Computer Science
Technical Report

Design-level Cohesion Measures: Derivation, Comparison, and Applications[*]

Byung-Kyoo Kang
kang@cs.colostate.edu

James M. Bieman
bieman@cs.colostate.edu

January 29, 1996
Submitted for Publication

Technical Report CS-96-104

Computer Science Department
Colorado State University
Fort Collins, CO 80523-1873

Phone: (970) 491-5792 Fax: (970) 491-2466
WWW: http://www.cs.colostate.edu

[*]Research partially supported by NASA Langley Research Center grant NAG1-1461.

Design-level Cohesion Measures: Derivation, Comparison, and Applications

Byung-Kyoo Kang **James M. Bieman**

Department of Computer Science
Colorado State University
Fort Collins, Colorado 80523 USA

January 29, 1996

Abstract

Cohesion was first introduced as a software attribute that could be used to predict properties of implementations that would be created from a given design. Unfortunately, cohesion, as originally defined, could not be objectively assessed, while more recently developed objective cohesion measures depend on code-level information. We show that association-based and slice-based approaches can be used to measure cohesion using only design-level information. Our design-level cohesion measures are formally defined, can be readily implemented, and can support software design, maintenance, and restructuring.

Index terms — cohesion, software measurement and metrics, software design, software maintenance, software restructuring and re-engineering, software visualization, software reuse.

1 Introduction

Module cohesion was defined by Yourdan and Constantine as "how tightly bound or related its internal elements are to one another"[10, p. 106]. They describe cohesion as an attribute of designs, rather than code, and an attribute that can be used to predict properties of implementations such as "ease of debugging, ease of maintenance, and ease of modification" [10, p. 140]. Since cohesion refers to the degree to which module components belong together, cohesion measurement should prove to be a very useful restructuring tool [3].

Following the original guidelines [7], the assessment of module cohesion is conducted by skilled engineers. These engineers would apply a set of subjective criteria to analyze associations between "processing elements" and classify the nature of these associations. Because of the subjective nature of the assessment, the measurement of module cohesion has been difficult to automate, and cohesion has not been effectively used as a software quality indicator [9].

Several approaches have been used to develop objective, automatable methods for measuring module cohesion. The first approach, an *association-based* approach, is used by Lakhotia [4] to formalize the notion of the associations between processing elements as a set of rules concerning data dependencies in module code. This method requires the analysis of code-level information and thus cannot be applied before code is written.

The second approach, a *slice-based* approach, is used by Bieman and Ott [2]. They measure functional cohesion in terms of the connections between code data tokens on module output slices. This method also requires code level information.

Research partially supported by NASA Langley Research Center grant NAG1-1461.

Address correspondence to: J. Bieman, Computer Science Dept, Colorado State Univ., bieman@cs.colostate.edu, Fort Collins, CO 80523, USA. (970)491-7096, Fax: (970) 491-2466.

Class cohesion measures for object-oriented software have also been defined using a slice-based approach, and by analyzing the connectivity between methods through common references to instance variables[1, 5, 6]. Method bodies are needed to apply these code-level class cohesion measures.

In this paper, we show that module cohesion can be objectively assessed using only design-level information. We develop and compare design-level cohesion measures using both the association-based and slice-based approaches, and we describe how these measures can be applied as design, maintenance, and restructuring tools.

2 Association-based Cohesion Measures

Stevens, Myers and Constantine define module cohesion (SMC Cohesion) on an ordinal scale. SMC Cohesion includes *coincidental*, *logical*, *temporal*, *procedural*, *communicational*, *sequential*, and *functional* cohesion where coincidental cohesion is the weakest and functional cohesion is strongest cohesion [7]. SMC Cohesion is determined by inspecting the association between all pairs of a module's processing elements.

Lakhotia uses the output variables of a module as the processing elements of SMC Cohesion and defines rules for designating a cohesion level which preserve the intent of the SMC Cohesion [4]. The associative principles of SMC Cohesion are transformed to relate the output variables based on data dependence relationships. A *variable dependence graph* models the control and data dependences between module variables. The rules for designating a cohesion level are defined using a strict interpretation of the association principles of SMC Cohesion. Because the rules are formal, a tool can automatically perform the classification. However, the technique can be applied only after the coding stage since it is defined upon the implementation details.

