.27, no.12, pp.1062–1084, Dec. 2001.
[7] M. Jorgensen, "Experience with the accuracy of software maintenance task effort prediction models," IEEE Trans. Softw. Eng., vol.21, no.8, pp.674–681, Aug. 1995.
[8] W. Li and S. Henry, "Object-oriented metrics that predict maintainability," J. Syst. Softw., vol.23, pp.111–122, 1993.
[9] S. Wake and S. Henry, "A model based on software quality factors which predicts maintainability," Proc. Conf. on Software Maintenance, pp.382–387, Oct. 1988.
[10] M. Alshayeb and W. Li, "An empirical validation of object-oriented metrics in two different iterative software processes," IEEE Trans. Softw. Eng., vol.29, no.11, pp.1043–1049, Nov. 2003.
[11] V.R. Basili, L.C. Briand, and W.L. Melo, "A validation of object-oriented design metrics as quality indicators," IEEE Trans. Softw. Eng., vol.22, no.10, pp.751–761, Oct. 1996.
[12] L.C. Briand, W.L. Melo, and J. Wust, "Assessing the applicability of fault-proneness models across object-oriented software project," IEEE Trans. Softw. Eng., vol.28, no.7, pp.706–720, July 2002.
[13] L.C. Briand, V.R. Basili, and W.M. Thomas, "A pattern recognition approach for software engineering data analysis," IEEE Trans. Softw. Eng., vol.18, no.11, pp.931–942, Nov. 1992.
[14] H. Aman, N. Mochiduki, H. Yamada, and M.T. Noda, "A simple predictive method for discriminating costly classes using class size metric," IEICE Trans. Inf. & Syst., vol.E88-D, no.6, pp.1284–1288, June 2005.
[15] C.J. Huberty, Applied Discriminant Analysis, John Wiley & Sons, New York, 1994.
[16] B.F. Manly, Multivariate Statistical Methods, Chapman & Hall, London, 1986.
[17] N. Mochiduki, H. Aman, H. Yamada, and M.T. Noda, "A predictive discrimination of software modification effort using class size metrics," in Foundation of Software Engineering XI, ed. M. Noro and S. Yamamoto, pp.93–96, Kindai Kagaku Sha, Tokyo, 2004.

[18] G. Taguchi, S. Chowdhury, and Y. Wu, The Mahalanobis-Taguchi System, McGraw-Hill, OH, 2000.

[19] L.C. Briand, J. Wust, J.W. Daly, and D.V. Porter, "Exploring the relationships between design measures and software quality in object-oriented systems," J. Syst. Softw., vol.51, pp.245–273, 2000.

[20] M. Lorenz and J. Kidd, Object-Oriented Software Metrics, Prentice Hall, NJ, 1994.

[21] S.R. Chidamber and C.F. Kemerer, "A metrics suite for object oriented design," IEEE Trans. Softw. Eng., vol.20, no.6, pp.476–493, June 1994.

[22] P.G. Hoel, Elementary Statistics, John Wiley & Sons, New York, 1976.

[23] E.L. Crow, F.A. Davis, and M.W. Maxfield, Statistics Manual, Dover, New York, 1955.

[24] http://www.eclipse.org/

[25] M.R. Spiegel, Theory and Problems of Probability and Statistics, McGraw-Hill, OH, 1975.

[26] http://azureus.sourceforge.net/

[27] http://www.jedit.org/

[28] http://sourceforge.net/

[29] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A multilinguistic token-based code clone detection system for large scale source code," IEEE Trans. Softw. Eng., vol.28, no.7, pp.657–670, July 2002.

[30] A.T.T. Ying, G.C. Murphy, R. Ng, and M.C. Chu-Carroll, "Predicting source code changes by mining change history," IEEE Trans. Softw. Eng., vol.30, no.9, pp.574–586, Sept. 2004.

[31] T.L. Graves, A.F. Karr, J.s. Marron, and H. Siy, "Predicting fault incidence using software change history," IEEE Trans. Softw. Eng., vol.26, no.6, pp.653–661, July 2000.

[32] M. Fischer, M. Pinzger, and H. Gall, "Analyzing and relating bug report data for feature tracking," Proc. 10th Working Conf. on Reverse Engineering (WCSE'03), pp.90–99, 2003.

[33] M. Fischer, M. Pinzger, and H. Gall, "Populating a release history database from version control and bug tracking systems," Proc. International Conf. on Softw. Maintenance (ICSM'03), pp.23–33, 2003.

