The Journal of Systems and Software 83 (2010) 2–17

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A systematic and comprehensive investigation of methods to build and evaluate fault prediction models

Erik Arisholm, Lionel C. Briand *, Eivind B. Johannessen

Simula Research Laboratory, P.O. Box 134, NO-1325 Lysaker, Norway
Dept. of Informatics, University of Oslo, Norway

ARTICLE INFO

Article history:
Received 31 October 2008
Received in revised form 18 March 2009
Accepted 26 June 2009
Available online 19 August 2009

Keywords:
Fault prediction models
Cost-effectiveness
Verification

ABSTRACT

This paper describes a study performed in an industrial setting that attempts to build predictive models to identify parts of a Java system with a high fault probability. The system under consideration is constantly evolving as several releases a year are shipped to customers. Developers usually have limited resources for their testing and would like to devote extra resources to faulty system parts. The main research focus of this paper is to systematically assess three aspects on how to build and evaluate fault-proneness models in the context of this large Java legacy system development project: (1) compare many data mining and machine learning techniques to build fault-proneness models, (2) assess the impact of using different metric sets such as source code structural measures and change/fault history (process measures), and (3) compare several alternative ways of assessing the performance of the models, in terms of (i) confusion matrix criteria such as accuracy and precision/recall, (ii) ranking ability, using the receiver operating characteristic area (ROC), and (iii) our proposed cost-effectiveness measure (CE).

The results of the study indicate that the choice of fault-proneness modeling technique has limited impact on the resulting classification accuracy or cost-effectiveness. There is however large differences between the individual metric sets in terms of cost-effectiveness, and although the process measures are among the most expensive ones to collect, including them as candidate measures significantly improves the prediction models compared with models that only include structural measures and/or their deltas between releases – both in terms of ROC area and in terms of CE. Further, we observe that what is considered the best model is highly dependent on the criteria that are used to evaluate and compare the models. And the regular confusion matrix criteria, although popular, are not clearly related to the problem at hand, namely the cost-effectiveness of using fault-proneness prediction models to focus verification efforts to deliver software with less faults at less cost.

1. Introduction

A significant research effort has been dedicated to defining specific quality measures and building quality models based on those measures (Briand and Wuest, 2002). Such models can then be used to help decision-making during development of software systems. Fault-proneness or the number of defects detected in a software component (e.g., class) are the most frequently investigated dependent variables (Briand and Wuest, 2002). In this case, we may want to predict the fault-proneness of classes in order to focus validation and verification effort, thus potentially finding more defects for the same amount of effort. Assuming a class is predicted as very likely to be faulty, one would take corrective action by investing additional effort to inspect and test the class. Given that software development companies might spend between 50% and 80% of their software development effort on testing (Collofello and Woodfield, 1989), research on fault-proneness prediction models can be motivated by its high cost-saving potential.

As a part of this study, we have reviewed a selection of relevant publications within the field of fault-proneness prediction models. Due to space constraints, details are not provided in this paper but are summarized in the related works section and are presented in Arisholm et al. (2008). The review revealed that a vast number of modeling techniques have been used to build such prediction models. However, there has been no comprehensive and systematic effort on assessing the impact of selecting a particular modeling technique.

To construct fault-proneness prediction models, most studies use structural measures such as coupling and cohesion as independent variables. Although some studies have investigated the possible benefits of including other measures such the number of

* Corresponding author. Address: Simula Research Laboratory, P.O. Box 134, NO-1325 Lysaker, Norway.
E-mail addresses: erika@simula.no (E. Arisholm), briand@simula.no (Lionel C. Briand), eivindjo@ifi.uio.no (E.B. Johannessen).

0164-1212/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2009.06.055

changes performed on components and their fault history in previous releases, none of the studies assess in a systematic way the impact of using various sets of measures, entailing different data collection costs, on the cost-effectiveness of the prediction models.

A large number of evaluation criteria have been used to evaluate and compare fault-proneness prediction models,. Among the most popular evaluation criteria are the ones that can be derived from the confusion matrix such as accuracy, precision, and recall. There is little consistency across the reviewed studies with respect to the criteria and methods that are used to evaluate the models, making it hard to draw general conclusions on what modeling technique or sets of independent variables seems the most appropriate. In addition, the popular confusion matrix criteria are somewhat abstract as they do not clearly and directly relate to the cost-effectiveness of using fault-proneness prediction models to focus verification and validation activities such as testing. Because there exists very little evidence of the economic viability of fault-proneness prediction models (Briand and Wuest, 2002), there is a need for evaluating and comparing fault-proneness prediction models not only by considering their prediction accuracy, but also by assessing the potential cost-effectiveness of applying such models.

To compare the potential cost-effectiveness of alternative prediction models, we need to consider (surrogate) measures of additional verification cost for the selected, faulty classes. For many verification activities, such as structural coverage testing or even simple code inspections, the cost of verification is likely to be roughly proportional to the size of the class.[1] What we want are models that capture other fault factors in addition to size, so that the model would select a subset of classes with high fault density.

To build fault-proneness prediction models there are a large number of modeling techniques to choose from, including standard statistical techniques such as logistic regression, and data mining techniques such as decision trees (Witten and Frank, 2005). The data mining techniques are especially useful since we have little theory to work with and we want to explore many potential factors (and their interactions) and compare many alternative models so as to optimize cost-effectiveness.

Although there are a large number of publications that have built and evaluated methods for building fault-proneness prediction models, it is not easy to draw practical guidelines from them in terms of what modeling techniques to use, what data to collect, and what practical gains to expect. This paper investigates in a systematic way three practical aspects of the building and evaluation of fault-proneness prediction models; (i) choice of modeling techniques, (ii) choice of independent variables (sets of measures), and (iii) choice of evaluation criteria. This assessment is performed by building a range of fault-proneness prediction models using a selection of relevant modeling techniques. The models are built using different sets of independent variables entailing different data collection costs. This allows us to assess the possible benefits of collecting certain sets of measures. The resulting models are then systematically compared and evaluated using a number of the most popular evaluation criteria such as accuracy, precision and recall. To assess the potential cost-effectiveness in applying the models to focus verification activities, we also compare the models according to a proposed measure of cost-effectiveness within this particular industrial context.

The remainder of this paper is organized as follows: Section 2 provides an overview of related works, whereas Section 3 presents our study design. In Section 4 we report our results, comparing several modeling techniques and sets of measures using a number of different evaluation criteria. Section 5 discusses what we consider the most important threats to validity, whereas Section 6 concludes and outlines directions for future research.

[1] Depending on the specific verification undertaken on classes predicted as fault prone, one may want to use a different size measure that would be proportional to the cost of verification.

2. Related work

A literature review of existing studies on building fault-proneness prediction models are given in Arisholm et al. (2008). In this section we focus on those studies that have attempted to compare techniques, measures or evaluation criteria to build the best possible fault-proneness prediction models.

Briand et al. (2002) compared traditional regression techniques with multivariate adaptive regression splines (MARS) (Friedman, 1991). The MARS model performed slightly better in terms of accuracy, completeness and correctness, compared to logistic regression. Also, the authors did a cost/benefit analysis, which suggested the MARS model outperformed the model built using logistic regression. Khoshgoftaar and Seliya (2004) compared seven models that were built using a variety of tools. The models were built using different regression and classification trees including C4.5, CHAID, Sprint-Sliq and different versions of CART. Also included in the study were logistic regression and case-based reasoning. The techniques were evaluated against each other by comparing a measure of expected cost of misclassification. The differences between the techniques were at best moderate. Vandecruys et al. (2008) compared Ant Colony Optimization against well-known techniques like C4.5, support vector machine (SVM), logistic regression, *K*-nearest neighbour, RIPPER and majority vote. In terms of accuracy, C4.5 was the best technique. However, the differences between the techniques in terms of accuracy, sensitivity and specificity were moderate. Kanmani et al. (2007) compared two variants of artificial neural networks against logistic regression and discriminant analysis. Neural network outperformed the traditional statistical regression techniques in terms of precision and recall. Gondra (2008) assessed the possible benefits of neural networks versus SVMs to perform simple classification. When considering fault-proneness as a binary classification problem (i.e. faulty vs. non-faulty) using a threshold of 0.5, the accuracy was 87.4% when using SVM compared to 72.61% when using neural networks – suggesting that SVM is a promising technique for classification within the domain of fault-proneness prediction. Elish and Elish (2008) compared SVM against eight other modeling techniques, among them Random Forest. The modeling techniques were evaluated in terms of accuracy, precision, recall and the F-measure using four data sets from the NASA Metrics Data Program Repository. All techniques achieved an accuracy ranging from approximately 0.83 to 0.94. As with the other studies reviewed here, there were some differences, but no single modeling technique was significantly better than the others across data sets. Guo et al. (2004) compared 27 modeling techniques including logistic regression and 20 techniques available through the WEKA tool. The study compared the techniques using five different datasets from the NASA MDP program, and although the results showed that Random Forests perform better than many other classification techniques in terms of accuracy and specificity, the results were not significant in four of the five data sets. Arisholm et al. (2007) evaluated the possible cost-effectiveness and classification accuracy (precision, recall, ROC) of eight data mining techniques. The results suggested that C4.5 classification trees performed well, and somewhat better than other techniques such as SVM and neural networks.

Most existing studies only considered code structural metrics and only a subset of studies (Arisholm et al., 2007; Khoshgoftaar and Kehan, 2007; Khoshgoftaar and Seliya, 2003; Kim et al., 2008; Nagappan and Ball, 2007; Ostrand and Weyuker, 2007; Ostrand et al., 2005; Ostrand et al., 2007; Weyuker et al., 2007;

Nagappan and Ball, 2005; Graves et al., 2000) have included other measures like deltas of structural metrics between subsequent releases, or measures related to fault history and the software development process itself (process metrics). Nagappan and Ball (2007, 2005) used *code churn* together with dependency metrics to predict fault-prone modules. Code churn is a measure of the amount of code change within a component over time. Kim et al. (2008) used deltas from 61 complexity metrics and a selection of process metrics, and achieved an accuracy ranging from 64% to 92% on twelve open source applications. Graves et al. (2000) counted the number of changes done in a module as well as the average age of the code. Weyuker et al. (2007) constructed a fault-count prediction model using a number of process measures in addition to structural measures. They accounted for the number of developers who modified a file during the prior release, and the number of *new* developers involved on a particular file. In addition, they counted the cumulative number of distinct developers who have modified a file during its lifetime. The model using these process measures showed only slight improvements compared with a model using only structural measures. Khoshgoftaar and Seliya (2003) considered 14 process metrics, such as the number of updates done by designers who had 10 or less total updates in their entire company career, the number of different designers making changes to a particular module, and the net increase in lines of code (LOC) for each module. Khoshgoftaar and Seliya did not study the impact of the individual measures on fault-proneness, but their prediction models achieved a balance of Types I and II misclassification rates of 25–30%.

In summary, there exist a few studies that have compared a comprehensive set of data mining techniques for building fault-proneness prediction models to assess which techniques are more likely to be accurate in various contexts. Most models were evaluated through different confusion matrix criteria and, as a result, it is difficult to provide general conclusions. However, it appears that the differences between modeling techniques might be relatively small. Most existing studies have used structural measures as candidate predictors whereas only a subset have also included other more expensive measures, such as code churn and process measures. However, no studies have so far attempted to evaluate the benefits of including such measures in comparison with models that contain only structural code measures. In this paper, we assess, in a systematic way, how both the choice of modeling technique and the selection of different categories of candidate measures affect the accuracy and cost-effectiveness of the resulting prediction models based on a complete set of evaluation criteria. We furthermore assess how the choice of evaluation criteria affects what is deemed to be the "best" prediction model.