SMC Cohesion defines an intuitive notion of the cohesion attribute of design components. In a previous paper [3], we used it as an empirical relation system to help us to define a cohesion measure that can be readily automated. This new measure can be applied to both the design and code of a module. It is derived from a design-level view of a module, an input-output dependence graph. In this section, the model and measure are summarized.

2.1 A Design-Level View of Module

The input-output dependence graph (IODG) is based on the data and control dependence relationships between input-output components of a module. Input components of a module include in-parameters and referenced global variables. Output components include out-parameters, modified global variables, and 'function return' values. The term 'component' refers to a static entity. An array, a linked list, a record, or a file is one component rather than a group of components. We define the data and control dependence informally; their formal definitions are given in compiler texts, for example, see reference [11].

Definitions:

- A variable y has a *data dependence* on another variable x ($x \overset{d}{\rightarrow} y$) if x 'reaches' y through a path consisting of a 'definition-use' and 'use-definition' chain.
- A variable y has a *control dependence* on another variable x if the value of x determines whether or not the statement containing y will be performed.
- A variable y is *dependent* on another variable x ($x \rightarrow y$) when there is a path from x to y through a sequence of data or control dependence. We call the path a *dependence path.*
- A variable y has *condition-control dependence* on another variable x ($x \overset{cc}{\rightarrow} y$) if y has a control dependence on x, and x is used in the predicate of a decision (i.e., if-than-else) structure.
- A variable y has *iteration-control dependence* on another variable x ($x \overset{ic}{\rightarrow} y$) if y has a control dependence on x, and x is used in the predicate of an iteration structure.
- A variable y has *c-control dependence* on another variable x ($x \overset{c}{\rightarrow} y$) if the dependence path between x and y contains a decision-control dependence.

- A variable y has *i-control dependence* on another variable x ($x \xrightarrow{i} y$) if the dependence path between x and y contains an iteration-control dependence but no condition-control dependence.

IODG Definition. The *input-output dependence graph* (IODG) of a module *M* is a directed graph, $G_M =$ *(V, E)* where V is a set of input-output components of *M*, and *E* is a set of edges labeled with dependence types such that $E = \{(x, y) \in V \times V \mid$ y has data, c-control, and/or i-control dependence on x$\}$

2.2 Design-Level Cohesion (DLC) Measure

In a manner similar to the approach used to develop SMC Cohesion, we define six relations between a pair of output components based on the IODG representation:

1. **Coincidental relation (R_1):**
   $R_1(o_1, o_2) \;=\; \neg(o_1 \to o_2) \;\wedge\; \neg(o_2 \to o_1) \;\wedge\; \neg\exists x\ [(x \to o_1) \wedge (x \to o_2)]$
   Two outputs o_1 and o_2 of a module have neither dependence relationship with each other, nor dependence on a common input.

2. **Conditional relation (R_2):**
   $R_1(o_1, o_2) \;=\; \exists x\ [((x \xrightarrow{c} o_1) \;\wedge\; (x \xrightarrow{c} o_2)) \;\vee\; ((x \xrightarrow{c} o_1) \;\wedge\; (x \xrightarrow{i} o_2)) \;\vee\; ((x \xrightarrow{i} o_1) \;\wedge\; (x \xrightarrow{c} o_2))]$
   Two outputs are c-control dependent on a common input, or one of two outputs has c-control dependence on the input and the other has i-control dependence on the input.

3. **Iterative relation (R_3):**
   $R_1(o_1, o_2) \;=\; \exists x\ [(x \xrightarrow{i} o_1) \;\wedge\; (x \xrightarrow{i} o_2)]$
   Two outputs are i-control dependent on a common input.

4. **Communicational relation (R_4):**
   $R_1(o_1, o_2) \;=\; \exists x\ [((x \xrightarrow{d} o_1) \;\wedge\; (x \to o_2)) \;\vee\; ((x \xrightarrow{d} o_1) \;\wedge\; (x \to o_2))]$
   Two outputs are dependent on a common input. One of two outputs has data dependence on the input and the other can have a control or a data dependence.

5. **Sequential relation (R_5):**
   $R_1(o_1, o_2) \;=\; (o_1 \to o_2) \;\wedge\; (o_2 \to o_1)$
   One output is dependent on the other output.

6. **Functional relation (R_6):**
   $R_1(o_1, o_2) \;=\; o_1 \equiv o_2$
   There is only one output in a module.

