Assessing the Applicability of Fault-Proneness Models Across Object-Oriented Software Projects

Lionel C. Briand, *Member*, *IEEE Computer Society*, Walcelio L. Melo, and Jürgen Wüst

Abstract—A number of papers have investigated the relationships between design metrics and the detection of faults in object-oriented software. Several of these studies have shown that such models can be accurate in predicting faulty classes within one particular software product. In practice, however, prediction models are built on certain products to be used on subsequent software development projects. How accurate can these models be considering the inevitable differences that may exist across projects and systems? Organizations typically learn and change. From a more general standpoint, can we obtain any evidence that such models are economically viable tools to focus validation and verification effort? This paper attempts to answer these questions by devising a general but tailorable cost-benefit model and by using fault and design data collected on two midsize Java systems developed in the same environment. Another contribution of the paper is the use of a novel exploratory analysis technique (MARS) to build such fault-proneness models, whose functional form is a priori unknown. Results indicate that a model built on one system can be accurately used to rank classes within another system according to their fault proneness. The downside, however, is that, because of system differences, the predicted fault probabilities are not representative of the system predicted. However, our cost-benefit model demonstrates that the MARS fault-proneness model is potentially viable, from an economical standpoint. The linear model is not nearly as good, thus suggesting a more complex model is required.

Index Terms—Object-oriented, metrics, measures, empirical validation, cross-validation.

1 Introduction

Measures of structural design properties such as coupling or complexity are widely considered to be indicators of external system quality attributes, such as reliability or maintainability. Many measures taking structural properties of object-oriented systems into account were proposed in the literature [2], [9], [7]. A number of case studies have provided empirical evidence that, by using regression analysis techniques, highly accurate prediction models for class fault-proneness can be built from existing OO design measures [5], [6]. The accuracy of the prediction models in these studies is usually quantified by some measure of goodness of fit of the model predictions to the actual fault data, or using within-system cross-validation techniques, such as V-cross-validation [14].

The purpose of building such models is to apply them to other systems (e.g., different systems developed in the same development environment), in order to focus verification and validation efforts on fault-prone parts of those systems. But, little is known about the accuracy of prediction models when they are actually applied under realistic conditions. In that sense, the existing studies can mostly be characterized as feasibility studies.

In this paper, we build a fault-proneness prediction model based on a set of OO measures using data collected from a midsized Java system and then apply the model to a different Java system developed by the same team. We then evaluate the accuracy the model's prediction in that system and the model's economic viability using a cost-benefit model.

A second objective is to use a novel exploratory, regression-based technique called MARS (Multivariate Adaptive Regression Splines [12]). We investigate whether this technique can result in more accurate fault-proneness models than typical logistic regression models assuming a linear relationship between covariates and the logit term.

Results indicate that a model built on one system can be used to accurately rank classes within another system according to their fault-proneness. However, only the MARS model is accurate whereas the linear logistic regression model does not seem appropriate. A cost-benefit model demonstrates such a fault-proneness model is potentially viable as it can provide clear benefits. However, the probabilities of a model built on one system applied to another suggest that differences in system factors make these probabilities meaningless, though the class ranking is preserved. In other words, if this is confirmed, such models' probabilities would need to be calibrated for each system. The implications of such a limitation are discussed below.

The paper is structured as follows: Section 2 describes the setting of the study. Section 3 briefly lays down our

- *L.C. Briand is with the Software Quality Engineering Laboratory, Systems and Computer Engineering, Carleton University, Colonel By Drive, Ottawa, ON, K1S 5B6, Canada. E-mail: briand@sce.carleton.ca.*
- *W.L. Melo is with Oracle Brazil and the University Católica de Brasilia, SQN Qd. 02- Bl A-Salas 604, Brasilia, DF Brazil 70712-900. E-mail: wmelo@computer.org.*
- *J. Wüst is with the Fraunhofer Institute for Experimental Software Engineering, Sauerwiesen 6, 67661 Kaiserslautern, Germany. E-mail: jwh69@gmx.net.*

Manuscript received 2 Apr. 2001; accepted 12 Sept. 2001.
Recommended for acceptance by M Sheppard.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 113914.

analysis methodology and introduces the modeling techniques, logistic regression, and MARS. Analysis results are detailed in Section 4 and we draw our conclusions in Section 5.

2 Description of Study Setting

This section describes the systems used for this study, the data collected, and the data collection procedures.

2.1 Systems

This section provides details about the two Java systems used in this study, *Xpose* and *Jwriter*, both developed at Oracle Brazil. Jwriter was developed first and then Xpose started. But, Xpose is the larger of the two systems in terms of fault data and number of classes and, as such, more fit for us to build a prediction model. We will therefore use Xpose for the fit data set from which we build our prediction models. Jwriter, the smaller system, will be used as a test data set. We try to put ourselves in a situation where one would have sufficient[1] design and fault data from one or a few projects and would use such data to build fault-proneness models to be used on new projects. Though, due to the characteristics of the data sets at hand, we had to use the two projects in their reverse order of performance (Xpose and Jwriter are used to build and test the model, respectively), we believe this has no effect on answering our research questions: The differences between projects are still there and are going to affect the model's accuracy. Our research design will simply have implications in the way we interpret the results.

2.1.1 Xpose

Xpose for Java is an application that allows its users to display and edit XML documents. With Xpose one can traverse an XML document's hierarchy, explore the attributes of a particular node in that hierarchy, or view and edit the XML source associated with it. The application offers a Document Browser, an XML rendering machine, and a property inspector and contents editor to view and modify XML elements.

Xpose was implemented under Sun's JDK 1.2, using the Swing library classes for the GUI and a prefabricated XML Parser[2] also developed at Oracle. Xpose consists of 144 Java classes, with a total of 1,774 methods. Some of the 144 classes were reused with modifications.

2.1.2 Jwriter

Jwriter is a component that provides basic word processing capabilities, i.e., text editing, formatting at the word, paragraph and document level, definition of styles (groups of formatting elements), support for file formats (RTF, HTML), in-document images, automated spell checking, and connectivity to Oracle Databases. Jwriter can be used standalone or integrated in other applications that need to provide word processing capabilities.

Jwriter was developed under Sun JDK 1.1, using Swing and a set of prefabricated toolkits for RTF and HTML data exchange and spell checking support. Jwriter consists of 68 Java classes, with a total of 933 methods.

2.2 Independent Variables

The set of design measures investigated in this study includes a subset of the coupling measures described in [4], [6], a set of measures related to polymorphism [2], a subset of the OO measurement suite of Chidamber and Kemerer [9], and some simple size measures based on counts of the methods and attributes. The appendix contains a summary of the measures we used.

The type of measures used here are design measures. Calculating these measures relies exclusively on information that is available from the definition of the class interfaces at the design stage (e.g., from UML diagrams): class methods and their parameter types, class attributes and their types, inheritance relationships.

Some of the classes in Xpose and Jwriter contain Java inner classes, in our case so-called "Listeners" which deal with Java events and usually are very small. These inner classes were not treated as individual observations (data points) in this study. Instead, in the calculation of our design measures, methods and attributes of inner classes were counted to contribute towards the containing class. Accordingly, faults traced back to an inner class were assigned to the outmost containing class. Clearly, other strategies to deal with inner classes are possible, e.g., by treating them as individual observations and adjusting the measures accordingly.

This choice entirely depends on how the resulting prediction models are intended to be used. If we want inner classes to be inspection objects that our prediction model can point to, we need to treat them as separate observations. Then, the fault data has to be collected at this finer level of granularity.

2.3 Dependent Variable

We want to evaluate whether existing OO design measures are useful for predicting the probability that a fault is detected in a class during field use. More precisely, the probability of fault detection that is meant here is a conditional probability: The probability that at least one failure occurring in the field is traced back to a fault in a class, depending on the obtained measurement values of the design measures for that class. This should be a good indicator of the class fault-proneness.

2.4 Data Collection Procedures and Measurement Instruments

For both Xpose and Jwriter the following relevant items were collected:

- the source code of the systems and
- data about faults found in the field by customers.

A static analyzer (Jmetrics [1], developed by Oracle Brazil) was developed to extract the values for the object-oriented design measures directly from the source code of Xpose and Jwriter. Even though the type of measures investigated here are design measures (as explained in

1. This is subjective and context dependent but, as a ballpark figure, our experience suggests that at least 100 observations (classes) are needed to account for the complexity of such fault-proneness models. As discussed below, Jwriter only contains 68 classes.

2. Oracle XML Parser, see http://technet.oracle.com.

Section 2.2), the measurement data was obtained from the source code since, in this case study, it is the only complete and readily available representation of the system amenable to automated measurement.

One issue that has usually to be dealt with when analyzing design measures and fault data is that systems are being measured at a given version and keep evolving. We therefore have to decide what version we measure and what fault data we consider. The way to address this issue is context dependent. When building fault-proneness models with Xpose fault data, we will use all faults due to failures reported by users of version 1.17 over a period of roughly three months. We also collected our design measurements on Xpose version 1.17 and the class interfaces, on which are based the design measures we used, have not changed significantly after that version. When we will apply our fault-proneness model to Jwriter, we will apply it to version 0.5 (the first delivered version) and use fault data due to failures reported by users of version 0.5, which was operational over a period of four months). We therefore emulate the situation where we would build a model based on the fault history of a system and then apply it to a new system that has just been developed.

For Xpose, the collected fault data comprises 132 faults, located in 31 out of the 144 classes, i.e., 21 percent of the Xpose classes are faulty. For Jwriter, 28 faults were located in 27 out of the 68 classes, i.e., 39 percent of the Jwriter classes are faulty.