3. Design of study

When building fault-proneness prediction models, many decisions have to be made regarding the choice of dependent and independent variables, modeling technique, evaluation method and evaluation criteria. At present, no systematic study has been performed to assess the impact of such decisions on the resulting prediction models (Arisholm et al., 2008). This paper compares alternative fault-proneness prediction models where we systematically vary three important dimensions of the modeling process: modeling technique (e.g., C4.5, neural networks, logistic regression), categories of independent variables (e.g., process measures, object-oriented code structural measures, code churn measures) and evaluation criteria (e.g., accuracy, ROC, and cost-effectiveness). We assess (i) to what extent different data mining techniques affect prediction accuracy and cost-effectiveness, (ii) the effects of using different sets of measurements (with different data collection costs) on the accuracy and cost-effectiveness of the fault-proneness predictions models, and (iii) how our decisions in terms of selecting the "best" model would be affected by using the different evaluation criteria. This section describes the development project, study variables, data collection, and model building and evaluation procedures.

3.1. The development project

The legacy system studied is a Java middleware system called COS, serving the mobile division in a large telecom company. COS provides more than 40 client systems with a consistent view across multiple back-end systems, and has evolved through 22 major releases during the past eight years. At any point in time, between 30 and 60 software engineers were involved in the project. The core system currently consists of more than 2600 Java classes amounting to about 148 KSLOC. In addition to this, the system consists of a large number of test classes, library classes, and about 1000 KSLOC of generated code, but this code is not considered in our study. As the system expanded in size and complexity, QA engineers felt they needed more sophisticated techniques to focus verification activities on fault-prone parts of the system. We used 13 recent releases of this system for model building and evaluation. As a first step, the focus was on unit testing in order to eliminate as many faults as possible early on in the verification process by applying more stringent test strategies to code predicted as fault-prone.

3.2. Data collection procedures

Perl scripts were developed to collect file-level change data for the studied COS releases through the configuration management system (MKS). In our context, files correspond to Java public classes. The data model is shown in Fig. 1. Each change is represented as a change request (CR). The *CR* is related to a given *releaseId* and has a given *changeType*, defining whether the change is a critical or non-critical fault correction, a small, intermediate, or large requirement change, or a refactoring change. An individual developer can work on a given CR through a logical work unit called a change package (CP), for which the developer can check in and out files in relation to the CR. For a CP, we record the number of CRs that the responsible developer has worked on prior to opening the given CP, and use this information as a surrogate measure of that person's coding experience on the COS system. For each *Class* (file) modified in a *CP*, we record the number of lines added and deleted, as modeled by the association class *CP_Class.* Data about each file in the COS system is collected for each release, and is identified using a unique *MKSId*, which ensures that the change history of a class can be traced even in cases where it changes location (package) from one release to the next. This traceability turned out to be crucial in our case because we wanted to keep track of historic changes and faults for each class, and there were quite a few refactoring changes in the project that would result in loss of historic data if we did not use the *MKDId* to uniquely identify each class. Finally, for each release, a code parser (JHawk) is executed to collect structural measures for the class, which are combined with the MKS change information. Independent (change, process, and code structure measurements) and dependent variables (Faults in the next release) were computed on the basis of the data model presented in Fig. 1.

3.3. Dependent variable

The dependent variable in our analysis was the occurrences of corrections in classes of a specific release which are due to field error reports. Since our main current objective was to facilitate

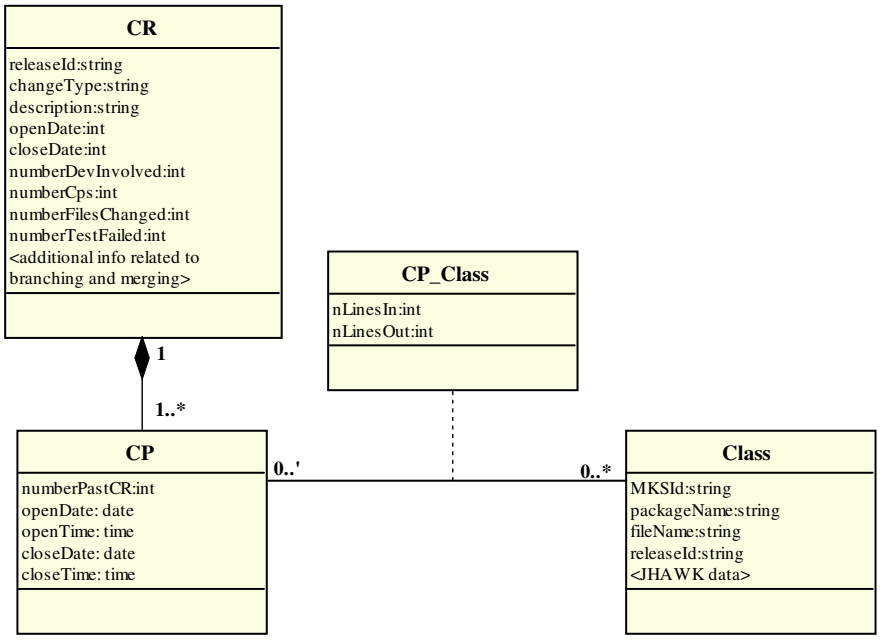

Fig. 1. Data model.

unit testing and inspections, the class was a logical unit of analysis. Given that our aim was to capture the fault-proneness of a class in a specific release n, and that typically a fault correction involved several classes, we decided to count the number of distinct fault corrections that was required in each class for developing release $n + 1$. Furthermore, in this project, only a very small portion of classes contained more than one fault for a given release, so class fault-proneness in release n is therefore treated as a classification problem and is estimated as the probability that a given class will undergo one or more fault corrections in release $n + 1$.

3.4. Explanatory variables

Though many studies on predicting fault-prone classes on the basis of the structural properties of object-oriented systems have been reported (Arisholm et al., 2008), a specificity of the study presented here is the fact that we needed to predict fault-proneness for a changing legacy system. Thus, in addition to structural measures, similar to other studies (Arisholm et al., 2007; Khoshgoftaar and Kehan, 2007; Khoshgoftaar and Seliya, 2003; Kim et al., 2008; Nagappan and Ball, 2007, 2005; Ostrand and Weyuker, 2007; Ostrand et al., 2005, 2007; Weyuker et al., 2007; Graves et al., 2000) we also use data on changes and fault corrections for specific releases and their impact on the code. In our context, past change and fault data could be useful to help predicting fault-proneness by identifying what subset of classes have shown to be inherently fault and change prone in the past. Our explanatory variables can be classified into three categories:

- Object-oriented (OO) code measures, i.e., measures of structural properties derived from the source code. In this study, the JHawk tool was used to collect such measures, as shown in Table 1.
- Delta measures: These measures capture the amount of change – sometimes called churn – in a file between two successive releases. In this study, the delta measures were computed from the JHawk measures given in Table 1.
- Process measures: In this study, the process measures were collected from the configuration management system (MKS), and included a surrogate measure of the experience of each developer performing each change, the number of developers that have made changes to a file, the number of faults in previous release(s) and simpler measures such as the accumulated number of lines added and/or removed in a given release.

The fundamental hypothesis underlying our work is that the fault-proneness of classes in a legacy, object-oriented system can be affected by these measures. Furthermore, it is also likely that these factors interact in the way they affect fault-proneness. For example, changes may be more fault-prone on larger, more complex classes. The data mining techniques used to build the models will account for such interactions.

The three categories of measures (OO, Delta and Process) incur different costs in terms of data collection effort and process instrumentation requirements. *OO* measures can be collected from simple code snapshots, *Deltas* require that different versions of the system be available, whereas *Process* measures require that developers record detailed information about their work (e.g., changes and fault corrections, developer info, time of changes, whether a change passed certain test procedures) in a systematic and consistent way in configuration management or change management systems. To assess the relative importance of the individual categories of explanatory variables (*OO*, *Delta* and *Process*), they were combined to construct seven different candidate metric sets (*OO*, *Delta*, *Process*, *OO + Delta*, *Process + OO*, *Process + Delta*, *Total*). In Section 4.2 we will show how the many different measures of accuracy and cost-effectiveness of the fault-proneness prediction models are affected by the choice of metric set. In this way, we will not only be able to compare individual categories of measures (e.g., *Process* vs. *OO*) but also assess the potential impact of *combining* measures (e.g., *Process + OO*) with regards to a comprehensive set of evaluation criteria (Section 3.7). Based on such analyses, we will be in a better position to determine whether the added cost of collecting, for example, process measures will result in payoffs in terms of better fault-proneness prediction models.

Table 1
Summary of the explanatory variables.

Variable	Description	Source
OO		
No_Methods|NOQ|NOC	Number of [implemented|query | command] methods in the class	JHawk
LCOM	Lack of cohesion of methods	JHawk
TCC|MAXCC|AVCC	[Total|Max|Avg] cyclomatic complexity in the class	JHawk
NOS | UWCS	Class size in [number of Java statements | number of attributes + number of methods]	JHawk
HEFF	Halstead effort for this class	JHawk
EXT/LOC	Number of [external | local] methods called by this class	JHawk
HIER	Number of methods called that are in the class hierarchy for this class	JHawk
INST	Number of instance variables	JHawk
MOD	Number of modifiers for this class declaration	JHawk
INTR	Number of interfaces implemented	JHawk
PACK	Number of packages imported	JHawk
RFC	Total response for the class	JHawk
MPC	Message passing coupling	JHawk
FIN	The sum of the number of unique methods that call the methods in the class	JHawk
FOUT	Number of distinct non-inheritance related classes on which the class depends	JHawk
R–R|S–R	[Reuse|Specialization] Ratio for this class	JHawk
NSUP|NSUB	Number of [super|sub] classes	JHawk
MI|MINC	Maintainability Index for this class[including|not including] comments	JHawk
Delta		
	For each OO measure X *above:*	
delta_<X>	The difference in each OO measure X between two successive releases	Calculated
Process		
[nm1|nm2|nm3]_CLL_CR	The number of large requirement changes for this class in release [$n-1$|$n-2$|$n-3$]	MKS
[nm1|nm2|nm3]_CFL_CR	The number of medium requirement changes for this class in release [$n-1$|$n-2$|$n-3$]	MKS
[nm1|nm2|nm3]_CKL_CR	The number of small requirement changes for this class in release [$n-1$|$n-2$|$n-3$]	MKS
[nm1|nm2|nm3]_M_CR	The number of refactoring changes for this class in release [$n-1$|$n-2$|$n-3$]	MKS
[nm1|nm2|nm3]_CE_CR	The number of critical fault corrections for this class in release [$n-1$|$n-2$|$n-3$]	MKS
[nm1|nm2|nm3]_E_CR	The number of noncritical fault corrections for this class in release [$n-1$|$n-2$|$n-3$]	MKS
numberCRs	Number of CRs in which this class was changed	MKS
numberCps	Total number of CPs in all CRs in which this class was changed	MKS
numberCpsForClass	Number of CPs that changed the class	MKS
numberFilesChanged	Number of classes changed across all CRs in which this class was changed	MKS
numberDevInvolved	Number of developers involved across all CRs in which this class was changed	MKS
numberTestFailed	Total number of system test failures across all CRs in which this class was changed	MKS
numberPastCr	Total developer experience given by the accumulated number of prior changes	MKS
nLinesIn	Lines of code added to this class (across all CPs that changed the class)	MKS
nLinesOut	Lines of code deleted from this class (across all CPs that changed the class)	MKS
	For CRs of type Y = {CLL, CFL, CKL, M, CE, E}:	
<Y>_CR	Same def as *numberCRs* but only including the subset of CR's of type Y	MKS
<Y>_CPs	Same def as *numberCpsForClass* but only including the subset of CR's of type Y	MKS
<Y>numberCps	Same def as *numberCps* but only including the subset of CR's of type Y	MKS
<Y>numberFilesChanged	Same def as *numberFilesChanged* but only including the subset of CR's of type Y	MKS
<Y>numberDevInvolved	Same def as *numberDevInvolved* but only including the subset of CR's of type Y	MKS
<Y>numberTestFailed	Same def as *numberTestFailed* but only including the subset of CR's of type Y	MKS
<Y>numberPastCr	Same def as *numberPastCr* but only including the subset of CR's of type Y	MKS
<Y>nLinesIn	Same def as *nLinesIn* but only including the subset of CR's of type Y	MKS
<Y>nLinesOut	Same def as *nLinesOut* but only including the subset of CR's of type Y	MKS

3.5. Model building techniques

A detailed description of many of the most popular techniques for building fault-proneness prediction models can be found in Arisholm et al. (2008). In this study we compared one classification tree algorithm (C4.5) as it is the most studied in its category, the most recent coverage rule algorithm (PART) which has shown to outperform older algorithms such as Ripper (Witten and Frank, 2005), Logistic Regression as a standard statistical technique for classification, Back-propagation neural networks as it is a widely used technique in many fields, and SVM.