Appendix A: Mahalanobis Distance

In measuring the "distance" between two points in a space, we commonly use Euclidean distance: for example, in a two-dimensional space, the (Euclidean) distance between $\boldsymbol{p_1} = (x_1, y_1)^T$ and $\boldsymbol{p_2} = (x_2, y_2)^T$ is expressed as

$$|\boldsymbol{p_1} - \boldsymbol{p_2}| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \ .$$

However Euclidean distance may not be useful for a classification of entity on the scatter diagram (*i.e.* discriminant analysis) since it can not consider "dispersion" of data. For example, Fig. A· 1 is a scatter diagram plotting entities that are categorized into two groups A and B. The entities of group A and B are denoted by circles ($\circ$) and crosses ($\times$), respectively. Now let $\overline{\boldsymbol{x}}_{\boldsymbol{A}} = (1.71, 2.92)^T$ and $\overline{\boldsymbol{x}}_{\boldsymbol{B}} = (7.52, 4.04)^T$ are the means of two groups, respectively. In Fig. A· 1, $\boldsymbol{p} = (4.70, 2.70)^T$ seem to be closer to group B than A. However $\boldsymbol{p}$ is nearer to $\overline{\boldsymbol{x}}_{\boldsymbol{A}}$ than $\overline{\boldsymbol{x}}_{\boldsymbol{B}}$ in Euclidean distance:

$$|\boldsymbol{p} - \overline{\boldsymbol{x}}_A|^2 (\simeq 8.98) < |\boldsymbol{p} - \overline{\boldsymbol{x}}_B|^2 (\simeq 9.74)$$

Thus a linear boundary based on Euclidean distance such as

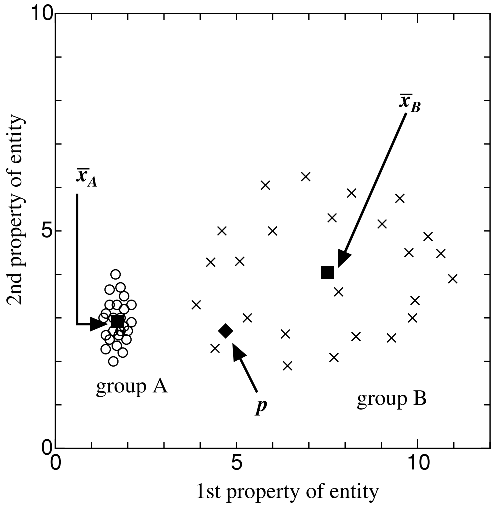


Fig. A· 1 Two groups of data.

the perpendicular bisector between $\overline{\boldsymbol{x}}_{\boldsymbol{A}}$ and $\overline{\boldsymbol{x}}_{\boldsymbol{B}}$ (see Fig. 1) classifies $\boldsymbol{p}$ into inappropriate group, A.

Mahalanobis distance is different from Euclidean distance in considering dispersion of data. The variance and covariance of data are also used to calculate Mahalanobis distance. While Euclidean distance between $\boldsymbol{p}$ and $\overline{\boldsymbol{x}}_{\boldsymbol{A}}$ ($|\boldsymbol{p} - \overline{\boldsymbol{x}}_A|_E$) has been given by

$$|\boldsymbol{p} - \overline{\boldsymbol{x}}_A|_E^2 = (\boldsymbol{p} - \overline{\boldsymbol{x}}_A)^T(\boldsymbol{p} - \overline{\boldsymbol{x}}_A) \ ,$$

Mahalanobis distance between them ($|\boldsymbol{p} - \overline{\boldsymbol{x}}_A|_M$) is calculated using the following equation:

$$|\boldsymbol{p} - \overline{\boldsymbol{x}}_A|_M^2 = (\boldsymbol{p} - \overline{\boldsymbol{x}}_A)^T S_A^{-1} (\boldsymbol{p} - \overline{\boldsymbol{x}}_A) \ ,$$

where S_A is the variance-covariance matrix for the data of group A. We can also calculate Mahalanobis distance for group B in the similar way. Mahalanobis distance is a group-specific distance that is based on dispersion of data, *i.e.* the variance-covariance matrix. For example, let S_A and S_B are

$$S_A = \begin{pmatrix} 0.0508 & 0.00748 \\ 0.00748 & 0.251 \end{pmatrix}, \ S_B = \begin{pmatrix} 4.87 & 0.256 \\ 0.256 & 1.76 \end{pmatrix} .$$

