Information and Software Technology 55 (2013) 1437–1449

Contents lists available at SciVerse ScienceDirect

Information and Software Technology

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

FCA–CIA: An approach of using FCA to support cross-level change impact analysis for object oriented Java programs ☆

Bixin Li [a,c], Xiaobing Sun [a,b,*], Jacky Keung [d]

[a] School of Computer Science and Engineering, Southeast University, Nanjing, China
[b] School of Information Engineering, Yangzhou University, Yangzhou, China
[c] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China
[d] Department of Computer Science, City University of Hong Kong, China

ARTICLE INFO

Article history:
Received 21 January 2012
Received in revised form 26 December 2012
Accepted 4 February 2013
Available online 5 March 2013

Keywords:
Formal concept analysis
Change impact analysis
Lattice of class and method dependence
Impact factor

ABSTRACT

Background: *Software Change Impact Analysis* (CIA) is an essential technique in software engineering to identifying the potential influences of a change, or determining change entities to accomplish such a change. The results derived, in many cases, ambiguous for the software maintainers, introduces the problem of unclear starting point of these impacted entities.
Objective: In an attempt to address this issue, this work proposes a novel approach for cross-level CIA, producing a ranked list of potentially impacted methods derived from class-level changes. Moreover, the approach of ranking the impact results is expected to be effective for maintainers to distinguish the probability of the impacted methods to be false-positives. Such results provide an eclectic approach for CIA.
Method: The approach, *FCA–CIA*, uses formal concept analysis (*FCA*) to produce an intermediate representation of the program based on the static analysis of the source code. The representation is called Lattice of Class and Method Dependence (*LoCMD*). *FCA–CIA* takes the changed classes in the change set as a whole, and determines the reachable set from the changed classes on the *LoCMD*. Based on the hierarchical property of the *LoCMD*, the impacted methods are ranked according to the impact factor metric which corresponds to the priority of these methods to be inspected.
Result: Empirical evaluations on four real-world software projects demonstrate the effectiveness of the impact factor metric and the *FCA–CIA* technique. The result shows the predicted impacted methods with higher impact factor values are more likely to be affected by the changes. Our study also shows that the *FCA–CIA* technique generates more accurate impact set than the *JRipples* and *ICP* coupling based CIA technique.

1. Introduction

Software maintenance and change are necessary to cope with new requirements, existing faults and change requests, etc. Changes made to software will inevitably have certain unpredictable and undesirable effects on the other parts of the software. Software Change Impact Analysis (*CIA*) is a technique commonly used to identify the potential effects caused by software changes [11]. CIA starts with a set of changed elements in a software system, called the *change set*, and attempts to determine a possibly larger set of elements, called the *impact set*, which requires attention or maintenance effort due to these changes [11]. It plays an important role in software development, maintenance, and regression testing [11,12,50,64]. CIA can be used before or after a change implementation. Before making changes, we can employ CIA for program comprehension, change impact prediction and cost estimation [11,12]. After changes have been implemented, CIA can be applied to trace ripple effects, select test cases, and perform change propagation [50,64,10,49]. The commonly used CIA techniques mainly contain *static CIA* and *dynamic CIA* techniques. Static CIA techniques are often performed by analyzing the syntax and semantic, or evolutionary dependence of the program (or its change history repositories) [2,1,55,44,56,28]. The resultant impact set often has many false-positives, with many of its elements not really impacted [37,43,38]. Thus this impact set they compute is very large and difficult for practical use [11]. Whereas dynamic

☆ This work is supported partially by National Natural Science Foundation of China under Grant No. 60973149, partially by the Open Funds of State Key Laboratory of Computer Science of Chinese Academy of Sciences under Grant No. SYSKF1110, partially by Doctoral Fund of Ministry of Education of China under Grant No. 20100092110022, and partially by the Scientific Research Foundation of Graduate School of Southeast University under Grant No. YBJJ1102.

* Corresponding author at: School of Computer Science and Engineering, Southeast University, Nanjing, China. Tel.: +86 25 5209089; fax: +86 25 52090879.
E-mail addresses: bx.li@seu.edu.cn (B. Li), xbsun@yzu.edu.cn (X. Sun), Jacky.Keung@cityu.edu.hk (J. Keung).

0950-5849/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.infsof.2013.02.003

CIA techniques consider some specific inputs, and rely on the analysis of the information collected during program execution (e.g., execution traces information, coverage information and execution relation information) to calculate the impact set [37,5,43]. Moreover, their impact set often includes some false-negatives, that is, some of the real impacted elements are not identified [11,38]. In addition, the cost of dynamic CIA techniques is usually higher than that of static CIA because of the overhead of expensive dependency analysis during program execution [13,38].

In spite of such a collection of CIA techniques, there still exist several problems with current CIA techniques as follows:

(1) The impact set is expected to have fewer false-positives and false-negatives. To provide an eclectic approach for CIA is necessary.
(2) The impact set computed by most of current CIA techniques is composed of a set of potentially impacted entities (classes, methods or statements). However, software maintainers do not know where to start to inspect the impacted entities in the impact set.
(3) The granularity level of impact set of a CIA technique is often corresponding to that of the change set, i.e., when the change set is at a certain granularity-level (files, classes, class members), the impact set is also at the same granularity-level. Sometimes a cross-level CIA which computes fine-level impact set from coarse-level change set is needed [47,38].
(4) Most current CIA techniques compute the impact set based on computing the union of the impact sets of each changed element in the change set. It does not consider the relation among the changed elements in the change set. But in practice, there exists some relationship among these changed elements.

