A Cost-Effectiveness Criterion for Applying Software Defect Prediction Models

Hongyu Zhang[1,2] and S.C. Cheung[3]
[1]Tsinghua University
[2]State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
Beijing, China
hongyu@tsinghua.edu.cn
[3]The Hong Kong University of Science and Technology, Hong Kong, China
scc@cse.ust.hk

ABSTRACT
Ideally, software defect prediction models should help organize software quality assurance (SQA) resources and reduce cost of finding defects by allowing the modules most likely to contain defects to be inspected first. In this paper, we study the cost-effectiveness of applying defect prediction models in SQA and propose a basic cost-effectiveness criterion. The criterion implies that defect prediction models should be applied with caution. We also propose a new metric FN/(FN+TN) to measure the cost-effectiveness of a defect prediction model.

Categories and Subject Descriptors
D.2.5 [**Software Engineering**]: Testing and Debugging - *Debugging Aids.*

General Terms
Measurement, Reliability

Keywords
Defect prediction, cost effectiveness, evaluation metrics.

1. INTRODUCTION
Software quality assurance (SQA) is important for the success of a software project. However, it is a resource and time-consuming activity, which may include manual code inspections, technical review meetings, and intensive software testing. Modern large and complex systems often consist of hundreds or even thousands of modules (methods, files, components, etc). It is thus desirable to predict which modules are more likely to contain defects so that project managers can allocate limited SQA resources in a cost-effective manner.

Software defect prediction has recently attracted immense interest from the software engineering community. Many defect prediction models have been proposed (e.g., [1, 3-15]). These models collect historical defects of software modules as well as various module features such as program complexity, structural dependency, changes, process and organizational factors. A classifier such as Decision Tree or Naïve Bayes is utilized to train a classification model, which can then be used to predict the defect proneness of a new module. To evaluate the accuracy of predictions, researchers have adopted many metrics such as Recall and Precision.

Despite recent advances in defect prediction models, these models are not able to detect all defective modules and can identify correct modules as defective. Although the cost of building a defect prediction is usually small as most features can be automatically extracted by mining software repositories [8], we still need to deploy models for defect prediction cautiously. For example, one may be hesitated to deploy defect prediction models to safety critical systems if the models cannot guarantee a low degree of false negatives. It is because the deployment cost is not restricted to the effort of inspecting the reported defective modules. It also includes the cost of failures arising from the defects that are missed by the prediction models. Therefore, studying the cost-effectiveness of defect prediction models is important for the practical application of these models. Current studies on defect prediction mostly focus on the improvement of data quality [3, 10], the identification of new features [12], the design of new classification techniques [4], and the selection of training projects [15]. The cost-effectiveness of applying defect prediction models in SQA practices is not well explored.

In this paper, we study the cost-effectiveness of applying defect prediction models. We propose a criterion that is derived from the intuitions that a defect prediction based SQA strategy should at least cost less than the strategy of inspecting all modules and the strategy of inspecting randomly sampled modules. We also propose a new metric to measure the cost-effectiveness of defect prediction models. We use a case study on Eclipse project to illustrate the usage of the proposed metric.

2. EVALUATION METRICS FOR DEFECT PREDICTION MODELS
Prediction of defective modules can be cast as a classification problem in machine learning: given training samples of modules with labels as defective (containing at least one defect) or non-defective (no defects). A prediction model has four results: true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN), as shown in Table 1. The total number of actual defective modules is denoted as POS and the total number of actual non-defective modules is denoted as NEG.

Recall and Precision are often adopted to evaluate defect predication models (e.g., [3, 11, 14]). These metrics are the accuracy measures widely used in Information Retrieval area. They are defined as follows:

ESEC/FSE'13, August 18–26, 2013, Saint Petersburg, Russia
Copyright 2013 ACM 978-1-4503-2237-9/13/08...$15.00
http://dx.doi.org/10.1145/2491411.2494581

$$\text{Recall} = \frac{TP}{TP+FN}, \quad \text{Precision} = \frac{TP}{TP+FP}$$

Recall defines the rate of true defective modules in comparison to the total number of defective modules, and Precision relates the number of true defective modules to the number of modules predicted as defective. The values of Recall and Precision are between 0 and 1, the higher the better.

