ELSEVIER

The Journal of Systems and Software 56 (2001) 63–75

The Journal of Systems and Software

www.elsevier.com/locate/jss

The prediction of faulty classes using object-oriented design metrics

Khaled El Emam [a,*], Walcelio Melo [b], Javam C. Machado [c]

[a] National Research Council, Institute for Information Technology, Building M-50, Montreal Road, Ottawa, Ont., Canada K1A OR6
[b] Oracle Brazil, SCN Qd. 2 Bl. A, Ed. Corporate, S. 604, 70712-900 Brasilia, DF, Brazil
[c] Dep. Computacao, Univ. Federal do Ceara, Brazil

Received 8 November 1999; received in revised form 8 February 2000; accepted 29 April 2000

Abstract

Contemporary evidence suggests that most field faults in software applications are found in a small percentage of the software's components. This means that if these faulty software components can be detected early in the development project's life cycle, mitigating actions can be taken, such as a redesign. For object-oriented applications, prediction models using design metrics can be used to identify faulty classes early on. In this paper we report on a study that used object-oriented design metrics to construct such prediction models. The study used data collected from one version of a commercial Java application for constructing a prediction model. The model was then validated on a subsequent release of the same application. Our results indicate that the prediction model has a high accuracy. Furthermore, we found that an export coupling (EC) metric had the strongest association with fault-proneness, indicating a structural feature that may be symptomatic of a class with a high probability of latent faults.

Keywords: Object-oriented metrics; Software metrics; Software quality; Java metrics; Java quality

1. Introduction

Recent evidence indicates that most faults in software applications are found in only a few of a system's components (Moller and Paulish, 1993; Kaaniche and Kanoun, 1996; Ohlsson and Alberg, 1996; Fenton and Ohlsson, 2000). The early identification of these components allows an organization to take mitigating actions, such as focus defect detection activities on high risk components, for example, by optimally allocating testing resources (Harrison, 1988), or redesigning components that are likely to cause field failures.

In the realm of object-oriented systems, one approach to identify faulty classes early in development is to construct prediction models using object-oriented design metrics. Such models are developed using historical data, and can then be applied for identifying potentially faulty classes in future applications or future releases. The usage of design metrics allows the organization to take mitigating actions early and consequently avoid costly rework.

A considerable number of object-oriented metrics have been constructed in the past, for example, Li and Henry (1993), Abreu and Carapuca (1994), Chidamber and Kemerer (1994), Lorenz and Kidd (1994), Henderson-Sellers (1996), Briand et al. (1997), Benlarbi and Melo (1999), Tang et al. (1999) and Cartwright and Shepperd (2000). There have also been empirical studies validating object-oriented metrics and constructing prediction models that utilize them, such as Li and Henry (1993), Abreu and Melo (1996), Basili et al. (1996), Briand et al. (1997, 1998b, 2000), Binkley and Schach (1998), Chidamber et al. (1998), Harrison et al. (1998), Nesi and Querci (1998), Benlarbi and Melo (1999), Melo et al. (1999), Tang et al. (1999) and Cartwright and Shepperd (2000). However, most of these studies did not focus exclusively on metrics that can be collected during the design stage.

In this paper, we report on a study that was performed to construct a model to predict which classes in a future release of a commercial Java application will be faulty. In addition to identifying the faulty classes, the model can be applied to give an overall quality estimate (i.e., how many classes in the future release will likely

* Corresponding author.
E-mail addresses: khaled.el-emam@iit.nrc.ca (K. El Emam), wmelo@br.oracle.com (W. Melo), javam@lia.ufc.br (J.C. Machado).

0164-1212/01/$ - see front matter
PII: S0164-1212(00)00086-8

have a fault in them). The model uses only object-oriented design metrics. Our empirical validation results indicate that the model has high accuracy in identifying which classes will be faulty and in predicting the overall quality level.

Furthermore, our results show that the most useful predictors of class fault-proneness are a metric measuring inheritance depth and a metric measuring export coupling (EC), with EC having a dominating effect. These results are consistent with a previous study on a C++ telecommunications system, which found that EC was strongly associated with fault-proneness (El-Emam et al., 1999).

The paper is organized as follows. In Section 2, we provide an overview of the object-oriented metrics that we evaluate, and our hypotheses. In Section 3, we present our research method, and in Section 4 the detailed results, their implications, and limitations. We conclude the paper in Section 5 with a summary and directions for future research.

2. Background

2.1. Metrics studied

Our focus in this study are the two metrics sets defined by Chidamber and Kemerer (1994) and Briand et al. (1997). In combination these constitute 24 metrics. Of these, only a subset can be reliably collected during the design stage of a project. This subset includes inheritance and coupling metrics (and excludes cohesion and traditional complexity metrics).

At the design stage, it is common to have defined the classes, the inheritance hierarchy showing the parent–child relationships among the classes, identified the methods and their parameters for each class, and the attributes within each class and their types. Detailed information that is commonly available within the definition of a method, for example, those methods from other classes which are invoked, would not be available at design time. The cohesion metrics defined in these metrics suites were not collected for the same reason. This leaves us with a total of 10 design metrics that can be collected, two defined by Chidamber and Kemerer (1994), and eight by Briand et al. (1997).

The two Chidamber and Kemerer metrics are DIT and NOC. The depth of inheritance tree (Chidamber and Kemerer, 1994) metric is defined as the length of the longest path from the class to the root in the inheritance hierarchy. It is stated that as one goes further down the class hierarchy, the more complex a class becomes, and hence more fault-prone. The number of children inheritance metric (Chidamber and Kemerer, 1994) counts the number of classes which inherit from a particular class (i.e., the number of classes in the inheritance tree down from a class).

The Briand et al. coupling metrics are counts of interactions among classes (Briand et al., 1997). The metrics distinguish the types of relationships among the classes (i.e., friendship, inheritance, or another type of relationship), different types of interactions, and the locus of impact of the interaction. The acronyms for the metrics indicate what type of interactions are counted. We define below the acronyms and their meaning, and then summarize the design metrics.

- The first two letters indicate the relationship (A: coupling to ancestor classes; D: descendents; and O: other, i.e., none of the above). Although the Briand et al., metrics suite covers it, friendship is not applicable in our case since the language used for the system we analyze is Java.
- The next two letters indicate the type of interaction between classes c and d (CA: there is a class–attribute interaction between classes c and d if c has an attribute of type d; and CM: there is a class–method interaction between classes c and d if class c has a method with a parameter of type class d). There is a method–method interaction between classes c and d if c invokes a method of d, or if a method of class d is passed as a parameter to a method of class c. Method–method interactions are typically not available at design time, however. [1]
- The last two letters indicate the locus of impact (IC: import coupling; and EC). A class c exhibits IC if it is the using class (i.e., client in a client–server relationship), while it exhibits EC if it is the used class (i.e., the server in a client–server relationship).

Ancestor-based coupling metrics that we considered were ACAIC and ACMIC. The descendent-based coupling metrics that we considered were DCAEC and DCMEC. The remaining coupling metrics cover the four combinations of type of interaction and locus of impact: OCAIC, OCAEC, OCMIC, and OCMEC.

