Journal of mathematics and computer Science 9 (2014) 149 - 156

Contents list available at JMCS

Journal of Mathematics and Computer Science

Journal Homepage: www.tjmcs.com

Measuring Cohesion and Coupling of Object-Oriented Systems

[1] Mahdi Saadati, [2] Homayoon Motameni

[1] *Islamic Azad University Sari Branch*
Mahdi_Saadati_1366@Yahoo.Com
[2] *Assistant Prof. Member of Faculty Computer Engineering Department,*
Islamic Azad University Sari Branch
Motameni@iausari.ac.ir

Article history:
Received August 2013
Accepted November 2013
Available online November 2013

Abstract

Cohesion and coupling are considered amongst the most important properties to evaluate the quality of a design. In the context of OO software development, cohesion means relatedness of the public functionality of a class whereas coupling stands for the degree of dependence of a class on other classes in OO system. Given two lines of code, A and B, they are coupled when B must change behavior only because A changed. They are cohesive when a change to A allows B to change so that both add new value. In this paper, a new metric has been proposed that measures the class cohesion on the basis of relative relatedness of the public methods to the overall public functionality of a class. The proposed metric for class cohesion uses a new concept of subset tree to determine relative relatedness of the public methods to the overall public functionality of a class. A set of metrics has been proposed for measuring class coupling based on three types of UML relationships, namely association, inheritance and dependency. The reasonable metrics to measure cohesion and coupling are supposed to share the same set of input data. Sharing of input data by the metrics encourages the idea for the existence of mutual relationships between them.

Keywords: class cohesion, class coupling, dependency, inheritance, association.

1. Introduction

Cohesion and coupling are considered amongst the most important metrics for measuring the structural soundness of OO system. In the OO paradigm of software development cohesion means extent to which the public methods of class perform the same task [Bieman & Kang 1995], whereas coupling means the degree of dependence of a class on other classes in the system [2]. If we look into the existing OO metrics for cohesion, coupling, size and reuse, we will notice that all of these metrics use the shared input data. Usage of the shared input data encourages our basic supposition about the existence of relationships among the mentioned OO metrics. Based on our basic supposition, we have attempted to find out relationships among OO metrics for cohesion, coupling.

For measuring class cohesion we have proposed a new metric that measures the relatedness among the public methods on the basis of their relative contribution to the overall public functionality of a class. To determine the relative contribution of public methods, a new concept of subset tree has been used. Our proposed metric, measures the class cohesion in interval of 0 to 1 inclusively. Almost all generally supported notions of OO paradigm are incorporated in the proposed metric.

As stated earlier class coupling means the degree of dependence of class among other class in OO system. We propose that one of the possible ways to measure the class coupling is by using the UML relationships among the classes. Such UML relationships are association, inheritance and dependency. This approach to measure class coupling has enabled us to distinguish among the different coupling types based on UML relationships and has also given us opportunity to analyze the effects of each UML relationships separately. Each type of UML relationship based coupling (association coupling, inheritance coupling and dependency coupling) has been studied against the class cohesion [4].

2. COHESION

Here, the derivation of a new metric for class cohesion is described.

2.1. Metric for Class Cohesion

The proposed metric for class cohesion measures the relatedness among the public methods of class on the basis of their relative contribution to overall public functionality of class. A new concept of subset tree has been used to determine the relative contribution of public methods to overall public functionality. Relatedness among the public methods of a class is determined on the basis of common usage of member attributes.

2.2. Set representation of class

We can express public methods and all attributes of class X in a form of a set S(X); such that set S(X) represents the union of set M(X) and A(X) in following manner [1].

$S(X) = M(X) \cup A(X)$

Set M(X) contains public methods of class X, whereas the set A(X) contains the all attributes of the class X regardless of their scope and type.

M(X) = {m1, m2, m3} set of member public methods

A(X) = {a1, a2, a3} set of all member attributes

2.3. Group determination in a class

A group of methods can be identified by determining the similarity of work in the public methods of the class. For identifying the relatedness of work, we will use the set M(X) that represents all public methods in class X. Methods in set M(X) can form different subsets of M(X) containing different numbers of methods as their elements. We will call these subsets of M(X) as groups and we will represent them with Gi. In set M(X) following are the possible groups excluding the empty set.

