The Journal of Systems and Software xxx (2014) xxx–xxx

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Change impact analysis and changeability assessment for a change proposal: An empirical study ☆☆☆,☆☆

Xiaobing Sun[a,d,*], Hareton Leung[c], Bin Li[a,d], Bixin Li[b]

[a] School of Information Engineering, Yangzhou University, Yangzhou, China
[b] School of Computer Science and Engineering, Southeast University, Nanjing, China
[c] Department of Computing, Hong Kong Polytechnic University, Hong Kong, China
[d] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China

ARTICLE INFO

Article history:
Received 25 August 2013
Received in revised form 8 April 2014
Accepted 15 May 2014
Available online xxx

Keywords:
Change impact analysis
Changeability assessment
Empirical study

ABSTRACT

Software change is a fundamental ingredient of software maintenance and evolution. Effectively supporting software modification is essential to provide a reliable high-quality evolution of software systems, as even a slight change may cause some unpredictable and undesirable effects on other parts of the software. To address this issue, this work used change impact analysis (CIA) to guide software modification. CIA can be used to help make correct decision on the change proposal, that is changeability assessment, and to implement effective changes for a change proposal. In this article, we conducted an empirical study on three Java open-source systems to show how CIA can be used during software modification. The results indicate that: (1) assessing changeability of a change proposal based on the impact results of the CIA is not accurate from the precision perspective; (2) the proposed impactness metric is an effective indicator of changeability assessment for the change proposal; and (3) CIA can make the change implementation process more efficient and easier.

1. Introduction

Software change is a fundamental ingredient of software maintenance and evolution. Effectively supporting software modification is essential to provide a reliable high-quality evolution of software systems, as even a slight change may cause some unpredictable and undesirable effects on other parts of the system. One of the most critical issues of the software maintenance process is to predict the impact of a change proposal (Schneidewind, 1987; Nesi, 1998; Kemerer and Slaughter, 1999; Fioravanti and Nesi, 2001; Lucia et al., 2002). In order to deal with a change proposal, some predictive measurement of its change ripples should be conducted. There have been a large amount of research work on the measurement and metrics for software development (Chidamber and Kemerer, 1994; Briand et al., 1999b; Gopal et al., 2002; Olague et al., 2007; Habra et al., 2008), but only a few on software maintenance (Bandi et al., 2003; Schneidewind, 2000). Accurate measurement is a prerequisite for all engineering disciplines, and software maintenance is no exception. Given a change proposal, software maintenance must address three problems: to make a preliminary estimation of the ripple effects affected by the modification, to determine whether to accept, reject, or further evaluate this given change proposal, and to implement changes according to the change proposal.

In this article, we integrate change impact analysis (CIA) and changeability assessment to perform a predictive measurement for the proposed change proposal. CIA, often simply called *impact analysis*, is an approach used to identify the potential effects caused by changes made to software (Bohner and Arnold, 1996). CIA starts with a set of proposed changed elements in a software system, called the *change set*, and attempts to determine a possibly larger set of elements, called the *impact set*, that requires attention or maintenance effort (Bohner and Arnold, 1996). The impact set can facilitate the change implementation process (Li et al., 2012).

☆ A preliminary edition of this article was accepted by COMPSAC 2012 as a short research track paper. This work extends and provides wider experimental evidence of the proposed method.

☆☆ This work is supported partially by the Natural Science Foundation of the Jiangsu Higher Education Institutions of China under Grant No. 13KJB520027, partially by the Open Funds of State Key Laboratory for Novel Software Technology of Nanjing University under Grant No. KFKT2014B13, partially by the Program for New Century Excellent Talents of Yangzhou University, and partially by the Cultivating Fund for Science and Technology Innovation of Yangzhou University under Grant No. 2013CXJ025.

* Corresponding author. Tel.: +86 18252740912.
E-mail addresses: xbsun@yzu.edu.cn, sundomore@163.com (X. Sun), hareton.leung@polyu.edu.hk (H. Leung), lb@yzu.edu.cn (B. Li), bx.li@seu.edu.cn (B. Li).

http://dx.doi.org/10.1016/j.jss.2014.05.036
0164-1212/

Specifically, maintainers can check the elements in the impact set to see whether they need further consideration. However, a critical threat to *CIA* is the accuracy of its impact set, i.e., the impact set may have some false-positives (a.k.a., the elements in the estimated impact set are not really impacted) and false-negatives (a.k.a.,some of the real impacted elements are not identified) (Li et al., 2012). Our work computes a ranked list of potentially impacted elements, which helps maintainers to estimate the probability of the impacted methods to be false-positives. Moreover, our work can remove the possibility of the false-negatives by choosing an appropriate impact set. Such impact results provide an eclectic approach for CIA. In addition, changeability assessment evaluates the ease to implement a change proposal (Board, in press). Based on changeability assessment, we can make a decision on the change proposal before actual change implementation. There has been some work that used CIA to assess the changeability of the proposed change proposal (Chaumun et al., 1999). As the impact set computed by CIA is often inaccurate, the computed changeability of the proposed change proposal is also not accurate (Sun et al., 2012).

