ABSTRACT

Approaches to detect fault-prone modules have been studied for a long time. As one of these approaches, the authors proposed a technique using a text filtering technique. They assume that bugs relate to words and context that are contained in a software module. The technique treats a module as text information. Based on the dictionary which was learned by classifying modules which induce bugs, the bug inducing probability over a target module is calculated, and it judges whether the given module is a fault-prone module. The predictive granularity of this technique is a module. In this study, the authors aimed at prediction with the finer granularity of the portion that induces a bug. Specifically, they tried to predict bug-inducing changes by using source code differences of bug inducing changes and previous changes and a text filtering technique. Similarly, the authors tried to predict bug fixing by using source code differences of bug fixing changes and previous changes and a text filtering technique. To show the effectiveness of the approach, the authors conducted two experiments and compared their approach with fault-prone filtering by applying it to two open source projects, and obtained higher accuracy.

Keywords: Bug-Inducing Changes, Faults Detection, Software Changes, Software Repositories Mining, Text Filtering Techniques

1. INTRODUCTION

In software development, it is expected that we can remove bugs efficiently and reduce the cost of development if we can predict changes of source code that induce bugs. In this study, we performed a prediction of bug inducing changes and bug fixing changes by using source code differences in software repositories which managed by a version control system, Git (Git, 2014). Specifically, we prepared Git repositories that include bug inducing and fixing information by SZZ algorithm (Sliwerski, Zimmermann, & Zeller, 2005) that integrates information of version control system and bug tracking system.

Next, we collected bug inducing and fixing...
changes by using source code differences that were extracted from those repositories. Finally, we classified these changes into bug inducing and fixing by a text filtering technique. In order to compare the accuracy of the prediction, we performed fault-prone modules prediction by using fault-prone filtering as the conventional technique.

2. RELATED WORKS

Fault-prone prediction is a mature area in software engineering with various studies having been done over the past 20 years. From 1999, for example, many studies have been conducted.

Software metrics related to program attributes such as lines of code, complexity, frequency of modification, coherency, coupling, etc., have been used in many previous studies. In those studies, such metrics are considered explanatory variables and fault-proneness is considered an objective variable. Mathematical models are constructed from those metrics. The selection of metrics varies according to studies. For example, studies such as Guo, Cukic, and Singh (2003), Menzies, Greenwald, and Frank (2007), and Seliya, Khoshgoftaar, and Zhong (2005) used NASA’s Metrics Data Program. Object oriented metrics are used in Briand, Melo, and Wust (2002), for example. Some studies used metrics based on metrics collection tools (Bellini, Bruno, Nesi, & Rogai, 2005; Denaro, & Pezze, 2002).

The selection of classification techniques also varies according to studies. Khoshgoftaar et al. performed a series of fault-prone prediction studies using various classification techniques; for example, the classification and regression trees (Khoshgoftaar, Shan, & Allen, 2000), the tree based classification with S-PLUS (Khoshgoftaar, Allen, & Deng, 2002), the Treedisc algorithm (Khoshgoftaar, & Allen, 2001), the Sprint-Sliq algorithm (Khoshgoftaar, & Seliya, 2002), and logistic regression (Khoshgoftaar, & Allen, 1999). The comparison was summarized in Khoshgoftaar and Seliya (2004). Logistic regression is a frequently used technique in fault-prone prediction (Briand et al., 2002; Denaro, & Pezze, 2002; Khoshgoftaar & Allen, 1999). Menzies et al. (2007) compared three classification techniques and reported that the naive Bayesian classifier achieved the best accuracy.

Prediction of bugs by using change history of version control system has been widely studied so far. For example, there are studies by Nagappan and Ball (2005), by Kim et al. (Kim, Pan, & Whitehead, 2006; Kim, Zimmermann, Whitehead, & Zeller, 2007), and so on.

Giger et al. empirically analyzed the relationship between fine-grained Source Code Changes (SCC) and the number of bugs in source files (Giger, Pinzger, & Gall, 2011). Their results showed that SCC outperforms code churn based on lines modified for learning bug prediction models. Hassan et al. proposed complexity metrics that are based on the code change process (Hassan, 2009). They presented models to quantify the complexity over time using historical code changes instead of source code attributes. Aversano et al. used a technique to identify bug-introducing changes to train a model that can be used to predict if a new change may introduces or not a bug (Aversano, Cerulo, & Grosso, 2007). They represented software changes as elements of an n-dimensional vector space of terms coordinates extracted from source code snapshots. Kim et al. introduced a bug prediction technique that uses a machine learning classifier to determine whether a new software change is more similar to prior buggy changes or clean changes (Kim, Whitehead, & Zhang, 2008). Their approach uses features extracted from the source code and change histories to train the Support Vector Machine classifier.

We have been tackled a fault-prone module prediction technique named “fault-prone filtering” so far (Mizuno, Ikami, Nakaichi, & Kikuno, 2007a, 2007b; Mizuno, & Kikuno, 2008, 2007a; Hata, Mizuno, & Kikuno, 2010). In this research, we focused on the historical textual information in software source code. However, the granularity of modules is relatively large. This affected the practical effectiveness of this approach.
11 more pages are available in the full version of this document, which may be purchased using the "Add to Cart" button on the product's webpage:

www.igi-global.com/article/classification-of-bug-injected-and-fixed-changes-using-a-text-discriminator/121547?camid=4v1


www.igi-global.com/e-resources/library-recommendation/?id=2

Related Content

An MDA Approach for Developing Executable UML Components
www.igi-global.com/chapter/mda-approach-developing-executable-uml/49162?camid=4v1a

Adaptive Computation Paradigm in Knowledge Representation: Traditional and Emerging Applications
www.igi-global.com/chapter/adaptive-computation-paradigm-knowledge-representation/29560?camid=4v1a
Developing Local Association Network Based IoT Solutions for Body Parts Tagging and Tracking
ZongWei Luo, Martin Lai, Mary Cheung, ShuiHua Han, Tianle Zhang, Zhongjun Luo, James Ting, Patrick Wong, Sam Chan, Kwok So and George Tipoe (2010). *International Journal of Systems and Service-Oriented Engineering* (pp. 42-64). [www.igi-global.com/article/developing-local-association-network-based/48518?camid=4v1a](www.igi-global.com/article/developing-local-association-network-based/48518?camid=4v1a)

A Decision Tree Analysis of a Multi-Player Card Game With Imperfect Information