Information Processing Letters 114 (2014) 469–474

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

# Combining the requirement information for software defect estimation in design time

Ying Ma [a,*], Shunzhi Zhu [a], Ke Qin [b], Guangchun Luo [b]

[a] School of Computer and Information Engineering, Xiamen University of Technology, Xiamen, 361024, China
[b] School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, 610054, China

ARTICLE INFO

*Article history:*
Received 16 November 2011
Received in revised form 16 March 2014
Accepted 16 March 2014
Available online 27 March 2014
Communicated by J.L. Fiadeiro

*Keywords:*
Machine learning
Software defect prediction
Design metric
Requirement metric
Software engineering

ABSTRACT

This paper analyzes the ability of requirement metrics for software defect prediction. Statistical significance tests are used to compare six machine learning algorithms on the requirement metrics, design metrics, and combination of both metrics in our analysis. The experimental results show the effectiveness of the predictor built on the combination of the requirement and design metrics in the early phase of the software development process.

## 1. Introduction

Software defect prediction is to predict the defect-prone modules for the next release of software or cross project software [1–4]. Menzies et al. [5] compared the performance of learning methods, and also endorsed the use of static code attributes for predicting defect-prone modules. But later, they reported the "ceiling effect", and held the idea that further progress in learning defect predictors might not come from better algorithms, but come from more information content of the training data [6].

There were a lot of researches investigating the ability of design metrics for defect prediction. Schroter et al. [7] found that the software design and past failure history, could be used to build predictor of fault-prone modules. Jiang et al. [8] compared the performance of predictive models based on design-level metrics with those based on code-level metrics. They found that models built on the two level metrics outperformed those built on either one metric group. Subramanyam and Krishnan [9] showed that *OO* design complexity metrics were significantly associated with defects. Ohlsson and Alberg [10] carried out an empirical study on Ericsson Telecom AB project, suggested that design metrics can be used to predict the most fault-prone modules. What is more, the design metrics, which extracted from design phase artifacts, design diagrams such as flow graphs and UML diagrams, were used at the NASA IV&V facility [11,12].

Recently, Boehm [13] described emergent requirement as one of the five future software measurement challenges. One of the earliest studies of the importance of the software requirements was conducted by Bell and Thayer [14]. They reported, "software requirements are important, and their problems are surprisingly similar across projects". Requirements that relate to a system's quality attributes are very useful to develop software with good quality. Requirement metrics also can be used to build predictor of fault-prone modules, when they are represented quantitatively, that is to say, they can be measured. Jiang et al. [15] built defect prediction models based on requirements, product/module code metrics, and both metrics. They suggested

* Corresponding author. Tel.: +86 0592 61830557; fax: +86 0592 61830580.
*E-mail address:* maying@xmut.edu.cn (Y. Ma).

http://dx.doi.org/10.1016/j.ipl.2014.03.012
0020-0190/© 2014 Elsevier B.V. All rights reserved.

**Table 1**
Data description.

| Data set | Modules with design | | Modules with requirement | |
|---|---|---|---|---|
| | #total | #defective modules | #total | #defective modules |
| cm1 | 505 | 81 | 114 | 58 |
| pc1 | 1107 | 73 | 320 | 44 |

**Table 2**
Requirement and design metrics.

| Requirement metrics | Description or formula |
|---|---|
| ACTION | Represents the number of actions the requirement needs to be capable of performing. |
| CONDITIONAL | Represents whether the requirement will be addressing more than one condition. |
| CONTINUANCE | Phrases such as the following: that follow an imperative and precede the definition of lower level requirement specification. |
| IMPERATIVE | Those words and phrases that command that something must be provided. |
| INCOMPLETE | Phrases such as 'TBD' or 'TBR'. They are used when a requirement has yet to be determined. |
| OPTION | Those words that give the developer latitude in the implementation of the specification that contains them. |
| RISK_LEVEL | A calculated risk level metric based on weighted averages from metrics collected for each requirement. |
| SOURCE | Represents the number of sources the requirement will interface with or receive data from. |
| WEAK_PHRASE | Clauses that are apt to cause uncertainty and leave room for multiple interpretations. |
| **Design metrics** | **Description or formula** |
| EDGE_COUNT:e | Those words and phrases that command that something must be provided. |
| NODE_COUNT:n | Number of nodes found in a given module. |
| BRANCH_COUNT | Branch count metrics. |
| CALL_PAIRS | Number of calls to other functions in a module. |
| CONDITION_COUNT | Number of conditionals in a given module. |
| v(G) | The cyclomatic complexity of a module: $v(G) = e - n + 2$. |
| DECISION_COUNT | Number of decision points in a given module. |
| DECISION_DENSITY | Condition count/Decision count. |
| iv(G) | The design complexity of a module. |
| DESIGN_DENSITY | Design density is calculated as: $iv(G)/v(G)$. |
| ev(G) | The essential complexity of a module. |
| ESSENTIAL_DENSITY | Essential density is calculated as: $(ev(G) - 1)/(v(G) - 1)$. |
| MAINTENANCE_SEVERITY | Maintenance severity is calculated as: $ev(G)/v(G)$. |
| MODIFIED_CONDITION_COUNT | The effect of a condition affect a decision outcome by varying that condition only. |
| MULTIPLE_CONDITION_COUNT | Number of multiple conditions that exist within a module. |
| PATHOLOGICAL_COMPLEXITY | A measure of the degree to which a module contains extremely unstructured constructs. |

that the early life cycle metrics can play an important role in fault prediction.

Different to prior works, we compare the performance of defect predictors built from requirement metrics, design metrics, and the combination of the both using Naive Bayes, AdaBoost, Bagging, Random Forest, Logistic regression, and K-Star methods. This empirical study shows that the defect predictor built on combining metrics can be applied at the very early phrase in software development process.

The rest of this paper is organized as follows. Section 2 describes software defect data sets. Sections 3 and 4 briefly present the performance metrics and the experiment design in this study, respectively. Section 5 shows the experimental result. Section 6 gives some discussion and threats to validity. Section 7 finalizes the paper with conclusions and future works.

## 2. Data set

The data we used for our experiments is obtained from the NASA Metrics Data Program (MDP) data repository [16], see Table 1 for an overview. We use the requirement metric data and design metric data in the repository directly. Note that, although thirteen data sets with design-level metrics are available in PROMISE repository, only two defect data sets have requirement-level metrics. These data sets have nine requirement metric attributes and sixteen design metric attributes. Table 2 displays the detail metric information.

## 3. Performance measures

To evaluate the performance of the prediction model, we can use the confusion matrix [17] which has four types of prediction as follows.

*True positive (TP):* the number of defective modules predicted as defective;
*False negative (FN):* the number of defective modules predicted as non-defective;
*False positive (FP):* the number of non-defective modules predicted as defective;
*True negative (TN):* the number of non-defective modules predicted as non-defective.

Since the performance metrics, F-measure and AUC are commonly used in defect prediction, we use them to evaluate the defect predictors based on different algorithms.