G1= {m1} G1 is subset of M(X)

G2= {m2} G2 is subset of M(X)

G3= {m3} G3 is subset of M(X)

G4= {m1, m2} G4 is subset of M(X)

G5= {m2, m3} G5 is subset of M(X)

G6= {m3, m1} G6 is subset of M(X)

G7= {m1, m2, m3} G7 is subset of M(X)

We supposed each of these groups is representing a functionally group by making the set of methods with respect to certain commonality. A coherent class is expected to represent minimum number of functionality groups through its public methods.

Different approaches have been used to find commonality among the methods of class. In design metrics for class cohesion (such as CAM) the commonality among parameter in the parameter list is a choice but in the implementation metrics for class cohesion common usage of member attribute by the member methods is considered as better choice. Usage of common attribute provides a clearer insight to a method than the common parameter in the parameter list.

Let's take the example class Queue to identify the similar groups of public methods. Class Queue is consisted of four member private attributes, namely count, front, end and temp and four public member methods, namely empty, count, append and serve.

In the figure 1, we have shown the public methods with rectangles and attributes with ovals. A link between a rectangle and an oval is representing the usage of attribute by a method. For instance, in figure 1, there are four lines which are connecting attribute count with four different methods count, empty, append and serve. That means attribute count is used by four methods [2].

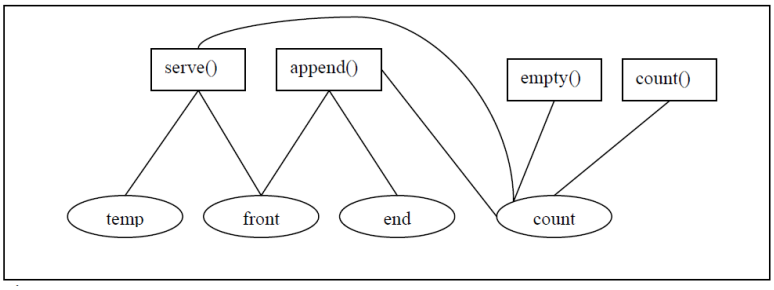

Figure 1 - the public methods with rectangles and attributes with ovals

From the figure 1, we can identify following groups:

Group 1: Attribute temp is used by serve ()

Group 2: Attribute front is used by serve () and append ()

Group 3: Attribute end is used by append ()

We can also express the groups as follows:

G1 = {serve ()}

G2 = {serve (), append ()}

G3 = {append ()}

G4 = {serve (), append (), empty (), count ()}

In order to calculate the class cohesion of class Queue, we will first measure the relatedness of each method with the identified groups. We define the relatedness of methods with identified groups in following way. Relatedness of public method M with identified group(s) is a ratio between number of its occurrences in all group(s) to total number of identified groups in class. Relatedness R of Method M is given by the following formula.

$$R(M) = \frac{MO}{TG} \qquad (1)$$

Where MO represents number of times a method M occurs in all groups and TG represents total number of groups. In following, we have calculated the values of relatedness for methods of class Queue.

R (append) = 3/4 = 0.75

R (count) = 1/4 = 0.25

R (empty) = 1/4 = 0.25

R (serve) = 3/4 = 0.75

Measurement of relatedness with identified groups for each method is an integral part of class cohesion; therefore we suppose the class cohesion as an average measurement of relatedness R (M) of all public methods and we define it by following expression for a class X.

$$Cohsion(X) = \frac{\sum R(M)}{TM} \qquad (2)$$

TM stands for total numbers of public methods in class X and R (M) stands for the measurement of relatedness for public method of class X. In following, the class cohesion for class Queue is calculated using the Equation 2.

