Information and Software Technology 51 (2009) 405–417

Contents lists available at ScienceDirect

Information and Software Technology

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

Revising cohesion measures by considering the impact of write interactions between class members

Gyun Woo, Heung Seok Chae *, Jian Feng Cui, Jeong-Hoon Ji

Department of Computer Science and Engineering, Pusan National University, 30, Jangjeon-dong, Geumjeong-gu, Busan 609-735, Republic of Korea

ARTICLE INFO

Article history:
Received 18 July 2007
Received in revised form 2 May 2008
Accepted 6 May 2008
Available online 23 May 2008

Keywords:
Software engineering
Metrics/measurement
Object-oriented design
Cohesion

ABSTRACT

Cohesion refers to the degree of the relatedness of the members in a class and several cohesion measures have been proposed to quantify the cohesiveness of classes in an object-oriented program. However, the existing cohesion measures do not differentiate write interactions from read interactions between class members, thus, do not properly reflect the cohesiveness of the class. This paper presents the revised versions of the existing five cohesion measures by considering the impact of write interactions between class members. In addition, we prove that the revised measures can be reduced into the original ones. To demonstrate the importance of write interactions, we have developed tools for automatic computation of the original and the revised cohesion measures and performed a case study where we found that write interactions are so commonly used in classes that they have much influence on cohesion measurement and the revised measures have stronger relations with change-proneness of classes than the original ones.

1. Introduction

Cohesion refers to the relatedness of the elements in a module [23]. A highly cohesive module is one whose elements have tight relationships among them in order to provide a single functionality of the module. On the contrary, a low cohesive module has some elements that have little relation with others, which indicates that the module may contain several unrelated functionalities. It is widely accepted that the higher the cohesion of a module is, the easier the module is to develop, maintain, and reuse.

In the object-oriented paradigm, various cohesion measures for classes have been proposed [3,10,13,14,22,18,21]. In the beginning of research on cohesion measures for classes, researchers just considered syntactic relationship between class members such as interactions between instance variables and methods. However, more recent researches have been tried to identify inherent characteristics of classes which can affect the cohesiveness of classes and incorporate them into cohesion metrics.

Bieman and Kang distinguished public methods from non-public methods and proposed *LCC* (Loose Class Cohesion) and *TCC* (Tight Class Cohesion) only with public methods in a class [3]. They also excluded constructors and destructors in order to remove the impact of artificial connection by those methods. In their comprehensive overview of object-oriented cohesion measures, Briand et al. [4] also addressed that access methods[1] artificially decrease class cohesion, whereas constructors and destructors artificially increase class cohesion. In order to resolve such problems due to access methods, constructors, and destructors, Chae et al. [10] introduced the notion of special methods. They noted that special methods have no influence on class cohesion because those methods are designed to show a specific behavior, interacting inherently with only some of instance variables for the specific purposes. They also tried to improve cohesion measures by considering characteristics of dependent instance variables in a class. That is, they attempted to enhance the existing cohesion measures by including the implicit and hidden interactions between class members due to data dependency [11].

In this paper, we propose an approach to enhancing the existing cohesion measures by incorporating some characteristics that are relevant to class cohesion, but have not been considered before. In general, cohesion depends on the interactions between class members; that is, the more interactions between class members have, the more cohesive the class is. The interactions between methods and instance variables can be classified into two categories: read interactions and write interactions. We note that a write interaction should be considered stronger than a read interaction because the write interaction can affect other methods that read

* Corresponding author.
E-mail addresses: woogyun@pusan.ac.kr (G. Woo), hschae@pusan.ac.kr (H.S. Chae), cuijf@pusan.ac.kr (J.F. Cui), jhji@pusan.ac.kr (J.-H. Ji).

[1] The only behavior of an access method is to provide read or write access to an instance variable of a class. An access method typically references only one instance variable that it provides access to.

0950-5849/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2008.05.014

the instance variable written. The existing cohesion measures do not distinguish write interactions from read interactions. However, we believe that write interactions have more contribution to cohesiveness of a class than read interactions.

We present an approach to improving the existing cohesion measures by emphasizing write interactions between class members. We also demonstrate the importance of our approach by performing a case study with Java class libraries. Discussion on effects of the write interactions is centered on *LCOM1*[2] [13], *LCOM2* [14], *TCC* [3], *Co* [18], and *LCOM5* [22] measures because they have been used in numerous empirical studies for investigating relationship between measures and quality factors such as development/maintenance effort [1,6,12,20,19], fault-proneness [2,8,7,16,21], and testability [9]. In addition, their use is gradually increasing in industry settings. This is manifested by the increasing number of industrial software tools, such as Together Control Center, which support automated computation of *LCOMs*.