The focus of this study is on CIA and changeability assessment for the code-level change proposal. As *class* is the basic element in object oriented programming environment, one of the most popular development environment, we assume that the change proposal is composed of a set of proposed changed classes. Given the class-level change proposal, we use *FCA–CIA* (Formal Concept Analysis–Change Impact Analysis) to calculate a ranked list of the potential impact set from these proposed changed classes since *FCA–CIA* has shown to be effective to compute the change effects (Sun et al., 2012; Li et al., 2013). *FCA–CIA* is a cross-level CIA, which starts from proposed class-level changes and produces a ranked list of potentially impacted methods. The potential impacted methods are ranked according to an *impact factor* metric which corresponds to the priority of these methods to be inspected. Then, we use an *impactness* metric based on the results of *FCA–CIA* to indicate the changeability of this change proposal. The *impactness* metric measures the degree the proposed change proposal may affect the original system. Finally, we use the impact results from *FCA–CIA* to facilitate change implementation according to the proposed change proposal. *FCA–CIA* have been evaluated in our previous work (Li et al., 2013). However, the following issues have not been addressed:

- Can CIA be directly used for changeability assessment?
- How to use CIA for changeability assessment and change implementation?

To answer these two questions, we conducted some empirical studies based on three open source systems. The main contributions of this article are threefold as follows:

- Our study shows that the impact results produced by CIA are inaccurate for changeability assessment. This implies that some other metrics should be developed for changeability assessment.
- The empirical studies show that the proposed *impactness* metric based on CIA is effective for changeability assessment, which can help users make correct decision on accepting or rejecting the change proposal.
- Based on a user study involving 16 students to fix four bugs in the *jEdit* subject program, from the results of time performance and the users' perception, it appears that CIA can make the change implementation process more efficient and effective. To the best of our knowledge, there is no other such evaluation in the literature.

The rest of the article is organized as follows. In the next section, we discuss the background to support CIA and changeability assessment. Section 3 presents our work of CIA and changeability assessment for software modification. We conduct some empirical studies to validate the effectiveness of our approach in Section 4. In Section 5, some related work in the field of CIA and changeability assessment is introduced. Finally, we present our conclusion and future work in Section 6.

Table 1
Formal context.

	M1	*M2*	*M3*	*M4*	*M5*	*M6*	*M7*	*M8*	*M9*	*M10*	*M11*	*M12*
C1	×	×	×		×	×						
C2	×	×	×	×	×			×			×	
C3	×					×	×		×			
C4				×				×				
C5						×	×		×	×	×	
C6				×						×	×	×

2. Background

FCA–CIA is performed based on concept lattice. In this section, we introduce the background of concept lattice. Concept lattice, also called *formal concept analysis* (FCA), is a field of applying mathematics to study the relation between entities and entity properties to infer a hierarchy of concepts (Ganter and Wille, 1986). For every binary relation between entities and their properties, a lattice can be constructed to provide insight into the structure of the original relation (Ganter and Wille, 1986). It has been shown that *FCA* is a powerful code analysis technique for software maintenance in the last few years (Snelting and Tip, 2000; Tilley et al., 2005). More details of *FCA* can be referred to Ganter and Wille (1986).

Typically, *FCA* follows three steps: (1) a formal context with formal object and formal attribute is provided; (2) concept lattice is generated by applying concept lattice construction algorithm to the formal context obtained from Step (1) (Ganter and Wille, 1986). (3) Analysis (for example, CIA and refactoring (Snelting and Tip, 2000; Tilley et al., 2005)) is conducted based on the properties (for example, hierarchical property) of the concept lattice.

A formal context can be easily represented by a relation table. In the relation table, rows are headed by classes and columns headed by methods. A *cross* in row *o* and column *a* means that the formal object *o* (corresponding to class *c*) has formal attribute *a* (corresponding to method *m*), in other words, class *c* depends on method *m*, defined as follows:

Definition 1 *(Dependence between class and method).* Given that a class *c* and a method *m* in a program, class *c* depends on method *m*, if and only if, at least one of the following conditions is satisfied:

1 *m* belongs to *c*;
2 *m* belongs to any superclass of *c*;
3 *c* depends on another method *k* calling *m*;
4 *c* depends on another method *k* called by *m*.

Table 1 shows the relation table between classes and methods for the Java program shown in Fig. 1. Such a table forms the formal context to be analyzed. Applying *FCA* technique to the formal context in Table 1, a set of formal concepts can be generated, which is composed of sets of classes sharing sets of methods as shown in Fig. 2. Formal concept is defined as a pair consisting of a set of formal objects (called the *extent*) and a set of formal attributes (called the *intent*) such that the extent consists of all formal objects that depend on the given formal attributes, and the intent consists of all formal attributes depended on by the given formal objects. The graphical representation of concept lattice uses the simple labeling approach to represent the formal concepts in a more compact and readable form (Ganter and Wille, 1986). This representation

Please cite this article in press as: Sun, X., et al., Change impact analysis and changeability assessment for a change proposal: An empirical study ☆☆. J. Syst. Software (2014), http://dx.doi.org/10.1016/j.jss.2014.05.036