Cohesion strength increases from relation R_1 to R_6. The six relations correspond to six association principles (temporal cohesion is not included) of SMC Cohesion with some degree of overlap.

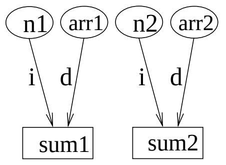
DLC Measure Definition. The cohesion level of a module is determined by the relation levels of output pairs. For each pair of outputs, the strongest relation for that pair is used. The cohesion level of the module is the weakest (lowest level) of all of the pairs. That is, the output pair with the weakest cohesion determines the cohesion of the module.

We have shown that the DLC measure is on an ordinal scale as long as we accept the ordering implied by the association principles of SMC Cohesion [3].

An IODG can be displayed visually in an IODG diagram. In such a diagram, the caller-callee relationship is represented by including the IODG of the callee in the IODG diagram of the caller. In an IODG digram, an input is represented by a circle, and an output by a square. The texts in each circle and square are the names of input and output variables. Each arrow indicates the dependence between two components. Figure 1 shows six cohesion levels for six simple modules.

(a)

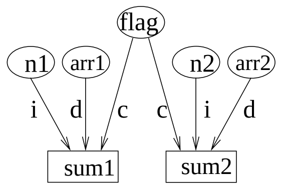
```
procedure Sum1_and_Sum2
   ( n1, n2 : integer;
     arr1, arr2 : int_array;
     var sum1,
         sum2 : integer );
var i : integer;
begin
    sum1 := 0;
    sum2 := 0;
      for i := 1 to n1 do
         sum1 := sum1 + arr1[i];
      for i := 1 to n2 do
         sum2 := sum2 + arr2[i];
end;
```



Coincidental cohesion

(b)

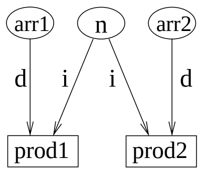
```
procedure Sum1_or_Sum2
   ( n1, n2, flag : integer;
     arr1, arr2 : int_array;
     var sum1,
         sum2 : integer );
var i : integer;
begin
    sum1 := 0;
    sum2 := 0;
    if flag = 1
      for i := 1 to n1 do
         sum1 := sum1 + arr1[i];
     else
      for i := 1 to n2 do
         sum2 := sum2 + arr2[i];
end;
```



Conditional cohesion

(c)

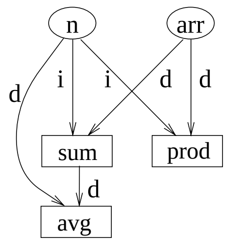
```
procedure Prod1_and_Prod2
   ( n : integer;
     arr1, arr2 : int_array;
     var prod1,
         prod2 : integer );
var i : integer;
begin
    prod1 := 1;
    prod2 := 1;
    for i := 1 to n do begin
       prod1 := prod1 * arr1[i];
       prod2 := prod2 * arr2[i];
    end;
end;
```



Iterative cohesion

(d)

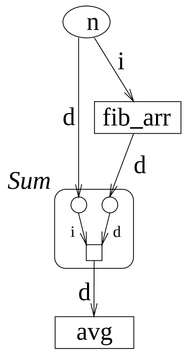
```
procedure Sum_and_Prod
   ( n : integer;
     arr : int_array;
     var sum,
         prod : integer;
     var avg : float );
var i : integer;
begin
    sum := 0;
    prod := 1;
    for i := 1 to n do begin
       sum := sum + arr[i];
       prod := prod * arr[i];
    end;
    avg := sum / n;
end;
```



Communicational cohesion

(e)

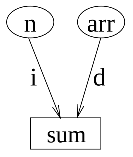
```
procedure Fibo_Avg
   ( n : integer;
     var fib_arr : int_array;
     var avg : float );
var asum : int_array;
    i : integer;
begin
   fib_arr[1] := 1;
   fib_arr[2] := 2;
   for i := 3 to n
      fib_arr[i] := fib_arr[i-1]
              + fib_arr[i-2];
   Sum(n, fib_arr, sum);
   avg := sum / n;
end;
```



Sequential cohesion

(f)

```
procedure Sum
   ( n : integer;
     arr : int_array;
     var sum : integer );
var i : integer;
begin
    sum := 0;
    for i := 1 to n do
       sum := sum + arr[i];
end;
```