For C4.5, we also applied the AdaBoost and Decorate metalearners (Witten and Frank, 2005), because decision trees are inherently unstable due to the way their learning algorithms work, and thus we wanted to assess the impact of using metalearners on C4.5. We included Decorate in addition to Adaboost because it is supposed to outperform boosting on small training sets and rivals it on larger ones.

Furthermore, as the outputs of leaves and rules are directly comparable, we combined C4.5 and PART predictions by selecting, for each class instance to predict, the rule or leaf that yields a fault probability distribution with the lowest entropy (i.e., the fault probability the furthest from 0.5, in either direction). This allows us to use whatever technique works best for each prediction instance.

For each metric set, we also used Correlation-based Feature Selection (CFS) (Hall, 2000) to pre-select variables, as further described in Arisholm et al. (2008), to assess the effect of such variable pre-selection on the prediction model performance.

All of the above techniques were applied using the WEKA tool and are described in Witten and Frank (2005). An attempt was made to optimize the parameters of various techniques, but in most cases the impact of varying these parameters was small and we resorted to using the WEKA default parameters.

3.6. Training and evaluation datasets

To build and evaluate the prediction models, class-level structural and change/fault data from 13 recent releases of COS were used. The data was divided into four separate subsets, as follows.

The data from the 11 first releases was used to form two datasets, respectively a training set to build the model and a test set to evaluate the predictions versus actual class faults. More specifically, following the default setting of most tools, two thirds of the data (16004 instances) were randomly selected as the *Training* dataset, whereas the remaining one third (8002 instances) formed the *Excluded* test dataset. Our data set was large enough to follow this procedure to build and evaluate the model without resorting to cross-validation, which is much more computationally intensive. Also, the random selection of the training set across 11 releases reduced the chances for the prediction model to be overly influenced by peculiarities of any given release. Note that in the training set, there were only 303 instances representing faulty classes (that is, the class had at least one fault correction in the next release). This is due to the fact that, in a typical release, a small percentage of classes turn out to be faulty. Thus, to facilitate the construction of unbiased models, we created a balanced subset (606 rows) from the complete training set, consisting of the 303 faulty classes and a random selection of 303 rows representing non-faulty classes. The proportions of faulty and correct classes were therefore exactly 50% in the training set and the probability decision threshold for classification into faulty and correct classes for the test sets can therefore be set to 0.5. Nearly all the techniques we used performed better (sometimes very significantly) when run on this balanced dataset. Consequently, the models reported in this paper were built using this subset of 606 instances.

Finally, the two most recent of the 13 selected releases formed the third and forth distinct datasets, hereafter referred to as the *COS 20* and *COS 21* datasets, which we also used as test sets. The *Excluded* test set allows us to estimate the accuracy of the model on the current (release 19) and past releases whereas the *COS 20* and *COS 21* test sets indicate accuracy on future releases. This will give us insights on any decrease in accuracy, if any, when predicting the future. The results given in Section 4 were obtained using only the test set (*Excluded*) and the two evaluation sets (*COS 20 and COS 21*), i.e., the training set was not included. By not including the training set, the results can be interpreted as what one could expect when applying the models on a new set of classes or a new system version.

3.7. Model evaluation criteria

Having described our model evaluation procedure, we now need to explain what model accuracy criteria we used. The alternative prediction models were assessed on the basis of all of the following criteria in order to (1) provide a comprehensive comparison of the models and (2) to assess how the choice of criteria affects the ranking of models.

First, we used several popular confusion matrix criteria (Witten and Frank, 2005), including *accuracy*, *precision* and *recall*, and *Type* I/II *misclassification rates*. Each of these measures can be defined on the basis of the confusion matrix given in Fig. 2. For example, in our context, accuracy is the percentage of classes correctly classified as either faulty or non-faulty ((TP + TN)/N). Precision is the percentage of classes classified as faulty that are actually faulty (TP/(TP + FP)) and is a measure of how effective we are at identifying where faults are located. Recall is the percentage of faulty classes that are predicted as faulty (TP/(TP + FN)) and is a measure of how many faulty classes we are likely to miss if we use the prediction model. Type I/II misclassification rates are the ratio of Type I errors (FP/N) and Type II errors (FN/N), respectively, as proposed in Khoshgoftaar and Allen (2001).

		Actual	
		Positive	**Negative**
Predicted by model	**Positive**	True positive (TP)	False positive (FP)
	Negative	False negative (FN)	True negative (TN)

Fig. 2. The confusion matrix.

All the measures described up to this point are evaluation criteria for classifiers. That is, in the context of fault-proneness models, these measures assess the accuracy (or inaccuracy) of a particular model with regards to fault classification. These measures require that one predefines a cut-off value for predicted probabilities, and although these measures are useful, the intent of fault-proneness prediction models is not only to classify instances. For example, if the prediction model was to be used to focus testing of fault-prone components, we would be more interested in the *ranking* of the components, and use the ranking to select a certain subset of components to test. Consequently, it would be preferable to be able to assess how well a particular model is at ranking instances in a correct manner. Further, it would be preferable to evaluate the performance of a prediction model without first having to choose a specific cut-off value. This is the objective of the receiver operating characteristic (ROC) curve. The ROC curve depicts the benefits of using the model (true positives) versus the costs of using the model (false positives) at different thresholds. The ROC curve allows one to assess performance of a prediction model in general – regardless of any particular cut-off value. The area under the ROC curve can be used as a descriptive statistic, and is the estimated probability that a randomly selected positive instance will be assigned a higher predicted p by the prediction model than another randomly selected negative instance (Hanley and McNeil, 1982). Hence, this statistic quantifies a model's ability to correctly rank instances. The larger the area under the ROC curve (the ROC area) the better the model. A perfect prediction model, that classifies all instances correctly, would have a ROC area of 100%.

The problem with the general confusion matrix criteria and ROC is that they are designed to apply to all classification problems and they do not clearly and directly relate to the cost-effectiveness of using class fault-proneness prediction models in our or any other given application context. Assuming a class is predicted as very likely to be faulty, one would take corrective action by investing additional effort to inspect and test the class. In our context, we consider the cost of such activities to be roughly proportional to the size of the class. For example, regarding control flow testing, many studies show that cyclomatic complexity (number of independent control flow paths) is strongly correlated with code size (Nagappan and Ball, 2005). Though this remains to be empirically investigated, this suggests that control flow testing over a large number of classes should be roughly proportional to the size of those classes.

Given the above assumption, if we are in a situation where the only thing a prediction model does is to model the fact that the number of faults is proportional to the size of the class, we are not likely to gain much from such a model. What we want are models that capture other fault factors in addition to size. Therefore, to assess cost-effectiveness, we compare two curves as exemplified in Fig. 3. Classes are first ordered from high to low fault probabilities. When a model predicts the same probability for two classes, we order them further according to size so that larger classes are selected last. The solid curve represents the actual percentage of faults given a percentage of lines of code of the classes selected to focus verification according to the abovementioned ranking procedure (referred to as the model cost-effectiveness (CE) curve). The dotted line represents a line of slope 1 where the percentage of faults would be identical to the percentage of lines of code (% NOS) included in classes selected to focus verifica-

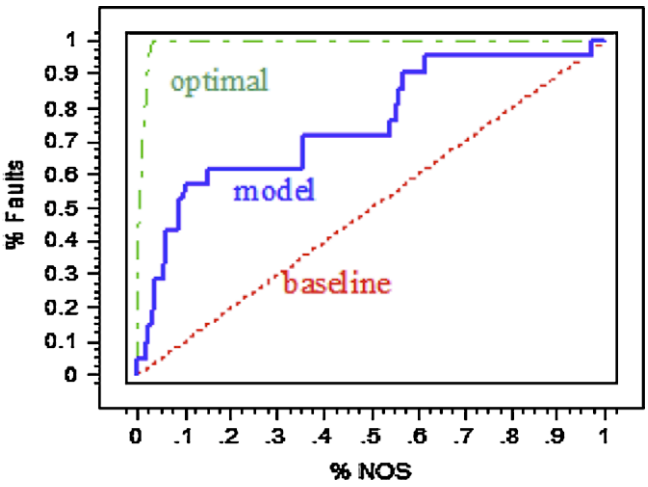

Fig. 3. Surrogate measure of cost-effectiveness.

tion. This line is what one would obtain, on average, if randomly ranking classes and is therefore a baseline of comparison (referred to as the *baseline*). Based on these definitions and the assumptions above, the overall cost-effectiveness of fault predictive models would be proportional to the surface area between the CE curve and the baseline. This is practical as such a surface area is a unique score according to which we can compare models in terms of cost-effectiveness regardless of a specific, possibly unknown, NOS percentage to be verified. If the model yields a percentage of faults roughly identical to the percentage of lines of code, then no gain is to be expected from using such a fault-proneness model when compared to chance alone. The exact surface area to consider may depend on a realistic, maximum percentage of lines of code that is expected to be covered by the extra verification activities. For example, if only 5% of the source code is the maximum target considered feasible for extra testing, only the surface area below the 5% threshold should be considered.

For a given release, it is impossible to determine beforehand what would be the surface area of an *optimal* model. For each release, we compute it by ordering classes as follows: (1) we place all faulty classes first and then order them so that larger classes are tested last, and (2) we place fault-free classes afterwards also in increasing order of size. This procedure is a way to maximize the surface area for a given release and set of faulty classes, assuming the future can be perfectly predicted. Once computed, we can compare, for a specific NOS percentage, the maximum percentage of faults that could be obtained with an optimal model and use this as an upper bound to further assess a model, as shown by the dashed line in Fig. 3.

To compare CE areas we need to account for the fact that the optimal model might differ across test sets. Thus, we compute a normalized cost-effectiveness measure as

$$\mathrm{CE}_\pi = (\mathrm{CE}_\pi(\mathrm{model}) - \mathrm{CE}_\pi(\mathrm{baseline}))/(\mathrm{CE}_\pi(\mathrm{optimal}) - \mathrm{CE}_\pi(\mathrm{baseline}))$$

where $CE_\pi(x)$ is the area under the curve x (*baseline*, *model*, or *optimal*) for a given π percentage of NOS. This measure can be interpreted as a proportion of the optimal cost-effectiveness, a measure which is comparable across evaluation datasets. Depending on the amount of resources available for testing, the percentage of NOS to be tested will vary, so we compute CE for respectively 1%, 5% and 20% of the NOS ($CE_{0.01}$, $CE_{0.05}$, $CE_{0.20}$). Computing a CE area is also a way to compare models without any specific percentage of classes in mind and based on a unique score. This is why we also choose to include the cost-effectiveness at 100% NOS ($CE_{1.00}$). Admittedly such CE values may not be easy to interpret but their purpose is to facilitate the comparison among models based on a measure that should be directly proportional to cost-effectiveness in the context of focusing verification and validation efforts.

