International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:4, No:2, 2010

Determining the Minimum Threshold for the Functional Relatedness of Inner-Outer Class

Sim Hui Tee, Rodziah Atan, and Abdul Azim Abd Ghani

***Abstract*—**Inner class is a specialized class that defined within a regular outer class. It is used in some programming languages such as Java to carry out the task which is related to its outer class. The functional relatedness between inner class and outer class is always the main concern of defining an inner class. However, excessive use of inner class could sabotage the class cohesiveness. In addition, excessive inner class leads to the difficulty of software maintenance and comprehension. Our research aims at determining the minimum threshold for the functional relatedness of inner-outer class. Such minimum threshold is a guideline for removing or relocating the excessive inner class. Our research provides a feasible way for software developers to define inner classes which are functionally related to the outer class.

***Keywords*—**Cohesion, functional relatedness of inner-outer class, inner class.

I. Introduction

INNER class is a specialized class that defined within a regular outer class [5][6]. It is a feature of some object-oriented programming languages, such as Java, that is used to assist the functionality of outer class. Hence, inner class is sometimes referred as helper class of its outer class. Although inner class is a component of outer class, it is different functionally from other class components such as attributes and methods. Inner class could be instantiated like a regular outer class [6]. Furthermore, inner class has all the features that a regular outer class possesses [6]. Inner class can contain methods and attributes. In addition, an inner class can contain other inner classes.

The functional relatedness between inner class and outer class is the main concern of defining an inner class. It is because the principle of inner class design is to assist its outer class. Thus, inner class should be cohesive to its outer class. Inner class should be defined to perform similar tasks as its outer class does. The functional dissimilarity between inner and outer class is always a bad programming practice because the former is functionally irrelevant to the outer class. The functional dissimilarity between inner-outer classes suggests that there is an excessive use of inner class. Excessive inner classes results in the difficulty of software maintenance and comprehension. Excessive inner class lowers the maintainability of software from the perspective of analyzability, stability and changeability that is set out by Heitlager et al [11]. In addition, excessive inner class adds complexity to the software. Thus, it is favorable to remove or relocate the excessive inner class in order to yield high class cohesion. A well designed inner-outer class relationship is more likely to be fault free and more adaptable [10].

Our research aims at determining the minimum threshold for the functional relatedness of inner and outer class. The proposed minimum threshold in our research is a guideline for removing or relocating the excessive inner class. Our research provides a feasible way for software developers to define inner classes which are functionally related to the outer class.

Sim Hui Tee is with Faculty of Creative Multimedia, Multimedia University, Malaysia (e-mail: shtee@mmu.edu.my).

Rodziah Atan, Abdul Azim Abd Ghani are with Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Malaysia.

II. Related Works

Chidamber and Kemerer proposed Lack of Cohesion in Methods (LCOM) to measure the functional relatedness of a class component [1][8]. LCOM is based on the number of disjoint sets of attributes that are used by the methods [1]. It is an inverse cohesion measure. A high value of LCOM indicates low cohesion and vice versa [1][9]. High value of LCOM is not favorable because it implies that the components of a class are not functionally related. LCOM was proposed as a measure for a general class [7], as Chidamber and Kemerer did not mention specifically that it can be used for inner class. However, LCOM is incapable of measuring the functional relatedness between inner and outer class because it is dealing with the common method-attribute connection within a single class.

Briand et al developed RCI that measures the extent to which individual methods use attributes or other methods [2]. RCI value of a class is the ratio of the number of actual interaction to that of possible interaction [2][3]. Briand et al define interaction as (1) data-data interaction, which is the usage between attributes or type declarations; and (2) data-method interaction, which involves the usage of data by methods. However, Briand et al asserts that RCI is not applicable to measuring inner class [2]. Furthermore, RCI cannot be used to measure the functional relatedness between inner and outer class.

LCOM and RCI measure the cohesion of a class based on method-attribute reference. These metrics do not concern the dependence relationships among class components. Zhou et al proposed a Dependence Relationships Based Cohesion Metric (DRC) to address the dependence relationships among

class components [4]. DRC defines the cohesion of a class as the average of dependence degrees of all methods and attributes [4]. Similar to LCOM and RCI, DRC is not applicable to measuring the functional relatedness between inner and outer class.

Previous works focus on the functional relatedness among class components. None of the works investigate the functional relatedness between inner and outer class. Previous works shed light primarily on method-attribute usage and dependency within a regular class. Our research fills the research gap by proposing a method to determine the minimum threshold for the functional relatedness between inner and outer classes. Such minimum threshold will eliminate the excessive inner class and hence enhance the cohesion of class.