Table 1. Defect prediction results

	Predicted Defective	Predicted Non-defective	
Actual Defective	TP	FN	POS
Actual Non-defective	FP	TN	NEG

To evaluate the accuracy of a prediction, many researchers (e.g., [5, 15]) also use the Probability of Detection (*pd*) and Probability of False Alarm (*pf*) metrics, which are defined as follows.

$$pd = \frac{TP}{TP+FN}, \quad pf = \frac{FP}{FP+TN}$$

The pd metric is the same as Recall. The pf metric measures how many defect-free modules are wrongly classified as defective. The values of pf are between 0 and 1, the lower the better.

2. ANALYZING THE COST OF APPLYING SOFTWARE DEFECT PREDICTION

2.1 Cost of Defect Prediction Strategy

Ideally, a defect prediction model should be able to correctly identify all defective software modules. Limited QA resources can then be allocated to these modules to make SQA more effective. However, there are two costs incurred by this strategy:

- Cost of inspection for the defective modules selected by defect prediction models. A defect prediction model returns *(TP+FP)* defective modules. Assuming the average cost required for inspecting a module is C_i, the cost of inspecting all predicted defective modules is $C_i*(TP+FP)$. In this study, to simplify the analysis, we assume that the inspection is perfect, i.e., after inspecting a module, the defects can be successfully revealed. The value of C_i is project-specific. For example, C_i for a complex project is usually higher than a simple project. Also, C_i for a project closer to its delivery deadline is usually higher than that for a project at its initial phase.
- Cost of failures arising from false negative modules. A defect prediction model often fails to identify all defective modules. We denote the total cost incurred by false negative modules $C_{fn}*FN$, where C_{fn} is the average cost of missing a defective module. The value of C_{fn} is project-specific, which could be high for some projects [2]. For example, C_{fn} for a safety-critical project is usually higher than that for a web-forum project.

In summary, the cost of inspecting modules selected by a defect prediction model C_p is defined as follows:

$$C_p = C_i*(TP+FP) + C_{fn}*FN \qquad (1)$$

As an example, Table 2 shows the results produced by a defect prediction model. The Precision is 62.07% and the Recall is 64.29%. If applied to SQA, the cost C_p is $29C_i + 10C_{fn}$.

Table 2. The accuracy of a defect prediction model

	Predicted Defective	Predicted Non-defective
Actual Defective	18	10
Actual Non-defective	11	6

2.2 Cost of Inspecting-All Strategy

A defect prediction model allows the developers to concentrate on the defect-prone modules. The other SQA strategy is to simply inspect all modules without prioritization. We define the cost of such strategy C_{all} as follows:

$$C_{all} = C_i*N = C_i*(TP+FP+FN+TN) \qquad (2)$$

Clearly, a cost-effective defect prediction model should satisfy the following requirement:

$$C_p < C_{all} \Rightarrow C_i*(TP+FP) + C_{fn}*FN < C_i*(TP+FP+FN+TN)$$
$$\Rightarrow FN/(FN+TN) < C_i/C_{fn} \qquad (3)$$

Considering the example shown in Table 2, the cost C_{all} is $45C_i$. $FN/(FN+TN)$ is 62.50%. Assuming C_i/C_{fn} is 1/3, then $FN/(FN+TN) > C_i/C_{fn}$, failing to satisfy the requirement defined in Equation (3). Applying such a defect prediction model is not more cost-effective than the simple strategy of inspecting all modules.

2.3 Cost of Random Sampling Strategy

Another simple SQA strategy is to randomly sample a subset of modules and test them. A defect prediction model should be able to correctly identify defective modules. It should perform better than randomly selecting and inspecting the same number of modules. We define the cost of the random inspecting strategy C_{ran} as follows:

$$C_{rnd} = C_i*(TP+FP) + C_{fn}*\%D*(N-TP-FP) = C_i*(TP+FP) + C_{fn}*\%D*(TN+FN) \qquad (4)$$

, where $C_i*(TP+FP)$ denotes the cost of randomly inspecting the same number of modules as a defect prediction model suggests. *%D* denotes the density of defective modules (i.e., the percentage of defective modules in the project). For a new project, the value of *%D* could be estimated based on the experience of similar projects. The item $C_{fn}*\%D*(TN+FN)$ denotes the cost of missing the defective modules. Clearly, a cost-effective defect prediction should satisfy the following requirement:

$$C_p < C_{rnd} \Rightarrow C_i*(TP+FP) + C_{fn}*FN < C_i*(TP+FP) + C_{fn}*\%D*(TN+FN) \Rightarrow FN/(FN+TN) < \%D \qquad (5)$$

Consider the example shown in Table 2, *%D* is 62.22% (= 28/45). The cost C_{rnd} is $29C_i + 9.96C_{fn}$. $FN/(FN+TN)$ is 62.50%, which is larger than *%D*. Therefore, applying such a defect prediction model is not more cost-effective than the simple strategy of random sampling and inspection.