3 Data Analysis Methodology

In this section we describe the methodology used to analyze the OO measurement data collected. The analysis procedure comprises an analysis of the descriptive statistics, principal component analysis, univariate and multivariate regression analysis against the fault data, and an evaluation of the prediction model. The reader is referred [5] to for a detailed description and justification of our analysis procedures.

3.1 Descriptive Statistics

The distribution and variance of each measure is examined to select those with enough variance for further analysis. Low variance measures do not differentiate classes very well and, therefore, will not be useful predictors in our data set. Furthermore, a comparison of the descriptive statistics between the two systems will help us better interpret the results of our analysis below.

3.2 Principal Component Analysis

If a group of variables in a data set are strongly correlated, these variables are likely to measure the same underlying dimension (i.e., class property) of the object to be measured. Principal component analysis (PCA) is a standard technique to identify the underlying, orthogonal dimensions that explain relations between the variables in the data set.

Principal components (PCs) are linear combinations of the standardized independent variables. The sum of the square of the coefficients in each linear combination is equal to one. PCs are calculated as follows: The first PC is the linear combination of all standardized variables that explain a maximum amount of variance in the data set. The second and subsequent PCs are linear combinations of all standardized variables, where each new PC is orthogonal to all previously calculated PCs and captures a maximum variance under these conditions. Usually, only a subset of all variables have large coefficients—also called the loading of the variable—and, therefore, contribute significantly to the variance of each PC. The variables with high loadings help identify the dimension the PC is capturing but this usually requires some degree of interpretation.

In order to identify these variables and interpret the PCs, we consider the rotated components. This is a technique where the PCs are subjected to an orthogonal rotation. As a result, the rotated components show a clearer pattern of loadings, where the variables either have a very low or high loading, thus showing either a negligible or a significant impact on the PC. There exist several strategies to perform such a rotation. We used the *varimax* rotation, which is the most frequently used strategy in the literature. See [11] for more details on PCA and rotated components.

3.3 Univariate and Multivariate Regression Analysis

Univariate logistic regression analysis is performed for each individual measure (independent variable) against the dependent variable, i.e., no fault/fault detection, in order to determine if the measure is a useful predictor of fault proneness.

Next, we build multivariate prediction models using logistic regression. This analysis is conducted to determine how well we can predict the fault proneness of classes, when the measures are used in combination. The covariates of the models are determined by using a mixed stepwise selection heuristic [13], which aims at identifying a subset of the available measures with a high goodness of fit to the data.[3] We build two types of prediction models

- Using the OO design measures directly as covariates.
- Using the *basic functions* from MARS as covariates. This combination of Mars and logistic regression is purported to yield higher accuracy models, having a more realistic functional form.

Details about logistic regression and MARS are given in the following sections.

3.3.1 Logistic Regression

The dependent variable we use to validate the design measures is the probability of an event, i.e., that at least one fault will be traced back to a particular class due to a failure occurring during system operation. Therefore, we use logistic regression, a standard technique based on maximum likelihood estimation, for the regression analysis. In the following, we give a short introduction to logistic regression and the user is referred to [5] for a more complete description of the application of logistic regression to build fault-proneness models. Full details on logistic regression can be found in [13].

A multivariate logistic regression model is based on the following relationship equation (the univariate model is a

3. Other heuristics, such as using principal component analysis to preselect variables, were tried but did not result in significant differences.

special case of this, where only one independent variable appears):

$$\pi(X_1, \cdots, X_n) = \frac{e^{(c_0 + c_1 X_1 + \cdots + c_n X_n)}}{1 + e^{(c_0 + c_1 X_1 + \cdots + c_n X_n)}}.$$

π is the probability that a fault will be traced back to a class after a failure during operation and the X_is are the design measures included as independent variables in the model (also called covariates).

Our dependent variable, π, is a conditional probability: the probability that a fault is found in a class, as a function of the class' structural properties. The plot of π vs. a single X_i—assuming that all other X_js are constant—takes a flexible S shape which ranges between two extreme cases:

- When X_i is not significant, then the curve approximates a horizontal line, i.e., π does not depend on X_i.
- When X_i entirely differentiates fault-prone software parts from the ones that are not, then the curve approximates a step function.

Such an S shape is perfectly suitable as long as the relationship between X_is and π is monotonic, an assumption consistent with our empirical hypotheses regarding the design measures. Otherwise, higher degree terms have to be introduced in the regression equation.

3.3.2 MARS

When analyzing and modeling the relationship between design properties and class fault proneness, one of the main issues is that relationships between these variables are expected to be complex (nonlinear) and to involve interaction effects. Because we currently know little about what nonlinearities and interaction to expect and because such relationships are also expected to vary from one organization to another, analyzing the measurement data in order to understand what affects fault proneness is usually a rather complex, exploratory process.

MARS is a novel statistical method presented in [12] and supported by a recent tool developed by Salford Systems.[4] At a high level, MARS attempts to approximate complex relationships by a series of linear regressions on different *intervals* of the independent variable ranges (i.e., subregions of the independent variable space). It is very flexible as it can adapt any functional form and is thus suitable to exploratory data analysis. One challenge though is to find the appropriate intervals on which to run independent linear regressions, for each independent variable, and identify interactions while avoiding overfitting the data. This is the purpose of the search algorithms proposed by the MARS methodology. Though these algorithms are complex and out of the scope of this paper, MARS is based on a number of simple principles. They are introduced below in order for the reader to understand the results presented in later sections.

There are alternative approaches to model complex relationships between dependent and independent variables, most notably artificial neural networks [8]. The results in [8] show that, for data sets of sizes comparable to what we use in this study, MARS models are more likely to be accurate than artificial neural networks, thus justifying our choice of MARS for this study.

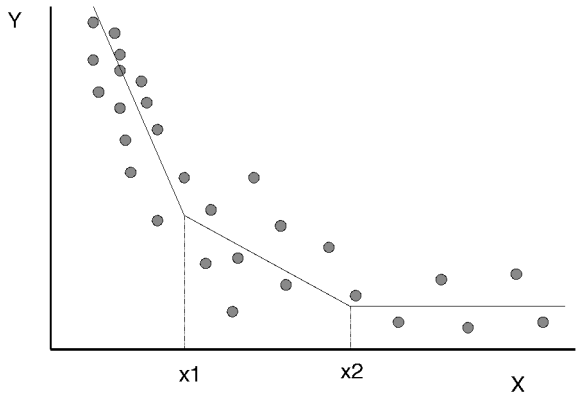


Fig. 1. Example knots in MARS.

Fig. 1 illustrates a simple example of how MARS would attempt to fit data, in a two dimension space (where Y and X are the dependent and independent variables, respectively), with piece-wise linear regression splines. A key concept is the notion of *knots*, that are the points that mark the end of region of data where a distinct linear regression is run, i.e., where the behavior of the modeled function changes. Fig. 1 shows two knots: x_1 and x_2. They delimit three intervals where different linear relationships are identified. MARS search algorithms identify appropriate knots in an automated way, though a number of search parameters have to be set by the user. Of course, in a case with higher dimensions and interactions between independent variables, the search becomes much more complex but the fundamental principles remain the same. The reader is referred to [12] for further details.

In order to model the concept of knots and piece-wise linear regression splines, MARS uses the concept of basis functions. These are functions of the form:

max(0, X-c) or
max(0, c-X),

where X is an independent variable and c a constant.

Such basis functions reexpress an independent variable X by mapping it to new variables, which are basis functions of the form described above. For max(0, X-c), X is set to 0 for all values of X up to some threshold value c and is equal to X for all values of X greater than c. By mixing the two types of basis functions presented above and providing adequate values for c, it is possible to approximate any functional shape. Determining the right knots (threshold values c) is a key challenge addressed by MARS search algorithms. In short, basis functions are used as the new independent variables of our regression estimation models. MARS also looks for interaction terms among basis functions, thus leading to the modeling of the interactions among independent variables.

In this paper, we are building a classification model. Our dependent variable is binary. From the short introduction above, it is clear that basis functions are identified assuming the dependent variable is continuous. In this context, as

4. www.salford-systems.com.

recommended in [12], we use MARS in two steps: 1) use the MARS algorithms to identify relevant basis functions and 2) refit the model with logistic regression, using the basis functions as covariates.

3.4 Model Evaluation

We evaluate the accuracy of our prediction models in terms of correctness and completeness, two standard measures used for classification techniques. The model predicts classes as either fault-prone or not fault-prone.

Correctness is the number of classes correctly classified as fault-prone, divided by the total number of classes classified as fault-prone. Low correctness means that a high percentage of the classes being classified as fault-prone do not actually contain a fault. We want correctness to be high, as inspections of classes that do not contain faults is a waste of resources.

Completeness is defined as the number of faults in classes classified as fault-prone, divided by the total number of faults in the system. It is a measure of the percentage of faults that would have been found if we used the prediction model in the stated manner. Low completeness indicates that many faults are not detected. These faults would then slip to subsequent development phases, where they are more expensive to correct.

In this study, we perform model evaluation in two different contexts:

V-cross-validation within Xpose. V-cross-validation [14] is a technique to obtain a realistic estimate of the predictive power of prediction models, when they are applied to data sets other than those the models were derived from, but no other test data set is available. In short, a V-cross validation divides the data set into V parts, each part being used to assess a model built on the remainder of the data set. In our study, we will divide our 144 observations data set into 10 randomly selected parts and perform a 10-cross validation.