2.2. Hypotheses

An articulation of a theoretical basis for developing quantitative models relating object-oriented metrics and external quality metrics has been provided in Briand et al. (1998b), and is summarized in Fig. 1. This illustrates that we hypothesize a relationship between the object-oriented metrics and fault-proneness due to the effect on cognitive complexity.

[1] According to the mapping between the development phases of a generic object-oriented development process and coupling metrics in (Briand et al., 1999a), metrics that count method–method interactions can be approximated during low level design, but are only stable at the implementation stage. Methods that count class–attribute and class–method interactions can be approximated during the analysis phase, but are stable before implementation.

Fig. 1. Theoretical basis for the development of object-oriented product metrics.

There, it is hypothesized that the structural properties of a software component (such as its coupling) have an impact on its cognitive complexity. Cognitive complexity is defined as the mental burden of the individuals who have to deal with the component, for example, the developers, testers, inspectors, and maintainers. High cognitive complexity leads to a component exhibiting undesirable external qualities, such as increased fault-proneness and reduced maintainability.

Therefore, our general hypothesis is that the metrics that we validate, and that were described above, are positively associated with the fault-proneness of classes. This means that higher values on these metrics represent structural properties that increase the probability that a class will have a fault that causes a field failure.

3. Research method

3.1. Data source and measurement

The system that was analyzed is a commercial Java application. The application implements a word processor that can either be used stand-alone or embedded as a component within other larger applications. The word processor provides support for formatted text at the word, paragraph, and document levels, allows the definition of groupings of formatting elements as styles, supports RTF and HTML external file formats, allows spell checking of a range of words on demand, supports images embedded within documents or pointed to through links, and can interact with external databases.

We consider two versions of this application: versions 0.5 and 0.6. Version 0.5 was fielded and feedback was obtained from its users. This feedback included reports of failures and change requests for future enhancements. For our study, we only used the failure reports. To address the additional functionalities, version 0.6 involved an extensive redesign of the application, partially to avoid using an externally provided GUI library which had critical limitations.

Version 0.5 had a total of 69 classes. The design metrics were collected using an especially developed Java static analysis tool (Farnese et al., 1999). No Java inner classes were considered. For each class, the design metrics were collected. Also, for each class, it was known how many field failures were associated to a fault in that class. In total, 27 classes had faults. Version 0.6 had 42 classes. This version was also used in the field and based on failure reports, a subsequent version of the application was released. For version 0.6, 24 of the classes had faults in them that could be traced to field failures.

In addition to the inheritance and coupling metrics, we collected two measures of size: the number of attributes defined in a class and the number of methods. Both were defined as size measures for object-oriented classes of the past (Briand et al., 2000). Since our conclusions are not changed by the choice of size measure, we only consider the ATTS measure in this paper.

In our analysis, we used data from version 0.5 as a training data set, and data from version 0.6 as the test data set. As noted above, all faults were due to field failures occurring during actual usage. For each class, we characterized it as either faulty or not faulty (i.e., a binary characterization). A faulty class had at least one fault detected during field operation.

It has been argued that considering faults causing field failures is a more important question to address than faults found during testing (Binkley and Schach, 1998). In fact, it has been argued that it is the *ultimate* aim of quality modeling to identify post-release fault-proneness (Fenton and Neil, 1999a). In at least one study, it was found that pre-release fault-proneness is not a good surrogate measure for post-release fault-proneness, the reason posited being that pre-release fault-proneness is a function of testing effort (Fenton and Ohlsson, 2000).

3.2. Data analysis methods

Our data analysis approach consists of four steps: [2]

1. variable selection;
2. calibration;
3. prediction;
4. quality estimation.

We describe the objectives of each of these steps as well as the analysis techniques employed below.

3.2.1. Variable selection

During this step, the objective is to identify the subset of the object-oriented metrics that are related to fault-proneness. These variables are then used as the basis for further modeling steps explained below. We select variables that are individually associated with fault-proneness and that are orthogonal to each other.

[2] The research method presented here and that we use in our study is a refinement of the methodology used in a previous study (El-Emam et al., 1999).

Variable selection is achieved by first looking at the relationship between each metric and fault-proneness individually. [3] The statistical modeling technique that we use is logistic regression (henceforth LR). This is further explained in Appendix A, as well as some diagnostics that are applied to check the stability of the resulting models.

A recent study highlighted the potential confounding effect of class size (El-Emam et al., 2000) and demonstrated it on the Chidamber and Kemerer metrics (Chidamber and Kemerer, 1994) and a subset of the Lorenz and Kidd (Lorenz and Kidd, 1994) metrics. Specifically, this demonstration illustrated that without controlling the confounding effect of class size, one obtain results that are systematically optimistic. It is therefore necessary to control class size to get accurate results. Our approach accounts for the potential confounding effect of class size.

A measured confounding variable can be controlled through a regression adjustment (Breslow and Day, 1980; Schlesselman, 1982). A regression adjustment entails including the confounder as another independent variable in a regression model. Our LR model is therefore

$$\pi = \frac{1}{1 + e^{-(\beta_0+\beta_1 x_1+\beta_2 x_2)}}, \quad (1)$$

where π is the probability of a class having a fault, x_1 the object-oriented metric being validated, and x_2 is the size, measured as ATTS. We construct such a model for each object-oriented metric being validated.

It should be noted that the object-oriented metric and the size confounding variable are not treated symmetrically in this model. Specifically, the size confounder (i.e., variable x_2) should always be included in the model, irrespective of its statistical significance (Breslow and Day, 1980). If inclusion of the size confounder does not affect the parameter estimate for the object-oriented metric (i.e., the β_1 parameter of variable x_1), then still we get a valid estimate of the impact of the metric on fault-proneness. The statistical significance of the parameter estimate for the object-oriented metric, however, is interpreted directly since this is how we test our hypothesis.

[3] We do not employ automatic selection procedures since they are known to be unstable. It is more common to use a forward selection technique rather than backward selection. The reason being that backward selection starts off with all of the variables and then eliminates variables incrementally. The number of observations is usually not large enough to justify constructing a model with all variables included. On the other hand, a Monte Carlo simulation of forward selection indicated that in the presence of collinearity among the independent variables, the proportion of 'noise' variables that are selected can reach as high as 74% (Derksen and Keselman, 1992). It is clear that many object-oriented metrics are inter-correlated (Briand et al., 1998b, 2000).

In constructing our models, we follow previous literature in that we do not present results for interactions nor higher order terms, for example, see Basili et al. (1996), Briand et al. (1997, 1998a,b, 2000), Benlarbi and Melo (1999), Tang et al. (1999) and El-Emam et al. (2000). This is to some extent justifiable given that there is no clear theoretical basis to assume any of the above as yet.

Building LR models as in Eq. (1) for each metric will result in a subset of the metrics that have statistically significant parameters being retained. However, it is likely that some of these retained metrics are associated with each other. We use the robust Spearman correlation coefficient for investigating such associations (Sheskin, 1997). [4] For metrics that are strongly associated with each other, we select only one of them for further consideration. The selected metric would be the one that has the largest change in odds ratio (i.e., the largest impact on fault-proneness).

