World Engineering & Applied Sciences Journal 6 (2): 104-108, 2015
ISSN 2079-2204

DOI: 10.5829/idosi.weasj.2015.6.2.22148

A Class Cohesion Measure for Evaluation of Resuablity

Neha Gehlot and Ritu Sindhu

Department of CSE, SGT University Gurgaon, Haryana, India

Abstract: Class cohesion is considered as one of most important object-oriented software attributes. It is defined as the degree of the relatedness of the members in a class. Based on instance variables usage criteria the major existing class cohesion metrics are measured. It is only a special and a restricted way of capturing class cohesion. We believe, as stated in many papers, that class cohesion should not exclusively be based on common instance variables usage criteria. We introduce, in this paper, a new criterion, which focuses on interactions between class methods and class instances. We developed a cohesion measurement tool for Java programs and performed a case study on several systems. This paper provides an account of new measures of cohesion developed to assess the reusability of Java classes. The obtained results demonstrate that our new class cohesion metric, based on the proposed cohesion criteria, captures several pairs of related methods, which are not captured by the existing cohesion metrics.

Key words: Metric · Cohesion · Reusability · Inheritance · Methods · Variables

INTRODUCTION

For some disciplines of software engineering software metrics have become essential [Pressman01]. Several software attributes (complexity, coupling, cohesion, etc.). Are assessed using metrics in the field of software quality? They provide, therefore, an important assistance to developers and managers in order to assess and improve software quality during the development process. During the last decade Object technology has been widely used in several areas. Cohesion refers to the degree of the relatedness of the members in a component. High cohesion is a desirable property of software components. It is widely recognized that highly cohesive components tend to have high maintainability and reusability [1] for the measurement of structure quality the cohesion of a component is measured. The cohesion degree of a component is high, if it implements a single logical function.

Several metrics have been proposed in the literature in order to measure class cohesion in object-oriented systems. The major existing class cohesion metrics have been presented in detail and are categorized in [2]. They are based on either instance variables usage or sharing of instance variables. These metrics capture class cohesion in terms of connections among members within a class. They count the number of instance variables used by methods or the number of methods pairs that share instance variables. We believe that it is only a special way of capturing class cohesion, which is based on instance variables usage criteria. These metrics have been experimented and widely discussed in the literature [Basili96, Briand00, Chae98, Chidamber98, ElEmam99 and Henderson-Sellers96]. Several studies have noted that the existing cohesion metrics fail in many situations to properly reflect the cohesiveness of classes [Kabaili00, Chae00]. According to many authors, they do not take into account some characteristics of classes, for example, sizes of cohesive parts as stated in [Aman02] and connectivity among members as stated in [Chae00].

Yourdon and Constantine introduced cohesion in the traditional applications as a measure of the extent of the functional relationships of the elements in a module. They have described cohesion as a criterion for the estimation of design quality. Grady Brooch describes high functional cohesion as existing when the elements of a component (such as a class) all work together to provide some well-bounded behaviour [Booch94]. In the object paradigm, a class is cohesive when its parts are highly correlated. It should be difficult to split a cohesive class. A class with low cohesion has disparate and non-related

Corresponding Author: Neha Gehlot, CSE Department, SGT University Gurgaon, Haryana, India.

members. Cohesion can be used to identify the poorly designed classes. Cohesion is an underlying goal to continually consider during the design process [3].

Beyond these aspects, we believe that the existing metrics fail to reflect properly the properties of class cohesion, particularly in terms of related methods. They are based on restricted criteria and could lead to unexpected values of cohesion in many situations. We believe that class cohesion should not exclusively be based on common instance variables usage as stated in [4] and will have to go beyond this aspect by considering the interaction patterns among class methods. We note that in many situations several methods are functionally related together without sharing any instance variables. We extended the existing criteria by considering different ways of capturing class cohesion. We introduce, in this paper, a new criterion, which focuses on interactions between class methods. We developed a cohesion measurement tool for Java programs and performed a case study on several systems. The obtained results demonstrate that our new class cohesion metric, based on the proposed cohesion criteria, captures several pairs of connected methods, which are not captured by the existing cohesion metrics [5].