Cohesion (Queue) = (0.75 + 0.25 + 0.75 + 0.25) / 4 = 0.50

Class cohesion value for class Queue indicates the problem in our supposition of metric for Class Cohesion (Equation 2), which we have presented so far. In fact, we have neglected the subsets formation among the identified groups of public methods. Neglecting of subsets formation amongst the groups has affected the results of class cohesion. We were considering occurrence of method in all groups equally. Occurrence of methods in groups cannot be weighted equally, because some groups are more associated with the overall public functionality of class and some groups are less associated with overall public functionality of class. Therefore, to measure class cohesion correctly, we need a mechanism to measure the relative relatedness of public methods to overall public functionality of class.

2.4. Subset tree formation

A subset tree is composed of nodes representing sets with public methods as their elements. Such sets group the public methods using a same attribute. Identified sets are placed in relationship of subset to parent nodes in the hierarchy of subset tree. Using a subset tree helps in determining relative contribution of method to overall public functionality of a class. From the example of the class Queue, we have identified following groups of related methods.

G1 = {serve ()} G1 is subset of G2 and G4

G2 = {serve (), append ()} G2 is subset of G4

G3 = {append ()} G3 is subset of G2 and G4

G4 = {serve (), append (), empty (), count ()} G4 is subset to none

As shown above groups are forming subsets, we can also show subset formation with help of a subset tree. In a subset tree, every node represents a group and child to that node represents the subset of the node. Figure 2 shows a subset tree for the groups of class Queue.

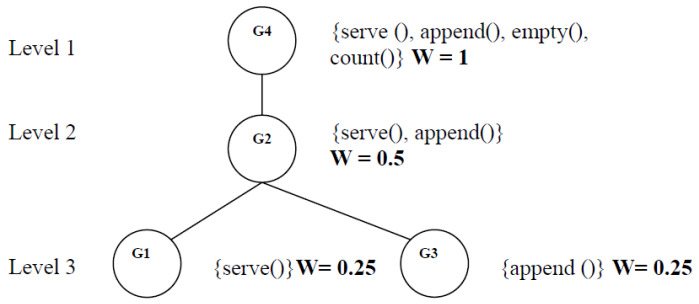

Figure 2- subset tree for the groups of class Queue

Subset tree in figure 2 is showing how different groups are coming on different level in the tree. The Group G4 on root level is weighted highly connected with the class as it contains all the public methods in it. The group G2 at the Level 2 is comparatively less connected with the class, whereas the groups G1 and G3 have a comparatively lower association than G2 and G4 with the class. As you can see, from the figure 2, we have assigned the weights Wi to all group Gi (nodes) in the subset tree using following equation 3.

$$\text{Weight of group G} = \frac{\text{Number of methods in a group}}{\text{Total number of public methods in subset tree}} \qquad (3)$$

Our previous definition for measuring the relatedness of a method R (M), is ignoring the effect of relative association of methods with class. Now, we will redefine the formula for relatedness of methods. To redefine relatedness R of method M; we will first define S (M) the Sum of weight for a method M based on M appearance in groups of a subset tree.

$$S(M) = \sum WG(M) \qquad (4)$$

WG (M) stands for the weight Wi of group Gi for a method M on it occurrence in a group Gi. If we recall the example of class *Queue* and keep the weights of subset tree in mind. The values of S(M) for method "append" will be following:

S (append) = WG2 (append) + WG4 (append) + WG3 (append) = 0.50+ 1.0 + 0.25 = 1.75

Method append occurs on three groups, namely G2, G3 and G4, and therefore, we have added the weights of these groups for method "append" to calculate S(M). We have followed the same guidelines for calculating the S(M) values for other three methods of the class Queue.

S (count) = WG4 (count) = 1.0

S (empty) = WG4 (empty) = 1.0

S (serve) = WG1 (serve) + WG3 (serve) + WG4 (serve) = 0.50 + 0.25 + 1.0 = 1.75

Every subset tree in a class contributes to overall functionality of the class and every method in the subset tree contributes to overall functionality presented by the subset tree, but such contribution may not be equal on both levels. If we just consider the contribution of method for a subset tree, we can argue that method which has highest S(M) value in the subset tree is the most related method to the overall functionality presented by the subset tree. Based on this argument, we redefine the relatedness of method as a relative measure to maximum S(M) for method in subset tree. Therefore, we have divided the S(M) for a method M by the maximum S(M) for a method in a given subset tree. The formula for measuring the relatedness of public methods can be defined using equation 5.

$$R(M) = \frac{S(M)}{MaxS(M)inTree} \qquad (5)$$

Based on the formula; we have calculated the following values of relatedness for all methods in the class Queue.