The consequence of this step is a small number of metrics remaining that are both associated with fault-proneness and that are orthogonal to each other.

3.2.2. Calibration

After identifying a subset of metrics that are associated with fault-proneness and that are orthogonal, we construct a multivariate model that combines all of these metrics. The construction of such a model follows the same procedure described in Appendix A, except that more variables will be incorporated.

The multivariate model can be practically applied in identifying which classes are likely to contain a fault (prediction step) and to estimate the overall fault content of a system (quality estimation step). However, first it must be calibrated.

During calibration, we identify the *optimal operating point* for the model. This operating point maximizes the prediction accuracy. Recall that a LR model makes predictions as a probability rather than a binary value (i.e., if we use a LR model to make a prediction, the predicted value is the probability of the occurrence of a fault). It is common to choose a cutoff value for this predicted probability. For instance, if the predicted probability is greater than 0.5, then the class is predicted to be of high risk. Instead of using such a generic cutoff, it is possible to select an optimal cutoff. In Appendix C, we describe the use of receiver operating characteristic (ROC) curves for identifying the optimal cutoff. This is achieved by computing accuracy measures for all

[4] We do not employ a multivariate technique such as a principal components analysis because usually a small number of metrics are retained and a simple correlation matrix makes clear the associations among them. If many metrics are retained, then it would be more appropriate to use a data reduction technique such as principal components analysis.

possible cutoff points and selecting the best one. Another useful measure from a ROC curve is the area under the curve (AUC). The AUC value characterizes the accuracy of the model across all possible cutoff values.

During calibration, only the training data set is used (version 0.5). It has been recommended that in studies where sample sizes are less than 100, as in our case, a leave-one-out approach provides reliable estimates of accuracy (Weiss and Kulikowski, 1991). Therefore, we use this approach during calibration.

3.2.3. Prediction

The calibrated model (with its optimal cutoff point already computed) can be used to predict which classes have a fault in the test data set (version 0.6). To evaluate the binary prediction accuracy of the calibrated model, we use the J coefficient. This is described further in Appendix B. The prediction results on a test data set provide a realistic assessment of how accurate this multivariate model will perform in actual projects.

3.2.4. Quality estimation

The calibrated model can also be used to estimate the overall quality of an object-oriented application. Here, quality is defined as the proportion of classes that are faulty. Appendix D explains how quality estimates may be calculated. We use the techniques in Appendix D to estimate the quality of the test data set (version 0.6), and then compare this estimate to the actual quality.

3.2.5. Summary

An overall summary of each of the four steps in our research method is provided in Table 1, including a description of the outputs.

4. Results

4.1. Descriptive statistics

The descriptive statistics for the object-oriented metrics and the size metric for the train and test data sets are presented in Tables 2 and 3, respectively. The tables show the mean, standard deviation, median, interquartile range (IQR), and the number of observations that are not equal to zero. In general, there are strong similarities between the two data sets. It is noticable that the OCAEC metric has a rather large standard deviation compared to the other metrics.

Variables ACAIC, ACMIC, DCAEC, and DCMEC have less than six observations that are non-zero on the training data set. Therefore, they were excluded from further analysis. This is the approach followed in previous studies (Briand et al., 2000; El-Emam et al., 2000).

4.2. Variable selection results

Table 4 contains the results of the LR models after controlling class size. The table only shows the parameters of the object-oriented metrics since this is what we draw conclusions from. This analysis was performed only on the training data set.

Only four metrics out of the six had a significant association with fault-proneness: OCAEC, OCMEC, OCMIC, and DIT. The change in odds ratio for the OCAEC metric is quite large. The change in odds ratio (see Appendix A) is a function of the standard deviation of the metric. OCAEC had a rather large standard deviation. This was due to a handful of observations that were extreme and hence inflated the variation. In general, the standard deviation of this metric was sensitive to a minority of observations (i.e., removing them would

Table 1
A summary of the steps in our research method

Step	Procedure	Outcome
Variable selection	For each metric, construct an LR model with two independent variables: a size measure and the object-oriented metric Retain the metrics that have a statistically significant parameter Look at the metrics' inter-correlations and select a subset that is orthogonal	A subset of the original metrics that are associated with fault-proneness and that are orthogonal
Calibration	Construct an LR model with size and all of the retained metrics from the above step Using a leave-one-out approach, construct the ROC curve and determine the optimal operating point	An LR model with the size metric and the retained metrics from the previous step as independent variables Identification of the optimal cutoff value
Prediction	Using the model from the above step, predict the fault status for the test data set at the optimal operating point Estimate the prediction accuracy	An estimate of the prediction accuracy using the J coefficient
Quality estimation	Estimate the proportion of faulty classes on the test data set (the quality estimate) using the calibrated model and evaluate its accuracy	The quality estimate on the test data set and its accuracy

Table 2
Descriptive statistics for all of the object-oriented metrics on the training data set (version 0.5)

	Mean	Median	Standard deviation	IQR	NOBS $\neq 0$
ACAIC	0.043	0	0.205	0	3
ACMIC	0.173	0	0.839	0	3
DCAEC	0	0	0	0	0
DCMEC	0	0	0	0	0
OCAIC	3.144	1	4.512	4	40
OCAEC	2.695	1	6.181	2	47
OCMIC	1.681	1	2.933	2	39
OCMEC	2.072	1	5.140	2	40
DIT	1.217	1	1.069	2	47
NOC	0.188	0	0.624	0	7
ATTS	9.159	7	10.745	13	52

Table 3
Descriptive statistics for all of the object-oriented metrics on the test data set (version 0.6)

	Mean	Median	Standard deviation	IQR	NOBS $\neq 0$
ACAIC	0.047	0	0.215	0	2
ACMIC	0.214	0	0.976	0	2
DCAEC	0	0	0	0	0
DCMEC	0	0	0	0	0
OCAIC	3.809	2	4.880	6	25
OCAEC	3.238	1	5.917	1	35
OCMIC	2.119	1	3.394	3	26
OCMEC	2.595	1	5.401	2	29
DIT	1.428	1	1.085	1	32
NOC	0.142	0	0.472	0	4
ATTS	12.642	10.5	12.270	11	37

Table 4
Results of the validation, including the LR parameters and diagnostics[a]

Metric	G (p-value)	R^2	η	β_1 coefficient (S.E.)	p-value	$\Delta\Psi$
OCAEC	42.45 (<0.0001)	0.49	5.80	2.5766 (0.6846)	**0.0001**	123
OCAIC	3.91 (0.1415)	0.0423	3.697	0.0541 (0.0801)	0.2498	1.276
OCMEC	29.57 (<0.0001)	0.33	3.64	1.2144 (0.3455)	**0.0002**	12.2
OCMIC	11.52 (0.0032)	0.124	2.61	0.3494 (0.1500)	**0.0099**	2.78
DIT	12.68 (0.0018)	0.137	3.869	0.7681 (0.2712)	**0.0023**	2.273
NOC	8.96 (0.0113)	0.099	2.239	−7.02 (25.07)	0.389	0.016

[a] The G coefficient tests the hypothesis if any of the regression parameters is different from zero. The R^2 is a goodness-of-fit measure, η the condition number to determine the extent of collinearity, β_1 the estimated coefficient for the object-oriented metric, (S.E.) the standard error of the coefficient estimate, the p-value the one-sided probability of getting an coefficient as extreme under the null hypothesis, and $\Delta\Psi$ is the change in odds ratio.

have non-negligible impacts on the standard deviation). Therefore, the estimate of the change in odds ratio for OCAEC is quite unstable.