Cross-system validation consists of applying the prediction model built from one data set (Xpose) to a different data set (Jwriter), and assess the accuracy of these predictions. We use the larger and, therefore, more representative data set of Xpose as fit data and Jwriter as test data.

3.5 Cost-Benefit Model

We need to define a cost-benefit model that helps us determine whether a given fault-proneness model would be economically viable. We propose a model below that is general, tailorable, and makes a number of assumptions.

- All classes predicted fault-prone are inspected.
- Usually, an inspection does not find all faults in a class. We assume an average inspection effectiveness e, $0 \leq e \leq 1$, where $e = 1$ means that all faults in inspected classes are being detected.
- Faults that are not discovered during inspection (faults that slipped through, faults in classes not inspected) later cause costs for isolating and fixing them. The average cost of a fault when not detected during inspection is *fc*.
- The cost of inspecting a class is assumed to be proportional to the size of the class.
- For computing a benefit, we need a baseline of comparison. As a baseline, we assume a simple model that ranks the classes by their size, and selects the n largest classes for inspection. The number n is chosen so that the total size of the selected classes is roughly the same as the total size of classes selected by the fault-proneness model based on design measures. It is thus ensured that we compare models where the investment—the cost of inspections—is the same or similar.

For the specification of the model, we need some additional definitions. Let $c_1, \ldots, c_N$ denote the N classes in the system. For $i = 1, \ldots, N$, let

- f_i be the number of actual faults in class i.
- p_i indicate if class i is predicted fault-prone by the model, i.e., $p_i = 1$ if class i is predicted fault-prone, $p_i = 0$, otherwise.
- s_i denote the size of class i (measured in terms of the number of methods, though other measures of size are possible).

The cost of inspection is $ic \cdot s_i$, where ic is the cost of inspection of one size unit.

Gains and losses are all expressed below in effort units, i.e., the effort saved and the effort incurred assuming inspections are performed on code, for example.

When using a fault-proneness model, the gain and cost of using the model can be expressed as:

Gain (effort saved):

g_m = defects covered and found.

$g\_m = e \cdot fc \cdot \Sigma_i(f_i \cdot p_i)$.

Cost (effort incurred):

c_m = direct inspection cost + defects not covered + defects that escape.

$c\_m = ic \cdot \Sigma_i(s_i \cdot p_i) + fc \cdot \Sigma_i(f_i \cdot (1 - p_i)) + (1 - e) \cdot fc \cdot \Sigma_i(f_i \cdot p_i)$.

In the same way, we express the cost and gains of using the size ranking model to select the n largest classes, so that their cumulative size is equal or close to $\Sigma_i(s_i \cdot p_i)$, the size of classes selected by the model.[5] For $i = 1, \ldots, N$, let $p'_i = 1$ if class i is among those n largest classes and $p'_i = 0$, otherwise.

$$g\_s = e \cdot fc \cdot \Sigma_i(f_i \cdot p'_i).$$
$$c\_s = ic \cdot \Sigma_i(s_i \cdot p'_i) + fc \cdot \Sigma_i(f_i \cdot (1 - p'_i)) + (1 - e) \cdot fc \cdot \Sigma_i(f_i \cdot p'_i).$$

We now want to assess the difference in cost and gain when using the fault-proneness model and size-ranking model, which is our comparison baseline:

$$\Delta gain = g\_m - g\_s = e \cdot fc \cdot (\Sigma_i(f_i \cdot p_i) - \Sigma_i(f_i \cdot p'_i)).$$
$$\Delta cost = c\_m - c\_s = ic \cdot (\Sigma_i(s_i \cdot p_i) - \Sigma_i(s_i \cdot p'_i)) + fc(\Sigma_i(f_i \cdot (1 - p_i)) - \Sigma_i(f_i \cdot (1 - p'_i))) + (1 - e) \cdot fc(\Sigma_i(f_i \cdot p_i) - \Sigma_i(f_i \cdot p'_i)).$$

5. We may not be able to get the exact same size, but we should be sufficiently close so that we perform the forthcoming simplifications. This is usually not difficult as the size of classes composing a system usually represent a small percentage of the system size. In practice, we can therefore make such an approximation and find an adequate set of n largest classes.

TABLE 1
Descriptive Statistics for Xpose and Jwriter

	Xpose, based on 144 classes							**Jwriter, based on 68 classes**						
Measure	Mean	StdDev	Max	P75	Median	P25	Min	Mean	StdDev	Max	P75	Median	P25	Min
SPA	0.2	0.7	5	0	0	0	0	0.3	1.1	8	0	0	0	0
DPA	1.1	3.0	16	0	0	0	0	1.7	4.0	20	1.5	0	0	0
SPD	0.1	0.5	5	0	0	0	0	0.0	0.0	0	0	0	0	0
DPD	0.8	6.7	78	0	0	0	0	0.8	3.7	28	0	0	0	0
SP	0.3	0.8	5	0	0	0	0	0.3	1.1	8	0	0	0	0
DP	1.9	7.3	78	1	0	0	0	2.4	5.9	30	2.5	0	0	0
NIP	30.9	37.4	199	44	17	4	0	46.2	40.2	145	63	41.5	11	0
OVO	2.4	5.4	42	2	0	0	0	1.2	2.7	19	2	0	0	0
ACAIC	0.2	0.5	4	0	0	0	0	0.0	0.2	1	0	0	0	0
ACMIC	1.5	3.4	24	2	0	0	0	0.2	0.8	5	0	0	0	0
DCAEC	0.1	0.8	9	0	0	0	0	0.0	0.0	0	0	0	0	0
DCMEC	0.4	4.3	51	0	0	0	0	0.0	0.0	0	0	0	0	0
OCAIC	1.1	2.1	14	1	0	0	0	3.2	4.5	17	4	1	0	0
OCAEC	0.8	1.8	14	1	0	0	0	2.7	6.2	38	2	1	0	0
OCMIC	2.2	3.5	21	3	1	0	0	1.7	3.0	17	2	1	0	0
OCMEC	2.7	7.7	69	2	0	0	0	2.1	5.2	40	2	1	0	0
DIT	2.3	1.8	7	3.5	1	1	1	1.2	1.1	4	2	1	0	0
NOC	0.6	3.1	29	0	0	0	0	0.2	0.6	3	0	0	0	0
Totatrib	3.3	4.7	24	5	1	0	0	9.3	10.8	41	14	7	0.5	0
Totprivatrib	2.9	4.7	24	4	0	0	0	6.4	9.7	39	9	1	0	0
Totprivmethod	2.1	5.1	37	2	0	0	0	2.4	4.4	19	2.5	1	0	0
Totmethod	12.3	13.9	83	18	7	3	0	13.7	12.1	66	19	9	6	1

We select n and, therefore, the p_i's so that

$$\Sigma_i(s_i \cdot p_i) - \Sigma_i(s_i \cdot p_i') \approx 0.$$

(Inspected classes have roughly equal size in both situations). We can thus, as an approximation, drop the first term from the Δcost equation. This also eliminates the inspection cost *ic* from the equation, and with it, the need to make assumptions about the ratio *fc* to *ic* for calculating values of $\Delta cost$. With this simplification, we have

$$\begin{aligned}\Delta\text{cost} = {} & fc \cdot (\Sigma_i(f_i \cdot (1 - p_i)) - \Sigma_i(f_i \cdot (1 - p_i'))) \\ & + (1 - e) \cdot fc \cdot (\Sigma_i(f_i \cdot p_i) - \Sigma_i(f_i \cdot p_i')).\end{aligned}$$

By doing the multiplications and adding summands it is easily shown that

$$\Delta\text{cost} = -e \cdot fc \cdot (\Sigma_i(f_i \cdot p_i) - \Sigma_i(f_i \cdot p_i')) = -\Delta\text{gain}.$$

The benefit of using the prediction model to select classes for inspection instead of selecting them according to their size is

$$\begin{aligned}\text{benefit} &= \Delta\text{gain} - \Delta\text{cost} \\ &= 2\Delta\text{gain} = 2 \cdot e \cdot fc \cdot (\Sigma_i(f_i \cdot p_i) - \Sigma_i(f_i \cdot p_i')).\end{aligned}$$

Thus, the benefit of using the fault-proneness model is proportional to the number of faults it detects above what the size-based model can find (if inspection effort is about equal to that of the baseline model, as is the case here). The factor 2 is because the difference between not finding a fault and having to pay *fc* and finding a fault and not having to pay *fc* is 2*fc*.

As we will see below, this will allow us, for a given range of *e* values, to determine the benefits of using a fault-proneness model in function of *fc*, the cost of a defect slipping through inspections. Both *e* and *fc* are context-dependent but it will still allow us to assess the order of magnitude of the model net benefits, if any.

4 Analysis Results

In this section, we present the details on our analysis results. We start off with the descriptive statistics of the design measures collected for Jwriter and Xpose (Section 4.1). We then perform principal component analysis (Section 4.2), univariate analysis (Section 4.3), and multivariate analysis using logistic regression and MARS to build and evaluate prediction models from the Xpose data set (Section 4.4). In Section 4.5, we apply the prediction model to Jwriter.

4.1 Descriptive Statistics

Table 1 presents the descriptive statistics for Xpose and Jwriter. Columns "Max," "75 percent," "Med.," "25 percent," "Min.," "Mean," and "Std Dev" state for each measure the maximum value, 75 percent quartile, median, 25 percent quartile, minimum, mean value, and standard deviation, respectively. For Xpose, the use of inheritance is sparse (low mean values of DIT, NOC). This has already been found in many systems before [5], [6], [9], [10]. As a consequence, the polymorphism measures from [2], and the measures counting coupling from/to ancestors/descendents too show low means and standard deviations. The measure DCAEC, DCMEC, SPD, and DPD have nonzero values only for very few (six or less) classes. We therefore drop these measures from the remainder of the analysis.

