Measuring Cohesion of Packages in Ada95

Baowen Xu
Department of Computer Science & Engineering, Southeast University
Nanjing, China, 210096
086-25-3793977
bwxu@seu.edu.cn

Zhenqiang Chen
Department of Computer Science & Engineering, Southeast University
Nanjing, China, 210096
086-25-3793977
chenzq@seu.edu.cn

Jianjun Zhao
Department of Computer Science & Engineering, Fukuoka Institute of Technology, Japan
081-92-606-4895
zhao@cs.fit.ac.jp

ABSTRACT

Ada95 is an object-oriented programming language. Pack-ages are basic program units in Ada 95 to support OO programming, which allow the specification of groups of logically related entities. Thus, the cohesion of a package is mainly about how tightly the entities are encapsulated in the package. This paper discusses the relationships among these entities based on dependence analysis and presents the properties to obtain these dependencies. Based on these, the paper proposes an approach to measure the package cohesion, which satisfies the properties that a good measure should have.

Categories and Subject Descriptors
D.2.8 [**Software Engineering**]: Metrics – *performance measures.*

General Terms
Measurement.

Keywords
Measurement, Cohesion, Object-Oriented.

1. INTRODUCTION

Cohesion is one of the most important software features during its development. It tells us the tightness among the components of a software module. The higher the cohesion of a module, the more understandable, modifiable and maintainable the module is. A software system should have high cohesion and low coupling. Researchers have developed several guidelines to measure cohesion of a module [1, 3, 4]. Since more and more applications are object-oriented, the approaches to measure cohesion of object-oriented (OO) programs have become an important research field.

Generally, each object-oriented programming language provides facilities to support OO features, such as data abstraction, encapsulation and inheritance. Each object consists of a set of attributes to represent the states of objects and a set of operations on attributes. Thus, in OO environment, the cohesion is mainly about how tightly the attributes and operations are encapsulated.

There are several approaches proposed in literature to measure OO program cohesion [2, 5, 6, 7, 11, 12]. Most approaches are based on the interaction between operations and attributes. The cohesion is measured as the number of the interactions. Generally only the references from operations to attributes are considered. And few care about the interactions of attributes to attributes and operations to operations at the same time. This might lead to bias when measuring the cohesion of a class. For example, when designing the trigonometric function lib class, we might set a global variable to record the temporal result. The variable is referred in all the operations of the class. According to methods based on the interaction between operations and attributes [6, 7], the cohesion is the maximum 1. In fact, there are no relations among the operations if the calls are not taken into account. In this view, its cohesion is 0. The difference is caused by considering only the references from operations to attributes, while not considering the inter-operation relations.

In our previous work, we have done some research in measuring OO program cohesion [10, 13, 14]. Our approach overcomes the limitations of previous class cohesion measures, which consider only one or two of the three facets. Since the OO mechanisms in different programming languages are different from each other, this paper applies our measure to Ada packages.

The remaining sections are organized as follows. Section 2 introduces the package in Ada 95. Section 3 discusses the basic definitions and properties for our measure. Based on the definitions and properties, Section 4 proposes approaches to measure package cohesion. Conclusion remarks are given in the last section.

2. PACKAGES IN ADA 95

In Ada 95[ISO95], packages and tagged types are basic program units to support OO programming. A package allows the specification of groups of logically related entities. Typically, a package contains the declaration of a type along with the declarations of primitive subprograms of the type, which can be called from outside the package, while its inner workings remain hidden from outside users. In this paper, we distinguish packages into four groups.

- PG1: Packages that contain any kind of entities except tagged types.
- PG2: Packages that only contain the declaration of one tagged type along with those primitive subprograms of the type. There are two subgroups in PG2:
  - PG2-1: The type is an original tagged type.
  - PG2-2: The type is a derived type.

- PG3: Combination of PG1 and PG2.
- PG4: Generic packages.

After a generic package is instantiated, it belongs to one of the former three groups. Thus, only cohesion measure of PG1, PG2 and PG3 is discussed in the paper.