Out of the remaining significant metrics, OCMEC had the largest change in odds ratio, indicating its strong impact on fault-proneness.

The Spearman correlations among the significant metrics are shown in Table 5. It is seen that all the coupling metrics are strongly associated with each other, with the strongest association between OCMEC and OCAEC. This is not surprising given that they are both EC metrics. The DIT metric has much weaker association with the coupling metrics.

We therefore select OCMEC and DIT for further investigation. We exclude OCAEC due to its unstable standard deviation (i.e., its usage would give us unstable results), [5] and exclude OCMIC since its change in odds ratio in Table 4 is smaller than that of OCMEC.

4.3. Calibration

At this juncture, we have identified two metrics that have value additional to class size, and that carry complementary information about the impact of class structure on fault-proneness. We now construct a multivariate LR model using these metrics.

[5] We also constructed a prediction model using the OCAEC metric instead of the OCMEC metric. The prediction accuracy was almost the same as that of the model using the OCMEC metric, but the model was, as expected, unstable.

Table 5
Spearman correlations among the metrics that were found to be associated with fault-proneness after controlling for size

	OCAEC	OCMEC	OCMIC
OCMEC	0.81		
	(<0.0001)		
OCMIC	0.50	0.71	
	(<0.0001)	**(<0.0001)**	
DIT	0.30	0.15	0.09
	(0.0015)	(0.1067)	(0.3480)

Table 6
LR results for the best model[a]

G	R^2		η	
38.98; $p < 0.0001$	0.4355		6.1737	
	Intercept	ATTS	OCMEC	DIT
β coefficient	−3.9735	0.0464	1.4719	1.0678
p-value	0.0001	0.1141	0.0004	0.0039
$\Delta\Psi$		1.603	20.746	3.156

[a] The G coefficient tests the hypothesis if any of the regression parameters is different from zero. The R^2 is a goodness-of-fit measure, η the condition number to determine the extent of collinearity, β the estimated coefficient for the variable, the p-value the one-sided probability of getting an coefficient as extreme under the null hypothesis, and $\Delta\Psi$ is the change in odds ratio.

Table 6 shows the multivariate LR model incorporating the two metrics and size. It will be noted that the effect of the size measure, ATTS, is not significant. However, as noted earlier, we keep it in the model to ensure that the parameter estimates for the remaining variables are accurate.

The ROC curve for the model in Table 6 is shown in Fig. 2. This curve was constructed using a leave-one-out approach. The area under this curve is 0.87, which is rather high in comparison to a previous study with object-oriented metrics (El-Emam et al., 1999). The optimal cutoff value for this LR model is 0.33, which is quite different from the traditonally utilized cutoff values (which are typically $\geqslant 0.5$). The sensitivity and specificity of this model at the optimal operating point are estimated to be 0.81 and 0.83, respectively. Sensitivity is the proportion of high risk classes that are correctly classified as high risk. Specificity is the proportion of low risk classes that are correctly classified as low risk.

Now that we have calibrated the model (i.e., determined its optimal operating characteristics), it can be applied to predict which classes in the test data set (version 0.6) are going to be faulty.

4.4. Prediction

We used the model in Table 6 to predict which classes in the test data set will have a fault. Note that since we only use design metrics, this prediction can be performed at design time.

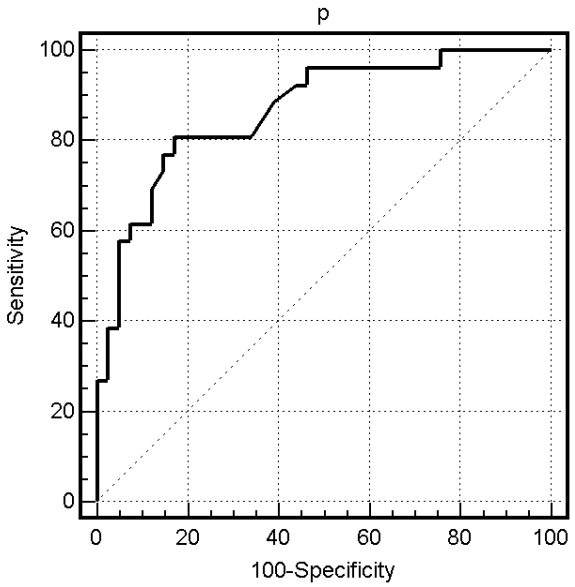

Fig. 2. ROC curve for the calibration model. The area under the ROC curve is 0.87. The optimal operating point is at a cutoff value of 0.33, with a sensitivity of 0.81 and a specificity of 0.83.

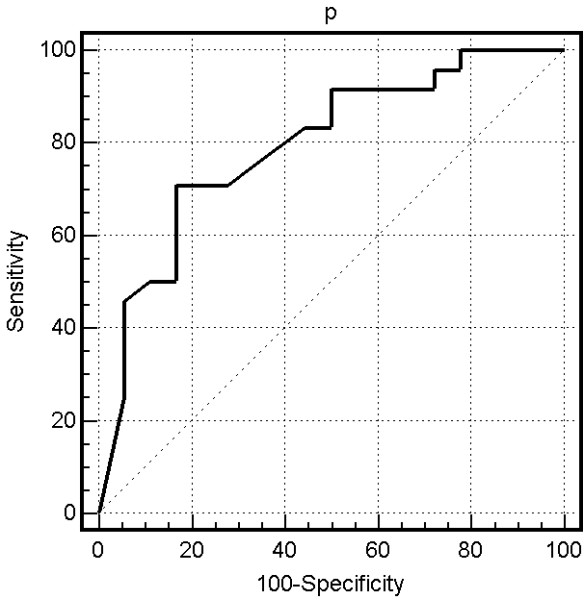

Fig. 3. ROC curve for the test set. The area under the ROC curve is 0.78.

The ROC curve for the predictions on the test data set is shown in Fig. 3. This curve has an area of 0.78, which is not far off from the leave-one-out estimate, and is very good. This indicates that this model would have a good prediction accuracy. The predictions at the optimal cutoff of 0.33 are shown in Table 7. The J value for this model is 0.49, which can be considered to be high.

In practice, it is also informative to calculate the proportion correct accuracy for a prediction model *when*

Table 7
Prediction results in a confusion matrix for the optimal cutoff. The J value for this is 0.49

		Predicted fault status		
		Not faulty	Faulty	
Real fault status	Not faulty	14	4	18
	Faulty	7	17	24
		21	21	42

used in a particular context. The following equation formulates the relationship between sensitivity, specificity, prevalence of faulty classes, and proportion correct accuracy:

$$A = (s \times h) + (f \times (1 - h)), \tag{2}$$

where A is the proportion correct accuracy, h the proportion of faulty classes, s sensitivity, and f is the specificity. For example, if our prediction model is to be used on an actual project where only 10% of the classes are expected to have faults in them, then the proportion correct accuracy would be approximately 0.77.