Turning the discussion to Jwriter, inheritance is even less frequently used there (average DIT is 1.2). Measures DCAEC, DCMEC, and SPD do not vary at all in this system. While the number of methods (totmethod, totprivmethod) shows remarkably similar distributions in both systems, the number of attributes (totattrib, totprivattrib) in Jwriter is about a factor three higher than in Xpose (and, therefore, aggregation coupling is higher, too). Measures OCMIC and OCMEC have about 25 percent lower means in Jwriter than in Xpose.

To conclude, even though the systems stem from the same development environment, the distributions of the measures vary considerably in some cases. This is certainly not an ideal situation for our goal to apply prediction models across systems but it may be a realistic one. The differences in distributions may be partly due to the fact that Xpose had a more experienced project manager in OO analysis and design, though the development team was the same. In addition, the team was more experienced when Xpose started and they used design by contract and enforced much more rigorous programming standards. All this may have helped them identify better objects and classes, and devise a better system class diagram. The Java libraries used for Jwriter were not yet mature, requiring work-around solutions to overcome shortcomings of the libraries, i.e., deviations from the originally intended design were necessary. This problem was less severe for Xpose. From a more general standpoint, in any organization, practices are likely to evolve over time, as people gain more experience and as key personnel changes. So, our data sets may reflect realistic conditions under which fault-proneness models have to be used. We will discuss further the implications in our analysis below.

4.2 Principal Component Analysis

Table 2 shows the results from PCA performed on the Xpose data set. We identified six orthogonal dimensions capturing 76 percent of the data set variance. For each PC, we provide its eigenvalue, the variance of the data set explained by the PC (in percent), and the cumulative variance. Below are the loadings of each measure in each rotated component. Values above 0.7 are set in boldface, these are the measures we call into play when we interpret the PCs.

Based on the loadings of the measures in each rotated component, the PCs are interpreted as follows:

- PC1: Class size, in terms of attributes and methods.
- PC2: The number of children / descendents of a class.
- PC3: Amount of polymorphism taking place.
- PC4: Export coupling to other classes.
- PC5: Import coupling from ancestor classes.
- PC6: Only measure OVO (overloading in stand-alone classes) has a strong weight in this PC.

These results do partially confirm earlier findings, that import and export coupling to ancestors/descendents/other classes tend to span orthogonal dimensions and that import coupling measures are somewhat associated with size measures. Note that the loadings in PC1 are not as strong as in the other PCs, the correlation between OCAIC and TotAtrib as measured by Pearson's r is 0.7, i.e., the measures explain less than 50 percent of the variation of each other.

TABLE 2
Rotated Components for Xpose Data Set

	PC1	PC2	PC3	PC4	PC5	PC6
EigenValue:	5.977	2.189	1.515	1.439	1.372	1.213
Percent:	33.205	12.161	8.415	7.995	7.621	6.738
CumPercent:	33.205	45.366	53.781	61.776	69.397	76.135
SPA	-0.176	0.062	**-0.924**	0.037	0.080	0.010
DPA	-0.072	-0.486	-0.156	-0.225	0.516	-0.182
ACAIC	-0.322	0.166	0.085	0.170	**0.747**	0.224
ACMIC	-0.118	-0.126	-0.202	0.067	**0.848**	-0.128
SP	-0.223	-0.175	**-0.891**	0.034	0.071	-0.021
DP	-0.219	**-0.922**	-0.026	0.083	0.056	-0.028
NIP	**-0.712**	-0.196	-0.139	-0.091	0.068	0.005
OCAIC	**-0.801**	0.063	-0.283	0.126	0.013	0.197
OCAEC	0.048	-0.026	-0.154	**0.801**	0.025	-0.245
OCMIC	-0.558	-0.578	-0.044	-0.088	-0.004	-0.166
OCMEC	-0.207	-0.191	0.101	**0.785**	0.081	0.071
OVO	-0.120	0.005	-0.055	0.087	0.085	**-0.862**
DIT	-0.211	0.120	-0.365	-0.094	0.172	0.456
NOC	-0.110	**-0.871**	0.016	0.256	-0.014	0.029
TotAtrib	**-0.850**	-0.243	-0.140	0.163	0.117	0.054
TotPrivAtrib	**-0.851**	-0.223	-0.153	0.182	0.118	0.057
TotPrivMeth.	**-0.736**	0.053	0.019	-0.082	0.272	-0.199
TotMethod	**-0.730**	-0.347	-0.136	0.116	0.191	-0.447

4.3 Univariate Analysis

The results of the univariate analysis on Xpose are summarized in Table 3. For each measure, the regression coefficient, along with its standard error, and the statistical significance (p-value), which is the probability that the coefficient is different from zero by chance are provided (see columns "Coef.," "S.E.," and "p-val," respectively).

As is visible from Table 3, most measures are found to be significantly (at $\alpha = 0.05$) related with fault-proneness.[6] All regression coefficients are positive, i.e., the higher coupling, polymorphism, or size of the class, the higher its fault-proneness. This is consistent with the general hypotheses associated with these measures [6], [2] and confirms results from earlier studies [5], [6].

Inheritance measures show a weak trend with fault-proneness (deeper classes and classes with a higher number of children are fault-prone). However, these trends are not significant at $\alpha = 0.05$. Also, this result is different from what was found in [5], where deeper classes too were more fault-prone, but classes with higher NOC were less fault-prone, and [6], where deeper classes were less fault-prone and NOC not significant at all. As discussed in [6], the impact of inheritance is likely to vary according to the inheritance strategy used and the experience of designers.

6. However, this may be due in part to repeated testing that increases the probability to find a significant relationship by chance. Though they are simple techniques to address this problem, e.g., Bonferroni procedure: divide your significance level (α) by the number tests performed, we use only univariate analysis here as a first variable preselection stage. As a result, here we will not use any technique to alleviate the problems due to repeated statistical testing.

TABLE 3
Univariate Analysis Results for Xpose

Msr.	Coef.	S.E.	p-val
SPA	0.719	0.306	**0.019**
DPA	0.241	0.075	**0.001**
SP	0.624	0.246	**0.011**
DP	0.242	0.069	**0.000**
NIP	0.042	0.008	**0.000**
OVO	0.026	0.033	0.428
ACAIC	0.625	0.334	0.062
ACMIC	0.171	0.066	**0.010**
OCAIC	0.536	0.142	**0.000**
OCAEC	0.147	0.099	0.137
OCMIC	0.447	0.092	**0.000**
OCMEC	0.099	0.041	**0.016**
DIT	0.182	0.106	0.086
NOC	0.740	0.380	0.052
TotAtrib	0.244	0.054	**0.000**
TotPrivAtrib	0.233	0.053	**0.000**
TotPrivMeth.	0.183	0.057	**0.001**
TotMethod	0.087	0.020	**0.000**

4.4 Multivariate Analysis

In this section, we present the multivariate prediction models that were built for the Xpose data set, using a mixed stepwise selection procedure [13]. Two models were built, a "linear" model (Section 4.4.1) that uses directly the design measures as independent variables and a MARS model (Section 4.4.2) that uses the MARS basis functions as covariates.

We also assess the model accuracy using a 10-cross-validation within Xpose. This is done for two purposes:

- To compare the model accuracy to results from previous studies, which also performed 10-cross-validation within the same system.
- To establish a comparison baseline when the models are then applied to the Jwriter data set in Section 4.5.

4.4.1 Linear Model

The multivariate linear model is shown in Table 4. The stepwise variable selection procedure retained measure NIP (belonging to the "class size" dimension PC1 as described in Section 4.2), an export coupling measure from PC3 (OCMEC), and an import coupling measure (OCMIC, not clearly associated with a particular PC). Overall, this is consistent with previous results [5], [6] where class size, import coupling and, to a lesser extent, export coupling play a role in fault proneness.

TABLE 4
Linear Model

Measure	Coeff	Std. Err	P Value
NIP	0.031	0.009	0.001
OCMIC	0.216	0.097	0.026
OCMEC	0.066	0.034	0.055
Intercept	-3.355	0.492	0.000

TABLE 5
Results of 10-Cross Validation of Linear Model

		Predicted		
		<0.5	>0.5	
Actual	No fault	108	5	113
	Fault	17 (50 faults)	14 (82 faults)	31 (132 faults)
		125	19	144

The results from performing a 10-cross validation are summarized in Table 5 and Fig. 2. In Table 5, classes with a predicted probability larger than 0.5 are classified as fault-prone, the others are predicted not fault-prone. The contingency table shows how predicted and actual faulty classes overlap and how many faults are contained by faulty classes, whether predicted faulty or not. Fourteen out of the 19 classes that are predicted fault-prone actually do contain a fault (73.6 percent correctness), these classes contain 82 out of the 132 faults in the system (62 percent completeness).

While the model is capable of identifying 19 out of 144 classes (i.e., 13 percent of all classes), which contain over 60 percent of all faults, the fact that we still miss 40 percent of the faults if we relied on the model prediction may hamper the model's practicality. We will investigate below this issue using the cost-benefit model introduced in Section 3.4.

The above figures are based on a cutoff value of $\pi = 0.5$ for the distinction fault-prone/not fault-prone, and the table only gives a partial picture, as other cut-off values are possible. Fig. 2 shows the correctness and completeness numbers as a function of the threshold π.