3. DEFINITIONS

3.1 Basic Definitions

In this section, we will present our definitions in the form of PG1. The cohesion of a package from PG1 is mainly about how tightly the objects and subprograms are encapsulated in the package. In this paper, the relationships among objects and subprograms are defined as three dependencies: inter-object, inter-subprogram and subprogram-object dependence.

Definition 1 In the package body or a subprogram of the package, if the definition (modification) of object A uses (refer, but not modify) object B directly or indirectly, or whether A can been defined is determined by the state of B, then A depends on B, denoted by $A \rightarrow B$.

Generally, if B is used in the condition part of a control statement (such as *if* and *while*), and the definition of A is in the inner statement of the control statement, the definition of A depends on B's state.

Definition 2 If object A is referred in subprogram P, P depends on A, denoted by $P \Rightarrow A$.

Definition 3 There are two types of dependencies between subprograms: call dependence and potential dependence. If P is called in M, then M call depends on P, denoted by $M \rightarrow P$. If the object A used in M is defined in P, the A used in M depends on the A defined in P, denoted by $M \xrightarrow{A,A} P$, where (A, A) is named as a tag. For each call edge, add a tag (*, *) for unification. i.e. if $P \rightarrow Q$, $P \xrightarrow{*,*} Q$.

To obtain these dependencies, we introduce four sets for each subprogram M:

- IN(M) is an object set, each element of which is an object referred before modifying its value in M;
- OUT(M) is an object set, each element of which is an object modified in M.
- DEP_A (M) is a dependence set which represents the dependencies from the objects referred in M to the objects defined outside M. Each element has the form <A, B>, where A and B are objects of the package.
- DEP_A_OUT(M) is a dependence set which records the dependencies from the objects referred in M to the objects defined outside M when exiting M.

In general, the intermediate results are invisible outside, and an object might be modified many times in a subprogram. We introduce DEP_A_OUT to improve the precision. Obviously, $DEP\_A\_OUT(M) \subseteq DEP\_A(M)$.

Property 1 $A \in IN(M), A \in OUT(P) \Rightarrow M \xrightarrow{A,A} P$.

Property 2 $<A, B> \in DEP\_A(M), B \in OUT(P)$

$$\Rightarrow M \xrightarrow{A,B} P.$$

Property 3 $M \xrightarrow{A,B} P, \forall <B, C>(<B, C> \in DEP\_A\_OUT(P), C \in OUT(Q)) \Rightarrow M \xrightarrow{A,C} Q$.

In our previous work [8, 9], we have proposed methods to analyze dependencies among statements for Ada programs. And these dependencies can be easily transformed to the dependencies proposed in this paper. Due to the space limitation, we do not discuss them in detail here.

To present our cohesion measure in a united model, we introduce package dependence graph to describe all types of dependencies.

Definition 4 The package dependence graph (PGDG) of a package PG is a directed graph, PGDG = <N, E, T>, where N is the node set and E is the edge set, T is the tag set. $N = N_O \cup N_P$, N_O is the object node set, each of which represents a unique object; N_P is the subprogram node set, each of which represents a unique subprogram. PGDG consists of three sub-graphs:

- Inter-Object Dependence Graph (OOG), OOG = $<N_O, E_O>$, where N_O is the object node set (the name of a node is the name of the object it represents); E_O is the edge set, if $A \rightarrow B$, then edge $<A, B> \in E_O$.
- Inter-Subprogram Dependence Graph (PPG), PPG = $<N_P, E_P, T>$, where N_P is the subprogram node set; E_P is the edge set which represents the dependencies between subprograms; $T \in (V \times V)$ is the tag set, where V is the union of objects and {*}.
- Subprogram-Object Dependence Graph (POG), POG = $<N, E_{PO}>$, where N is the node set which represents objects and subprograms; E_{PO} is the edge set representing dependencies between subprograms and objects. If $P \Rightarrow A$, $<P, A> \in E_{PO}$.

