Ain Shams Engineering Journal (2015) 6, 133–144

Ain Shams University

Ain Shams Engineering Journal

www.elsevier.com/locate/asej
www.sciencedirect.com

ELECTRICAL ENGINEERING

Extracting software static defect models using data mining

Ahmed H. Yousef *

Department of Computers and Systems, Faculty of Engineering, Ain Shams University, Cairo, Egypt

Received 2 June 2014; revised 13 August 2014; accepted 20 September 2014
Available online 13 October 2014

KEYWORDS

Defect models;
Software testing;
Software metrics;
Defect prediction

Abstract Large software projects are subject to quality risks of having defective modules that will cause failures during the software execution. Several software repositories contain source code of large projects that are composed of many modules. These software repositories include data for the software metrics of these modules and the defective state of each module. In this paper, a data mining approach is used to show the attributes that predict the defective state of software modules. Software solution architecture is proposed to convert the extracted knowledge into data mining models that can be integrated with the current software project metrics and bugs data in order to enhance the prediction. The results show better prediction capabilities when all the algorithms are combined using weighted votes. When only one individual algorithm is used, Naïve Bayes algorithm has the best results, then the Neural Network and the Decision Trees algorithms.

1. Introduction

The data mining approach is used to discover many hidden factors regarding software. This includes the success factors of software projects that attracted researchers a long time ago [1], the support of software testing management [2] and the defect pattern discovery [3]. The software defects estimation and prediction processes are used in the analysis of software quality [4]. They are also used to estimate the required effort and cost of maintenance after delivery [5]. The maintainability evaluation depends mainly on the use of software design metrics [6].

The quality of software depends on the maturity level of the software development processes [7]. In [8], Kelly stressed the importance of inspections to minimize the densities of defects, especially the requirements and design inspections. The number of defect densities decreased exponentially in the coding phase because defects were fixed when detected and did not migrate to subsequent phases. This peer review based inspection process becomes very expensive and unrealistic in the coding phase because the code review process is labor intensive; 8–20 LOC/minute can be inspected and this effort repeats for all members of the review team [9].

Therefore, the prediction of potential defective modules (on the code level) will be useful to prioritize which modules are the best candidates for this expensive code reviews, inspection and through testing. This prediction process is a complementary approach that should be used carefully to avoid the trend

* Tel.: +20 1001701882.
E-mail address: ahassan@eng.asu.edu.eg.
Peer review under responsibility of Ain Shams University.

Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.asej.2014.09.007

of leaving blind portions of the system that may contain defects which may be missed. Because assessing software goes toward areas that are believed to be mission critical, several defect detectors based on static code measures are proposed [10].

With the existence of software repositories including [11], several attempts are done to use empirical data to construct and validate different static defect models for multiple software projects or different versions of the same project [12]. One of these attempts is the NASA data sets online [13] which contains five large software projects with thousands of modules. Each set includes introductory text that lists and explains the static measures and other variables of each project. Data points correspond to modules and their statistic measures as well as a binary variable indicating whether the module is defective or not.

In order to model the causes of defective modules, static measures obtained from source code, mainly size, coupling, cohesion, inheritance, and complexity measures are used to determine whether a module is defective or not [14]. In [2,3,15], data mining techniques are used to search for rules that indicate modules with a high probability of being defective. Other techniques include the use of SVM and Service oriented architecture using expert COCOMO [16,17]. Koru and Liu used PROMISE repository to analyze the datasets representing the five projects provided by NASA [14,18]. The data sets are stratified according to module size. They found improved prediction performance in the subsets that included larger modules. Based on these results, they developed some guidelines that practitioners can follow in their defect-prediction efforts. These guidelines can help to overcome the low prediction performance, reported by Khoshgoftaar and Menzies [19,20]. However, these guidelines are not converted to a real software system that is integrated with compilers to guarantee that software developers follow the guidelines and write code that minimize the probability of having defective modules.

The absence of an established benchmark makes it hard, if not impossible, to compare approaches. Refs. [21,22] present a benchmark for defect prediction, in the form of a publicly available data set consisting of several software systems, and provide an extensive comparison of the explanative and predictive power of well-known bug prediction approaches. Ref. [23] proposed a framework for benchmarking the classification models for software defect prediction and found that no significant performance differences could be detected among the top 17 classifiers. An analysis of several software defect models is found in [24].

An extensive comparison is performed of many machine-learning algorithms on the PROMISE data sets [19]. However, the results of this comparison do not indicate that a specific algorithm has significant better results in the capabilities of prediction.

The aforementioned challenges can be summarized to three points. There is a need for effective software tools that uses data from available software repositories to predict the defective modules of a new software project in order to enable the project managers and quality team to know where to invest their time in testing and debugging. Because there is no conclusive finding of the best data mining technique which solve the problem of predicting defective modules in software projects, these tools should apply and combine different data mining techniques and compare the results.

The contributions of this paper are worth consideration for two reasons. Firstly, it provides a solution architecture that can enhance software development based on data in software repositories. Most of the aforementioned references converted the software metrics into knowledge and rules that should be used by experienced development team members to enhance their code. However, there is a lack of a software architecture that can be implemented and integrated with compilers to use these extracted knowledge to warn the less experienced software development team members of potential defective modules and enable the project manager to assign more resources to these potential defective modules. The integration of the data mining models with bugs tracking databases and metrics extracted from software source code leads to have more accurate results. Secondly, the paper provides a benchmark that provides an ensemble of data mining models in the defective modules prediction problem and compares the results. This benchmark has the same conditions including the same input dataset, the same percentage of the training dataset and the same feature selection approach. The paper proposes a combined approach to get better prediction results according to the values of precision, recall, accuracy and F-measure.

The remainder of this paper is organized as follows: Section 2 discusses related work and highlighted the most important approaches that are used to tackle the problem of extracting software defect models. This section can be skipped if the reader is familiar with software defect models literature. Section 3 presents the methodology used in this research. This includes the used metrics, proposed architecture, data collection methodology and the used data mining algorithms. In Section 4, the results of applying the data mining tools on several projects are presented. A thorough analysis and discussion are presented. Finally, Section 5 concludes the paper.

2. Related work

A software defect is defined as a deficiency or an issue arising from the use of a software product and causes it to perform unexpectedly. The ways to practically identify, and prioritize defects are addressed in several researches including [25]. Ref. [26] listed the top ten practices to reduce defects. An effective defect fighting strategy has many techniques that are described in [27,28].

Defects are real, observable indications of the software development progress from a schedule viewpoint [29]. Determining whether a new software change is buggy or clean is used to predict latent software defects before releasing the software to users [30]. Boehm found that about 80% of the defects come from 20% of the modules, and about half the modules are defect free [26]. Defects in the software cause failures of the programs during operation. Software failure intensity is related to the number of defects in the program [31]. Systematically conducted unit and system tests play an important role in revealing faults that lead to failures during the software execution [27]. It is natural to expect the continued discovery of defects after the software is placed into operation. Fixing Defects in the operational phase is considerably more expensive than doing so in the development or testing phase. Cost-escalation factors ranges from 5:1 to 100:1 [26].

The study of defect prediction can be classified into two categories: dynamic models and static models. The dynamic