4.4.2 MARS Model

We used the MARS tool to run the MARS algorithms and we obtained the following basis functions:

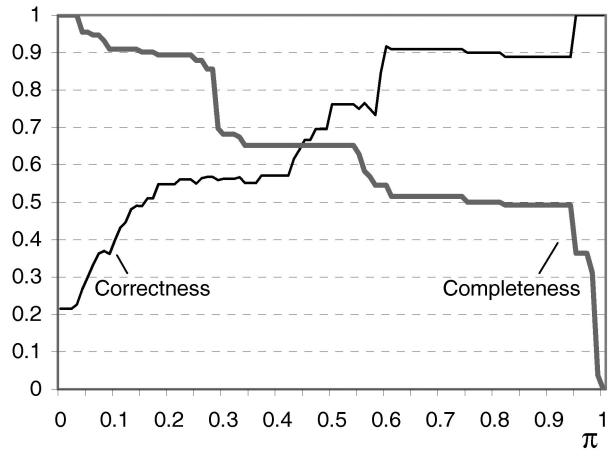


Fig. 2. Correctness and completeness-linear model.

TABLE 6
Mars Model

Measure	Coeff	Std. Err	p-value
OCMIC	0.947	0.188	< 0.001
BF5	-0.244	0.064	0.0002
BF6	0.096	0.055	0.0818
Intercept	2.867	0.408	< 0.001

BF3 = max(0, OCMIC - .129804E-07) ~ OCMIC
BF5 = max(0, 4.000 - DIT) * BF3
BF6 = max(0, OCMEC - 1.000)

After rerunning a logistic regression with the basis functions above (using OCMIC for BF3), we obtain the results in Table 6.

The MARS model is similar to the linear model in the sense that it involves OCMIC and OCMEC, the latter being a weaker predictor. The difference is that DIT (depth of class in inheritance tree, not associated with any of the principal components identified above) is selected as opposed to NIP in the linear model. Also, DIT clearly interacts with OCMIC. The sign of the coefficient of BF5 suggests that OCMIC has a stronger impact when DIT increases, that is import coupling impacts fault proneness further at deeper levels of inheritance. Though more investigation is required, the compound effect of inherited and imported class elements may generate an even higher complexity in terms of understanding and verifying classes.

Table 7 shows the contingency table that results when we apply the 10-cross validation to the MARS model. Fifteen of the 22 classes predicted fault-prone actually are faulty (68 percent correctness), these classes contain 97 of the 132 classes (73 percent completeness).

The MARS model does not show a strong difference with the linear model in terms of correctness. But, in terms of completeness, it performs far better. In other words, it is more accurate for the classes containing larger numbers of faults. Correctness and completeness as a function of π are shown in Fig. 3. As can be seen, the chosen cutoff value of 0.5 is in fact a good tradeoff between correctness and completeness for this model.

Overall, the accuracy of the models presented here is lower than what was found in previous studies [5], [6], where models achieved over 80 percent correctness and 90 percent completeness in 10-cross-validation. One explanation for the worse performance in the models here is that the previous studies used a more complete set of

TABLE 7
Results of 10-Cross Validation of MARS Model

		Predicted		
		<0.5	>0.5	
Actual	No fault	106	7	113
	Fault	16 (35 faults)	15 (97 faults)	31 (132 faults)
		122	22	144

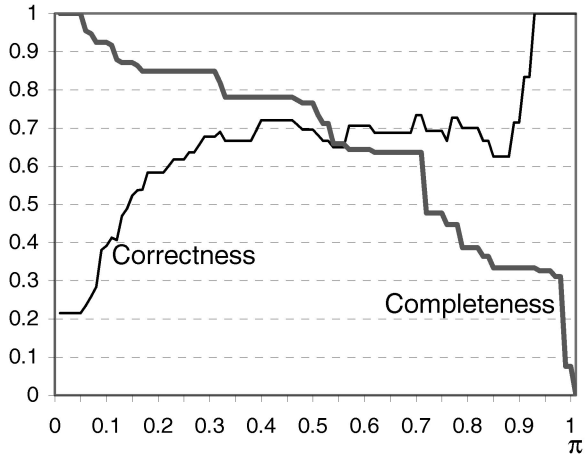


Fig. 3. Correctness and completeness—MARS model.

design measures. Especially, coupling measures based on method invocations were found to be very significant indicators of fault proneness. These measures were not available for the present study.

4.4.3 Cost/Benefit Analysis for Linear and MARS Models

Fig. 4 shows the benefit of using the linear and MARS model, respectively, to select classes for inspection over a simple size-based selection of classes for inspection, as defined in Section 3.5. This is possibly optimistic as there may be other ways to select classes, e.g., based on expert opinion. But, we need a quantifiable comparison baseline for the benefit model and, in addition, a model-based class selection allows us to take advantage of the previous experience of the development organization and making it available to less experienced developers. Moreover, in practice, finding the right experts at the right time may not be practical and asking them to rank classes may turn out to be a tedious and difficult task to perform.

The benefit is shown as a function of the number of classes selected for inspection by the fault-proneness models, which is itself dependent on the threshold π (An

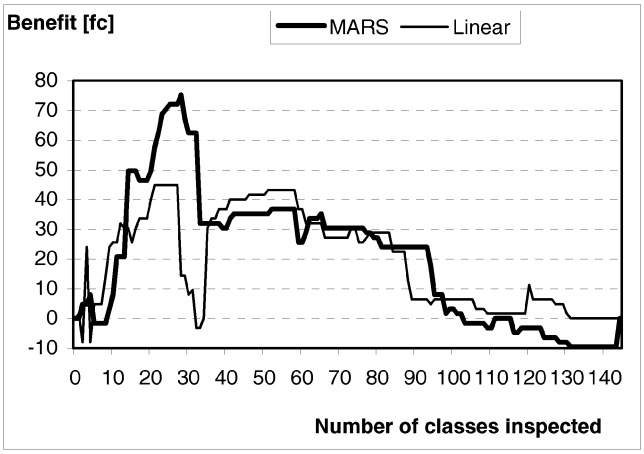


Fig. 4. Benefit graph for MARS model (inspection effectiveness e = 80 percent).

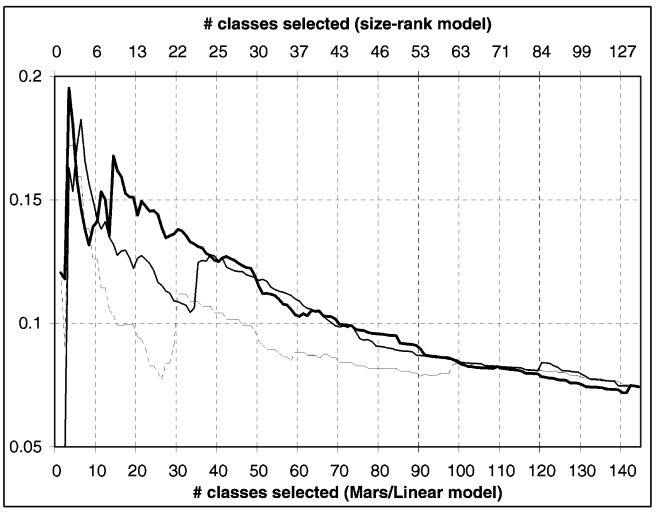


Fig. 5. Fault density of classes selected for inspection for Mars, Linear, and size-rank (comparison benchmark) model.

alternative way to look at this is that we rank the classes by predicted fault-proneness and then select a given number of classes based on this ranking). The benefit is expressed in multiples of *fc*, the average cost of a fault that goes undetected in inspections, and assumes an inspection effectiveness *e* of 0.8 (cf. Section 3.5 for how *e* and *fc* affect the model benefit). In a literature survey of industry data [3], *fc* typically ranged between 4.5 and 17 person hours, with a most likely value of six person hours. So, we can see that the type of benefits illustrated in Fig. 4 can represent substantial savings.

As a general trend, both curves first show a steep rise until they reach their maximum of 44.8*fc* (linear) and 75.8*fc* (MARS) at around n = 28 classes and then gradually descend until the benefit is back to zero when all classes are selected for inspection. As a comparison, for our data set, a hypothetical optimal model that can perfectly rank the classes by the actual number of faults they contain reaches its maximum benefit of 118*fc* at n = 31 classes (still assuming the same inspection efficiency e = 0.8). The linear model only achieves 38 percent of this theoretical maximum, whereas the MARS model with 65 percent is much closer, though there is still room for improvement.

Fig. 5 shows the fault density (number of faults per method) of the classes selected for inspection by the MARS and Linear model, as well as the size-rank model, as a function of the number of classes selected. The figure features two horizontal axes: The axis at the bottom applies to the curve of the Mars and Linear models, the axis shown at the top of the graph applies to the size-rank model. The top axis was chosen so that, at any vertical cross section in Fig. 5, the size of the classes selected for inspection by the size-rank model is about the same as those selected by the Mars and Linear model. This is in line with the strategy of our cost-benefit model to select the number of classes for the size-rank model, as defined in Section 3.5. We thus achieve traceability of the fault density curves with the cost benefit curves in Fig. 4—high benefits correspond with large discrepancies in fault density with the size-rank model and vice versa. Naturally, the number of classes at the top axis grows slower first, as the size-rank model selects bigger and, therefore, fewer classes, but catches up for higher values of n in the rightmost third of Fig. 5.

For n < 10, none of the models show a consistently significant benefit. The steep benefit rise begins with n = 10 classes. This is also visible from the fault densities in Fig. 5. All three models achieve their peak in fault density very early (before n = 5). This is due to an individual class with a large absolute number of faults (24) that is found early by all three models—hence, the models achieve similar fault densities and the MARS and linear model have no benefit over the size-ranking model for small values of n.