Functional cohesion

Figure 1: IODG's and DLC levels for six simple procedures.

sum	max	avg	statement
			procedure Sum_Max_Avg
1	1	1	(n : integer;
1	1	1	var arr,
1		1	sum,
	1		max : integer;
1		1	var avg : float);
1	1	1	i : integer;
			begin
2		2	sum := 0;
	3		max = arr[1];
3	3	3	for i := 1 to n do begin
4		4	sum := sum + arr[i];
	3		if arr[i] > max
	3		max = arr[i];
			end;
3		3	avg := sum / n;
			end;

SMC Cohesion :
Communicational cohesion

FC measures :
WFC = 17 / 27 = 0.63
A = (11*2 + 6*3) / (27*3) = 0.49
SFC = 6 / 27 = 0.22

Figure 2: Data slice profile for *Sum_Max_Avg*.

3 Slice-based Cohesion Measures

A program *slice* is the portion of the program that might affect the value of a particular identifier at a specified point in the program [8]. In developing cohesion measures, slices can be used to represent the functional components of a module.

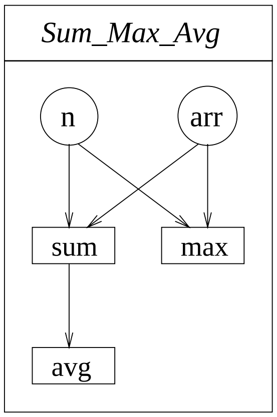
3.1 Functional Cohesion (FC) Measures

Bieman and Ott developed cohesion measures that indicate the extent to which a module approaches the ideal of functional cohesion [2]. They introduced three measures of functional cohesion as the relative number of “glue” or “adhesive” data tokens based on “data slices” of a module (procedure). The *data slice* of a variable is the sequence of data tokens which have a dependence relationship with the variable. A data slice is computed for each output of a procedure. *Glue tokens* are data tokens common to more than one data slice. The glue tokens common to every data slice of a module are *superglue tokens*. The adhesiveness of a data token is the number of data slices to which the data token is common.

The three measures of functional cohesion are *Weak Functional Cohesion* (*WFC*), *Strong Functional Cohesion* (*SFC*), and *Adhesiveness* (*A*). WFC is the ratio of glue tokens to the total number of tokens in a procedure. SFC is the ratio of superglue tokens to the total number of data tokens in a procedure. Adhesiveness the ratio of the amount of adhesiveness to the total possible adhesiveness, which is the adhesiveness when all data tokens are superglue tokens.

Figure 2 shows example functional cohesion computations. Each column in the figure corresponds to a data slice for each output. For example, the numbers in the first column are the number of data tokens in the corresponding line that affect the output or are affected by the output. The data tokens that are counted on more than two columns are glue data tokens and those that are counted on all columns are superglue data tokens. In this example, we find 17 glue data tokens and 6 superglue tokens.

The functional cohesion measure is formally defined. Thus, measurement tools can be (and have been) readily implemented. However, the measures depend on the implementation details and can be applied only after the body of a module has been coded.



(a)

com- ponent \ output	sum	max	avg
n	1	1	1
arr	1	1	1
sum	1	0	1
max	0	1	0
avg	1	0	1

(b)

Figure 3: An example (a) the IODG and (b) IODT of a procedure *Sum_Max_Avg*.

3.2 Design-level Functional Cohesion (DFC) Measures

We derive DFC measures following the approach used to develop the functional cohesion measures. Rather than analyzing code details, we use a design level view modeled by the IODG to define the measure. The DFC measures use a 'simplified' IODG which includes only dependence relationships between input-output components, without classifying the dependences. Figure 3(a) shows a graphical (IODG) and Figure 3(b) shows a tabular (IODT) representation of procedure *Sum_Max_Avg* of Figure 2.

In Figure 3 (b), the names of the output are listed in the first row and the names of the components (inputs and outputs) are in the first column of the figure. The "1" in the figure indicates that the corresponding component has a dependence relation with the named output, and the "0" indicates no dependence relation.

The IODG and IODT show the relationship between input-output components of a module. The DFC measures are expressed in terms of *isolated*, *essential*, and *cohesiveness*:

Definition: A component is *isolated* if it affects only one local functionality, i.e., it has a dependence relationship with only one output.