4.5. Quality estimation

In version 0.6 of the system, the prevalence of classes that were faulty was 0.57, as can be seen in Table 7. The naïve estimate of the prevalence of faulty components is 0.50 (21/42). This is smaller than the actual prevalance. By using Eq. (A.7) in Appendix D and the estimated sensitivity and specificity, we can correct this estimate to obtain an estimate of 0.52. The corrected estimate is closer to the actual value.

In this particular example the LR model that we constructed was rather accurate on the test data set (version 0.6). Therefore, the naïve and corrected estimates were not far apart. However, it is clear that the corrected estimate is an improvement over the naïve estimate.

In general, an LR model constructed from the training data set can provide rather good quality estimates using the formula provided in Appendix D. This is a considerable advantage as the LR model is usable at the design stage of a project.

4.6. Discussion of results

Our results indicate that, in addition to a simple size metric, the OCMEC and DIT metrics can be useful indicators of fault-prone classes. They are both associated with fault-proneness after controlling size, and when combined in a multivariate model can provide accurate predictions of which classes are likely to contain a fault. The added advantage of both of the above metrics is that they can all be collected at design time, allowing early management of software quality.

We have also added to the methodology initially presented in El-Emam et al. (1999, 2000) by providing a correct technique for estimating the quality of an object-oriented system using design metrics. Our results indicate that such an estimate is rather accurate.

We found that an inheritance metric and an EC metric are both associated with fault-proneness. We discuss this finding and its implications on the design of object-oriented systems.

4.6.1. Relationship between inheritance depth and fault-proneness

Previous studies suggest that depth of inheritance has an impact on the understandability of object-oriented applications, and hence would be expected to have a detrimental influence on fault-proneness (Cartwright, 1998; Unger and Prechelt, 1998). However, this conclusion is equivocal as a contradictory result was found in Daly et al. (1996). Some authors (Unger and Prechelt, 1998) contend that inheritance depth per se is not the factor that affects understandability, but the number of methods that have to be traced. Further support for this argument can be found in a recent study of a telecommunications C++ system (El-Emam et al., 1999), whereby an ancestor-based IC metric was found to be associated with fault-proneness, whereas depth of inheritance tree was not.

The fact that we found that the depth of inheritance tree to be associated with fault-proneness may be, according to previous literature reviewed above, due to two reasons:

- a badly designed inheritance hierarchy such that inherited classes are inconsistent with their superclasses;
- the DIT metric is confounded with method invocations up the inheritance hierarchy, and in fact inheritance depth is not the cause of fault-proneness.

Further focused studies are required to determine which of the above explanations are closer to reality.

4.6.2. Relationship between export coupling and fault-proneness

The effect of the EC metric in our results was much stronger than that of DIT. The EC metric that we selected considered class–method interactions, although we did find that it is strongly associated with class–attribute interactions as well. Therefore, a priori it seems reasonable to talk about EC in general since these two types of interactions tend to co-occur.

A previous study of a C++ telecommunications system (El-Emam et al., 1999) also noted that EC is associated with fault-proneness. While two studies do not make a trend, there appears to be some initial consistency in findings.

One can make two hypotheses about why EC is strongly associated with fault-proneness:

- Classes that have a high EC are used more frequently than any other classes. This means that in operational systems their methods are invoked most frequently. Hence, even if all classes in a system have exactly the same number of faults in them, more faults will be discovered in those with high EC simply because they are exercised more. This hypothesis suggests that cognitive complexity is not the causal mechanism that would explain our findings.
- A client of a class d makes assumptions about d's behavior. A class with more EC has more clients and therefore more assumptions are made about its behavior due to the existence of more clients. Since the union of these assumptions can be quite large, it is more likely that this class d will have a subtle fault that violates this large assumption space, compared to other classes with a smaller set of assumptions made about their behavior.

If either of the above hypotheses is true, it remains that they are not specific to object-oriented programs. The same phenomena can occur in traditional applications that followed structured design methodology.

4.6.3. Summary

Our results have highlighted certain structural properties of object-oriented systems that are problematic. While we cannot provide exact causal explanations for the findings that inheritance depth and EC are strongly associated with fault-proneness, we have posited some precise hypotheses that can be tested through further empirical enquiry.

4.7. Limitations

This study has a number of limitations which should be made clear in the interpretation of our results. These limitations are not unique to our study, but are characteristics of most of the product metrics validation literature. However, it is of value to repeat them here.

This study did not account for the severity of faults. Lack of accounting of fault severity was one of the criticisms of the quality modeling literature in Fenton and Neil (1999b). In general, unless the organization has a reliable data collection program in place where severity is assigned, it is difficult to retrospectively obtain this data. Therefore, the prediction models developed here can be used to identify classes that are prone to have faults that cause any type of failure.

It is also important to note that our conclusions are pertinent only to the fault-proneness dependent variable, albeit this seems to be one of the more popular dependent variables in validation studies. We do not make claims about the validity (or otherwise) of the studied object-oriented metrics when the external attributes of interest are, for example, maintainability (say measured as effort to make a change) or reliability (say measured as mean time between failures).

It is unwise to draw broad conclusions from the results of a single study. Our results indicate that two structural properties of object-oriented metrics are associated with fault-proneness. While these results provide guidance for future research on the impact of coupling and inheritance on fault-proneness, they should not be interpreted as the last word on the subject. Further validations with different industrial systems are necessary so that we can accumulate knowledge and draw stronger conclusions, and perhaps explain the causal mechanisms that are operating.

5. Conclusions

In this paper, we performed a validation of object-oriented design metrics on a commercial Java system. The objective of the validation was to determine which of these metrics were associated with fault-proneness. This would allow the prediction of the classes that will be fault-prone and estimating the overall quality of future systems. Our results indicate that an inheritance and an EC metric were strongly associated with fault-proneness. Furthermore, the prediction model that we constructed with these two metrics has good accuracy, and the method we employed for predicting the quality of a future system using design metrics also has a good accuracy.

While this is a single study, it does suggest that perhaps there are a small number of metrics that are strongly associated with fault-proneness, and that good prediction accuracy and quality estimation accuracy can be attained. This conclusion is encouraging from a practical standpoint, and hence urges further studies to corroborate (or otherwise) our findings and conclusions.

Appendix A. Overview of logistic regression

In this appendix, we provide an overview of LR and the various diagnostics and tests that were applied during the construction of our models.

Binary LR is used to construct models when the dependent variable is binary, as in our case. The general form of an LR model is

$$\pi = \frac{1}{1 + e^{-\left(\beta_0 + \sum_{i=1}^{k} \beta_i x_i\right)}}, \tag{A.1}$$

where π is the probability of a class having a fault, and the x_i's are the independent variables. The β parameters are estimated through the maximization of a log-likelihood (Hosmer and Lemeshow, 1989).