Also, worth noticing in Fig. 5 is that overall the fault densities of the classes selected steadily decrease as n increases. This is an encouraging result as we want our models to show such a pattern. However, in the range $n \in [15;35]$, the MARS model clearly selects higher density classes than both the Linear and size-based models. This explains the higher benefits observed in Fig. 4 and this is an important advantage of the MARS model as the range where it shows clearly superior benefits is the range where, in practice, the threshold between inspected and noninspected classes could very likely be selected (namely, the inspection of 10-25 percent of all classes).

The benefit curves for the linear and MARS model (Fig. 4) both show a sharp decline immediately following their respective peaks at about n = 28. To explain this observation, recall that the model benefit is expressed relative to the performance of the size-ranking benchmark model. At n = 30, the size-ranking model selects a (midsized) class containing a large number of faults (19). At this point, the performance of the benchmark model greatly improves (also visible by the increase in fault-density for the size-ranking model at n-30 in Fig. 5)—hence, the relative decrease in the benefit of the MARS and Linear model at this value of n.

Varying the inspection effectiveness *e* amounts to multiplying the benefit curves by a constant factor. For instance, if we assume an inspection effectiveness only half as high (e = 0.4), the peak of the MARS curve will be at 37.9fc.

4.5 Application of Xpose Models to Jwriter

The straightforward way to apply the models built from Xpose to the Jwriter classes is to calculate the predicted fault-proneness for each class in Jwriter; if it is above a certain threshold, the class is considered fault-prone and undergoes inspection, otherwise not. Fig. 6 and Fig. 7 show the resulting correctness and completeness graphs for the linear and MARS model, respectively.

A good tradeoff between correctness and completeness can be achieved at the very low cutoff values of 0.22 (linear) and 0.06 (MARS), at which point we have completeness and correctness values of about 60 percent for both models. These values are slightly lower than what was found for the models in 10-cross-validation. This result suggests that we have to expect an additional deterioration of the fault-proneness models' accuracy when they are applied to new systems. Moreover, the predicted probabilities cannot be subjected to usual interpretations for the selection of cut-off values for class fault-proneness estimation. MARS suggests a reasonable cut-off value at 0.06, which is significantly

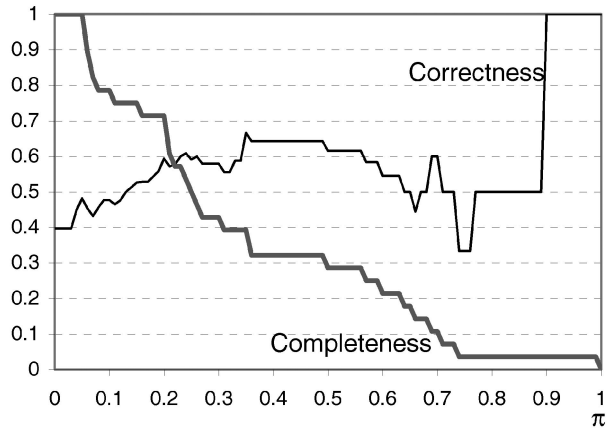


Fig. 6. Correctness/completeness for linear model applied to Jwriter.

different from the typical 0.5 threshold. However, as investigated below, cross-systems fault-proneness models can still present benefits if they are used in a different way.

The probabilities predicted by the Xpose model for Jwriter classes are very low by usual interpretation standards. Table 8 shows the distributions of predicted probabilities for the MARS and Linear model, for the 10-cross-validation within Xpose (shown in Section 4.4), and when applied to Jwriter. The MARS and Linear models show similar distributions for the 10-cross-validation within Xpose. When applied to Jwriter (Table 9), the predicted probabilities of the MARS model clearly exhibits a lower distribution, though the Linear model has a slightly higher distribution.

This latter observation is explained by the distributions across the two systems in terms of NIP, inheritance (DIT), and import/export coupling (OCMIC/OCMEC), the predictor variables of our model. For the MARS model, DIT and the coupling measures exhibit lower mean values in Jwriter than in Xpose (cf. Section 4.1 on descriptive statistics). It can therefore explain the low predicted probabilities. For the Linear model, NIP shows a much higher distribution in Xpose than in Jwriter, hence the higher predicted probabilities.

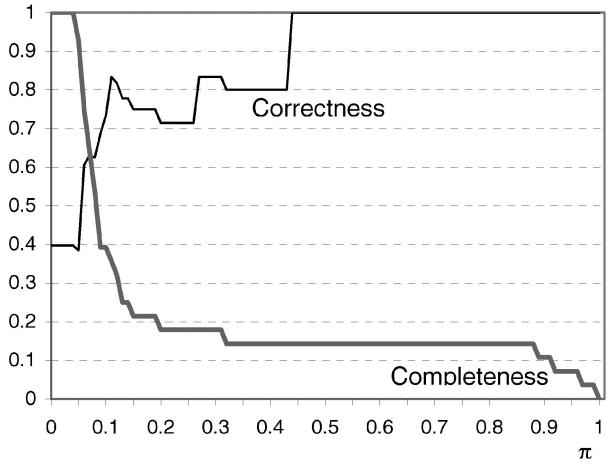


Fig. 7. Correctness/completeness for MARS model applied to Jwriter.

TABLE 8
Distribution of Predicted Probabilities
(10-Cross Validation within Xpose)

	Mean	StdDev	P75	Median	P25
MARS	0.210	0.259	0.189	0.079	0.061
Linear	0.211	0.263	0.285	0.086	0.043

Furthermore, the history of these systems can explain the difference between actual and predicted fault frequencies. Jwriter was developed first. The team was less experienced with Java, the Java libraries were less mature and contained more bugs, and the project manager was different. Even the design strategy (design by contract) and the coding standards were much stricter on Xpose. Therefore, relative to its structural properties and for the reasons explained above, Jwriter is much more fault-prone, as a system, than Xpose, and the model built from Xpose underestimates the fault-proneness of the Jwriter classes. In Xpose, 31 out of 144 classes actually are faulty (21 percent), in Jwriter, 27 out of 68 classes are faulty (39 percent). In Xpose, a threshold of 0.5 was good enough to classify roughly 20 percent of all classes as fault-prone. Though in Jwriter a higher percentage of classes must be selected to achieve acceptable correctness and completeness values, a lower threshold is required due to the shift to lower predicted probabilities.

From a more general standpoint, this suggests that, in the common situation where development practices are changing and teams are evolving and learning, an absolute, general interpretation of predicted probabilities will not be possible when they come from prediction models built on different systems. There are system factors that are likely to come into play, even in high maturity development environments. Unfortunately, this hinders us from using a predetermined cut-off value and predicted probabilities to classify classes according to their fault-proneness.

In order to still be able to use and assess the prediction models from Xpose, we apply them in a context where we rank the classes of Jwriter by their predicted fault-proneness. Then, we can assume that the N most fault-prone classes will undergo specific and thorough inspection. The question then is, how do we select N in practice? The results of the current study suggest that the selection of N cannot be based on historical data such as the typical proportion of faulty classes in past systems, as such a proportion can vary significantly from system to system. Typically, resources for inspections or testing are constrained, so N will in practice be driven by available time and budget. We can use the model to decide which classes

TABLE 9
Distribution of Predicted Probabilities
(Cross-System Application to Jwriter)

	Mean	StdDev	P75	Median	P25
MARS	0.133	0.214	0.086	0.059	0.054
Linear	0.257	0.248	0.335	0.190	0.070

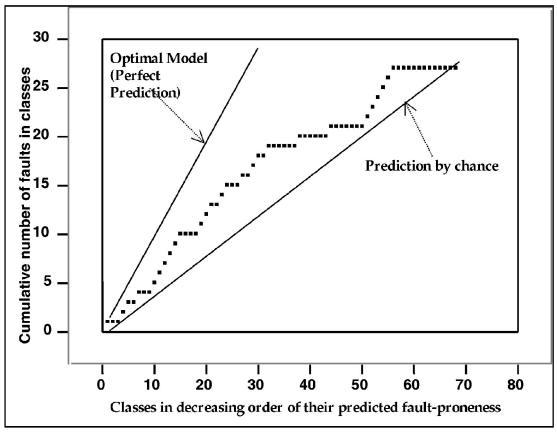


Fig. 8 Cumulative faults versus fault-proneness ranking for linear model applied to Jwriter.

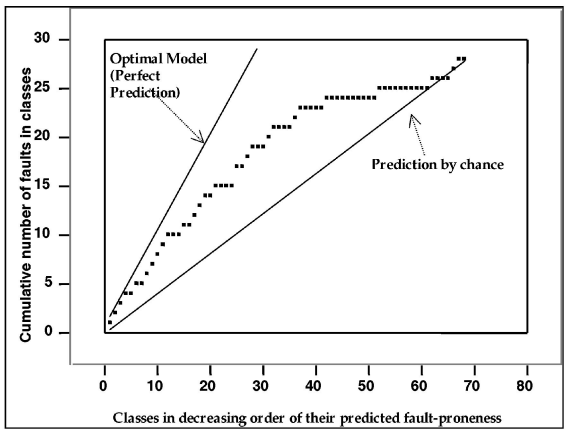


Fig. 9 Cumulative faults versus fault-proneness ranking for MARS model applied to Jwriter.

to inspect/test first, in order to find the largest number of defects as early as possible and continue until the available resources are depleted.