2.4 A Criterion for Cost-Effectiveness

Combining Equations (3) and (5), a defect prediction model should at least satisfy the following basic criterion in order to be cost-effective:

$$FN/(FN+TN) < min(C_i/C_{fn}, \%D) \qquad (6)$$

Considering the example shown in Table 2, the upper bound value of $FN/(FN+TN)$ a cost-effective defect prediction model should achieve is 1/3.

We also propose to use $FN/(FN+TN)$ as a metric to measure the cost-effectiveness of a defect prediction model, complementing

other metrics for measuring classification accuracy (e.g., Precision and Recall). The lower is the *FN/(FN+TN)* value, the better is the cost-effectiveness of the prediction model. Such a metric can be used for determining the benefit of applying a defect prediction model, and for selecting among alternative prediction models.

3. A CASE STUDY

To illustrate the usefulness of the proposed metric for evaluating cost-effectiveness of a defect prediction model, we re-examine the defect prediction results of a study performed by Zimmermann et al. on Eclipse [11]. Eclipse is a widely used integrated development platform for creating Java, C++ and web applications. It has been used as an experimental subject by many studies on software defect prediction (e.g., [3, 4, 14]). Note that although our case study uses the Eclipse experiment described in [11], the proposed metric is a general metric that is independent of the methods and data used for constructing prediction models.

In this study, we focus on the Eclipse 3.0 defect data (v2.0) at the package level. The data was collected by mining Eclipse's bug databases and version achieves. There are 429 packages in Eclipse 3.0, 57% of them contain post-release defects (defects reported in the first six months after release). The authors of [11] build several defect prediction models based on the data collect from Eclipse 2.0 and Eclipse 2.1, to predict defect-prone packages (packages contain at least one defect) in Eclipse 3.0. Table 3 summarizes their prediction results.

3.1 Evaluation with Recall and Precision Metrics

In Table 3, for the two prediction models, the Precision values are 0.641 and 0.713, the Recall values are 0.724 and 0.664, respectively. These results may or may not be satisfactory for some application scenarios. To evaluate the cost-effectiveness of the results, we apply the proposed metric as defined in Equation (6). Assume that the C_i/C_{fn} value is 1/3, the *FN/(FN+TN)* values are 0.44 and 0.41 for the two prediction models, respectively. These values are all above the C_i/C_{fn} value. Therefore, both models are unsatisfactory when cost-effectiveness is considered. The results suggest that cautions should be taken when these prediction models are applied in practice.

Table 3. Defect prediction results for Eclipse 3.0 packages

Train	Test	Precision	Recall	%D	FN/(FN+TN)
Eclipse 2.0	Eclipse 3.0	0.641	0.724	57%	0.44
Eclipse 2.1	Eclipse 3.0	0.713	0.664	57%	0.41

Figure 1 illustrates how the value of *FN/(FN+TN)* changes with different Precision and Recall values, for the Eclipse example. From Figure 1, we can see that, in order to satisfy the criterion that the value of *FN/(FN+TN)* should be lower than 1/3, the Precision values should be: 1) higher than 0.79 if Recall is 0.724, 2) higher than 0.91 if Recall is 0.664. The current Precision values are not "good enough" in terms of cost-effectiveness. Figure 1 also shows that, if the C_i/C_{fn} value is 1/2, given the same Recall values, the Precision values shown in Table 3 satisfy the cost-effective criterion.

Figure 2 illustrates how the values of *FN/(FN+TN)* vary with different *%D*, Precision (P) and Recall (R) values. Given the criterion that the value of *FN/(FN+TN)* should be lower than 1/3, both prediction models given in Table 3 are not cost-effective when *%D* is 57%. However, if *%D* is less than 50%, these two models satisfy the cost-effective criterion. Suppose that the C_i/C_{fn} value becomes 1/5, the two models satisfy the cost-effective criterion only if *%D* is less than 40%.