3.8. Model assessment procedure

We built a total of 112 different fault-proneness models on the basis of our training dataset, i.e., individual prediction models for each of the seven metric sets presented in Section 3.4 (*OO*, *Delta*, *Process*, *OO + Delta*, *Process + OO*, *Process + Delta*, *Total*) with and without CFS, using each of the eight candidate mining techniques presented in Section 3.5 (*Neural network*, *C4.5*, *Decorate C4.5*, *Boost C4.5*, *SVM*, *Logistic regression*, *PART*, *C4.5 + PART*). Each of the 112 models was evaluated on the three distinct evaluation datasets presented in Section 3.6 (*Excluded*, *COS 20*, *COS 21*) and using the evaluation criteria presented in Section 3.7 (*Accuracy*, *Precision*, *Recall*, *Type I/II misclassification rate*, *ROC*, $CE_{0.01}$, $CE_{0.05}$, $CE_{0.20}$, $CE_{1.00}$).

To assess the magnitude of the differences between the model building techniques and the metric sets, we report a number of statistics including the mean, the minimum, and maximum of each criterion. As it is difficult to make any assumptions about the underlying distribution for many of the evaluation criteria we use non-parametric tests to assess the significance of the differences. More specifically, for each evaluation criterion, we report *p*-values from a matched pair Wilcoxon's signed rank test for

- all pairs of techniques aggregated across metric sets, and
- all pairs of metric sets aggregated across techniques.

Given the large number of tests being performed, we set the level of significance to $\alpha = 0.001$. In practice it is useful to not only know the *p*-values, but also the *size* of the effect. Thus, in addition to the Wilcoxon *p*-value on the difference between respectively all pairs of techniques and all pairs of metric sets, we also report effect sizes on these differences using Cohen's *d* (Cohen, 1988).

4. Results

This section reports the results from the assessment procedure that was summarized in Section 3.8. As mentioned in Sections 3.4 and 3.5, a number of different models were built; both using a complete set of independent variables and using a CFS-reduced version of the same metric sets. Surprisingly, the performance of the models that were built using the reduced set of metrics were consistently but marginally poorer than the complete set of metrics across most of the evaluation criteria considered. Consequently, to simplify the already quite complex analyses, and since the results would anyway be very similar, we do not provide separate results for respectively the CFS-reduced models and the non-reduced models, but instead combine the two in one analysis.

First, we give an evaluation of the metric sets and modeling techniques using ROC and CE as we consider these criteria the most appropriate to evaluate prediction models in our context. Then, we show the results when considering a selection of the most popular confusion matrix criteria: accuracy, precision and recall, and Type I- and Type II-misclassification rates. At the end of this section we summarize and discuss the results.

The detailed results are reported in tables that form the basis for our discussion in the following subsections. The tables compare metric sets and modeling techniques against one another in terms of the different evaluation criteria. In the tables we report the mean, standard deviation, minimum and maximum value for each metric set and technique. These descriptive statistics are shown in the leftmost columns of the tables – next to the name of the metric

set or modeling technique. In the right part of the tables we report the difference between each combination of metric set/modeling technique in terms of effect size and theWilcoxon test. The latter appears in the upper right side of the diagonal, while the effect size appears in the lower left side of the diagonal. The effect size is shown in bold face if the corresponding Wilcoxon test is significant at $\alpha = 0.001$. The results for metric sets and techniques are sorted according to their mean values in each table; either descending or ascending depending on whether higher or lower values are better. Finally, the technique and metric set with the highest average rank when considering the ROC area and the four CE measures in combination are included as the "best technique" and "best metric set", respectively. The average results for the best technique are included in the tables that compare the metric sets, whereas the average results for the best metric set is included in the tables that compare the techniques.

4.1. Evaluation of modeling techniques using ROC and CE

Table 2 shows that the differences among techniques in terms of mean ROC area are in most cases very small, or at least too small to be of practical significance. If we were to use the median as a ranking criterion instead, the ranking of the techniques would be similar. The average ROC area ranges from 0.70 for C4.5 to above 0.75 using Decorate C4.5 and Neural network. That is, the probability that a faulty class will be assigned a higher fault probability than a non-faulty one is on average above 0.7, for all modeling techniques. Decorate C4.5 is the data mining technique which has the lowest standard deviation, and thus yields the most stable results regardless of metric set; the minimum is right below 0.6 while the maximum is 0.9, and the standard deviation is 0.08. C4.5 and PART and the combination of the two are perhaps the techniques that yield the models that are the easiest to interpret, as further explained in Arisholm et al. (2008). At the same time, C4.5 and PART are also the ones that yield the smallest ROC area among the techniques assessed in this study; the mean ROC area for C4.5 and PART is significantly smaller than the mean ROC area of the two best techniques. Although C4.5 has the lowest average ROC area overall, the ROC area when using C4.5 in combination with the Process metrics is similar to the mean ROC area using Neural network when not considering any particular metric set, suggesting that C4.5 is in fact a technique that may give fairly good results given that the optimal set of metrics (Process) is used. Considering the ease of interpretation of decision trees, one might

Table 2
Area under ROC curve for the modeling techniques.

	Mean	Std. Dev.	Min	Max	Best metricset (Process)	Neural network	Decorate C4.5	SVM	Logistic regression	Boost C4.5	PART	C4.5 + PART	C4.5
						Effectsize (Wilcoxon ($\alpha = 0.001$))							
Neural network	0.756	0.091	0.543	0.935	0.826	–	0.902	0.811	0.045	0.036	0.001	0.000	0.000
Decorate C4.5	0.752	0.077	0.598	0.899	0.779	0.048	–	0.515	0.109	0.006	0.000	0.000	0.000
SVM	0.749	0.112	0.453	0.942	0.724	0.072	0.034	–	0.556	0.164	0.011	0.004	0.001
Logistic regression	0.737	0.097	0.454	0.919	0.722	0.205	0.174	0.114	–	0.551	0.026	0.013	0.009
Boost C4.5	0.732	0.085	0.510	0.856	0.806	0.279	0.252	0.173	0.057	–	0.006	0.000	0.005
PART	0.708	0.086	0.468	0.861	0.776	**0.548**	**0.543**	0.412	0.317	0.280	–	0.661	0.467
C4.5 + PART	0.703	0.087	0.468	0.862	0.778	**0.599**	**0.599**	0.459	0.370	**0.336**	0.059	–	0.579
C4.5	0.699	0.091	0.470	0.873	0.762	**0.629**	**0.630**	**0.489**	0.403	0.372	0.099	0.041	–

Table 3
Cost-effectiveness for modeling techniques at $\pi = 0.01$ NOS.

	Mean	Std. Dev.	Min	Max	Best metricset (Process)	Logistic regression	Neural network	Decorate C4.5	Boost C4.5	C4.5 + PART	PART	C4.5	SVM
						Effectsize (Wilcoxon ($\alpha = 0.001$))							
Logistic regression	0.137	0.161	−0.043	0.665	0.185	–	0.012	0.024	0.052	0.074	0.025	0.012	0.007
Neural network	0.104	0.197	−0.043	0.807	0.142	0.186	–	0.421	0.724	0.848	0.700	0.661	0.292
Decorate C4.5	0.101	0.230	−0.043	0.870	0.339	0.179	0.010	–	0.445	0.347	0.130	0.833	0.427
Boost C4.5	0.099	0.161	−0.043	0.556	0.254	0.236	0.026	0.013	–	0.742	0.821	0.715	0.361
C4.5 + PART	0.090	0.129	−0.043	0.371	0.160	0.319	0.079	0.059	0.058	–	0.853	0.505	0.510
PART	0.087	0.120	−0.043	0.371	0.139	0.353	0.103	0.080	0.085	0.030	–	0.618	0.349
C4.5	0.080	0.135	−0.043	0.371	0.152	0.383	0.139	0.114	0.126	0.079	0.052	–	0.873
SVM	0.079	0.162	−0.043	0.689	0.153	0.358	0.135	0.112	0.122	0.077	0.053	0.006	–

Table 4
Cost-effectiveness for modeling techniques at $\pi = 0.05$ NOS.

	Mean	Std. Dev.	Min	Max	Best metricset (Process)	Logistic regression	Boost C4.5	PART	Neural network	C4.5 + PART	Decorate C4.5	C4.5	SVM
						Effectsize (Wilcoxon ($\alpha = 0.001$))							
Logistic regression	0.099	0.082	−0.029	0.255	0.160	–	0.055	0.095	0.003	0.130	0.029	0.001	0.000
Boost C4.5	0.076	0.088	−0.037	0.301	0.143	0.272	–	0.878	0.763	0.954	0.584	0.230	0.134
PART	0.074	0.070	−0.037	0.202	0.096	0.333	0.027	–	0.688	0.855	0.274	0.124	0.113
Neural network	0.073	0.085	−0.035	0.263	0.134	0.309	0.029	0.004	–	0.897	0.456	0.225	0.027
C4.5 + PART	0.070	0.085	−0.037	0.239	0.127	0.347	0.066	0.045	0.038	–	0.449	0.138	0.208
Decorate C4.5	0.062	0.085	−0.037	0.294	0.174	0.443	0.160	0.150	0.134	0.097	–	0.518	0.230
C4.5	0.052	0.072	−0.037	0.184	0.097	**0.607**	0.293	0.302	0.270	0.228	0.302	–	0.924
SVM	0.051	0.083	−0.035	0.220	0.113	**0.583**	0.291	0.296	0.268	0.229	0.131	0.017	–

Table 5
Cost-effectiveness for modeling techniques at $\pi = 0.20$ NOS.

	Mean	Std. Dev.	Min	Max	Best metricset (Process)	Boost C4.5	PART	Decorate C4.5	Logistic regression	C4.5 + PART	Neural network	C4.5	SVM
						Effectsize (Wilcoxon ($\alpha = 0.001$))							
Boost C4.5	0.168	0.132	−0.061	0.389	0.289	–	0.956	0.576	0.137	0.717	0.010	0.046	0.017
PART	0.162	0.140	−0.078	0.382	0.302	0.051	–	0.494	0.326	0.463	0.093	0.031	0.068
Decorate C4.5	0.156	0.119	−0.052	0.377	0.300	0.096	0.040	–	0.936	0.897	0.072	0.186	0.021
Logistic regression	0.155	0.154	−0.111	0.458	0.274	0.090	0.041	0.007	–	0.763	0.002	0.464	0.064
C4.5 + PART	0.152	0.148	−0.079	0.423	0.326	0.119	0.068	0.035	0.025	–	0.199	0.063	0.213
Neural network	0.130	0.150	−0.097	0.524	0.286	0.274	0.219	0.196	0.169	0.148	–	0.735	0.518
C4.5	0.129	0.139	−0.092	0.398	0.273	0.294	0.237	0.215	0.183	0.162	0.009	–	0.745
SVM	0.123	0.152	−0.090	0.511	0.230	0.316	0.261	0.241	0.209	0.189	0.042	0.035	–

Table 6
Cost-effectiveness for modeling techniques at $\pi = 1.0$ NOS.