Faced with these problems, we propose a novel approach, *FCA–CIA*, which uses *Formal Concept Analysis* (*FCA*) to support static CIA for object oriented Java programs in this article. *FCA* is a field of applying mathematics to deal with the study of the relation between entities and entity properties to infer a hierarchy of concepts [23]. For every binary relation between entities and their properties, a lattice can be constructed to provide a remarkable insight into the structure of the original relation [23]. It can be employed to produce a decomposition of an existing program based on the static analysis of the source code [54]. It has been shown that *FCA* is an elegant and powerful code analysis technique for software maintenance in the last few years [54,58]. This article firstly presents a novel representation, called *Lattice of Class and Method Dependence* (LoCMD), a compact and effective representation for CIA. *LoCMD* organizes methods into a hierarchical order. Based on this observation, *FCA–CIA* computes a ranked list of impact set by considering the change set as a whole. In addition, *FCA–CIA* technique is particularly useful for multiple proposed changes. Another advantage of using *LoCMD* is its easy application to CIA. *FCA–CIA* can be easily performed on the *LoCMD* by determining the reachable set from the lattice nodes labeled by the changed classes. The main contributions of this article are as follows:

- Construct a representation *LoCMD* for objected oriented Java programs, which can be easily applied to CIA.
- Propose a cross-level CIA technique, *FCA–CIA*, which starts from class-level changes and produces a ranked list of potentially impacted methods. These methods are ranked according to a novel *impact factor* metric, which displays the priority for inspecting this method.
- Show the effectiveness of the impact factor metric and the *FCA–CIA* technique on four real-world case studies.

This article is organized as follows: in the next section, we propose a novel representation, *Lattice of Class and Method Dependence* (LoCMD), to support *CIA*. Section 3 gives an introduction about the *FCA–CIA* technique based on the *LoCMD*. We conduct some case studies to validate the effectiveness of the *FCA–CIA* technique in Section 4. In Section 5, some related work of *CIA* techniques and applications of *FCA* in software maintenance are presented. Finally, we conclude our CIA technique and show some future work in Section 6.

2. Concept lattice for CIA

2.1. Basics for concept lattice

Formal Concept Analysis (FCA), also called concept lattice, is a field of applying mathematics based on the schematization of *concept* and *conceptual hierarchy* [23]. The basic notions of concept lattice are those of the *formal context* and the *formal concept*, which are defined as follows:

Definition 1 (*Formal context*). A formal context is defined as a triple $\mathbb{K} = (\mathcal{O}, \mathcal{A}, \mathcal{R})$, where $\mathcal{R}$ is a binary relation between a set of formal objects $\mathcal{O}$ and a set of formal attributes $\mathcal{A}$. Thus $\mathcal{R} \subseteq \mathcal{O} \times \mathcal{A}$.

Definition 2 (*Formal concept*). A formal concept, which is simply called concept, is a maximal collection of formal objects sharing common formal attributes. It is defined as a pair (O,A) with $O \subseteq \mathcal{O}$, $A \subseteq \mathcal{A}$, $O = \tau(A)$ and $A = \sigma(O)$, where $\tau(A) = \{o \in \mathcal{O} | \forall a \in A : (o,a) \in \mathcal{R}\} \wedge \sigma(O) = \{a \in \mathcal{A} | \forall o \in O : (o,a) \in \mathcal{R}\}$.

In the above definition, $\tau(A)$ is often said to be the *extent* of the concept and $\sigma(O)$ is said to be its *intent*. Relation between concepts often forms a partial order on the set of all concepts. We often use the following *subconcept* definition to construct a concept lattice [23]:

Definition 3 (*Subconcept*). Given two concepts (O_1,A_1) and (O_2,A_2) of a formal context, (O_1,A_1) is called the subconcept of (O_2,A_2), provided that $O_1 \subseteq O_2$ (or $A_1 \supseteq A_2$). we usually mark such relation as: $(O_1,A_1) \leqslant (O_2,A_2) \Leftrightarrow O_1 \subseteq O_2 \Leftrightarrow A_1 \supseteq A_2$

The set of all concepts of a formal context forms a partial order. Birkhoff has found in 1940 that it was also a complete lattice [9], defined as:

Definition 4 (*Concept lattice*). The concept lattice $\mathcal{L}(Co)$ is a complete lattice. $\mathcal{L}(Co) = \{(O,A) \in 2^{\mathcal{O}} \times 2^{\mathcal{A}} | O = \tau(A) \wedge A = \sigma(O)\}$, where infimum and supremum of two concepts (O_1,A_1) and (O_2,A_2) are defined as: $(O_1,A_1) \wedge (O_2,A_2) = (O_1 \cap O_2, \sigma(O_1 \cap O_2))$, and $(O_1,A_1) \vee (O_2,A_2) = (\tau(A_1 \cap A_2), A_1 \cap A_2)$, respectively.

The complete information for each concept of $\mathcal{L}(Co)$ is given by their extents and intents. However, if the concepts are all labeled with such complete information, the lattice is really very hard to understand. Fortunately, there is a simple way to represent their extents and intents in a more compact and readable form. A lattice element is labeled with $a \in \mathcal{A}(o \in \mathcal{O})$, if it is the most general (special) concept having a (o) in its intent (extent). The lattice element marked with a is thus [23]:

$$\mu(a) = \vee\{co \in \mathcal{L}(Co) | a \in int(co)\}$$

In this formula, $int(co)$ represents the intent of the concept co. And it indicates that all concepts smaller than $\mu(a)$ have a in its intent. Similarly, the lattice element marked with o is:

$$\gamma(o) = \wedge\{co \in \mathcal{L}(Co) | o \in ext(co)\}$$