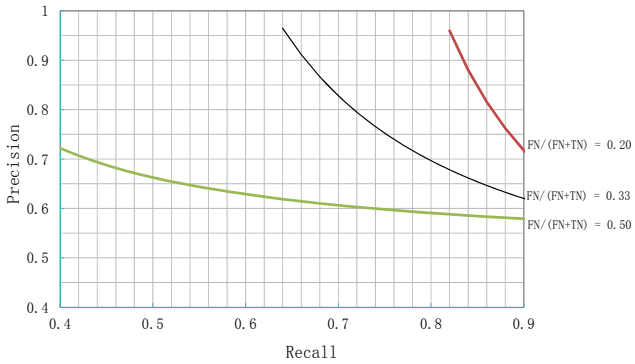

Figure 1. FN/(FN+TN) under different recall and precision values

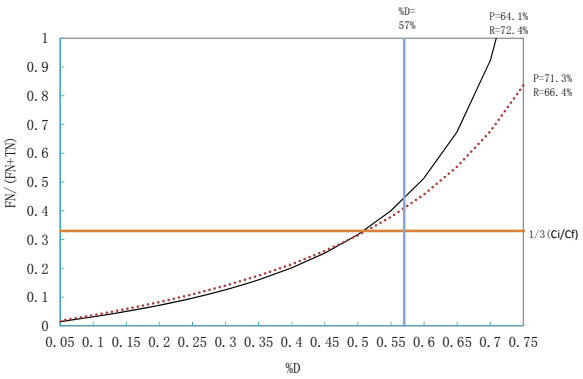

Figure 2. FN/(FN+TN) under different %D values

3.2 Evaluation with PD and PF Metrics

In Table 3, the Recall and Precision metrics are used to measure the classification accuracy of prediction models. We also examine how the proposed metric works with the pd (probability of detection) and pf (probability of false alarm) metrics. The original paper [11] does not give the evaluation results measured in terms of pd and pf. However, we can derive these values from Table 3 directly. In our prior work [13, 15], we discovered the relationship between the Precision and pd/pf as follows:

$$\Pr ecision = \frac{TP}{TP+FP} = \frac{1}{1+\frac{FP}{TP}} = \frac{1}{1+\frac{NEG \times PF}{POS \times PD}} \qquad (7)$$

Clearly, the NEG/POS item in the above equation can be calculated as: $NEG/POS = (1-\%D)/\%D$ (8)

Based on Equations (7) and (8), we transform the Recall/Precision values in Table 3 to pd and pf values in Table 4. To evaluate whether the pd and pf values shown in Table 4 are "good enough", we plot a figure (Figure 3) that illustrates how the value of *FN/(FN+TN)* changes with different pd and pf values, for the same Eclipse example. From Figure 3, we can see that, in order to satisfy the criterion that the value of *FN/(FN+TN)* should be lower than 1/3, the pf values should be: 1) less than 0.25 if pd is 0.724, 2) less than 0.08 if pd is 0.664. The current pf values are 0.538 and

0.354, which do not satisfy the cost-effectiveness criteria. Figure 3 also shows that, if the C_i/C_{fn} value is 1/2, given the same pd values, the pf values shown in Table 4 satisfy the cost-effective criterion.

Table 4. Defect prediction results in terms of pd and pf

Train	Test	pd	pf
Eclipse 2.0	Eclipse 3.0	0.724	0.538
Eclipse 2.1	Eclipse 3.0	0.664	0.354

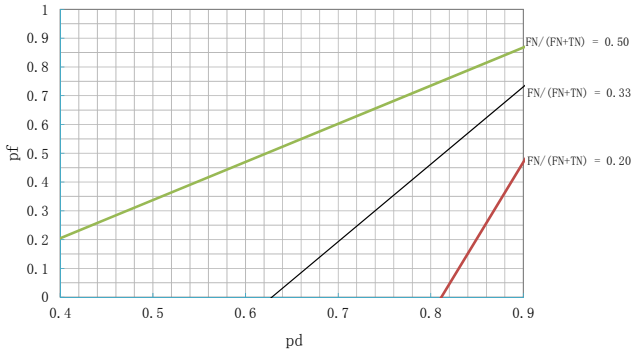

Figure 3. FN/(FN+TN) under different pd and pf values

4. DISCUSSIONS

In this section, we answer the questions specified in the call for paper[1]:

- **What is the new idea?** We propose a new criterion and metric for evaluating the cost-effectiveness of a defect prediction model.
- **Why is it new?** Currently, defect prediction models are evaluated using conventional accuracy metrics such as Precision, Recall, pd, and pf. There is a lack of explicit criterion and metric that can evaluate a prediction model from the cost-effectiveness perspective.
- **What is the single most related paper by the same author(s)? By others?** In our prior work [13], we studied the metrics used for evaluating defect prediction models. We explore the relationship between the Recall/Precision and pd/pf metrics. Our finding reveals a fundamental limit on the ability to improve defect predictors for domains where the defective modules are relatively infrequent. In this work, we further explore the cost-effectiveness aspect of the evaluation, which is more significant when the defective modules are relatively frequent.