	Mean	Std. Dev.	Min	Max	Best metricset (Process)	Boost C4.5	Decorate C4.5	Neural network	Logistic regression	PART	SVM	C4.5	C4.5 + PART
						Effectsize (Wilcoxon ($\alpha = 0.001$))							
Boost C4.5	0.272	0.208	−0.259	0.607	0.536	–	0.320	0.049	0.037	0.037	0.033	0.000	0.001
Decorate C4.5	0.259	0.236	−0.294	0.650	0.526	0.062	–	0.083	0.051	0.098	0.005	0.007	0.019
Neural network	0.227	0.235	−0.249	0.720	0.535	0.205	0.135	–	0.441	0.839	0.487	0.985	0.584
Logistic regression	0.217	0.247	−0.262	0.674	0.362	0.241	0.171	0.040	–	0.849	0.130	0.907	0.735
PART	0.213	0.243	−0.216	0.656	0.499	0.262	0.191	0.058	0.017	–	0.681	0.441	0.208
SVM	0.200	0.281	−0.331	0.742	0.342	0.292	0.225	0.103	0.064	0.048	–	0.745	0.839
C4.5	0.196	0.252	−0.202	0.636	0.515	**0.333**	0.259	0.129	0.087	0.071	0.018	–	0.811
C4.5 + PART	0.192	0.237	−0.214	0.654	0.510	**0.359**	0.281	0.147	0.103	0.086	0.031	0.013	–

choose this technique if the goal is not only to predict fault-proneness, but also to interpret the model and explain it to practitioners. If the results from using C4.5 are not sufficient, Adaboost can be applied to further improve the model, as the combination of C4.5 and boosting is the technique that yields the best overall ranking across all evaluation criteria.

In Tables 3–6 the data mining techniques are compared using the surrogate measure of cost-effectiveness described in Section 3.7. The difference in average cost-effectiveness between the most and least cost-effective techniques ranges from 0.04 to 0.08 percentage points depending on which threshold π is used. Although there is to some degree a significant difference between the extremes, the differences are negligible considering the uncertainty in the data. Using the optimal set of measures (*Process*), all techniques yield a cost-effectiveness of approximately 30% of the optimal model at $\pi = 0.20$ NOS. Although there is still room for improvement, this is more than three times as cost-effective compared to a model based on random selection.

4.2. Evaluation of metric sets using ROC and CE

As shown in Table 7, the differences in average ROC area between the metric sets (across techniques) are moderate. The average ROC area ranges from 0.65 for Deltas up to 0.77 when using the Process metric set. The Delta metric set is significantly worse than the other combinations of metrics. The ROC area for all but the Delta set is above 0.7.

Though the smallest ROC area (0.45) is obtained when using the Process metrics,[2] this set of metrics is at the same time best in terms of mean and maximum ROC area. Compared to the Process metrics alone, there seems to be no immediate gain by combining them with the OO metrics. However, as can be seen from Table 7, by adding the OO metrics, the minimum ROC area is lifted above 0.6, and the standard deviation is lower.

[2] It is worth noting that all ROC areas below 0.5 were obtained using the CFS-reduced data sets.

If we turn to cost-effectiveness, the results for the metric sets are quite different. In Tables 8–11 we compare the metric sets in terms of cost-effectiveness.

Looking back at (Table 7), we can see that the OO metrics are on par with the Process metrics when considering the ROC area. However from Tables 8–11, we observe that in terms of cost-effectiveness the difference between these two sets of metrics is much larger. At $\pi = 0.20$ NOS (Table 10), the cost-effectiveness using OO metrics are not even 1% of the optimal model, while the cost-effectiveness by using the Process metrics alone are one third of the optimal model, and over three times as cost-effective than the baseline (random model).

As explained in Section 3.8, a number of models were built by using different data mining techniques. Because three separate test sets were applied to the each of these prediction models, we obtained a fairly large number of observations for each metric set. These samples form distributions which we can compare. Fig. 4 depicts the distribution in cost-effectiveness for the prediction models built and evaluated using the Process metrics and the OO metrics, respectively. The plot shows the median cost-effectiveness for each group of prediction models. In addition to the median shown as a solid line, the area between the 25 and 75 percentile is shaded. This visualization can be interpreted as simplified boxplots of the cost-effectiveness when using the two metric sets at discrete levels of NOS. As can be seen from the figure, the distribution in cost-effectiveness using the process metrics is far from the baseline, and nearly not overlapping with the corresponding distribution obtained from using the OO metrics. Looking at the plot for the process metrics, we observe that the 25 percentile for the process metrics are close to 50% *Total faults* at $CE_{0.20}$. This shows that among the models using the process metrics alone, a majority of them (3/4) located more than 50% of the faults in 20% of the most fault-prone classes as predicted by the model. Further, the 75 percentile at $CE_{0.20}$ for the process metrics is at 70% *Total faults*, indicating that 25% of the most cost-effective models in fact identified over 70% of the faults in the 20% most fault-prone classes. This is comparable to the results obtained by Ostrand and Weyuker (2007) and Ostrand et al. (2007, 2005).

Table 7
Area under ROC curve for the metric sets.

	Mean	Std. Dev.	Min	Max	Best technique (Boost C4.5)	Process	Process + OO	Total	Process + Delta	OO + Delta	OO	Delta
						Effectsize (Wilcoxon (α = 0.001))						
Process	0.772	0.097	0.453	0.942	0.806	–	0.968	0.852	0.034	0.041	0.004	0.000
Process + OO	0.768	0.072	0.608	0.915	0.763	0.041	–	0.438	0.004	0.000	0.000	0.000
Total	0.759	0.089	0.546	0.884	0.761	0.132	0.108	–	0.011	0.000	0.000	0.000
Process + Delta	0.736	0.086	0.510	0.929	0.703	0.387	0.402	0.264	–	0.880	0.103	0.000
OO + Delta	0.720	0.080	0.562	0.840	0.736	0.578	**0.627**	**0.460**	0.192	–	0.003	0.000
OO	0.702	0.085	0.532	0.849	0.690	0.761	**0.834**	**0.654**	0.398	0.220	–	0.001
Delta	0.648	0.079	0.468	0.821	0.665	**1.397**	**1.584**	**1.317**	**1.069**	**0.910**	0.659	–

Table 8
Cost-effectiveness for the metric sets at π = 0.01 NOS.

	Mean	Std. Dev.	Min	Max	Best technique (Boost C4.5)	Process	Process + Delta	Process + OO	Total	Delta	OO	OO + Delta
						Wilcoxon (α = 0.001)						
Process	0.190	0.209	−0.043	0.775	0.254	–	0.402	0.030	0.078	0.000	0.000	0.000
Process + Delta	0.175	0.212	−0.043	0.870	0.157	0.072	–	0.279	0.279	0.002	0.000	0.000
Process + OO	0.123	0.151	−0.043	0.511	0.129	0.367	0.281	–	0.630	0.004	0.000	0.000
Total	0.116	0.176	−0.043	0.689	0.109	0.388	0.307	0.048	–	0.068	0.000	0.000
Delta	0.049	0.088	−0.043	0.360	0.008	**0.879**	0.774	0.597	0.475	–	0.263	0.000
OO	0.025	0.071	−0.043	0.208	0.009	**1.061**	**0.950**	**0.832**	**0.676**	0.306	–	0.017
OO + Delta	0.001	0.070	−0.043	0.362	0.025	**1.215**	**1.102**	**1.036**	**0.856**	**0.605**	0.335	–

Table 9
Cost-effectiveness for the metric sets at π = 0.05 NOS.

	Mean	Std. Dev.	Min	Max	Best technique (Boost C4.5)	Process	Process + Delta	Total	Delta	Process + OO	OO + Delta	OO
						Effectsize (Wilcoxon (α = 0.001))						
Process	0.130	0.075	−0.029	0.301	0.143	–	0.227	0.001	0.000	0.000	0.000	0.000
Process + Delta	0.116	0.079	−0.027	0.289	0.117	0.185	–	0.019	0.001	0.004	0.000	0.000
Total	0.083	0.085	−0.037	0.255	0.105	**0.590**	0.404	–	0.479	0.177	0.000	0.000
Delta	0.071	0.070	−0.026	0.201	0.027	**0.817**	**0.606**	0.156	–	1.000	0.000	0.000
Process + OO	0.071	0.081	−0.037	0.258	0.102	**0.761**	0.566	0.147	0.001	–	0.000	0.000
OO + Delta	0.009	0.044	−0.037	0.163	0.035	**1.957**	**1.669**	**1.087**	**1.048**	**0.938**	–	0.939
OO	0.006	0.035	−0.037	0.101	0.000	**2.121**	**1.810**	**1.192**	**1.177**	**1.042**	0.096	–

Table 10
Cost-effectiveness for the metric sets at π = 0.20 NOS.

	Mean	Std. Dev.	Min	Max	Best technique (Boost C4.5)	Process	Process + Delta	Delta	Total	Process + OO	OO + Delta	OO
						Effectsize (Wilcoxon (α = 0.001))						
Process	0.285	0.088	0.102	0.524	0.289	–	0.000	0.000	0.000	0.000	0.000	0.000
Process + Delta	0.233	0.092	0.041	0.389	0.276	**0.574**	–	0.005	0.000	0.000	0.000	0.000
Delta	0.183	0.140	−0.030	0.458	0.147	**0.874**	0.426	–	0.936	0.141	0.000	0.000
Total	0.170	0.112	−0.071	0.387	0.207	**1.143**	**0.619**	0.103	–	0.023	0.000	0.000
Process + OO	0.129	0.121	−0.076	0.331	0.192	**1.476**	**0.971**	0.410	0.348	–	0.000	0.000
OO + Delta	0.022	0.091	−0.106	0.261	0.057	**2.933**	**2.303**	**1.356**	**1.440**	**0.997**	–	0.320
OO	0.007	0.078	−0.111	0.222	0.011	**3.343**	**2.652**	**1.550**	**1.683**	**1.202**	0.184	–

Table 11
Cost-effectiveness for the metric sets at π = 1.0 NOS.

	Mean	Std. Dev.	Min	Max	Best technique (Boost C4.5)	Process	Process + Delta	Delta	Total	Process + OO	OO + Delta	OO
						Effectsize (Wilcoxon (α = 0.001))						
Process	0.478	0.165	−0.122	0.742	0.536	–	0.000	0.000	0.000	0.000	0.000	0.000
Process + Delta	0.394	0.107	0.119	0.669	0.372	**0.604**	–	0.000	0.000	0.000	0.000	0.000
Delta	0.236	0.195	−0.216	0.674	0.236	**1.343**	**1.008**	–	0.400	0.701	0.000	0.000
Total	0.224	0.219	−0.213	0.531	0.318	**1.308**	**0.984**	0.054	–	0.479	0.000	0.000
Process + OO	0.199	0.194	−0.223	0.470	0.273	**1.552**	**1.247**	0.190	0.125	–	0.000	0.000
OO + Delta	0.037	0.185	−0.306	0.357	0.137	**2.512**	**2.358**	**1.045**	**0.925**	**0.853**	–	0.004
OO	−0.013	0.178	−0.331	0.294	0.036	**2.863**	**2.774**	**1.334**	**1.191**	**1.139**	0.275	–

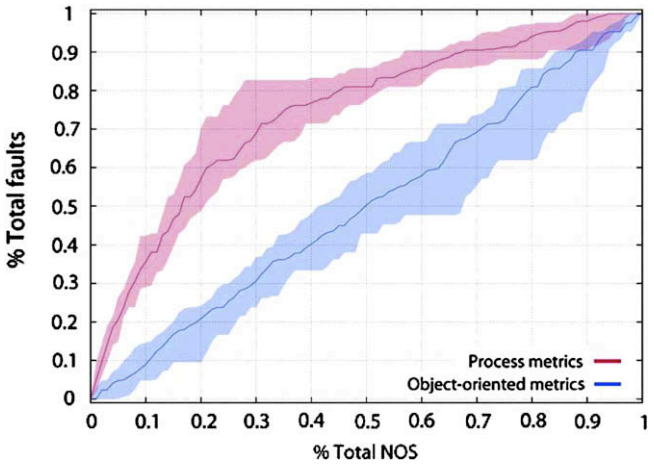

Fig. 4. Median and 25 percentile/75 percentile for process metrics and object-oriented metrics.