Class Cohesion: Existing Metrics: Classes are considered as the basic units of object-oriented software. Classes should then be designed to have a good quality [6]. However, improper modelling in the design phase, particularly improper responsibilities assignment decisions can produce classes with low cohesion. In order to assess class cohesion in object-oriented systems several metrics have been proposed in the literature. Most of the proposed class cohesion metrics are inspired from the LCOM (Lack of Cohesion in Methods) metric defined by Chid amber and Kemmerer [7]. Many authors have redefined the LCOM metric as referenced in the following paragraphs.

Class Cohesion: A New Measure: Class cohesion in our approach, as stated initially in [Badri95], refers essentially the relatedness of public methods of a class, which represent the functionalities used by its clients. It is defined in terms of the relative number of related public methods in the class. The others methods of the class are included indirectly through the public methods. Our approach is comparable to the one adopted by Baseman and Kang in.

Table 1: The major existing cohesion metrics.

Metric	Definition
LCOM1	Lack of cohesion in methods. The number of pairs of methods in the class using no Instance variables in common.
LCOM2	Let P be the pairs of methods without shared instance variables and Q be the pairs of Methods with shared instance variables. Then LCOM2 = |P| - |Q|, if |P| > |Q|. If this Difference is negative, LCOM2 is set to zero.
LCOM3	Consider an undirected graph G, where the vertices are the methods of a class and there is an edge between two vertices if the corresponding methods share at least one instance Variable. Then LCOM3 = | connected components of G |
LCOM4	Like LCOM3, where graph G additionally has an edge between vertices representing Methods Mi and My, if Mi invokes My or vice versa. Co Connectivity. Let V be the vertices of graph G from LCOM4 and E its edges. Then $Co = 2.\frac{\|E\| - (\|V\| - 1)}{(\|V\| - 1).(\|V\| - 2)}$
LCOM5	Consider a set of methods {Mi} (I = 1, …, m) accessing a set of instance variables {Am} (j = 1, a). Let μ (Am) be the number of methods that reference Am. Then $LCOM5 = \frac{(1/a)\Sigma\, 1 \le j \le a\, \mu(Am) - m}{1 - m}$
Coho	$Coho = \frac{\Sigma\, 1 \le j \le a\, \mu(Am)}{m.a}$
TCC	Tight Class Cohesion. Consider a class with N public methods. Let NP be the maximum Number of public method pairs: NP = [N * (N – 1)] / 2. Let NDC be the number of direct Connections between public methods. Then TCC is defined as the relative number of Directly connected public methods. Then, TCC = NDC / NP.
LCC	Loose Class Cohesion. Let NIC be the number of direct or indirect connections between Public methods. Then LCC is defined as the relative number of directly or indirectly Connected public methods. LCC = NIC / NP.

We have revised our initial definition of class cohesion proposed in [Badri95] by extending the methods invocation criterion in the one hand and introducing the concept of indirect usage of attributes defined by Baseman and Kang in [8] in the other hand. We have also extended this concept to the methods invocation criterion.

Direct Relation Between Methods: Two public methods Mi and My may be directly connected in many ways: they share at least one instance variable in common (UA relation), or interact at least with another method of the class (IM relation), or both. It means that: Tami ∩ UA My ≠ ☐ or Mimi n Imp ≠ ☐. The maximum number of public methods pairs, as stated in [Badri95, Bieman95], is n * (n-1) / 2.

Indirect Relation Between Methods: However, two public methods Mi and Mj can be indirectly related if they are directly or indirectly related to a method Mk. The indirect relation, introduced by Bieman and Kang in [Bieman95], is the transitive closure of the direct relation. We use this concept in our approach for identifying the indirect related methods. Thus, a method M1 is indirectly connected with a method Mk if there is a sequence of methods M1, M2, M3, …, Mk such that Mi is directly connected to Mi+1 (i= 1, k-1)[9].