We will perform one ranking of the entire Jwriter class set according to their predicted fault proneness. Based on this ranking, we select the N classes with the highest predicted fault-proneness (leaving N variable) and investigate how the maximum number of faults that can be potentially detected increases with N? If the prediction model is useful, we would expect the number of faults found to increase sharply in the beginning and then reach a plateau.

Fig. 8 shows the cumulative number of faults versus class fault-proneness ranking, based on the predictions of the linear model for the Jwriter data. The lower straight line is the percentage of faults that we could expect when we randomly rank the classes (number of faults is assumed to be proportional to the number of classes). The upper straight line is what the ideal model would yield, if we were able to rank the 27 faulty classes, containing one fault each, as the 27 most fault-prone classes.

We can see in Fig. 8 a large hump among the least fault prone classes. This is caused by seven classes that are unexpectedly faulty, according to the linear model. In short, the curve does not have the shape we would expect and this model does not work very well.

Fig. 9 shows the graph that results when we rank the classes by predicted fault proneness of the MARS model. This figure fits better in our expectations than the linear model in the sense that the curve reaches a plateau after rank 35. There is a very small hump towards the end caused by two classes having one fault each, but no major abnormality. We can also see that the first 15 most fault-prone classes are almost exclusively faulty and close to the optimal line.

Fig. 10 shows the benefit curves for the linear and MARS model, respectively, again assuming inspection effectiveness of e = 0.8. And again, the general trend is that, both curves first show a rise until they reach their maximum of 14.4*fc* at n = 35 classes (linear) and 17.6*fc* at n = 31 classes (MARS), respectively, and then gradually descend until the benefit is back to zero when all classes are selected for inspection. The models reach their peaks rather late (more than half of the Jwriter classes for the linear model) and the maximum benefit values of 17.6fc/14.4fc is lower than for Xpose, too. Note, however, that Jwriter is a smaller system, an absolute comparison with the 10-cross validation results in Xpose is not possible. A theoretically optimal model that can perfectly rank the classes by the actual number of faults they contain reaches its maximum benefit of 32*fc* at n = 27 classes (still assuming the same inspection efficiency e = 0.8); that is less than twice that of the MARS model. The MARS model, however, clearly shows benefits for practically interesting numbers of classes inspected but, as would be expected, not on the extreme right of the graph when more than 60 classes are selected for inspections. The MARS model benefit curves clearly shows that clear benefits can be obtained in using such a model to select classes to undertake (additional) inspections or testing.

We further investigate the performance of the MARS model and try to explain why there are significant benefits in Fig. 10. The positive benefit of the model would suggest that the detection of faults increases faster than the number of methods inspected. We therefore normalize the number of methods and faults by their respective total to express them as a percentage. The result is shown in Fig. 11, which

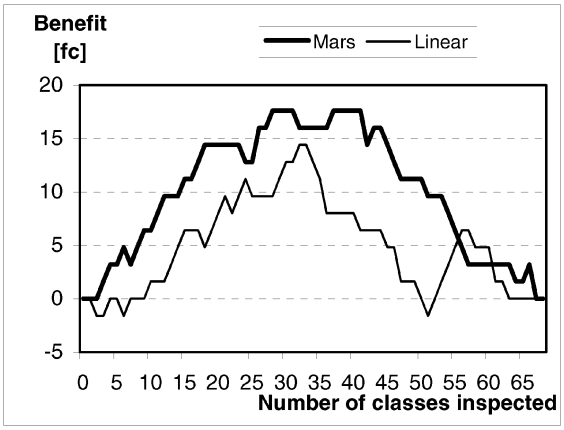


Fig. 10. Benefit curve of linear and MARS models for Jwriter (inspection effectiveness e = 80 percent).

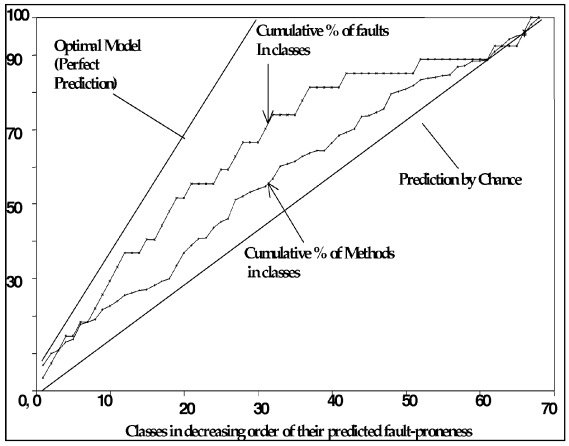


Fig. 11. Cumulative faults and class size versus predicted fault proneness of MARS model applied to Jwriter.

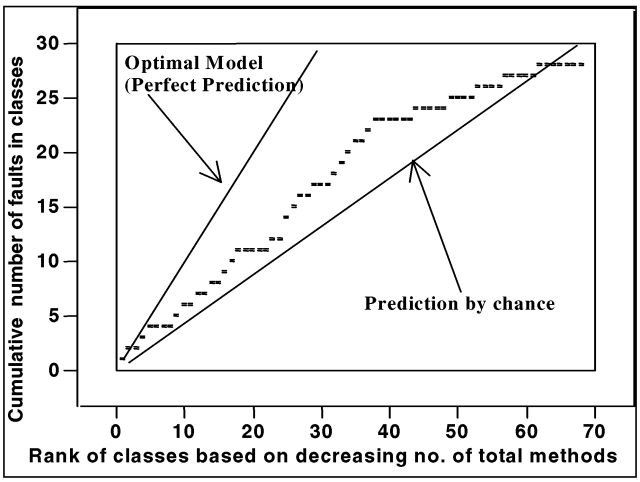


Fig. 12. Cumulative faults versus size ranking (based on number of methods in the class).

contains the cumulative number of faults (upper curve) versus cumulative number of methods (lower curve) on the same scatterplot.

We reach a maximum of 20 percent difference between the percentages of methods and faults, before starting a plateau. This demonstrates that the ranking is able to identify classes with higher fault-density and explains the benefit results presented earlier. Also, the lower curve approximates a linear growth, which indicates that the ranking performed by our model is significantly different from the ranking of the classes by their number of methods.

Furthermore, we rank the classes by the number of methods in decreasing order and look at the cumulative number of faults (Fig. 12). The curve appears close to the chance line for the 20 predicted most fault-prone classes. In other words, with this ranking, we only catch as many faults as could be expected by the number of selected classes alone. Thus, the class ranking produced by the MARS model built from the Xpose data set is not outperformed by a simple strategy that would be immediately applicable within Jwriter based on class size. This further justifies the use of a model such as the one we developed using the Xpose data. Note that, when using the number of data members or attributes as a size measure, results are very similar to what we observe for number of methods.

4.6 Using Fault-Proneness Models to Quantify Quality Improvement

We have seen above that Jwriter was far more fault-prone than what would be expected based on the fault-proneness model built using Xpose data. After investigation, as discussed above, our best explanations were learning effects and the differences in conditions under which the two systems were developed, e.g., practices, management. This is interesting as it shows how such fault-proneness models can be used in order to quantify the improvement in quality across systems after factoring out the structural properties that make such systems difficult to compare. For example, the number of faulty classes or faults can be predicted based on history (completed systems). Then, the relative difference between predicted and actual fault counts can be used to quantify improvement.

4.7 Class Ranking and Iterative Development

Though ranking classes allows us to use the predictions of the Xpose model in Jwriter, there are practical implications if this strategy is to be applied on projects. In order to perform such a ranking, we must first have all classes available for measurement. Object-oriented development, however, is typically iterative. To rank the classes by fault-proneness, we would have to hold inspections until the last iteration, when all classes are fully implemented and ready to be measured. In an iterative development, we could rank the classes within each new iteration and select a certain percentage of the classes with the highest ranks in that iteration for inspection. However, we may have iterations with relatively many high-risk classes, then we would select too few classes for inspection. For iterations with few high-risk components, too many classes would be selected for inspection. So, we would expect a higher misclassification rate this way, than when we select classes for inspections based on one ranking of the entire data set. The smaller the iterations, the stronger this effect should be.

5 Conclusions

One of the main objectives of this paper is to assess whether fault-proneness models, based on design measurement, are applicable and can be viable decision making tools when applied from one object-oriented system to the other, in a given environment. This question is of high practical interest as it determines whether such models are worth investigating. But, little is currently known on the subject. In this paper, we apply a fault-proneness model built on one system to another system. These two systems have been developed with a nearly identical development team (a different project manager), using a similar technology (OO analysis and design and Java) but different design strategies and coding standards. We believe that the context of our study represents realistic conditions that are often encountered: change in personnel, learning effects, and evolution of technology. In this context, we want to assess whether an organization can build viable models to focus verification and validation on fault-prone parts of systems.