Fig. 4 supports the results in the tables comparing metric sets, showing that the cost-effectiveness obtained by using the OO metrics is close to zero. It is worth noting that there are in fact a large number of models using the OO metrics that have negative cost-effectiveness: the median of the OO metrics is close to the baseline with slope 1 indicating that 50% of the observations are below this baseline, and thus these models are not more cost-effective than a completely random model. It is interesting that the average cost-effectiveness for OO metrics is close to zero across all thresholds. Turning back to Tables 8–11, note also that the cost-effectiveness of the models built using other metric sets decreases when the OO metrics are added. For example, this is visible when comparing the cost-effectiveness of the process metrics with that of the process metrics in combination with the OO metrics (Process + OO): The process metrics are consistently more cost-effective, but when adding the OO metrics, this combination is consistently ranked among the least cost-effective. That is, adding the OO metrics consistently degrades the cost-effectiveness of a model. Further, we observe that although the deltas have the smallest average ROC area, these metrics are consistently more cost-effective than the OO metrics. The low cost-effectiveness of the OO metrics may be due to their correlation with size measures, which has been reported in many other papers (El Emam et al., 2001).

If we were to use the prediction models to focus verification and validation efforts by, say, inspecting the 20% most fault-prone classes – the gain from using the process metrics (finding 60% of the faults on average) compared to the average of what would be obtained with random orders (finding 20% of the faults) is substantial. Of course, this is a somewhat simplified view for both scenarios, as we probably cannot expect to find *all* faults by applying a particular fault-proneness model to focus verification and validation.[3] Still, the gain from using a prediction model based on process metrics is substantial compared with the baseline model. On the other hand, we also see that there is much room for improvement when compared to an optimal ranking of the classes: the best model is approximately 50% of the optimal model in terms of cost-effectiveness.

The results show that the OO metrics are good predictors of faulty classes (i.e., large ROC area), but these metrics do not result in cost-effective prediction models. Many OO metrics have been shown to be associated with size (El Emam et al., 2001), and this fact might explain the low cost-effectiveness of the OO metrics, because the surrogate measure for cost-effectiveness penalize models which mostly capture a size effect. Although the process metrics are presumably more expensive to collect, the results show that collecting process metrics is likely to be cost-effective.

[3] A suitable cost-benefit model that accounts for the percentage of faults that are not discovered during verification efforts is given in Briand and Wust (2002).

4.3. Evaluating techniques and metric sets using other evaluation criteria

In the two previous subsections, metric sets and modeling techniques were compared using two evaluation criteria: ROC area and cost-effectiveness (CE). This section presents the results when using some of the more commonly used evaluation criteria. More specifically, we will consider the most popular measures that can be derived from the confusion matrix as explained in Section 3.7. We did not investigate in detail how these classification accuracy measures are affected by different probability cut-off values. Still, the results given in this section are comparable to most studies, which in most cases do not vary the threshold, but rather use the default value of 0.5 (Arisholm et al., 2008). We first consider *accuracy* as it is the most prominent measure in the studies reviewed. Then, we show our results for *precision*, *recall* and *Type I-* and *Type II-misclassification rates* as these evaluation criteria are also widely used (Arisholm et al., 2008).

One of the conclusions in the two previous subsections was that the Process metrics set seems to be the overall best metric set and Boost C4.5 the best modeling technique in terms of average ROC area and cost-effectiveness. Consequently, to facilitate comparisons with the previous subsections, we still show the Process/Boost C4.5 results in a separate column.

4.3.1. Accuracy

Tables 12 and 13 show the accuracy for modeling techniques and metric sets, respectively. As higher accuracy is considered better than lower accuracy, the tables are sorted in descending order according to the mean values.

The differences in accuracy among modeling techniques are smaller than the differences among metric sets. If one were to select a particular modeling technique based on the average accuracy, one would probably select SVM or logistic regression, although these techniques yield lower accuracy when used in conjunction with the optimal metric set (Process).

It is worth pointing that the Delta metric set yields the highest accuracy. Looking at the results for ROC area in Table 7 in Section 4.2, Delta was the metric set giving the smallest average ROC area, and thus one would probably conclude that using these metrics to predict fault-proneness is not optimal, thus running counter to what one would conclude when considering the accuracy measure.

Furthermore, what is considered the best metric set is highly dependent on which cut-off that is used. Here we have used a threshold of 0.5 because it is commonly used in the existing literature, however, it is difficult to give a rule of thumb as to what cut-off to use because there would probably be large variations across studies as these results are highly dependent on properties of the data set. In our case, the most accurate models are obtained when using cut-off values above 0.8. This is due to the highly unbalanced nature of our data sets: only a small percentage of the classes are faulty. Although high accuracy is intuitively a desired property, our results suggest that accuracy is not necessarily an appropriate measure for evaluating how useful fault-proneness prediction models are.

4.3.2. Precision and recall

Two other evaluation criteria that are widely used are the *precision* and *recall* measures, as explained in Section 3.7. Tables 14 and 15 show the results for these measures using the different metric sets. The metric sets are sorted in descending order according to their mean precision/recall.

Table 12
Accuracy of modeling techniques.

	Mean	Std. Dev.	Min	Max	Best metricset (Process)	SVM	Logistic regression	C4.5 + PART	Neural network	Decorate C4.5	C4.5	Boost C4.5	PART
						Effectsize (Wilcoxon ($\alpha = 0.001$))							
SVM	0.863	0.061	0.744	0.985	0.869	–	0.000	0.145	0.007	0.000	0.000	0.000	0.000
Logistic regression	0.845	0.060	0.753	0.983	0.867	0.295	–	0.830	0.265	0.007	0.004	0.000	0.000
C4.5 + PART	0.838	0.105	0.650	0.970	0.934	0.287	0.077	–	0.806	0.000	0.000	0.000	0.000
Neural network	0.830	0.090	0.681	0.986	0.916	0.432	0.199	0.089	–	0.017	0.014	0.000	0.000
Decorate C4.5	0.807	0.104	0.634	0.970	0.915	0.652	**0.443**	0.298	0.230	–	0.059	0.002	0.000
C4.5	0.793	0.125	0.568	0.969	0.912	0.709	**0.527**	0.391	0.334	0.122	–	0.494	0.270
Boost C4.5	0.783	0.107	0.658	0.961	0.903	**0.925**	**0.719**	**0.528**	0.477	0.234	0.092	–	0.452
PART	0.771	0.125	0.526	0.964	0.901	**0.932**	**0.750**	**0.582**	**0.537**	0.314	0.177	0.099	–

Table 13
Accuracy of metric sets.

	Mean	Std. Dev.	Min	Max	Best technique (Boost C4.5)	Delta	Process	Process + Delta	Total	Process + OO	OO + Delta	OO
						Effectsize (Wilcoxon ($\alpha = 0.001$))						
Delta	0.908	0.085	0.739	0.986	0.889	–	0.367	0.000	0.000	0.000	0.000	0.000
Process	0.902	0.050	0.744	0.982	0.903	0.089	–	0.000	0.000	0.000	0.000	0.000
Process + Delta	0.871	0.070	0.760	0.971	0.868	**0.475**	**0.504**	–	0.000	0.000	0.000	0.000
Total	0.797	0.084	0.612	0.945	0.684	**1.319**	**1.519**	**0.959**	–	0.351	0.000	0.000
Process + OO	0.776	0.070	0.642	0.899	0.711	**1.697**	**2.065**	**1.354**	0.267	–	0.031	0.000
OO + Delta	0.744	0.085	0.526	0.896	0.745	**1.925**	**2.252**	**1.622**	**0.620**	0.408	–	0.037
OO	0.715	0.074	0.568	0.834	0.680	**2.420**	**2.947**	**2.156**	**1.029**	**0.845**	0.364	–

Table 14
Precision for the metric sets.

	Mean	Std. Dev.	Min	Max	Best technique (Boost C4.5)	Delta	Process	Process + Delta	Total	Process + OO	OO + Delta	OO
						Effectsize (Wilcoxon ($\alpha = 0.001$))						
Delta	0.104	0.094	0.040	0.429	0.076	–	0.288	0.000	0.000	0.000	0.000	0.000
Process	0.082	0.047	0.020	0.273	0.082	0.294	–	0.000	0.000	0.000	0.000	0.000
Process + Delta	0.067	0.035	0.019	0.160	0.061	**0.521**	**0.362**	–	0.000	0.000	0.000	0.000
Total	0.044	0.021	0.014	0.101	0.030	**0.871**	**1.024**	**0.768**	–	0.486	0.000	0.000
Process + OO	0.039	0.020	0.013	0.110	0.031	**0.941**	**1.162**	**0.942**	0.223	–	0.000	0.000
OO + Delta	0.032	0.014	0.013	0.061	0.031	**1.063**	**1.426**	**1.288**	**0.671**	**0.432**	–	0.012
OO	0.029	0.013	0.013	0.058	0.025	**1.108**	**1.520**	**1.411**	**0.854**	**0.620**	0.227	–

Table 15
Recall (or Sensitivity, TP rate) for the metric sets.

	Mean	Std. Dev.	Min	Max	Best technique (Boost C4.5)	Process + OO	Total	OO	OO + Delta	Process + Delta	Process	Delta
						Effectsize (Wilcoxon ($\alpha = 0.001$))						
Process + OO	0.623	0.113	0.389	0.889	0.677	–	0.689	0.925	0.252	0.000	0.000	0.000
Total	0.612	0.138	0.278	0.833	0.723	0.087	–	0.752	0.490	0.000	0.000	0.000
OO	0.609	0.117	0.333	0.781	0.597	0.122	0.023	–	0.408	0.001	0.000	0.000
OO + Delta	0.593	0.137	0.361	0.833	0.609	0.235	0.134	0.122	–	0.005	0.000	0.000
Process + Delta	0.518	0.175	0.167	0.755	0.556	**0.712**	**0.595**	**0.611**	0.480	–	0.203	0.000
Process	0.492	0.162	0.139	0.833	0.554	**0.936**	**0.794**	**0.826**	**0.674**	0.151	–	0.000
Delta	0.362	0.160	0.056	0.616	0.429	**1.884**	**1.671**	**1.762**	**1.552**	**0.929**	**0.810**	–

From Table 14 we see that the precision ranges from 3% to approximately 10%. This indicates that when using a cut-off of 0.5 to distinguish faulty classes from non-faulty ones, only a small part of the fault-prone classes identified by the prediction model is in fact faulty – that is, most of the classes predicted as faulty are false positives. Although the maximum for Delta is above 0.4, the precision of our models is much lower than comparable studies who typically achieved precision in the range of 0.7 to 0.95 (Elish and Elish, 2008; Pai and Dugan, 2007; Denaro and Pezze, 2002). The reason we get a relatively low precision is probably because only 0.5–2% of the classes in our data sets are in fact faulty. Thus, even a few false positives have a huge impact on the precision of the prediction models.