III. Excessive Inner Classes

Inner classes are nested within outer class. Fig. 1 illustrates the basic structure of inner class.

```
class A{

   class B{

   }

}
```

Fig. 1 Basic structure of inner class

In Fig. 1, class *A* is an outer class which contains an inner class named *B*. Fig. 1 represents the most basic structure of inner class. However, inner class may contain attributes, methods, constructors and further inner classes, as shown in Fig. 2.

```
class A{

   class B{
      int x,y;
      B( ){  }
      void run( ){  }

      class C{

      }

   }

}
```

Fig. 2 Inner class with its components

In Fig. 2, class *A* is the root outer class (First level). Class *B* is the direct inner class of *A*; whereas class *C* is the indirect inner class of *A*. In class *B*, there are attributes *x* and *y*, constructor, method *run ( )* and class *C*. Class *C* represents an inner class that contained by class *B*. Class *B* is the direct outer class of class *C*. On the same vein, class *C* may contain further inner classes.

There is a possibility of excessive inner classes that defined in a root outer class when the depth of inner class increases. The phenomena of excessive inner class imply that the functional relationship between inner and outer class is loosely related. A loose functional relationship of inner-outer class is a sign of low cohesion. Fig. 3 depicts a scenario where the phenomenon of excessive inner class occurs.

```
class House{

   class Room{

      int getSize( ){
       //code implementation..
      }
      class Fan{

        int getWeight( ){
         //code implementation..
        }

      }

   }

}
```

Fig. 3 Excessive inner class

An inner class is a functional subset of its outer class. It is supposed to carry out a specific function for its outer class. In Fig. 3, inner class *Room* has a method *getSize()* which returns the size of room of a house. The definition of class *Room* is conceptually appropriate because it is functionally related to its outer class *House*. However, class *Room* contains an inner class *Fan*, which returns the weight of a fan. Class *Fan* is not functionally related to its outer class *Room* because it does not perform a task which is related to the class *Room*. Thus, the phenomenon of excessive inner class occurs at class *Fan*. It should be removed from the program for the sake of class cohesiveness.

There are some cases where an excessive inner class should be relocated instead of being removed from the program. Relocation of inner class should be considered when an excessive inner class is found functionally related to other part of program within the scope of root outer class. Fig. 4 depicts an example of an excessive inner class that needs relocation.

```
class Objects{

   class Furniture{

   }

   class Book{
       class Chair{

           int getWeight( ){
            //code implementation..
           }

       }

   }
}
```

Fig. 4 Excessive inner class that needs relocation

In Fig. 4, class *Objects* is the root outer class which contains two direct inner classes, which are *Furniture* and *Book*. Class *Book* contains an inner class *Chair*. However, class *Chair* is not supposed to perform a *Book*-specific task. Hence, it should be relocated to class *Furniture*, as shown in Fig. 5.

```
class Objects{

   class Furniture{
          class Chair{

           int getWeight( ){
            //code implementation..
           }

       }
   }

   class Book{


   }
}
```

Fig. 5 Relocation of excessive inner class

Relocation of class *Chair* to class *Furniture* implies that the former performs a *Furniture*-specific task. After relocation of class *Chair*, the phenomenon of excessive inner class has been eliminated.

IV. Determining the Minimum Threshold for the Functional Relatedness of Inner-Outer Class

We propose a new method to determine the minimum threshold for the functional relatedness of inner-outer class. The method-attribute and method-method reference between inner class and its direct outer class are the factors that determine the functional relationship between inner-outer classes. Total number of methods and attributes are counted, for inner and its direct outer class. The minimum threshold for the functional relatedness of inner-outer class is defined as below:

$$\frac{R}{C} > 0$$

R is a set of inner class components that reference to its direct outer class components. Reference of outer class components implies that an inner class is functionally using or connecting to its outer class. C is a set of components of the direct outer class. C encompasses methods and attributes of the direct outer class. Greater ratio of R to C implies greater functional relatedness between an inner class and its direct outer class. When the set R is an empty set, the R to C ratio is 0, which implies that there is no component of inner class references to the component of its outer class. In the event that R is an empty set, the inner class should be removed or relocated because the inner class is not functionally related to its direct outer class.

Fig. 6 illustrates an inner class *B* which contains a method called *BMethod( )*.

```
class A{

  int x,y;

  void AMethod( ){

  }

  class B{

    void BMethod( ){

       AMethod( );
       x=10;

    }
  }
}
```

Fig. 6 Reference of inner class to outer class

In Fig. 6, class *A* is the direct outer class of inner class *B*. Class *A* contains two attributes (*x, y*) and a method called *AMethod( )*. Class *B* contains a method *BMethod( )* that references to attribute *x* and invokes *AMethod( )*. Hence, there are two references between the inner and outer class. Based on Fig. 6, Table I shows a record of the R to C ratio.

TABLE I
R to C Ratio between Class *B* and Class *A*

R	C	R/C
2	3	0.667

The R to C ratio of inner class *B* and its direct outer class *A* is 0.667, which passes the minimum threshold value (>0). The obtained value means that out of three outer class components, the inner class makes two references to its direct outer class. The observed reference between inner-outer classes suggests that inner class is functionally related to its direct outer class in two aspects.

