Expert Systems With Applications 45 (2016) 185–194

Contents lists available at ScienceDirect

Expert Systems With Applications

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

Co-changing code volume prediction through association rule mining and linear regression model

Shin-Jie Lee [a,b,*], Li Hsiang Lo [b], Yu-Cheng Chen [b], Shi-Min Shen [b]

[a] Computer and Network Center, National Cheng Kung University, Tainan 701, Taiwan
[b] Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan 701, Taiwan

ARTICLE INFO

Keywords:
Co-changing code volume prediction
Co-changing methods identification

ABSTRACT

Code smells are symptoms in the source code that indicate possible deeper problems and may serve as drivers for code refactoring. Although effort has been made on identifying divergent changes and shotgun surgeries, little emphasis has been put on predicting the volume of co-changing code that appears in the code smells. More specifically, when a software developer intends to perform a particular modification task on a method, a predicted volume of code that will potentially be co-changed with the method could be considered as significant information for estimating the modification effort. In this paper, we propose an approach to predicting volume of co-changing code affected by a method to be modified. The approach has the following key features: co-changing methods can be identified for detecting divergent changes and shotgun surgeries based on association rules mined from change histories; and volume of co-changing code affected by a method to be modified can be predicted through a derived fitted regression line with *t*-test based on the co-changing methods identification results. The experimental results show that the success rate of co-changing methods identification is 82% with a suggested threshold, and the numbers of correct identifications would not be influenced by the increasing number of commits as a project continuously evolves. Additionally, the mean absolute error of co-changing code volume predictions is 133 lines of code which is 95.3% less than the one of a naive approach.

1. Introduction

Code smells are certain structures in the code that suggest the possibility of refactoring and may complicate the maintenance of software (Eick, Graves, Karr, Marron, & Mockus, 2001; Fowler, 1999; Khomh, Di Penta, & Guéhéneu, 2009; Olbrich, Cruzes, Basili, & Zazworka, 2009; Sjoberg, Yamashita, Anda, Mockus, & Dyba, 2013; Slinger, 2005; van Emden & Moonen, 2002). In Fowler (1999), 22 code smells are elaborated and their related refactorings are also investigated. In the past decade, a number of approaches have been proposed to enable automatically identifying code smells. Approaches on detections of duplicated code are investigated (Bellon, Koschke, Antoniol, Krinke, & Merlo, 2007; Roy & Cordy, 2007; Tiarks, Koschke, & Falke, 2009). In Moha, Guéhéneuc, Duchien, and Le Meur (2010), a smell detection tool is proposed to identify design smells of Blob, Functional decomposition, Spaghetti code, Swiss army knife and the code smells related to the design smells. In Tsantalis and Chatzigeorgiou (2009), a methodology for the identification of Move method refactoring opportunities is proposed for solving Feature envy bad smells. In Ligu, Chatzigeorgiou, Chaikalis, and Ygeionomakis (2013), a technique for the identification of refused bequest code smells is developed based on the intentional introduction of errors in the inherited methods of objects.

Among the code smells, Divergent change and Shotgun surgery are the ones that are directly related to co-changes of code. As described in Fowler (1999), a divergent change occurs when one class is commonly changed in different ways for different reasons, and a shotgun surgery occurs when every time a kind of change is made on a class, a lot of little changes to a lot of different classes have to be made. Although effort has been made on identifying divergent changes and shotgun surgeries, little emphasis has been put on predicting the volume of co-changing code that appears in the code smells. More specifically, when a software developer intends to perform a particular modification task on a method, a predicted volume of code that will potentially be co-changed with the method could be considered as significant information for estimating the modification effort. In this work, we propose a system for predicting volume of

* Corresponding author at: Computer and Network Center, National Cheng Kung University, Tainan 701, Taiwan.
E-mail addresses: jielee@mail.ncku.edu.tw (S.-J. Lee), roraibar@gmail.com (L.H. Lo), mooc0102@gmail.com (Y.-C. Chen), thanatos1710@gmail.com (S.-M. Shen).

http://dx.doi.org/10.1016/j.eswa.2015.09.023
0957-4174/

co-changing code affected by a method to be modified. The system has the following key features:

- Co-changing methods can be identified based on association rules mined from change histories.
- Volume of co-changing code affected by a method to be modified can be predicted through a derived linear regression line with *t*-test based on the co-changing methods identification results.

We also conducted experiments to evaluate the success rate of identifying co-changing methods, numbers of correct predictions of co-changes as a project evolves, and the mean absolute error of predicting co-changing code volume.

This paper is organized as follows. The proposed approach is elaborated in Section 2. Section 3 describes the experimental evaluations of the proposed system. Related works are discussed in Section 4. In Section 5, we conclude the contributions of the proposed approach and the future work.