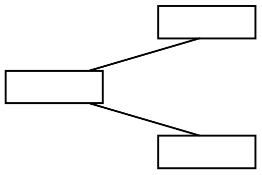
Example1 shows the package Tri, which contains three objects: temp, temp1 and temp2, and four subprograms: sin, cos, tg and ctg. Figure 1 shows the PGDG of the package Tri in Example1 (all the Tags on PPG are (*, *), because there are only call dependencies in this example. We omit the Tags for convenience).

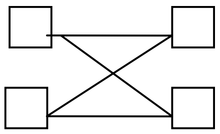
Example1: package Tri.

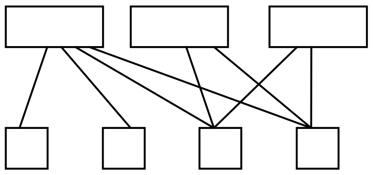
```
package Tri is
    temp, temp1, temp2: real;
    function sin (x: real) return real;
    function cos (x: real) return real;
    function tg (x: real) return real;
    function ctg (x: real) return real;
end Tri;


package body Tri is
    function sin (x: real) return real is
    begin temp:=…; return temp; end sin;
    …
```

```
      function tg (x: real) return real is
      begin
         temp1:=sin(x);temp2:=cos(x);
         temp:=temp1/temp2; return temp;
      end tg;
      …
   end Tri;
```







3.2 Extended Definitions

Since there is no object in the package of PG2, the definitions of Section 3 can not be applied to these packages directly. Therefore, this section will extend the definitions of Section 3.1 to a more general model by the following steps:

- For PG1, if there is an embedded package, the package is taken as an object.
- For PG2, take the components of the type as objects of the package.

Let A, B be object of a type T, M, P primitive subprograms, and Com1 and Com2 are components of T. Then

$\exists$ A, B (A.Com1 $\rightarrow$ B.Com2) $\Rightarrow$ Com1 $\rightarrow$ Com2.

$\exists$ A, P (P $\Rightarrow$ A.Com) $\Rightarrow$ P $\Rightarrow$ Com.

$\exists$ A, B, M, P (M $\xrightarrow{A.Com1, B.Com2}$ P)

$\Rightarrow$ M $\xrightarrow{Com1, Com2}$ P.

- For PG3, take the types as objects of the package.
- To present our measure in a unified model, we add powers for different objects.

$$PW(O) = \begin{cases} Cohesion(O) & O \text{ is a package object} \\ Cohesioin(PG(O)) & O \text{ is a type object} \\ 1 & others \end{cases}$$

where Cohesion (O) is the cohesion of O, PG (O) returns the package containing O.

4. MEASURING PACKAGE COHESION

According to the PGDG, this section will propose our method to measure the package cohesion. In the following discussions, we assume package PG contains *n* objects and *m* subprograms, where *m*, *n* ≥*0*.

4.1 Measuring Inter-Object Cohesion

Inter-object cohesion is about the tightness among objects in a package. To measure this cohesion, for each object A, introduce a set A_DEP to record the objects on which A depends, i.e.

O_DEP(A) = {B| A$\rightarrow$B, A $\neq$ B}.

Let

$$PW\_DEP(A) = \sum_{B \in O\_DEP(A)} PW(B).$$

Then, we define the inter-object cohesion as:

$$Cohesion(O\_O, PG) =$$

$$\begin{cases} 0 & n = 0 \\ PW(A) & n = 1 \\ 1/n \sum_{i=1}^{n} \frac{PW\_DEP(A_i)}{n-1} & n > 1 \end{cases}$$

where $\frac{PW\_DEP(A)}{n-1}$ represents the degree on which A depends on other objects.

If *n*=0, there is no object in the package, we set it to 0. If *n*=1, there is one and only one object in the package, the cohesion is its power.

4.2 Measuring Subprogram-Object Cohesion

Subprogram-object cohesion is the most important field in measuring cohesion. Until now, there have been several approaches proposed in literature, such as Chae's methods [6, 7]. But most approaches are based on the POG. As we have mentioned above, all these methods describe the object reference in a simple way and subprograms are connected by the objects referred. Whether there are related among these subprograms are not described exactly. Thus, these approaches should be improved to describe these relations. For completeness, we use *Co*(*Prev*) to represent a previous cohesion measure, which satisfies Briand's four properties.