Maximum of S (M) for all methods in the subset tree is 1.75

R (append) = 1.75/1.75 = 1.0

R (count) = 1.0/1.75 = 0.57

R (empty) = 1.0/1.75 = 0.57

R (serve) = 1.75/1.75 = 1

For measuring the cohesion for the subset tree, we will calculate the average measurement of relatedness R(M) of all methods in the subset tree with the help of following equation 6.

$$Cohsion(SubsetTree) = \frac{\sum R(M)}{TM} \qquad (6)$$

TM stands for the total number of methods in subset tree. For the class Queue we will get following results:

Cohesion (Class Queue) = (1.0+0.57+0.57+1.0)/4 = 0.785

In the case of class Queue, we have only one subset tree, therefore the cohesion of subset tree is actually the cohesion of class Queue. This value of cohesion for the class Queue helps us to recognize it as a cohesive class. The difference between our previously calculated value (0.50) for cohesion of class Queue and this value is because of weights, we have assigned to each group in the subset tree. And, previously, we have ignored the fact that each group contributes to overall functionality of a class differently.

3. COUPLING

In OO design, the coupling of a class means the measurement of the interdependence of class with the other classes.

3.1. UML relationships

According to [Booch el at. 1999], in OO modeling there are three types important relationships among the classes, namely dependency, inheritance and association relationships .Dependency represents the using relationships among the classes; inheritance relationship connects generalized classes to their specialized classes; and association relationship shows the structural relationship among the objects. In following section, we will discuss these relationships briefly [3].

3.2. Dimensions of class coupling

Based on the types of UML relationships, we have found three types of coupling among the classes namely, dependency, inheritance and association. A class in OO implementation may have all such types of coupling with other classes. This observation makes the coupling a three dimension function that can be expressed by something like following equation.

$$Coupling(Class) = aA_c + bD_c + cI_c \qquad (7)$$

Where a, b and c are the coefficients of unknown values and Ac, Dc and Ic are representing association, dependency and inheritance types of coupling respectively. We are also not sure about the degree of stated equation. Therefore, we are unable to form an equation based on three dimensions of coupling.

We can imagine intuitively all dimensions of coupling have a different contribution to over class coupling. But the weights of their contributions by the types of coupling are undetermined. Undetermined weights also hinder us to define an ordinal scale classification for measuring the overall class coupling.

4. CONCLUSION

Our proposed metric, that measures class cohesion on the basis of relative relatedness of public methods to the overall public functionality of class, is resulted better than the metrics that only measure the class cohesion on the basis of relatedness among the public methods. Approach to measure class coupling by counting the number of instances where a class is dependent on other classes via UML relationships (Association, Inheritance and Dependency), can enable us to analyze the class coupling more granularly.

5. REFERENCES

[1] J. Bieman and L. Ott, "Measuring functional cohesion". IEEE Trans. Software Engineering, 20(8):644– 657, Aug. 1994.

[2] James M. Bieman and Byung-Kyooh Kang "Measuring Coupling and Cohesion: An Information-Theory Approach" Edward B. Allen, Taghi M. Khoshgoftaar, Florida Atlantic University, Boca Raton, Florida USA

[3] S. Chidamber and C. Kemerer, "Measurement of Cohesion and Coupling in OO Analysis Model Based on Crosscutting Concerns, O. Ormandjieva, M. Kassab, C. Constantinides, 20 (1994) 476–493.

[4] Dirk Beyer, Claus Lewerentz, and Frank Simon, "Measuring Cohesion and Coupling of Object-Oriented Systems - Derivation and Mutual Study of Cohesion and Coupling", IWSM 2000.