Then we obtain

$$|\boldsymbol{p} - \overline{\boldsymbol{x}}_A|_M^2 = (\boldsymbol{p} - \overline{\boldsymbol{x}}_A)^T S_A^{-1} (\boldsymbol{p} - \overline{\boldsymbol{x}}_A) \simeq 178 \ ,$$

and

$$|\boldsymbol{p} - \overline{\boldsymbol{x}}_B|_M^2 = (\boldsymbol{p} - \overline{\boldsymbol{x}}_B)^T S_B^{-1} (\boldsymbol{p} - \overline{\boldsymbol{x}}_B) \simeq 2.44 \ .$$

Therefore we see $|\boldsymbol{p} - \overline{\boldsymbol{x}}_B|_M < |\boldsymbol{p} - \overline{\boldsymbol{x}}_A|_M$, and $\boldsymbol{p}$ will be categorized into group B by the Mahalanobis distance-based discriminator.

Appendix B: LOC Modified in Version-Upgrade

In this paper, LOC modified in a version-upgrade is considered to be the number of different lines of source codes in two different versions, excepting comment statements and empty (only white space(s)) lines.

Given a version-upgrade case of a source code: let S_1 and S_2 be the older code and the newer one, respectively. We can get the same code as S_2 by iterating one of the following operations:

(1) insert consecutive n_1 (≥ 1) lines into S_1;

(2) remove consecutive n_2 (≥ 1) lines from S_1;

(3) replace consecutive n_3 (≥ 1) lines with another consecutive n'_3 (≥ 1) lines in S_1.

Regard the above operations (1), (2), and (3) as "adding," "removing," and "changing," respectively. Then define modified LOC as the total count of lines involved in the above operations: (1) n_1, (2) n_2, and (3) $\max(n_3, n'_3)$, where $\max(a, b)$ is a if $a > b$ otherwise b.

We can capture the lines involved in the above operations using Unix command `diff`; our empirical studies used "GNU diffutils ver.2.8.1" with the option "`-cbwB`." These options have the following functions:

- "`-c`": use the context output format;
- "`-b`": ignore changes in amount of white space;
- "`-w`": ignore white space when comparing lines;
- "`-B`": ignore changes that just insert or delete blank lines.

Figure A· 2 shows an example of use of `diff` command for two Java source files in which all comment statements are erased in advance; we have developed a tool for erasing all comment statements from a Java source file, and the tool has used in our empirical studies. In Fig. A· 2, the lines whose header are "+," "-," and "!" correspond to the added lines, the removed ones, and the changed ones, respectively. From the above definition, we can get the modified LOC in Fig. A· 2 as $4 (= n_1 + n_2 + \max(n_3, n'_3) = 1 + 1 + 2)$.

```
*** HelloWorld.java  2004-08-31 15:12:38.000000 +0900
--- HelloWorld.java  2005-06-30 17:05:29.000000 +0900

***************
*** 1,8 ****
  public class HelloWorld {
      public static void main(String[] args){
!         for ( int i = 0 ; i < 10 ; i++ ){
              System.out.println("Hello World!!");
-             System.out.println("Hello");
          }
      }
  }
--- 1,9 ----
  public class HelloWorld {
      public static void main(String[] args){
!         for ( int i = 0 ; i < 5 ; i++ ){
!             System.out.print("[" + i + "] ");
              System.out.println("Hello World!!");
          }
+         System.out.println("END");
      }
  }
```

Fig. A· 2 Example of output of `diff`.

Hirohisa Aman received the Dr. degree in Engineering from Kyushu Institute of Technology, Kitakyushu, Japan, in 2001. Since 2001, he has been with Ehime University. He is currently an assistant professor of the Faculty of Engineering, Ehime University. His primary research interest is in software maintenance cost estimation using metrics. He is a member of Information Processing Society of Japan (IPSJ), Japan Society for Software Science and Technology (JSSST), the Institute of Electrical and Electronics Engineering (IEEE), and Japan Society for Fuzzy Theory and Intelligent Informatics (SOFT).

Naomi Mochiduki received the Masters degree in Computer Science from Ehime University, Matsuyama, Japan, in 2005. She is currently with the FUJITSU Broad Solution & Consulting, Inc. She is a member of IPSJ.

Hiroyuki Yamada received the Dr. degree in Engineering from Osaka University, Suita, Japan, in 1988. Since 1988, he has been with Ehime University. He is currently an associate professor of the Faculty of Engineering, Ehime University. His primary research interests are in the field of knowledge-based software engineering, especially, software requirements acquisition and software evolution. He is a member of IPSJ, JSSST, IEEE, Association for Computing Machinery (ACM), and Japanese Society for Artificial Intelligence (JSAI).