For each subprogram P, we introduce another two sets: P_O and P_O_OUT. Where

- P_O(P) records all the objects referred in P.

- P_O_OUT(P) records the objects referred in P, but these objects relate to objects referred by other subprogram, i.e.,

$$\text{P\_O\_OUT(P)}=\{A \mid \exists B, M\ (P \xrightarrow{B,A} M \vee M \xrightarrow{A,B} P) \wedge A,B \neq '*'\}.$$

Let

$$\rho(P) = \frac{\sum_{A_i \in P\_O\_OUT(P)} PW(A_i)}{\sum_{A_i \in P\_O(P)} PW(A_i)}$$

Then, we define the subprogram-object cohesion as:

$$Cohesion(P\_O, PG) = \begin{cases} 0 & m = 0 \vee n = 0 \\ \dfrac{\sum_{A_i \in P\_O(P)} PW(A_i)}{\sum PW(A_i)} & m = 1 \\ \frac{1}{m}\sum_{i=1}^{m} Co(Prev) * \rho(P_i) & Others \end{cases}$$

If P_O(P) = Φ, i.e. no objects are referred in P, we set $\rho(P)$ =0. If the objects referred in P are not related to other subprograms, these objects can work as local variables. It decreases the cohesion to take a local variable for a subprogram as an object for all subprograms. If there is no object or subprogram in the package, no subprogram will depend on others. Thus, $Cohesion(P\_O, PG) = 0$.

4.3 Measuring Inter-Subprogram Cohesion

In the PGDG, although subprograms can be connected by objects, this is not necessary sure that these subprograms are related. To measure the inter-subprogram cohesion, we introduce another set P_DEP(P) = {M| P→M} for each P. The inter-subprogram cohesion *Cohesion*(*P_P*, *PG*) is defined as following:

$$Cohesion(P\_P, PG) = \begin{cases} 0 & m = 0 \\ 1 & m = 1 \\ \frac{1}{m}\sum_{i=1}^{m} \dfrac{|P\_DEP(P_i)|}{m-1} & m > 1 \end{cases}$$

where $\dfrac{|P\_DEP(P)|}{m-1}$ represents the tightness between P and other subprograms in the package.

If each subprogram depends on all other subprograms, *Cohesion*(*P_P*, *PG*) = 1.

If all subprograms have no relations with any other subprogram, *Cohesion*(*P_P*, *PG*) = 0.

4.4 Measuring Package Cohesion

After measuring the three facets independently, we have a discrete view of the cohesion of a package. We have two ways to measure the package cohesion:

1) Each measurement works as a field, the package cohesion is 3-tuple,

   Cohesion(*PG*) = < *Cohesion*(*O_O, PG*), *Cohesion*(*P_O, PG*), *Cohesion*(*P_P, PG*)>.

2) Integrate the three facets as a whole

$$Cohesion(PG) = \begin{cases} 0 & m = 0 \\ k * Cohesion(P\_P, PG) & n = 0, m \neq 0 \\ \sum_{i=1}^{3} k_i * Cohesion_i(PG) & Others \end{cases}$$

where $k \in (0,1]$; k_1, k_2, k_3 >0, and $k_1 + k_2 + k_3$ =1.

$Cohesion_1(PG) = Cohesion(O\_O, PG)$

$Cohesion_2(PG) = Cohesion(P\_P, PG)$

$Cohesion_3(PG) = Cohesion(P\_O, PG)$

If n=0, m≠0, the package cohesion describes only the tightness of the call relations, thus we introduce a parameter *k* to constrain it.

For the example shown in Figure 1, the cohesion of Tri describes as follows:

Cohesion(O_O, Tri)= 1/3
Cohesion(*P_O, Tri*)=0
Cohesion(*P_P, Tri)*=1/3

Let $k_1 = k_2 = k_3 = 1/3$, *Co*(*Prev*)=1, then

Cohesion(*Tri*)= 2/9.