A.1. Magnitude of association

The magnitude of an association can be expressed in terms of the change in odds ratio as the x_1 variable (i.e., object-oriented metric) changes by one standard deviation, and is denoted by $\Delta\Psi$, and is given by [6]:

$$\Delta\Psi = \frac{\Psi(x_1 + \sigma)}{\Psi(x_1)} = e^{\beta_1 \sigma}, \quad \text{(A.2)}$$

where σ is the standard deviation of the x_1 variable. When computing the change in odds ratio, it is also necessary to inspect each independent variable for outliers since a single extreme observation can increase the standard deviation and hence inflate the change in odds ratio. This only involves the detection of univariate outliers, and therefore can be performed by inspecting the variable distribution.

A.2. Collinearity

Since we control through regression adjustment for the size confounder, careful attention should be paid to the detection and mitigation of potential collinearity. Strong collinearity can cause inflated standard errors for the estimated regression parameters. We use the condition number (denoted by η) as described by Belsley et al. (1980). Further discussion of collinearity diagnostics in the context of validating object-oriented metrics can be found in (El-Emam et al., 2000). Belsley et al. suggest that a condition number greater than 30 indicates mild to severe collinearity.

A.3. Hypothesis testing

The next task in evaluating the LR model is to determine whether any of the regression parameters are different from zero, i.e., test H_0: $\beta_1 = \beta_2 = \cdots = \beta_k = 0$. This can be achieved by using the likelihood ratio G statistic (Hosmer and Lemeshow, 1989). If the likelihood ratio test is found to be significant at $\alpha = 0.05$, then we can proceed to test each of the individual parameters. This is done using a Wald statistic, $\hat{\beta}_j/(\text{S.E.}(\hat{\beta}_j))$, which follows a standard normal distribution. These tests were performed at an one-tailed alpha level of 0.05. We used one-tailed test since all of our alternative hypotheses are directional: there is a positive association between the metric and fault-proneness.

[6] In some instances the change in odds ratio is defined as: $\Delta\Psi = \Psi(x_1 + 1)/\Psi(x_1)$. This gives the change in odds when the object-oriented metric increases by one unit. Since different metrics utilize different units, this approach precludes the comparison of the change in odds ratio value. By using an increment of one standard deviation rather than one unit, as we did, we can compare the relative magnitudes of the effects of different object-oriented metrics since the same unit is used.

For each object-oriented metric, if the parameter of the object-oriented metric is statistically significant, then this metric is considered further. Statistical significance indicates that the metric is associated with fault-proneness. If the parameter for the object-oriented metric is not statistically significant then that metric is dropped from further consideration.

A.4. Goodness of fit

In previous studies another descriptive statistic has been used, namely an R^2 statistic that is analogous to the multiple coefficient of determination in least-squares regression (Briand et al., 1998b, 2000). We use a corrected version of this suggested by Hosmer and Lemeshow (1989). It should be recalled that this descriptive statistic will in general have low values compared to what one is accustomed with in a least-squares regression. In our study, we will use the corrected R^2 statistic as an indicator of the quality of the LR model.

A.5. Influence analysis

Influence analysis is performed to identify influential observations (i.e., ones that have a large influence on the LR model). Pergibon (1981) has defined the $\Delta\beta$ diagnostic to identify influential groups in LR. The $\Delta\beta$ diagnostic is a standardized distance between the parameter estimates when a group of observations with the same x_i values is included and when they are not included in the model. We use the $\Delta\beta$ diagnostic in our study to identify influential groups of observations. For groups that are deemed influential, we investigate this to determine if we can identify substantive reasons for them being over influential. In all cases in our study where a large $\Delta\beta$ was detected, its removal, while affecting the estimated coefficients, did not alter our conclusions.

Appendix B. Measures of prediction accuracy

It is common that prediction models using object-oriented metrics are cast as a binary classification problem. We first present some notation before discussing the binary accuracy measure that we use.

Table 8 shows the notation in obtained frequencies when a binary classifier is used to predict the class of unseen observations in a confusion matrix. We consider a class as being high risk if it has a fault and low risk if it does not have a fault.

Such a confusion matrix also appears frequently in the medical sciences in the context of evaluating diagnostic tests, for example, see Gordis (1996). Two important parameters have been defined on such a matrix

Table 8
Notation for a confusion matrix

		Predicted Risk		
		Low	High	
Real Risk	Low	n_{11}	n_{12}	N_{1+}
	High	n_{21}	n_{22}	N_{2+}
		N_{+1}	N_{+2}	N

that will be used for our exposition, namely sensitivity and specificity.

The *sensitivity* of a binary classifier is defined as

$$s = \frac{n_{22}}{n_{21} + n_{22}}. \quad (A.3)$$

This is the proportion of high risk classes that have been correctly classified as high risk classes.

The *specificity* of a binary classifier is defined as

$$f = \frac{n_{11}}{n_{11} + n_{12}}. \quad (A.4)$$

This is the proportion of low risk classes that have been correctly classified as low risk classes.

Ideally, both the sensitivity and specificity should be high. A low specificity means that there are many low risk classes that are classified as high risk. Therefore, the organization would be wasting resources reinspecting or focusing additional testing effort on these classes. A low sensitivity means that there are many high risk classes that are classified as low risk. Therefore, the organization would be passing high risk classes to subsequent phases or delivering them to the customer. In both the cases, the consequences may be expensive field failures or costly defect correction later in the life cycle.

The J coefficient of Youden (1950) was suggested in El-Emam et al. (2001) as an appropriate measure of accuracy for binary classifiers in software engineering. This is defined as

$$J = s + f - 1. \quad (A.5)$$

This coefficient has a number of desirable properties. First, it is prevalence independent (i.e., it does not depend on the proportion of faulty classes in the data set). For example, if our classifier has specificity and sensitivity equal to $f = 0.9$ and $s = 0.7$, then its J value is 0.6 irrespective of prevalence. The J coefficient can vary from −1 to +1, with +1 being perfect accuracy and −1 being the worst accuracy. A guessing classifier (i.e., one that guesses high/low risk with a probability of 0.5) would have a J value of 0. Therefore, J values greater than zero indicate that the classifier is performing better than would be expected from a guessing classifier.

Appendix C. Overview of ROC curves

Previous studies have used a plethora of LR cutoff values to decide what is high risk or low risk, for example, 0.5 (Basili et al., 1996; Morasca and Ruhe, 1997; Briand et al., 1998b, 1999b), 0.6 (Briand et al., 1998b), 0.65 (Briand et al., 1998b, 2000), 0.66 (Briand et al., 1998a), 0.7 (Briand et al., 2000), and 0.75 (Briand et al., 2000). In fact, and as noted by some authors (Morasca and Ruhe, 1997), the choice of cutoff value is arbitrary, and one can obtain different results by selecting different cutoff values, for example, see El-Emam et al. (1999).

A general solution to the arbitrary thresholds problem mentioned above is ROC curves (Metz, 1978). One selects many cutoff points, from 0 to 1 in our case, and calculates the sensitivity and specificity for each cutoff value, and plots sensitivity against one-specificity as shown in Fig. 4. Such a curve describes the compromises that can be made between sensitivity and specificity as the cutoff value is changed. One advantage of expressing the accuracy of our prediction model (or for that matter any diagnostic test) as an ROC curve is that it is independent of the cutoff value, and therefore no arbitrary decisions need be made as to where to cut off the predicted probability to decide that a class is high risk (Zweig and Campbell, 1993). Furthermore, using an ROC curve, one can easily determine the optimal operating point, and hence obtain an optimal cutoff value for an LR model.