Fig. 7 illustrates an example where the inner class fails to meet the minimum threshold. There is no reference between inner and outer classes.

```
class C1{

  int i,j,k;
  void printSum( ){
    System.out.println(i+j+k);
  }

   class C2{

     void C2Method( ){

       int v;

     }

   }
}
```

Fig. 7 Absence of inner-outer reference

Fig. 7 shows an example of the absence of inner-outer reference. Class *C1* has four components (three attributes and one method) and an inner class *C2*. Inner class *C2* contains a method named *C2Method( )*. A local variable *v* is declared in *C2Method( )*. It is apparent that the inner class *C2* makes no reference to any components of *C1*. Based on Fig. 7, Table II shows a record of R to C ratio.

TABLE II
R TO C RATIO BETWEEN CLASS *C2* AND CLASS *C1*

R	C	R/C
0	4	0

The R to C ratio of inner class *C2* and its direct outer class *C1* is 0, which does not pass the minimum threshold value. The obtained result implies that out of four outer class components, the inner class makes no references to its direct outer class. The absence of reference between inner-outer classes suggests that inner class is not functionally related to its direct outer class in any aspects. Hence, inner class *C2* is suggested to be removed.

There is possibility of set R containing more element than set C. In such scenario, R to C ratio will be greater than 1. There is no upper limit for the ratio. Greater R to C ratio implies that the degree of functional relatedness between inner and outer class is high. High functional relatedness implies high cohesion between inner and outer class. Fig. 8 illustrates an example of highly related inner-outer class.

```
class COuter{
  String message;

  void printMsg(String msg){
   System.out.println(msg);
  }

    class CInner{

        void run( ){
           message="hi";
           printMsg(message);
           message="How are you?";
           printMsg(message);
         }

    }

}
```

Fig. 8 Highly related inner-outer class

Class *COuter* contains two components and an inner class *CInner*. *CInner* contains a method *run( )* that makes four references to the components of outer class. Based on Fig. 8, Table III shows a record of R to C ratio.

TABLE III
R TO C RATIO BETWEEN CLASS *CINNER* AND CLASS *COUTER*

R	C	R/C
4	2	2

The R to C ratio of inner class *CInner* and its direct outer class *COuter* is 2, which passes the minimum threshold value (>0). Observing that set R is greater than set C, it implies that the inner class is highly related to its outer class. The obtained ratio value means that out of two outer class components, the inner class makes four references to its direct outer class. The observed reference between inner-outer classes suggests that inner class is functionally related to its direct outer class in four aspects.

V. CONCLUSION AND FUTURE WORKS

Our research proposes a new method to determine the minimum threshold for the functional relatedness of inner-outer class. We have shown that greater R to C ratio implies a high degree of functional relatedness between inner-outer classes. In addition, we suggest the software practitioners to remove the excessive inner classes which fail to meet the minimum threshold.

In future, investigation should be carried out to study a mechanism for the relocation of excessive inner classes. Relocation of excessive inner classes is more complicated than removal of excessive inner classes because it may add complexity to software if not carefully planned. Furthermore, future works should focus on the impact of inheritance on the functional relatedness of inner-outer class.

REFERENCES

[1] S.R.Chidamber and C.F.Kemerer. A metrics suite for object oriented design. In *IEEE Transactions on Software Engineering*, Vol 20. No 6. 1994. Pp476-493Xxxxx

[2] L.C. Briand, J.W.Daly, and J.Wust. A unified framework for cohesion measurement in object-oriented systems. In *Empirical Software Engineering*, 3, 1998, pp.65-117.

[3] Y.Zhou, B.Xu, J.Zhao, and H.Yang. ICBMC: an improved cohesion measure for classes. In *Proceedings of the International Conference on Software Maintenance (ICSM'02)*, 2002.

[4] Y.Zhou, L.Wen, J.Wang, Y.Chen, H.Lu, and B.Xu. DRC: a dependence relationships based cohesion measure for classes. In *Proceedings of the Tenth Asia-Pacific Software Engineering Conference (APSEC'03)*, 2003. Pp1-9.

[5] H.Schildt. *Java 2: A Beginner's Guide*. USA: McGraw-Hill/Osborne. 2003.

[6] C.S.Horstmann and G.Cornell. *Core Java 2*. USA: Prentice Hall. 2004.

[7] R.Barker and E.Tempero. A large-scale empirical comparison of object-oriented cohesion metrics. In *Proceedings of 14th Asia-Pacific Software Engineering Conference (APSEC 2007)*. 2007. Pp414-421.

[8] S.Counsell, S.Swift, A.Tucker. Object-oriented cohesion as a surrogate of software comprehension: an empirical study. *5th IEEE International Workshop on Source Code Analysis and Manipulation (SCAM'05)*. 2005.

[9] S.Counsell, E.Mendes, S.Swift. Comprehension of object-oriented software cohesion: the empirical quagmire. *10th International Workshop on Program Comprehension (IWPC'02)*. 2002.

[10] G.Gui. Component reusability and cohesion measures in object-oriented systems. In *Information and Communication Technologies*, 2nd Volume. 2006. Pp 2878-2882.

[11] I.Heitlager, T.Kuipers and J.Visser. A practical model for measuring maintainability. In *Proceedings of Sixth International Conference on the Quality of Information and Communications Technology*. 2007. Pp30-39.