Table 15 shows the corresponding results for *recall*. We see that the models typically capture somewhere between 36% and 62% of the faulty classes on average using a cut-off equal to 0.5. This is comparable to other studies, e.g. (Kim et al., 2008; Pai and Dugan, 2007; Denaro and Pezze, 2002), while other studies achieved recall close to 1 (Elish and Elish, 2008). With respect to recall, the Total metric set is best, and looking at the results when using the overall

best modeling technique (Boost C4.5) in combination with the total set of metrics, we observe that 72% of the faults are captured on average by these models.

Among the modeling techniques, the differences in average precision are small, typically in the range from 0.05 to 0.07 (Table 16).The rule- and tree-based modeling techniques are techniques that seem to yield low precision, whereas these techniques are at the same time those that yield higher recall than SVM, neural network and logistic regression (Table 17).

4.3.3. Types I and II misclassification rates

Ostrand and Weyuker argue that Type II errors are the most expensive, and that prediction models should be selected and evaluated by their *Type II misclassification rate* (Ostrand and Weyuker, 2007). This measure is also used by Khoshgoftaar and Seliya (2004, 2003). In Table 18 we report the average Type II misclassification rate for each technique using a default cut-off equal to 0.5. As smaller numbers are considered better (less errors in predictions), the table is sorted in ascending order according to the average for each technique.

The Type II misclassification rate is typically small, suggesting that a large part of the prediction models assigns a predicted fault probability above 0.5 to most of the faulty classes. Our Type II misclassification rates are slightly smaller (better) than those reported in earlier studies, where this rate typically ranged from 0.01 (Ostrand and Weyuker, 2007) to 0.3 (Kanmani et al., 2007). Although the differences among the modeling techniques presented here are small, if we were to select a particular technique based on the results in this table, we would conclude that the decision trees or rule-based techniques, i.e., C4.5 (with or without boosting) or PART, yield the best prediction models in terms of Type II misclassification rates. This contradicts our conclusion based on the ROC area in Section 4.1.

Because the Types I and II misclassification rates are inversely correlated – that is, in most cases decreasing the number of Type II errors leads to an increase in the number of Type I errors – it is useful to compare the results in Table 18 with the Type I misclassification rates given in Table 19. Table 19 clearly illustrates that modeling techniques that have lower Type II misclassification rates have higher Type I misclassification rates. If we were to select the modeling technique that would yield best results in terms of Type I misclassification rates, we would probably choose another modeling technique than when considering Type II misclassification rates. That is, considering Type II misclassification rates we concluded that the rule- or decision tree-based techniques were best, while from Table 19 we conclude that these are signif-

Table 16
Precision for each of the modeling techniques.

	Mean	Std. Dev.	Min	Max	Best metricset (Process)	SVM	Neural network	C4.5 + PART	Logistic regression	Decorate C4.5	C4.5	Boost C4.5	PART
						Effectsize (Wilcoxon ($\alpha = 0.001$))							
SVM	0.073	0.081	0.019	0.429	0.069	–	0.009	0.290	0.007	0.005	0.021	0.000	0.000
Neural network	0.064	0.074	0.016	0.429	0.107	0.125	–	0.471	0.954	0.378	0.132	0.014	0.001
C4.5 + PART	0.059	0.037	0.015	0.175	0.097	0.226	0.078	–	0.526	0.051	0.000	0.000	0.000
Logistic regression	0.058	0.050	0.020	0.316	0.062	0.233	0.095	0.033	–	0.300	0.267	0.025	0.006
Decorate C4.5	0.053	0.037	0.014	0.150	0.084	0.316	0.176	0.155	0.097	–	0.138	0.001	0.000
C4.5	0.052	0.033	0.013	0.143	0.078	0.341	0.201	**0.198**	0.129	0.035	–	0.171	0.139
Boost C4.5	0.048	0.032	0.014	0.131	0.082	**0.412**	0.278	**0.326**	0.233	0.160	0.131	–	0.363
PART	0.047	0.036	0.013	0.148	0.075	**0.416**	0.284	**0.329**	0.241	**0.171**	0.143	0.019	–

Table 17
Recall for each of the modeling techniques.

	Mean	Std. Dev.	Min	Max	Best metricset (Process)	Boost C4.5	PART	C4.5	Decorate C4.5	Neural network	Logistic regression	SVM	C4.5 + PART
						Effectsize (Wilcoxon ($\alpha = 0.001$))							
Boost C4.5	0.592	0.155	0.222	0.833	0.554	–	0.492	0.268	0.055	0.000	0.000	0.000	0.000
PART	0.571	0.151	0.278	0.833	0.482	0.139	–	0.706	0.694	0.022	0.015	0.003	0.000
C4.5	0.570	0.161	0.194	0.795	0.488	0.140	0.005	–	0.776	0.014	0.007	0.001	0.000
Decorate C4.5	0.567	0.158	0.167	0.781	0.465	0.163	0.027	0.021	–	0.003	0.013	0.003	0.000
Neural network	0.522	0.194	0.056	0.778	0.467	**0.399**	0.280	0.269	0.251	–	0.426	0.134	0.918
Logistic regression	0.519	0.164	0.167	0.778	0.548	**0.461**	0.331	0.317	0.298	0.020	–	0.379	0.904
SVM	0.507	0.192	0.111	0.889	0.494	**0.487**	0.368	**0.355**	0.338	0.078	0.065	–	0.356
C4.5 + PART	0.505	0.155	0.194	0.775	0.440	**0.561**	**0.428**	**0.410**	**0.392**	0.096	0.083	0.010	–

Table 18
Type II misclassification rates for each modeling technique.

	Mean	Std. Dev.	Min	Max	Best metric set (Process)	Boost C4.5	PART	Decorate C4.5	C4.5	Neural network	Logistic regression	C4.5 + PART	SVM
						Effectsize (Wilcoxon ($\alpha = 0.001$))							
Boost C4.5	0.005	0.003	0.001	0.011	0.006	–	0.758	0.144	0.132	0.000	0.000	0.000	0.000
PART	0.006	0.003	0.001	0.010	0.007	0.072	–	0.350	0.272	0.012	0.034	0.000	0.006
Decorate C4.5	0.006	0.003	0.002	0.011	0.007	0.118	0.045	–	0.598	0.011	0.020	0.000	0.002
C4.5	0.006	0.003	0.002	0.011	0.007	0.142	0.071	0.027	–	0.034	0.032	0.000	0.001
Neural network	0.006	0.003	0.002	0.013	0.007	**0.311**	0.247	0.208	0.181	–	0.831	0.965	0.061
Logistic regression	0.006	0.003	0.002	0.012	0.006	**0.337**	0.271	0.231	0.203	0.015	–	0.961	0.062
C4.5 + PART	0.006	0.003	0.002	0.011	0.008	**0.411**	**0.342**	**0.301**	**0.269**	0.069	0.055	–	0.203
SVM	0.007	0.003	0.002	0.012	0.007	**0.420**	0.358	0.320	**0.292**	0.106	0.094	0.045	–

Table 19
Type I misclassification rates for each modeling technique.

	Mean	Std. Dev.	Min	Max	Best metric set (Process)	SVM	Logistic regression	C4.5 + PART	Neural network	Decorate C4.5	C4.5	Boost C4.5	PART
						Effectsize (Wilcoxon ($\alpha = 0.001$))							
SVM	0.130	0.061	0.003	0.248	0.125	–	0.000	0.162	0.006	0.000	0.000	0.000	0.000
Logistic regression	0.149	0.061	0.005	0.242	0.127	**0.297**	–	0.839	0.276	0.006	0.004	0.000	0.000
C4.5 + PART	0.155	0.106	0.020	0.346	0.058	0.287	0.075	–	0.787	0.000	0.000	0.000	0.000
Neural network	0.164	0.091	0.002	0.311	0.078	0.433	0.198	0.090	–	0.019	0.016	0.000	0.000
Decorate C4.5	0.187	0.105	0.020	0.362	0.077	**0.657**	0.445	**0.302**	0.233	–	0.060	0.002	0.000
C4.5	0.201	0.126	0.021	0.426	0.081	**0.712**	0.528	**0.394**	0.336	0.120	–	0.452	0.259
Boost C4.5	0.212	0.108	0.029	0.336	0.091	**0.933**	**0.724**	**0.534**	**0.481**	0.236	0.095	–	0.483
PART	0.223	0.126	0.026	0.470	0.092	**0.935**	**0.750**	**0.585**	**0.537**	**0.312**	0.177	0.095	–

icantly worse than SVM. In practice, one would probably consider a trade-off between these types of misclassification rates. One option is to investigate the consistency in ranking for each technique across the evaluation criteria. Then, neural networks would perhaps be a good compromise.

As can be seen from the results above, which modeling technique or metric set can be considered "best" is highly dependent on the criteria used for evaluation. The prediction models in this case study yield a recall and accuracy comparable to recent studies. However, the precision of our models is very low due to the unbalanced nature of our data sets, and choosing another cut-off than 0.5 can possibly yield very different results.

4.4. Discussion

In the subsections above we have evaluated and compared several carefully selected modeling techniques and metric sets that entail different data collection costs. Our goal was to assess what measures are necessary to achieve practically useful predictions, what modeling techniques seem to be more helpful, and how our conclusions differ depending on the evaluation criteria used.

We observe that the *Process* measures on average yield the most cost-effective prediction models, whereas the *OO* metrics on average is no better than a model based on random selection of classes. Although the *Delta* measures alone does not yield particularly large ROC areas, these measures still yield more cost-effective prediction models than the *OO* metrics.

Turning to the evaluation criteria, a first observation is that using general confusion matrix criteria raises a number of issue: (i) it is difficult to assess if the default cut-off of 0.5 is appropriate and if not, what other cut-off value should be used; (ii) none of these criteria strongly relate to the main goal in our context, that is *ranking* classes according to their fault-proneness to prioritize and increase the cost-effectiveness of verification; (iii) none of these criteria are clearly related to the possible cost-effectiveness of applying a particular prediction model.

Further, another issue when evaluating prediction models is that what can be considered the best modeling technique or set of measures is highly dependent on the evaluation criteria used for evaluation. Consequently, it is crucial that the criteria used to evaluate fault-proneness prediction models are closely linked to the intended, practical application of the prediction models.

We argue that ROC and CE capture two properties that are of high importance within our context, namely class ranking and cost-effectiveness: The area under the ROC curve reflects the probability that a faulty class is assigned a higher fault probability than a non-faulty one, while the CE measure allows us to compare prediction models according to their cost-effectiveness based on a number of assumptions. As shown in Section 4.2, these two measures capture two different dimensions of model performance: The difference between the Process and the OO metric sets was not clearly visible when only considering the ROC area, whereas the differences considering CE were relatively large. The results showed that an apparently accurate model is not necessarily cost-effective. Consequently, we emphasize the importance of considering not only measures such as the ones that can be derived from the confusion matrix, but also specific measures that are more closely related to the possible cost-effectiveness of applying fault-proneness prediction models to focus verification efforts.

5. Threats to validity

The evaluation of techniques and metric sets were done using data from one single environment. The data collected were from 13 major releases over a period of several years. The system has endured a large extent of organizational and personnel change. Thus, it is unlikely that the results are heavily affected by individual developers and their experience, or the traits of certain releases of the system. Still, as with most case studies, one should be careful to generalize the specific results to all systems or environments. However, at a more general level, we believe that many methodological lessons can be learned from this study, including the need for doing systematic and comprehensive evaluations to ensure that the prediction models have the desired properties (e.g., cost-effectiveness) for the purpose at hand. Moreover, in many ways, the environment we worked in has the typical characteristics of telecom software development: parallel patches and new feature development, configuration management and fault reporting systems, Java technologies, and a significant level of personnel turnover.