There are public methods of the class C, class c is inherited by some other classes having their own public methods. So the degree of cohesion in the class C based on the direct and indirect relations between its public methods is measured.

The new definition that we propose for class cohesion assessment seems to be more appropriate than the others, particularly the ones supposed taking into account the interactions between methods. It allows capturing more properties of classes, particularly, in terms of connections between methods. Two public methods can be related by calling directly or indirectly, for instance, private (or protected) methods, which do not use any attribute of the class. Such characteristics are captured in our cohesion metric.

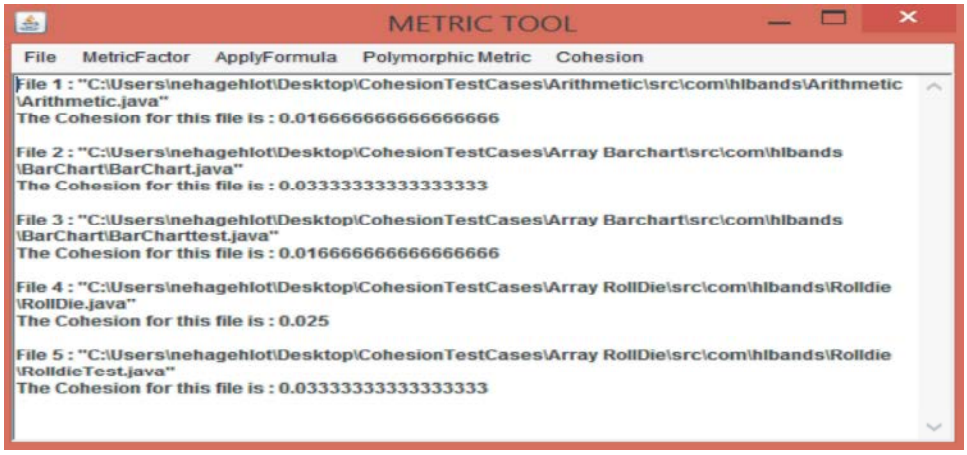
As stated earlier, In OO programming, a class can inherit the methods and attributes of other classes. For example, a class X can inherit the methods and attributes of other class Y if class X is the derived from class Y. This notion of OO programming makes the class Y as a subset of class X. In other words; this phenomena adds methods and attributes to derived class X from the base class Y. In our approach of measuring class cohesion, we will use relatedness among the public methods of a class. We can see that the notion of inheritance may add new members (i.e. protected and public attributes and methods) in the derived class and may produce an effect on the value of cohesion for the derived class. For measuring the class cohesion, we will add public and protected members of base class to the sub class and we will treat such derived members similar to other members of the class [10].

Class Cohesion reusability metric = K* [(MP+Vp)*IC]
MP = total number of public methods
Vp = total number of public variables
IC = No.of inherited classes

K is proportionality constant with a value 0.5

Using proportionality constant k the raw metrics values are empirically adjusted for calculation in order to map the metrics values into a scale from 0.0 to <1.00 (0 ± 100%)

Implementation



Test Case Analysis: Measure of cohesive metric is calculated as a measure of single functionality in a java class the analysis shows measure of cohesion in the code. Measure of the proposed metric that is cohesive metric is calculated for several test cases and presented. It is observed as the public methods in project increases, the value of the metric increases to very high values due to presence of high degree of cohesion thus enhancing reusability measure within the code.

Cohesion has been claimed to improve reuse and reusability, to fully measure productivity we may need some effective means to measure the degree of cohesion. We also need to measure cohesion in order to establish the veracity of claims is made about it with respect to reusability. It is the measurement of cohesion that is the subject of this paper. In this paper, we present our attempt to measure cohesion by means of methods used in a java class. This is a technique which has been traditionally used for understanding other forms of behaviour, such as improving performance or determines the coverage of test cases. We apply similar techniques in an attempt to measure the cohesion that takes place during the execution of an application. Thus in this paper cohesion due to interactions between methods and variables is taken as directly proportional to reusability factor which is a quality factor and thus gave a cohesive reusability present due to cohesion in a java class.