TABLE 10
Definitions of Measures

Name	Definition
spa [2]	Static polymorphism in ancestors
dpa [2]	Dynamic polymorphism in ancestors
spd [2]	Static polymorphism in descendants
dpd [2]	Dynamic polymorphism in descendants
sp [2]	Static polymorphism in inheritance relations. sp=spa+spd
dp [2]	Dynamic polymorphism in inheritance relations. dp=dpa+dpd
nip [2]	Polymorphism in non-inheritance relations.
ovo [2]	Overloading in stand-alone classes
acaic [4]	These coupling measures are counts of interactions between classes. The measures distinguish the relationship between classes (inheritance or not), different types of interactions, and the locus of impact of the interaction. The acronyms for the measures indicates what interactions are counted: The first letter indicates the relationship (A: coupling to ancestor classes, D: Descendents, O: Others, i.e., no inheritance relationships). The next two letters indicate the type of interaction: CA: Class-Attribute interactions between classes c and d: c has an attribute of type d. CM: Class-Method interactions between classes c and d: c has a method with a parameter of type class d. The last two letters indicate the locus of impact: IC for import coupling, EC for export coupling.
acmic [4]	
dcaec [4]	
dcmec [4]	
ocaic [4]	
ocaec [4]	
ocmic [4]	
ocmec [4]	
dit [9]	The depth of the class in the inheritance tree.
noc [9]	The number of children of a class.
totatrib	The total number of attributes in the class
totprivatrib	The number of private attributes in the class
totprivmethod	The number of private methods in the class
totmethod	The total number of methods in the class

Our results suggest that applying the models across systems is far from straightforward. Even though the systems stem from the same development environment, the distributions of measures change and, more importantly, system factors (e.g., experience, design method) affect the applicability of fault detection predicted probabilities. It can be argued that, in a more homogeneous environment, this effect might not be as strongly present as in the current study. But, we doubt whether such environments exist or are in any case representative. It is likely, however, that the more stable the development practices and the better trained the developers, the more stable the fault-proneness models.

We then use the prediction model to rank classes of the second system according to their predicted fault-proneness (using the model built on the first system). When used in this manner, the model can in fact be helpful at focusing verification effort on faulty classes. Though predicted defect detection probabilities are clearly not realistic based on actual fault data, the fault-proneness class ranking is accurate. The model clearly performs better than chance and also outperforms a simple model based on class size, e.g., number of methods.

Another objective of the paper is to apply a novel exploratory analysis technique (Multivariate Adaptive Regression Splines: MARS) to the construction of fault-proneness models. Because we know little about what functional form to expect, such exploratory techniques that help finding optimal model specifications may be very useful. Our results support and suggest that the model generated by MARS outperforms a logistic regression model where the relationship between the logit and the independent variables is linear (log-linear model).

A third contribution of the paper is that we propose and apply a specific cost-benefit model for fault-proneness models. The goal is to assess the economic viability of such fault-proneness models as a function of a number of factors, e.g., defect detection effectiveness. Results suggest that, under realistic assumptions, fault-proneness models can provide substantial benefits, even when used across systems.

Future research needs to focus on collecting larger data sets, involving large numbers of systems from a same environment. It is crucial, in order to make the research on fault-prone models of practical relevance, that they be usable from project to project. Their applicability depends on their capability to predict accurately and precisely (fine grain predictions) where faults lie in new systems, based on the development experience accumulated on past systems.

APPENDIX

DEFINITION OF DESIGN MEASURES

In Table 10, we provide short definitions of all measures used in the paper. For detailed definitions, references to the source of each measure are provided.

ACKNOWLEDGMENTS

The authors would like to thank Dan Steinberg, from Salford Systems, for his advice on the use of the MARS tool (www.salford-systems.com). Also, they thank the software engineers of the Advanced Technoloy Solution Group from Oracle Brazil: Gonsalo Nunes, Caricio Afonso, Umberto Mancebo, Paulo Neto, and Aristeu Pires for building Xpose, Jwriter, and Jmetrics. Their crucial help was very much

appreciated. Amirul Khan should also be thanked for his support in generating some of the graphs and The Band for the Last Waltz. L. Briand was supported in part by the US National Science Foundation under grant number 0082574 and by the Canadian National Science and Engineering Research Council (NSERC).

Disclaimer: The fault data of the software systems studied in this paper corresponds to early versions of these systems and is not representative of the quality of final products. The system versions currently in production are robust and reliable.

REFERENCES

[1] A. Barbosa da Veiga, R. Farnese, and W. Melo, "JMetrics Java Metrics Extractor: An Overview," Univ. of Brasilia, Dept. of Computer Science, Under-Graduating Final Project, Brasilia, DF, Brazil, 1999, Also published as a Catholic Univ. of Brasilia, Master on Informatics, Software Quality Engineering Group, Technical Report, UCB-QSW-TR-1999/01.

[2] S. Benlarbi and W. Melo, "Polymorphism Measures for Early Risk Prediction," *Proc. 21st Int'l Conf. Software Eng., ICSE '99,* pp. 335-344, 1999.

[3] L. Briand, K. El Emam, O. Laitenberger, and T. Fussbroich, "Using Simulation to Build Inspection Efficiency Benchmarks for Development Projects," Technical Report ISERN-97-21, Fraunhofer Inst. for Experimental Software Eng., Germany, 1997, available at http://www.iese.fhg.de/ISERN.

[4] L. Briand, W. Melo, and P. Devanbu, "An Investigation into Coupling Measures for C++," *Proc. IEEE Int'l Conf. Software Eng., (ICSE),* 1997.

[5] L. Briand, J. Wüst, J. Daly, and V. Porter, "Exploring the Relationship between Design Measures and Software Quality in Object-Oriented Systems," *J. Systems and Software,* vol. 51, pp. 245-273, 2000.

[6] L. Briand, J. Wüst, H. Lounis, and S. Ikonomovski, "Investigating Quality Factors in Object-Oriented Designs: an Industrial Case Study," *Proc. 21st Int'l Conf. Software Eng., (ICSE '99),* pp. 345-354, 1999.

[7] L. Briand, S. Morasca, and V. Basili, "Property-Based Software Engineering Measurement," *IEEE Trans. Software Eng.,* vol. 22, no. 1, pp. 68-86, Jan. 1996.

[8] R.D. De Veaux, D.C. Psichogios, and L.H. Ungar, "A Comparison of Two Nonparametric Estimation Schemes: MARS and Neural Networks," *Computers Chemical Eng.,* vol. 17, no. 8, pp. 819-837, 1993.

[9] S.R. Chidamber and C.F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Trans. Software Eng.,* vol. 20, no. 6, pp. 476-493, June 1994.

[10] S. Chidamber, D. Darcy, and C. Kemerer, "Managerial use of Metrics for Object-Oriented Software: An Exploratory Analysis," *IEEE Trans. Software Eng.,* vol. 24, no. 8, pp. 629-639, Aug. 1998.

[11] G. Dunteman, *Principal Component Analysis.* SAGE Publications, 1989.

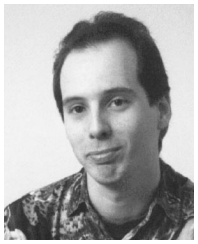
[12] J. Friedman, "Multivariate Adaptive Regression Splines," *The Annals of Statistics,* vol. 19, pp. 1-141, 1991.

[13] D.W. Hosmer and S. Lemeshow, *Applied Logistic Regression.* John Wiley & Sons, 1989.

[14] M. Stone, "Cross-Validatory Choice and Assessment of Statistical Predictions," J. Royal Stat. Soc., vol. 36, pp. 111-147, 1974.

Lionel C. Briand holds a PhD degree in computer science, with high honors, from the University of Paris, XI, France. He is currently heading the Software Quality Engineering Laboratory at the Department of Systems and Computer Engineering, Carleton University, Canada. Before that, he was the software quality engineering department head at the Fraunhofer Institute for Experimental Software Engineering, Germany. Lionel also worked as a research scientist for the Software Engineering Laboratory, a consortium of the NASA, Goddard Space Flight Center, CSC, and the University of Maryland. But, his first experiences were in the trenches, designing and developing large software systems, and he has, over the years, acted as a consultant to many industrial and government organizations. He has been on the program, steering, or organization committees of many international IEEE conferences. He also belongs to the steering committee of METRICS and is on the editorial board of *Empirical Software Engineering: An International Journal* and the *IEEE Transactions on Software Engineering*. His research interests include object-oriented analysis and design, inspections and testing in the context of object-oriented development, quality assurance and control, project planning and risk analysis, and technology evaluation. He is a member of the IEEE Computer Society.

Walcelio L. Melo holds the following degrees in computer science: the BSc degree in 1983 from University of Brasilia, Brazil, the MSc degree in 1988 from the Federal University of Rio Grande do Sul, Brazil, and the PhD degree in 1993 from University of Grenoble—Universite Joseph Fourrier, France. He is a technology manager at the Advanced Technology Solutions, Oracle Consulting, USA, where he is responsible for developing, teaching, and introducing Oracle's J2EE Component-Based Development Method for use by Oracle Consulting teams worldwide. He is also a software architect for J2EE-based solutions. Dr. Melo is a professor in the Computer Science Department at the Catolica University of Brasilia, Brazil, where he leads the software quality engineering group. Before joining Oracle in 1997, Dr. Melo had been the software engineering lead researcher at CRIM (Centre de Recherche Informatique de Montreal) and an adjunct professor in the School of Computer Science at McGill University, Montreal, Canada. From 1994 to 1989, Dr. Melo was faculty research associate at the Institute for Advanced Computer Studies at University of Maryland, College Park, and a member of the NASA Software Engineering Laboratory. From 1989 to 1993, he developed research related to software process programming languages in the Software Engineering Laboratory of Grenoble, France. He is (co-)author of more than 45 scientific papers published in workshops, conferences, journals, books, and encyclopaedias. His research interests are component-based software development methods, software quality tools, techniques and methods, object-oriented software measurement, software reuse, J2EE frameworks, and e-business applications.



Jürgen Wüst received the degree Diplom-Informatiker (MS) in computer science from the University of Kaiserslautern, Germany, in 1997. He is currently a researcher at the Fraunhofer Institute for Experimental Software Engineering (IESE) in Kaiserslautern, Germany. His research activities and industrial activities include software measurement, software product evaluation, and object-oriented development techniques.