Briand *et al.* [3, 4] have stated that a good cohesion measure should be

(1) Non-negative and standardization.
(2) Minimum and maximum.
(3) Monotony.
(4) Cohesion does not increase when combining two modules.

These properties give a guideline to develop a good cohesion measure.

According to the definitions, it is easy to prove our measure satisfies these properties.

4.5 Cohesion for PG2-2

In the hierarchies of types, the derived type inherits the components and primitive subprograms of the super types. Generally, inheritance will increase the coupling and decrease the cohesion. For the package from PG2-2, we will discuss its cohesion in four cases:

Case 1: Take the package independently.

Case 2: Take all the primitive subprograms and components (contains those from super type) into consideration.

Case 3: If the primitive subprograms of the derived type might access the components (or subprogram) of the super type, take these components (or subprogram) as those of the derived type.

Case 4: Take the super type as an object of the derived type.

The shortcoming of Case 1 is that: It only measures the cohesion of the additional components and primitive subprograms of the derived type, not the complete type.

The primitive subprograms in the super type can not access the components of the derived type except dispatched subprograms. Consequently, in Case 2 or 3, the deeper the hierarchy of types is, the smaller the cohesion. And it is hard to design a package which cohesion is big enough.

Although we present four cases in this section, none is good enough to describe the cohesion for a package from PG2-2. To measure the cohesion of a derived type, much more aspects should be considered.

5. RELATED WORKS

There have several methods proposed in literatures to measure class cohesion. This section gives a brief review of these methods.

(1) Chidamber's *LCOM1* $\in [0, \frac{m*(m-1)}{2}]$, it measures the cohesion by similar methods and non-similar methods. It is a reverse cohesion measure. The bigger the measure, the lower the cohesion.

(2) The PPG in **Hitz's *LCOM2*** is represented by an undirected graph. *LCOM2* is the number of sub-graphs connected. When there is one and only one sub-graph, he introduces *connectivity* to distinguish them.

(3) Briand's *RCI* is the ratio of the number of edges on POG to the max interaction between subprograms and objects.

(4) Henderson's *LCOM3* can be described as follows.

$$LCOM\,3(C) = \frac{\frac{1}{n}\sum_{j=1}^{n} |\mu(A_j)| - m}{1 - m}$$

where μ(A)= {M| A∈P_O(M)}, A is attribute and M is method.

(5) Chae's *CO* [6] introduces *glue methods*, and **Xu-Zhou's *CO*** [13] introduces *cut set* (*glue edges*) to analyze the interact pattern. These two measures are more rational than other measures.

From the introductions above, we can see that

- All these methods consider the attribute reference in a simple way. Whether the methods are related or not are not described exactly.
- *LCOM1*, *LCOM2* and *LCOM3* are non-standard, because their up-bounds are related to the number of methods in the class. *LCOM1* is non-monotonous. The measuring results might be inconsistent with intuition in some cases
- *RCI* has the basic four properties proposed by Briand. But it does not consider the patterns of the interactions among its members, neither *LCOM1* and *LCOM2* nor *LCOM3*.
- Chae's *CO* overcomes most limitations of previous measures. But it is non-monotonous [13]. Xu-Zhou's *CO* improves Chae's cohesion measure, and makes its result more consistent with intuition. The chief disadvantage of both measures is that they can be applied to connected POG; otherwise the result will always be 0.
- *LCOM1* and *LCOM2* measure the cohesion among methods in a class. We can improve the similar function using the dependencies among methods proposed in this paper.
- *LCOM3*, Chae and Xu-Zhou's *CO* measure the cohesion among methods and attributes in a class. In this paper we improve them by introducing $\rho(M)$ for each method M.

