Computers and Electrical Engineering 67 (2018) 15–24

Contents lists available at ScienceDirect

Computers and Electrical Engineering

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

Empirical analysis of change metrics for software fault prediction[☆]

Garvit Rajesh Choudhary[a], Sandeep Kumar[b,*], Kuldeep Kumar[c,*], Alok Mishra[d], Cagatay Catal[e]

[a] Google Inc., Mountain View California, USA
[b] Department of Computer Science and Engineering, Indian Institute of Technology Roorkee, India
[c] Department of Computer Science and Engineering, Dr B R Ambedkar National Institute of Technology Jalandhar, Punjab, India
[d] Department of Software Engineering, Atilim University, Ankara, Turkey
[e] Information Technology Group, Wageningen University, Wageningen, The Netherlands

ARTICLE INFO

Article history:
Received 17 November 2017
Revised 24 February 2018
Accepted 26 February 2018

Keywords:
Software fault prediction
Eclipse
Change log
Metrics
Software quality
Defect prediction

ABSTRACT

A quality assurance activity, known as software fault prediction, can reduce development costs and improve software quality. The objective of this study is to investigate change metrics in conjunction with code metrics to improve the performance of fault prediction models. Experimental studies are performed on different versions of Eclipse projects and change metrics are extracted from the GIT repositories. In addition to the existing change metrics, several new change metrics are defined and collected from the Eclipse project repository. Machine learning algorithms are applied in conjunction with the change and source code metrics to build fault prediction models. The classification model with new change metrics performs better than the models using existing change metrics. In this work, the experimental results demonstrate that change metrics have a positive impact on the performance of fault prediction models, and high-performance models can be built with several change metrics.

1. Introduction

Software fault prediction studies aim to create prediction models that detect software components with more likelihood of having faults. Software metrics data and fault information from previous software releases are used to train the classification model and this model is then used to predict the fault-proneness of the modules in new releases [1]. Each software metrics value can be used to evaluate the software quality; for example, the average value of the Cyclomatic Complexity (CC) metrics in a class can help quality assurance experts to know whether or not that class is risky and needs changes. Risky class indicates that there is a high potential for a bug if that class is modified by software changes.

From a broader perspective, software fault prediction activity might be considered as a way of reliability prediction [2]. Some reliability prediction approaches predict the fault-prone modules before the testing phase; others use reliability

☆ Reviews processed and recommended for publication to the Editor-in-Chief by Associate Editor Dr. A. Isazadeh.
* Corresponding authors.
E-mail addresses: iitgarvit@gmail.com (G.R. Choudhary), sgargfec@iitr.ac.in (S. Kumar), kumark@nitj.ac.in (K. Kumar), alok.mishra@atilim.edu.tr (A. Mishra), cagatay.catal@wur.nl (C. Catal).

https://doi.org/10.1016/j.compeleceng.2018.02.043
0045-7906/© 2018 Elsevier Ltd. All rights reserved.

growth models to understand how the reliability changes over time. This study mainly focuses on the prediction of fault-prone modules before the testing phase and does not address reliability growth models. In addition to the fault prediction models, researchers recently developed new models to predict the security dimension of software quality. This research area is known as software vulnerability prediction and machine learning algorithms have also been applied to build these vulnerability prediction models. In this study, we do not address vulnerabilities directly but focus on general software faults.

Instead of using single metrics for the evaluation, these kinds of metrics can be combined in a prediction model. In order to build such a fault prediction model, software metrics and previous fault data must be extracted from software repositories [3]. If not automated, this task can be time consuming, error prone, and costly. However, there are several utilities in the industry that help to automate this process.

Software quality assurance (SQA) involves monitoring the software development process to ensure quality. In this respect, testing is the most time-consuming activity. Organizations are still looking for alternative solutions to reduce these testing efforts and ensure software quality at a lower cost. Software fault prediction models help project managers to distribute verification resources effectively. Recent studies show that the combination of novel metrics, such as product and process metrics, provide better performance than those that only use one type of metric. Madeyski and Jureczko (2015) [4] suggest using Number of Distinct Committers (NDC) metrics in addition to product metrics when building fault prediction models.

From the machine learning perspective, building models using software metrics in conjunction with the fault data is considered as supervised learning. However, there are some cases where either the previous fault data do not exist or there are very limited fault data. Also, it might be the first project undertaken by an organization in that domain, or the organization might not have access to the historical data. Either way, the organization will have no previous fault data. In a global software engineering project, some companies might not necessarily compile and store these kinds of data, and, hence, only limited data exist.

Recently researchers developed several models to address these problems [5]. In these works, product metrics are generally investigated to build a prediction method. Researchers are primarily concerned with these metrics because they are collected more easily from the repositories than other metrics types, such as process metrics. Public prediction datasets mostly contain product metrics.

Researchers have also investigated software metrics threshold values to build prediction models and design noise detection techniques [6]. As shown in these studies, software metrics are one of the most important components in building high-performance fault prediction models. A recent work [7] has emphasized the importance of the characteristics belonging to input metrics datasets in the selection of suitable software fault prediction models. For this reason, the authors of the present paper focus on change metrics to increase the performance of models. Recent studies apply metrics that are related to the developers, organization metrics [8], and network metrics.

In this study, we analyze the impact of change metrics on fault prediction models in detail. Several change metrics are extracted from software repositories, and models are built using machine learning algorithms, namely Decision Tree, K-Nearest Neighbor, and Random Forest.

Software metrics can be classified into the following types:

- Code and complexity metrics [9–11]: Complexity metrics indicate how complex a code block is, and it is widely known that the more complex the code is, the more vulnerability and risk exist for that module. So far, researchers have suggested and applied many complexity metrics (e.g., the well-known CC metric of Thomas McCabe), which are mostly calculated based on the source code. Therefore, we show this category as code and complexity metrics. CC metric shows how many independent paths, which must be tested separately, exist for that module (method/class/package). These metrics are the traditional product metrics derived directly from the software. Some examples for code metrics are lines of code and lines of comment. Examples for complexity metrics are system complexity, McCabeâs cyclomatic complexity, and essential complexity. If a procedural programming language such as C language is used in the project, McCabe and Halstead metrics [12] are the most popular.
- Object-oriented metrics [13,14]: These metrics are valid for an object-oriented programming paradigm, and they are related to specific programming concepts, such as inheritance, class, cohesion, and coupling. Several object-oriented metrics suites have been proposed. The most popular is the CK (Chidamber-Kemerer) metrics suite. The CK metrics are: weighted method count (WMC), coupling between object classes (CBO), number of children (NOC), lack of cohesion in methods (LCOM), depth of inheritance tree (DIT), and response for a class (RFC).
- Change or process metrics [15–18]: These metrics are related to the changes made during the software development process. They are collected throughout the software lifecycle across its multiple releases. Some process metrics are code churn measures, change bursts, and code deltas. Code churn metric shows the rate at which your code evolves [19]. Change burst metric takes into account the consecutive change sequences [18]. Code delta metric computes the difference between two builds in terms of a specific metric such as lines of code.
- Developer metrics [20,21]: These metrics are directly related to the software developer and are extracted for the different developers who contribute to the software. These metrics include the cumulative number of developers revising a module, the cumulative number of developers who changed the file over all the releases, and the lines of code modified by a developer.