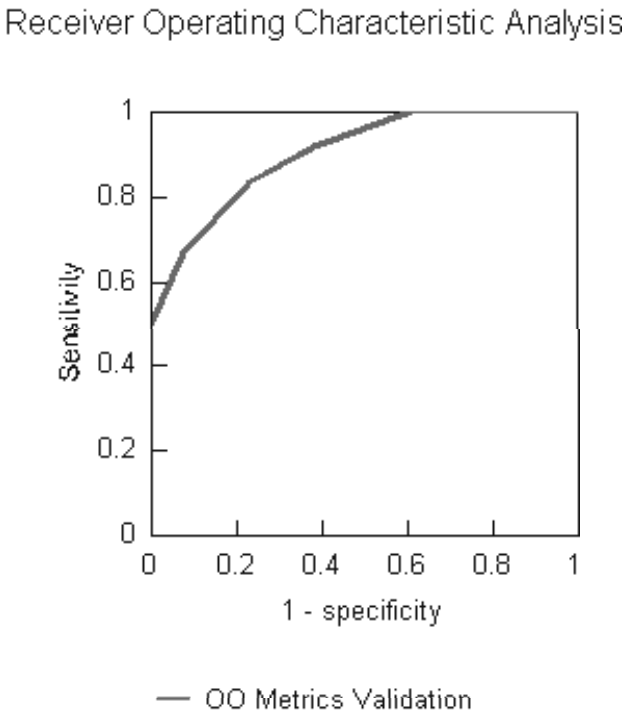

Fig. 4. Hypothetical example of an ROC curve.

For our purposes, we can obtain a summary accuracy measure from an ROC curve by calculating the AUC using a trapezoidal rule (Hanley and McNeil, 1982). The area under the ROC curve has an intuitive interpretation (Hanley and McNeil, 1982;Spiegelhalter, 1986): it is the estimated probability that a randomly selected class with a fault will be assigned a higher predicted probability by the LR model than another randomly selected class without a fault. Therefore, an AUC of say 0.8 means that a randomly selected faulty class has an estimated probability larger than a randomly selected not faulty class 80% of the time.

When a model cannot distinguish between faulty and not faulty classes, the area will be equal to 0.5 (the ROC curve will coincide with the diagonal). When there is a perfect separation of the values of the two groups, the area under the ROC curve equals 1 (the ROC curve will reach the upper left corner of the plot).

Therefore, to compute the accuracy of a prediction LR model, we use the area under the ROC curve, which provides a general and non-arbitrary measure of how well the probability predictions can rank the classes in terms of their fault-proneness.

The optimal operating point on the ROC curve is the point closest to the top-left corner. This gives the cutoff value that will provide the highest sensitivity and specificity. At the optimal cutoff, one can also estimate the sensitivity, $\hat{s}$, and specificity, $\hat{f}$. These values are then used for quality estimation.

Appendix D. Quality estimation

Using a calibrated LR model (i.e., where the optimal cutoff point has been identified), it is possible to estimate quality on a new data set. Here, quality is defined as the proportion of classes that have at least one fault. A naïve estimate of the proportion of faulty classes is

$$\hat{t} = \frac{N_{+2}}{N}. \tag{A.6}$$

However, as shown in Rogan and Gladen (1978), this will only be unbiased if both sensitivity and specificity are equal to 1. The corrected estimate of the proportion of faulty components is given by

$$\hat{p} = \frac{\hat{t} + \hat{f} - 1}{\hat{s} + \hat{f} - 1}. \tag{A.7}$$

If both $\hat{s}$ and $\hat{f}$ are equal to 1, then $\hat{p} = \hat{t}$. Since in practice this is unlikely to be the case, we should use Eq. (A.7) to make the estimate.

References

Abreu, F.B., Carapuca, R. 1994. Object-oriented software engineering: measuring and controlling the development process. In: Proceedings of the Fourth International Conference on Software Quality.

Abreu, F.B., Melo, W., 1996. Evaluating the impact of object-oriented design on software quality. In: Proceedings of the Third International Software Metrics Symposium, pp. 90–99.

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

Belsley, D., Kuh, E., Welsch, R., 1980. Regression Diagnostics: Identifying Influential Data and Sources of Collinearity. Wiley, New York.

Benlarbi, S., Melo, W., 1999. Polymorphism measures for early risk prediction. In: Proceedings of the 21st International Conference on Software Engineering, pp. 334–344.

Binkley, A., Schach, S., 1998. Validation of the coupling dependency metric as a predictor of run-time failures and maintenance measures. In: Proceedings of the 20th International Conference on Software Engineering, pp. 452–455.

Breslow, N., Day, N. 1980. Statistical Methods in Cancer Research – Volume 1 – The Analysis of Case Control Studies. IARC.

Briand, L., Devanbu, P., Melo, W., 1997. An investigation into coupling measures for C++. In: Proceedings of the 19th International Conference on Software Engineering.

Briand, L., Daly, J., Porter, V., Wuest, J., 1998a. Predicting fault-prone classes with design measures in object oriented systems. In: Proceedings of the International Symposium on Software Reliability Engineering, pp. 334–343.

Briand, L., Wuest, J., Ikonomovski, S., Lounis, H., 1998b. A comprehensive investigation of quality factors in object-oriented designs: an industrial case study. International Software Engineering Research Network, ISERN-98-29.

Briand, L., Daly, J., Wuest, J., 1999a. A unified framework for coupling measurement in object-oriented systems. IEEE Transactions on Software Engineering 25 (1), 91–121.

Briand, L., Wuest, J., Ikonomovski, S., Lounis, H., 1999b. Investigating quality factors in object-oriented designs: an industrial case study. In: Proceedings of the International Conference on Software Engineering.

Briand, L., Wuest, J., Daly, J., Porter, V., 2000. Exploring the relationships between design measures and software quality in object oriented systems. Journal of Systems and Software 51, 245–273.

Cartwright, M., 1998. An empirical view of inheritance. Information and Software Technology 40, 795–799.

Cartwright, M., Shepperd, M., 2000. An empirical investigation of an object-oriented software system. IEEE Transactions on Software Engineering, to appear.

Chidamber, S., Kemerer, C., 1994. A metrics suite for object-oriented design. IEEE Transactions on Software Engineering 20 (6), 476–493.

Chidamber, S., Darcy, D., Kemerer, C., 1998. Managerial use of metrics for object-oriented software: an exploratory analysis. IEEE Transactions on Software Engineering 24 (8), 629–639.

Daly, J., Brooks, A., Miller, J., Roper, M., Wood, M., 1996. Evaluating inheritance depth on the maintainability of object-oriented software. Empirical Software Engineering: An International Journal 1 (2), 109–132.

Derksen, S., Keselman, H., 1992. Backward, forward and stepwise automated subset selection algorithms: frequency of obtaining authentic and noise variables. British Journal of Mathematical and Statistical Psychology 45, 265–282.