  Recently, other researchers also explored effort-aware defect prediction models. For example, Mende and Koschke [7] discovered that when effort is considered, many classifiers perform not significantly better than a random selection of modules. They considered the ordering of modules and measure effort using a CE metric proposed by Arisholm et al. [1]. The CE metric is defined in a cumulative lift-chart and plots the percentage of identified faults against the examined lines of code. Posnett et al. [9] also proposed a revised CE metric called AUCCE. Unlike their work, our criterion and metric are based on defect prediction results alone (Table 1) and do not consider ordering and size of defective modules.
- **What feedback do the authors expect from the forum?** We would like to exchange ideas with other researchers and explore the applicability of the proposed metric in evaluating defect prediction models.

5. CONCLUSION

In this paper, we study the cost-effectiveness of applying defect prediction models in SQA. We propose a basic cost-effectiveness criterion (Equation 6) derived from the intuition that a defect prediction based SQA strategy should at least cost less than the strategy of inspecting all modules and the strategy of inspecting randomly sampled modules. The criterion implies that defect prediction models should be applied with cautions. Besides the conventional metrics (such as Recall and Precision) for measuring the accuracy of a defect prediction model, we propose a new metric *FN/(FN+TN)* to measure the cost-effectiveness of the prediction model. We believe our work can help better understand and apply defect prediction research results in practice.

ACKNOWLEDGMENTS

This research is supported by the NSFC projects 61073006 and 61272089, and Hong Kong SAR RGC/GRF project 611811.

REFERENCES

[1] E. Arisholm, L. C. Briand, M. Fuglerud, Data Mining Techniques for Building Fault-proneness Models in Telecom Java Software, in *Proc. ISSRE 2007*, 215-224, Nov 2007.

[2] R. Charette, Why Software Fails, *IEEE Spectrum*, September 2005.

[3] S. Kim, H. Zhang, R. Wu and L. Gong, Dealing with Noise in Defect Prediction, in *Proc. ICSE'11*, May 2011.

[4] M. Li, H. Zhang, R. Wu, and Z.-H. Zhou, Sample-based Software Defect Prediction with Active and Semi-supervised Learning, *Journal of Automated Software Engineering*, Springer, Jan 2012, pp.1-30.

[5] T. Menzies, J. Greenwald and A. Frank, Data Mining Static Code Attributes to Learn Defect Predictors, *IEEE Trans. Software Engineering*, 32(11), pp. 1-12, 2007.

[6] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, A. Bener, Defect prediction from static code features: current results, limitations, new approaches, *Journal Automated Software Eng*, Dec 2010.

[7] T. Mende and R. Koschke, Effort-Aware Defect Prediction Models, in *Proc. CSMR 2010,* pp.107-116, March 2010.

[8] N. Nagappan, T. Ball,and A. Zeller, Mining Metrics to Predict Component Failures, in *Proc. ICSE'06*, May 2006.

[9] D. Posnett, V. Filkov, and P. Devanbu, Ecological inference in empirical software engineering, in *Proc. ASE'11*, 362-371.

[10] R. Wu, H. Zhang, S. Kim, and S.C.Cheung, ReLink: Recovering Links between Bugs and Changes, in *Proc. ESEC/FSE'11*, Szeged, Hungary, Sep 2011.

[11] T. Zimmermann, R.Premraj and A. Zeller, Predicting Defects for Eclipse, in *Proc. PROMISE'07,* Minneapolis, USA, May 2007.

[12] T. Zimmermann and N. Nagappan, Predicting Defects using Network Analysis on Dependency Graphs, in *Proc. ICSE 2008,* Leipzig, Germany, May 2008.

[13] H. Zhang and X. Zhang, Comments on "Data Mining Static Code Attributes to Learn Defect Predictors", *IEEE Trans. on Software Engineering*, 33(9), 635-636, 2007.

[14] H. Zhang, An Investigation of the Relationships between Lines of Code and Defects, in *Proc. ICSM'09*, Sep 2009.

[15] H. Zhang, A. Nelson, T. Menzies, On the Value of Learning From Defect Dense Components for Software Defect Prediction, *Proc. PROMISE'10*, Sep 2010.

[1] http://esec-fse.inf.ethz.ch/cfp_new_ideas.html