Table 2: Table measuring metrics

Project	*Class*	*Cohesion Metric*
Revenue	Revenue Predictin. java	0.14
	Revenue Test. java	0.83
Show quality	Average.java	0.33
	Averagetest.java	0.33
Turtle graphics	Turtle.java	0.33
	Turtletest.java	0.33
Pythagorean table	Pythagorean.java	FI
	Pythagoreantest.java	0.16
Population growth	Populationgrowth.java	0.1
	Populationgrwothtest.java	0.16
Dice Rolling	Dicerolling.java	0.1
	Dicerollingtest.java	0.16
Barchart	Barchart.java	0.03
	Barcharttest.java	0.016
Code	Signin.java	0.20
	Signup.java	0.20
	Testrun.java	0.20
	Uploadfile.java	0.20
App	Uploadnewfile.java	0.20
	Beeper.java	0.03
Atm	atm.java	0.01

CONCLUSION

In the compiler research literature and provide important and relatively easily-computed information the static metrics are quite commonly used. In this paper we have focused on computing metrics as a means of assessing the actual behaviour of a program. More relevant views of the program to compiler and runtime optimization developers can be provided to by this information collected. In this paper both the compile time runtime behaviour is calculated by measuring the runtime and compile time cohesion which is associated with the concept of method reusability to provide the resultant proposed metric. The proposed metric is validated on a java metric tool and results are validated through test cases to ensure that the results are accurate and validated. Thus providing with a cohesion measure in a java class and scope to enhance the functionality and reusability of a java code.

REFERENCE

1. Hristov Danail, Oliver Hummel, Mahmudul Huq and Werner Janjic, 2012. Structuring Software reusability Metrics for Component-Based Software Development, ICSEA 2012: The Seventh International Conference on Software Engineering Advances.
2. Washizaki Hironori, Hirokazu Yamamoto and Yoshiaki Fukazawa, 2003. A Metrics Suite for Measuring Reusability of Software Components, Proceedings of the 9th International Symposium on Software Metrics September, IEEE Computer Society, Washington, DC, USA, pp: 211-223.
3. Gui, G. and P.D. Scott, 2008. New Coupling and Cohesion Metrics for Evaluation of Software Component Reusability, Proc. Of the Intern. Conference for Young Computer Scientists, pp: 1181-1186.
4. Sagar Shrddha, N.W. Nerurkar and Arun Sharma, 2010. A soft computing based approach to estimate reusability of software components, ACM SIGSOFT Software Engineering Notes, 35(5): 1-5.
5. Boxall, M.A.S. and S. Araban, 2006. Interface Metrics for Reusability Analysis of Components, Australian Software Engineering Conference (ACWEC'04), Melbourne, Australia, pp: 40-50.
6. Eun Sook Cho, Min Sun fim and Soo Dong Kim, 2005. Component Metrics to Measure Component Quality, Proceedings of the eighths Asia-Pacific Software Engineering Conference, pp: 1530-1362/01.

7. Hummel O. and C. Atkinson, 2006. Using the Web as a Reuse Repository, Reuse of Off-the-Shelf Components, Lecture Notes in Computer Science, Springer, 4039: 298-311.
8. Poulin, J., 1994. Measuring Software Reusability, Proceedings of the 3rd International Conference on Software Reuse: Advances in Software Reusability, IEEE Computer Society Press, Los Alamitors, pp: 126-138.
9. Rotaru, O.P. and M. Dobre, 2009. Reusability Metrics for Software Components, In Proceedings of the ACS/IEEE 2005 International Conference on Computer Systems and Applications, pp: 24.
10. Yingmei Li, Shao Jingbo and Xia Weining, 2012. On Reusability Metric Model for Software Component, Software Engineering and Knowledge Engineering: Theory and Practice Advances in Intelligent and Soft Computing, 114: 865-870.