2. An approach to predicting co-changing code volume

Fig. 1 shows the system architecture of the proposed approach. Through the component of Code Parser, the source code of a project being analyzed will be parsed into abstract syntax trees, and each commit information will be retrieved from the change history of the project. The code smells of large classes, long methods, and long parameter lists will be detected by the component of Large Class/Long Method/Long Parameter List Detector through comparing the LOC of classes/methods/numbers of parameters of the project being analyzed against the thresholds determined based on 50 open source projects (Lee, Lin, Lo, Chen, & Lee, 2014). The code smells of divergent changes and shotgun surgeries will be detected by the component of Divergent Change/Shotgun Surgery Detector based on association rule mining (Agrawal, Imieliński, & Swami, 1993).

Subsequently, the component of Class Relationships with Code Smells Network Generator generates a set of networks representing the class relationships, and adds the code smells information to the networks, which was also developed based on our previous work. The networks will be visualized through an open source network visualization tool, Cytoscape (Cytoscape official website). Additionally, Divergent Change/Shotgun Surgery Detector component also derives fitted regression lines for predicting co-changing code volume based on the lines of changed code in methods. When a user inputs the number of lines of code of a method to be modified, the Co-Changing Code Volume Prediction component will predict the volume of co-changing code through a derived fitted regression line.

Fig. 2 shows a snapshot of the visualization of detected code smells in Eclipse Equinox project. The networks of the relationships between all classes in the project can be visualized by clicking the tag "all" in the control panel, and the networks of the relationships between a class, its methods and its attributes can be visualized by clicking the tag of the class name. The networks will be visualized on the panel in right hand side. In the system, four basic types of the relationships between two classes, i.e., extension, implementation, dependency and association, are identified through parsing the source code and are visualized according to the notations of UML class diagrams. Identified code smells are then highlighted on the networks in red or yellow. In Fig. 2, the red and yellow rectangles denote large classes, and the red lines denote the co-changing relations between classes. The following sections will detail how to identify co-changing methods and visualize divergent changes and shotgun surgeries.

2.1. Co-changing methods identification

In this section, we introduce how to identify co-changing methods based on association rule mining and how to detect divergent changes and shotgun surgeries based on identifications of co-changing methods. Co-change of two methods is identified by the following definition:

Definition 1. (Unidirectional co-change of two methods): Let α *and* β be two methods in a class. Let C_α be the commits in which α is changed, and let C_β be the commits in which β is changed. β is said to co-change with α if the following association rule is mined:

$$\{\alpha\} \Rightarrow \{\beta\},$$

where $|C_\alpha \cup C_\beta| \geq \theta$, the support of the rule is support $(\{\alpha\} \Rightarrow \{\beta\}) = \frac{|C_\alpha \cap C_\beta|}{|C_\alpha \cup C_\beta|} \geq \gamma$, and the confidence of the rule is confidence $(\{\alpha\} \Rightarrow \{\beta\}) = \frac{|C_\alpha \cap C_\beta|}{|C_\alpha|} \geq \delta$.

In this work, we developed a code parser to extract changed lines of code of each method in each commit from the Git repositories of Eclipse projects based on Eclipse AST parser and JGit. The change types include the changes of the signature and the body of a method.

Table 1 shows an example of the change histories of two methods *getServiceReferences* and *getServices* in the Equinox project. $c_1, \ldots, c_{10}$ denote all of the commits in which at least one of the two methods

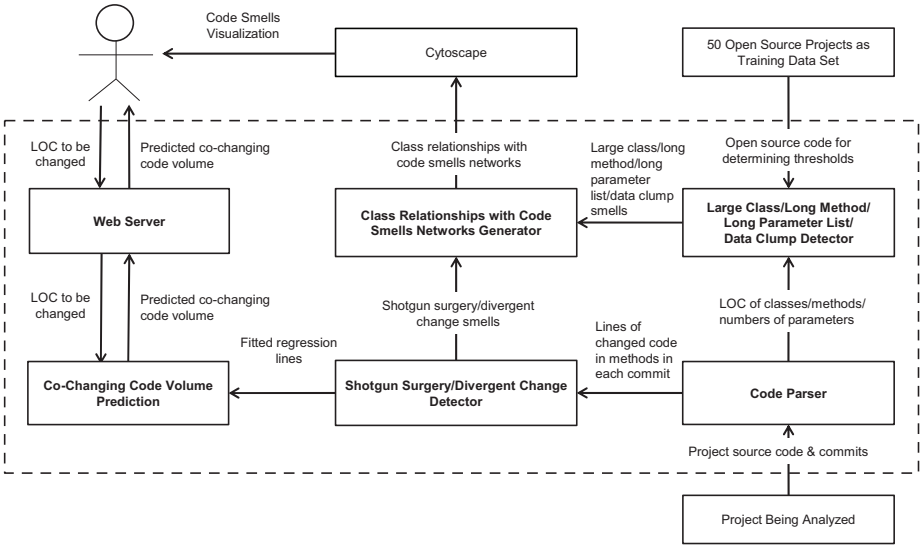

Fig. 1. System architecture of the proposed approach.