In this study, we have not accounted for the actual cost of making the measures available and collecting them. Consequently, there are some initial costs associated with this process improvement activity that we do not account for. In particular, the Process metric set, being most cost-effective, is at the same time the measures that have the highest cost with respect to data reporting and collection.

The prediction models built in this case study were built using default parameters. That is, we have not systematically investigated how the models are affected by varying the parameters. There are possibly a large number of potential combinations of parameters for each modeling technique and optimizing the parameters with respect to some criteria for each technique would be very computational intensive. Furthermore, optimizing the modeling parameters might also lead to overfitted models that is highly specific to the training set. One way to alleviate this potential threat would be to apply evolutionary programming to optimize the parameters with respect to some property, e.g., cross-validated measures of ROC or CE.

Note also that the use of statistical tests in this study to test the differences between techniques and metric sets are somewhat exploratory in nature. In particular, from a formal standpoint, the notion of p-values is questionable in our context, because we have not taken a random sample from a target population, but rather

used all the data we had available and computed *p*-values on differences between subsets of our data. For this reason we have also reported effect sizes, which are not problematic in this regard.

6. Conclusions and further work

Our review of recent studies (Arisholm et al., 2008) revealed that many studies do not comprehensively and systematically compare modeling techniques and types of measures to build fault prediction models. Many works also do not apply suitable evaluation methods and show little consistency in terms of criteria and methods that are used to evaluate the prediction models. Thus, it is hard to draw general conclusions on which measures and modeling techniques to use to build fault-proneness prediction models based on the existing body of studies. Further, most studies evaluate their models using confusion matrix criteria while we have shown that the metric set or technique that is put forward as the best is highly dependent on the specific criteria used.

Except for a few studies, i.e. (Elish and Elish, 2008; Guo et al., 2004), there has been no systematic and comprehensive effort on comparing modeling techniques to build accurate and useful fault-proneness prediction models. In this paper, we do not only compare a carefully selected set of modeling techniques in a systematic way, but we also compare the impact of using different types of measures as predictors, based on different evaluation criteria. By doing so, we also propose a systematic process and associated data analysis procedures for the rigorous comparison of models in terms of their cost-effectiveness.

More precisely, we have empirically evaluated all combinations of three distinct sets of candidate measures (OO structural measures, code churn measures, and process change and fault measures) and eight, carefully selected modeling techniques, using a number of evaluation criteria. Overall, the findings are that the measures and techniques that are put forward as the "best" is highly dependent on the evaluation criteria applied. Thus, it is important that the evaluation criteria used to evaluate the prediction models are clearly justified in the context in which the models are to be applied.

Within the field of software verification we propose a surrogate measure of cost-effectiveness (CE) that enables us to assess and compare the possible benefits of applying fault-proneness prediction models to focus software verification efforts, e.g., by ranking the classes according to fault-proneness and focusing unit testing on the π% most fault-prone components. Using this CE measure to evaluate the prediction models in our case study revealed that using OO metrics to build fault-proneness prediction models does not necessarily yield cost-effective models – possibly because these metrics show strong correlation with size related measures, and prediction models that merely capture size are not cost-effective under the assumption that verification costs are proportional to size. Further, this case study clearly suggests that one should consider process-related measures, such as measures related to the history of changes and faults, to improve prediction model cost-effectiveness. Regarding the choice of modeling technique, the differences appear to be rather small in terms of cost-effectiveness, although Adaboost combined with C4.5 overall gave the best results. Note however that we have only compared techniques using default parameters, and as future work we will try to optimize the parameters while attempting to avoid overfitting.

The CE measure proposed in this paper is a *surrogate* measure to facilitate comparisons of prediction models using a criterion that is directly linked to the assumed cost-effectiveness of using such models to focus verification efforts. In order to assess the *real* cost-effectiveness and possible return on investment, we have recently performed a pilot study where the C4.5 prediction model was applied in a new release of the COS system. In this pilot study, developers spent an additional week of unit testing on the most fault-prone classes and several serious faults that otherwise would have slipped through to later testing phases or even the production system was discovered and corrected. Preliminary results suggest a return of investment of about 100% by preventing these faults from slipping through to later phases where they would have been more expensive to correct (Fuglerud, 2007). Due to these promising preliminary results, plans are underway to perform large-scale evaluations of the costs and benefits of using the prediction models to focus testing in the COS project.

References

Arisholm, E., Briand, L.C., Fuglerud, M., 2007. Data mining techniques for building fault-proneness models in Telecom Java Software. In: The 18th IEEE International Symposium on Software Reliability, 2007. ISSRE '07, pp. 215–224.

Arisholm, E., Briand, L.C., Johannessen, E., 2008. Data mining techniques, candidate measures and evaluation methods for building practically useful fault-proneness prediction models. Simula Research Laboratory Technical Report 2008-06.

Briand, L.C., Wuest, J., 2002. Empirical studies of quality models in object-oriented systems. Advances in Computers 59, 97–166.

Briand, L.C., Wust, J., 2002. Empirical studies of quality models in object-oriented systems. Advances in Computers 56, 97–166.

Briand, L.C., Melo, W.L., Wust, J., 2002. Assessing the applicability of fault-proneness models across object-oriented software projects. IEEE Transactions on Software Engineering 28, 706–720.

Cohen, J., 1988. Statistical Power Analysis for the Behavioral Sciences, second ed. Lawrence Erlbaum Associates.

Collofello, J.S., Woodfield, S.N., 1989. Evaluating the effectiveness of reliability-assurance techniques. Journal of Systems and Software 9, 191–195.

Denaro, G., Pezze, M., 2002. An empirical evaluation of fault-proneness models. In: 24th International Conference on Software Engineering, pp. 241–251.

El Emam, K., Benlarbi, S., Goel, N., Rai, S.N., 2001. The confounding effect of class size on the validity of object-oriented metrics. IEEE Transactions on Software Engineering 27, 630–650.

Elish, K.O., Elish, M.O., 2008. Predicting defect-prone software modules using support vector machines. Journal of Systems and Software 81, 649–660.

Friedman, J.H., 1991. Multivariate adaptive regression splines. Annals of Statistics 19, 1–67.

Fuglerud, M.J., 2007. Implementing and evaluating a fault-proneness prediction model to focus testing in a Telecom Java Legacy System. M.Sc. Thesis, Dept. of Informatics, University of Oslo.

Gondra, I., 2008. Applying machine learning to software fault-proneness prediction. Journal of Systems and Software 81, 186–195.

Graves, T.L., Karr, A.F., Marron, J.S., Siy, H., 2000. Predicting fault incidence using software change history. IEEE Transactions on Software Engineering 26, 653–661.

Guo, L., Ma, Y., Cukic, B., Singh, H., 2004. Robust prediction of fault-proneness by random forests. In: 15th International Symposium on Software Reliability Engineering, pp. 417–428.

Hall, M., 2000. Correlation-based feature selection for discrete and numeric class machine learning. In: 17th International Conference on Machine Learning, pp. 359–366.

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

JHawk, http://www.virtualmachinery.com/jhawkprod.htm.

Kanmani, S., Uthariaraj, V.R., Sankaranarayanan, V., Thambidurai, P., 2007. Object-oriented software fault prediction using neural networks. Information and Software Technology 49, 483–492.

Khoshgoftaar, T.M., Allen, E.B., 2001. Modeling software quality with classification trees. In: Pham, H. (Ed.), Recent Advances in Reliability and Quality Engineering, vol. 2. World Scientific Publishing, Singapore, pp. 247–270.

Khoshgoftaar, T.M., Kehan, G., 2007. Count models for software quality estimation. IEEE Transactions on Reliability 56, 212–222.

Khoshgoftaar, T.M., Seliya, N., 2003. Analogy-based practical classification rules for software quality estimation. Empirical Software Engineering 8, 325–350.

Khoshgoftaar, T.M., Seliya, N., 2004. Comparative assessment of software quality classification techniques: an empirical case study. Empirical Software Engineering 9, 229–257.

Kim, S., Whitehead, J.E.J., Zhang, Y., 2008. Classifying software changes: clean or buggy? IEEE Transactions on Software Engineering 34, 181–196.

NASA Metrics Data Program Repository, <http://mdp.ivv.nasa.gov/>.

Nagappan, N., Ball, T., 2005. Use of relative code churn measures to predict system defect density. In: 27th International Conference on Software Engineering, St. Louis, MO, USA.

Nagappan, N., Ball, T., 2007. Using software dependencies and churn metrics to predict field failures: an empirical case study. In: First International Symposium on Empirical Software Engineering and Measurement, pp. 364–373.

Ostrand, T.J., Weyuker, E.J., 2007. How to measure success of fault prediction models. In: Fourth International Workshop on Software Quality Assurance (SOQUA), ACM, Dubrovnik, Croatia.
Ostrand, T.J., Weyuker, E.J., Bell, R.M., 2005. Predicting the location and number of faults in large software systems. IEEE Transactions on Software Engineering 31, 340–355.
Ostrand, T.J., Weyuker, E.J., Bell, R.M., 2005. Predicting the location and number of faults in large software systems. IEEE Transactions on Software Engineering 31, 340–355.
Ostrand, T.J., Weyuker, E.J., Bell, R.M., 2007. Automating algorithms for the identification of fault-prone files. In: 2007 International Symposium on Software Testing and Analysis, London, United Kingdom.
Pai, G.J., Dugan, J.B., 2007. Empirical analysis of software fault content and fault proneness using Bayesian methods. IEEE Transactions on Software Engineering 33, 675–686.
Vandecruys, O., Martens, D., Baesens, B., Mues, C., De Backer, M., Haesen, R., 2008. Mining software repositories for comprehensible software fault prediction models. Journal of Systems and Software 81, 823–839.
Weyuker, E.J., Ostrand, T.J., Bell, R.M., 2007. Using developer information as a factor for fault prediction. In: International Workshop on Predictor Models in Software Engineering.
Witten, I., Frank, E., 2005. Data Mining: Practical Machine Learning Tools and Techniques, second ed. Morgan Kaufman.

Erik Arisholm received the MSc degree in electrical engineering from University of Toronto and the PhD degree in software engineering from University of Oslo. He has several years of industry experience in Canada and Norway as a lead engineer and design manager. He is now a professor in software engineering at Simula Research Laboratory and University of Oslo.

His current research interests include model-driven development, verification and software evolution, with an emphasis on empirical studies.

Lionel C. Briand is a professor of software engineering at the Simula Research laboratory and University of Oslo, leading the project on software verification and validation. Before that, he was on the faculty of the department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, where he was full professor and held the Canada Research Chair in Software Quality Engineering. He has also been the software quality engineering department head at the Fraunhofer Institute for Experimental Software Engineering, Germany, and 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, USA.

He has been on the program, steering, or organization committees of many international, IEEE and ACM conferences.

He is editor-in-chief of Empirical Software Engineering (Springer) and is a member of the editorial boards of Systems and Software Modeling (Springer) and Software Testing, Verification, and Reliability (Wiley).

He was on the board of IEEE Transactions on Software Engineering from 2000 to 2004.

His research interests include: model–driven development, testing and quality assurance, and empirical software engineering.

Eivind received the MSc in computer science from the University of Oslo in 2008. He is currently employed as an IT consultant at Bekk Consulting AS, where he is a member of the quality assurance and testing competency group. His research interests include software quality prediction models, software verification and validation and object-oriented design principles. His master thesis formed the basis for this paper.