The basic concept discussed in this paper was proposed in the previous work[25]. This paper describes extensions in three respects.

- In this paper, two more cohesion measures, namely *Co* and *LCOM5*, are additionally revised to consider the impact of the write interactions. In addition, we give an intuitive description on the necessity of considering write interactions using an example. Furthermore, the reducibility from new measures to the old ones is also proved.
- We have performed a further case study. In this experiment, we computed the original and the revised cohesion values from 1183 classes of JDK. This experiment shows that write interactions are frequently used and they have much influence on cohesion measurement. In addition we performed correlation analysis with change-proneness. This experiment shows that the revised cohesion measures could be better indicators to predicting the change-proneness of classes better than the original ones.
- To perform a case study, we have developed new tools for automating measurement of the original and the revised measures. In the previous paper, we performed a case study with C++ libraries by the tool based on GEN++ [15]. However, GEN++ fails to handle advanced C++ features such as namespace and template. Based on APIs of Together tool, the newly developed tool can handle such features. In addition, Java as well as C++ can be processed.

The rest of this paper is organized as follows: Section 2 briefly describes five existing cohesion measures. Section 3 describes a rationale for the consideration of write interactions, proposes revised versions of the existing cohesion measures, and proves that the revised versions can be reduced into the original ones. Section 4 describes a case study to show the importance of write interactions on cohesion measurement. Finally, conclusions and suggestions for the future works are given in Section 5.

2. The original cohesion measures

This section describes five cohesion measures under study in this paper. First we introduce some basic notations for the clear description of the cohesion measures and then briefly define the cohesion measures using them.

2.1. Basic definitions

This section introduces some basic definitions in order to define the existing five measures (*LCOM1*, *LCOM2*, *TCC*, *Co*, and *LCOM5*) in a unified vocabulary.

Definition 1. For a class C, $V(C)$ denotes the set of instance variables of C that are implemented in C and $M(C)$ denotes the set of methods of C that are implemented in C.

Definition 2. The elements of $V(C)$ can be indexed by unique natural numbers and can be enumerated as $v_1, v_2, \ldots, v_n$ where $n = |V(C)|$. Then, the set of (unordered) pairs of distinct variables $V_p(C)$ can be defined as:

$$V_p(C) = \{(v_i, v_j) \mid v_i \in V(C), v_j \in V(C), i < j\}$$

Similarly, the methods of a class C can be enumerated as $m_1, m_2, \ldots, m_l$ where $l = |M(C)|$ and the set of (unordered) pairs of distinct methods $M_p(C)$ can also be defined as:

$$M_p(C) = \{(m_i, m_j) \mid m_i \in M(C), m_j \in M(C), i < j\}$$

Definition 3. For a method m of a class C, i.e. $m \in M(C)$, $V_U(m)$ denotes the set of instance variables that are directly referenced (read or written) by m. Similarly, $M_U(m)$ denotes the set of methods of C that are directly invoked by m.

Definition 4. For a method m of a class C, $V_R(m)$ denotes the set of instance variables whose values are read by m and $V_W(m)$ denotes the set of instance variables whose values are written by m. For V_R,V_W, and V_U, the following property holds: $V_R(m) \cup V_W(m) = V_U(m)$.

2.2. The definitions of the original measures

In this section, every cohesion measure of interest is briefly defined using the basic notations defined above.

2.2.1. LCOM1

The original definition of *LCOM1* [13] is defined as the number of *unrelated* pairs of methods in a class. When two methods have no instance variable that is referenced in common, the methods are considered unrelated in *LCOM1*. Since the set of variables referenced by m is $V_U(m)$, the original *LCOM1* can be defined as follows:

Definition 5.

$$LCOM1(C) = \left|\{(m_i, m_j) \in M_p(C) \mid V_U(m_i) \cap V_U(m_j) = \emptyset\}\right| \quad (1)$$

Note that $M_p(C)$ itself denotes the set of unordered pairs of distinct methods in C so the condition $m_i \neq m_j$ does not have to be present.

Let us examine a simple example. Fig. 1 shows a *member interaction graph* for C, which shows the interactions between the members of $V(C)$ and the members of $M(C)$. The nodes of the interaction

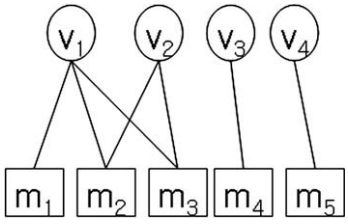

Fig. 1. A member interaction graph for class C.

[2] Lack of COhesion in Methods.