6. CONCLUSION

This paper proposes an approach to measure the cohesion of a package based on dependence analysis. In this method, we discussed the tightness of a package from the three facets: inter-object, subprogram-object and inter-subprogram. These three facets can be used to measure the package cohesion independently and can also be integrated as a whole. Our approach overcomes the limitations of previous class cohesion measures, which consider only one or two of the three facets. Thus, our measure is more consistent with the intuition. In the future work, we will verify and improve our measure by experiment analysis

When measuring package cohesion, the following should be paid attentions.

(1) In the hierarchies of types, the primitive subprograms of super type might access the objects of the derived type by dispatching. Therefore, when measuring the cohesion of PG2, it is hard to determine whether the accession of derived typed is considered or not.

(2) We can determine polymorphic calls in an application system. However it is impossible for a package, which can be reused in many systems.

(3) How to deal with some special subprograms, such as access subprograms, since such subprograms can access some special objects in the package.

(4) How to apply the domain knowledge to cohesion measure.

In all, if a package can be applied to many applications, the cohesion is mainly about itself without considering the application environments. Otherwise, it is the cohesion in the special environments.

7. ACKNOWLEDGMENTS

This work was supported in part by the National Natural Science Foundation of China (NSFC) (60073012), National Grand Fundamental Research 973 Program of China (G1999032701), and National Research Foundation for the Doctoral Program of Higher Education of China (20020286004).

8. REFERENCES

[1] Allen, E.B., Khoshgoftaar, T.M. Measuring Coupling and Cohesion: An Information-Theory Approach. in Proceedings of the Sixth International Software Metrics Symposium. Florida USA, IEEE CS Press, 1999, 119-127.

[2] Bansiya, J.L., et al. A Class Cohesion Metric for Object-oriented Designs. Journal of Object-oriented Programming, 1999, 11(8): 47-52.

[3] Briand, L.C., Morasca, S., Basili, V.R. Property-Based Software Engineering Measurement. IEEE Trans. Software Engineering, Jan. 1996, 22(1): 68-85.

[4] Briand, L.C., Daly, J., Wuest, J. A Unified Framework for Cohesion Measurement in Object-Oriented Systems. Empirical Software Engineering, 1998, 3(1): 65-117.

[5] Briand, L.C., Morasca, S., Basili, V.R. Defining and Validating Measures for Object-Based High-Level Design. IEEE Trans. Software Engineering, 1999, 25(5): 722-743.

[6] Chae, H.S., Kwon, Y.R., Bae, D.H. A Cohesion Measure for Object-Oriented Classes. Software —— Practice & Experience, 2000, 30(12): 1405-1431.

[7] Chae, H.S., Kwon, Y.R. A Cohesion Measure for Classes in Object-Oriented Systems. in Proceedings of the Fifth International Software Metrics Symposium. Bethesda, MD USA, 1998, IEEE CS Press, 158-166.

[8] Chen, Z., Xu, B., Yang, H. Slicing Tagged Objects in Ada 95. in Proceedings of AdaEurope'2001, LNCS 2043: 100-112.

[9] Chen, Z., Xu, B., Yang, H., Zhao, J. Static Dependency Analysis for Concurrent Ada 95 Programs. in Proceedings of AdaEurope 2002, LNCS 2361, 219-230.

[10] Chen, Z., Xu, B. Zhou, Y., Zhao, J., Yang, H. A Novel Approach to Measuring Class Cohesion Based on Dependence Analysis. in Proceedings of ICSM 2002, IEEE CS Press, 377-383

[11] Chidamber, S.R., Kemerer, C.F. A Metrics Suite for Object-Oriented Design. IEEE Trans. Software Engineering, 1994, 20(6): 476-493.

[12] Hitz, M., Montazeri, B. Measuring Coupling and Cohesion in Object-Oriented Systems. in Proceedings of International Symposium on Applied Corporate Computing, Monterrey, Mexico, October 1995: 25-27.

[13] Xu, B., Zhou, Y. Comments on A cohesion measure for object-oriented classes. Software —— Practice & Experience, 2001, 31(14): 1381-1388.

[14] Zhou, Y., Guan, Y., Xu, B. On Revising Chae's Cohesion Measure for Classes. J. Software. 2001, 12(Suppl.): 295-300 (in Chinese)