El-Emam, K., Benlarbi, S., Goel, N., Rai, S., 1999. A Validation of Object-Oriented Metrics. National Research Council of Canada, NRC/ERB 1063.

El-Emam, K., Benlarbi, S., Goel, N., Rai, S., 2000. The confounding effect of class size on the validity of object-oriented metrics. IEEE Transactions on Software Engineering, to appear.

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

Farnese, R., Melo, W., Veiga, A.da., 1999. JMetrics: Java Metrics Extractor. Oracle Consulting Services, Brazil.

Fenton, N., Neil, M., 1999a. Software metrics: successes, failures, and new directions. Journal of Systems and Software 47, 149–157.

Fenton, N., Neil, M., 1999b. A critique of software defect prediction models. IEEE Transactions on Software Engineering 25 (5), 676–689.

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

Gordis, L., 1996. Epidemiology. Saunders, London.

Hanley, J., McNeil, B., 1982. The meaning and use of the area under a receiver operating characteristic curve. Diagnostic Radiology 143 (1), 29–36.

Harrison, R., Counsell, S., Nithi, R., 1998. Coupling metrics for object oriented design. In: Proceedings of the Fifth International Symposium on Software Metrics, pp. 150–157.

Harrison, W., 1988. Using software metrics to allocate testing resources. Journal of Management Information Systems 4 (4), 93–105.

Henderson-Sellers, B., 1996. Object-Oriented Metrics: Measures of Complexity. Prentice-Hall, Englewood cliffs, NJ.

Hosmer, D., Lemeshow, S., 1989. Applied Logistic Regression. Wiley, New York.

Kaaniche, M., Kanoun, K., 1996. Reliability of a commercial telecommunications system, In: Proceedings of the International Symposium on Software Reliability Engineering, pp. 207–212.

Li, W., Henry, S., 1993. Object-oriented metrics that predict maintainability. Journal of Systems and Software 23, 111–122.

Lorenz, M., Kidd, J., 1994. Object-Oriented Software Metrics. Prentice-Hall, Englewood cliffs, NJ.

Melo, W., Lounis, H., Sahroui, H., 1999. In: H. Webster (Eds.), Software quality. Wiley Encyclopedia of Electrical and Electronics Engineering, p. 19.

Metz, C., 1978. Basic principles of ROC analysis. Seminars in Nuclear Medicine 8 (4), 283–298.

Moller, K-H., Paulish, D., 1993. An empirical investigation of software fault distribution. In: Proceedings of the First International Software Metrics Symposium, pp. 82–90.

Morasca, S., Ruhe, G., 1997. Knowledge discovery from software engineering measurement data: a comparative study of two analysis techniques. In: Proceedings of the International Conference on Software Engineering and Knowledge Engineering.

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

Ohlsson, N., Alberg, H., 1996. Predicting fault-prone software modules in telephone switches. IEEE Transactions on Software Engineering 22 (12), 886–894.

Pergibon, D., 1981. Logistic regression diagnostics. The Annals of Statistics 9 (4), 705–724.

Rogan, W., Gladen, B., 1978. Estimating prevalence from the results of a screening test. American Journal of Epidemiology 107 (1), 71–76.

Schlesselman, J., 1982. Case-Control Studies: Design, Conduct, Analysis. Oxford University Press, Oxford.

Sheskin, D., 1997. Handbook of Parametric and Nonparametric Statistical Procedures. CRC Press, Boca Raton.

Spiegelhalter, D., 1986. Probabilistic prediction in patient management in clinical trials. Statistics in Medicine 5, 421–433.

Tang, M-H., Kao, M-H., Chen, M-H., 1999. An empirical study on object oriented metrics. In: Proceedings of the Sixth International Software Metrics Symposium, pp. 242–249.

Unger, B., Prechelt, L., 1998. The Impact of Inheritance Depth on Maintenance Tasks – Detailed Description and Evaluation of Two Experiment Replications. Fakultat fur Informatik – Universitaet Karlsruhe, 19/1998.

Weiss, S., Kulikowski, C., 1991. Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems. Morgan Kaufmann Publishers, Los Altos, CA.

Youden, W., 1950. Index for rating diagnostic tests. Cancer 3, 32–35.

Zweig, M., Campbell, G., 1993. Receiver-operating characteristic (ROC) plots: a fundamental evaluation tool in clinical medicine. Clinical Chemistry 39 (4), 561–577.

Khaled El Emam is currently at the National Research Council in Ottawa. He is the editor of the IEEE TCSE Software Process Newsletter, the current International Trials Coordinator for the SPICE Trials, which is empirically evaluating the emerging ISO/IEC 15504 International Standard world-wide, co-editor of ISO's project to develop an international standard defining the software measurement process, member of the CMMI product development team, focusing on empirical evaluation of the CMMI product set, and Knowledge Area specialist for the Software Engineering Process in the IEEE's project to define the Software Engineering Body of Knowledge. He is also an adjunct professor at the School of Computer Science at McGill University, and an adjunct professor at the Department of Computer Science at the University of Quebec in Montreal. Previously, he worked in both small and large software research and development projects for organizations such as Toshiba International Company, Yokogawa Electric, and Honeywell Control Systems. Khaled El Emam obtained his Ph.D. from the Department of Electronic Engineering, King's College, the University of London (UK) in 1994. He was previously the head of the Quantitative Methods Group at the Fraunhofer Institute for Experimental Software Engineering in Germany, a research scientist at the Centre de recherche informatique de Montreal (CRIM), and a research assistant in the Software Engineering Laboratory at McGill University.

Walcelio Melo is a Technical Practice Manager at Oracle Brazil responsible for commerce custom development applications. He is also a Professor in the Computer Science Department at the Católica University of Brasilia, Brazil, and the chairman of the Post-graduate Diploma on Software Engineering. Before joining Oracle Brazil, Dr. Melo had been the Software Engineering Lead Researcher at CRIM (Centre de recherche informatique de Montréal) and Adjunct Professor in the School of Computer Science at McGill University, Montréal, Canada. From 1994 to 1996, Dr. Melo was Faculty Research Associate in the Institute for Advanced Computer Studies at University of Maryland, College Park, Maryland, 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. From 1983 to 1989, he worked as a System Analyst for the Brazilian Federal Bureau of Data Processing. Dr. Melo holds the following degrees in Computer Science: B.Sc. in 1983 from University of Brasilia, Brazil, M.Sc. in 1988 from the Federal University of Rio Grande do Sul, Brazil, and Ph.D. with honors in 1993 from University of Grenoble – Université Joseph Fourrier, France. His research interests are software measurement, software reuse, object technologies, and e-commerce software applications.

Javam C. Machado is an assistant professor of the Computer Science Department of the Federal University of Ceará - Brazil, where he is the chairman of the object technology and distributed system post-graduate Diploma. Dr. Javam received his Ph.D. in computer science from the University of Grenoble – France in 1995, his M.Sc. from the Federal University of Rio Grande do Sul – Brazil in 1990 and his B.Sc. in computer science from the Federal University of Ceará in 1987. His research interests are advanced databases, object technologies and WEB systems.