For example, in Figure 3, component 'max' is isolated since it has dependence a relationship with only one output, itself. The other components are not isolated.

Definition: A component is *essential* if it affects (or is affected by) all functionalities of the module, i.e., it has dependence relationships with all outputs of the module.

If a module contains only one output, the output is the only functionality of the module. Thus, every component in the module is not isolated and is essential. In Figure 3, components 'n' and 'arr' are essential since they affect all outputs.

We define the *cohesiveness* of a component as the degree of "relatedness" of the component to the outputs. Cohesiveness provides more information than simply classifying a component as isolated or essential. The cohesiveness of a component represents the relative number of outputs that the component relates together. We do not address the cases where input does not contribute to the computation of output, i.e., in our model, every component has dependence relation with at least one output. Therefore, the cohesiveness of a component is the relative number of the other output(s) with which the component has a dependence relation. If a module contains only one output, the cohesiveness of every component in the module is 1.

Definition: For an arbitrary module, the cohesiveness of i'th component is:

$$C_i = \begin{cases} \frac{N_i - 1}{O - 1} & \text{if } O > 1 \\ 1 & \text{otherwise} \end{cases}$$

where N_i is the number of outputs with which the ith component has a dependence relation, and O is the total number of output in the IODG model of the module.

The cohesiveness of an isolated component is 0 and the cohesiveness of an essential one is 1. In Figure 3(b), the cohesiveness of *n* and *arr* is 1, the cohesiveness of *sum* and *avg* is 1/2, and the cohesiveness of *max* is 0.

Three measures, *Loose Cohesiveness* (LC), *Tight Cohesiveness* (TC) and *Module Cohesiveness* (MC), are defined as the relative number of non-isolated components, the relative number of essential components, and the average cohesiveness of the components of the model, respectively:

DFC Measure Definition.

$$\begin{aligned} \mathsf{LC}(m) &= D/T \\ \mathsf{TC}(m) &= E/T \\ \mathsf{MC}(m) &= \frac{\sum_{i=1}^{T} C_i}{T} \end{aligned}$$

where D, E, and C_i are the number of non-isolated components, the number of essential components, and the cohesiveness of i'th component, respectively, in the IODG of module m. T is the total number of components in m.

Using the definition of component cohesiveness, module cohesiveness can be expressed as

$$\mathsf{MC}(m) = \frac{\sum_{i=1}^{T}(N_i - 1)}{T * (O - 1)} = \frac{\sum_{i=1}^{T} N_i - T}{T * O - T}$$

The three measures for the procedure *Sum_Max_Avg* in Figure 2 and 3 are

$$\begin{aligned} &\mathsf{LC}(Sum\_Max\_Avg) = 4/5 = 0.8 \\ &\mathsf{TC}(Sum\_Max\_Avg) = 2/5 = 0.4 \\ &\mathsf{MC}(Sum\_Max\_Avg) = (2*2+2*1)/(5*2) = 0.6 \end{aligned}$$

An isolated component has zero cohesiveness, a non-isolated component has cohesiveness of greater than 0, and essential component has cohesiveness of one. Thus, for a given module m:

$$E \leq \sum_{i=1}^{T} C_i \leq D$$

where D, E, C_i, and T are defined as above. Therefore,

$$\mathsf{TC}(m) \leq \mathsf{MC}(m) \leq \mathsf{LC}(m)$$

3.3 Relationship between the DFC and FC measures

The DFC measures correspond closely to the FC measures. Each of the DFC measures (LC, TC, and MC) was defined to correspond to one of the FC measures (weak functional cohesion, strong functional cohesion, and adhesiveness) respectively. However, DFC measures are defined in terms of relations between the components of a module interface, while FC measures are based on the relationship between the components in a module body.

Figure 4 contains unlabeled IODG diagrams for different module configurations. Input, output, and selected internal data tokens are represented by circles, squares, and square bars, respectively. Figure 4(d) shows three modules with the same number of inputs and outputs, and the same dependence relations. Thus, their DFC measures are equal. However, the second module contains more essential data tokens. As the result, the FC measures of the second module are higher than those of the first module. The third module contains more isolated data tokens. As the result, the FC measures of the third module are lower than those of the first module. Figure 4 (e) and (f) also show that an increase in the number of essential or isolated data tokens affects the FC measures.

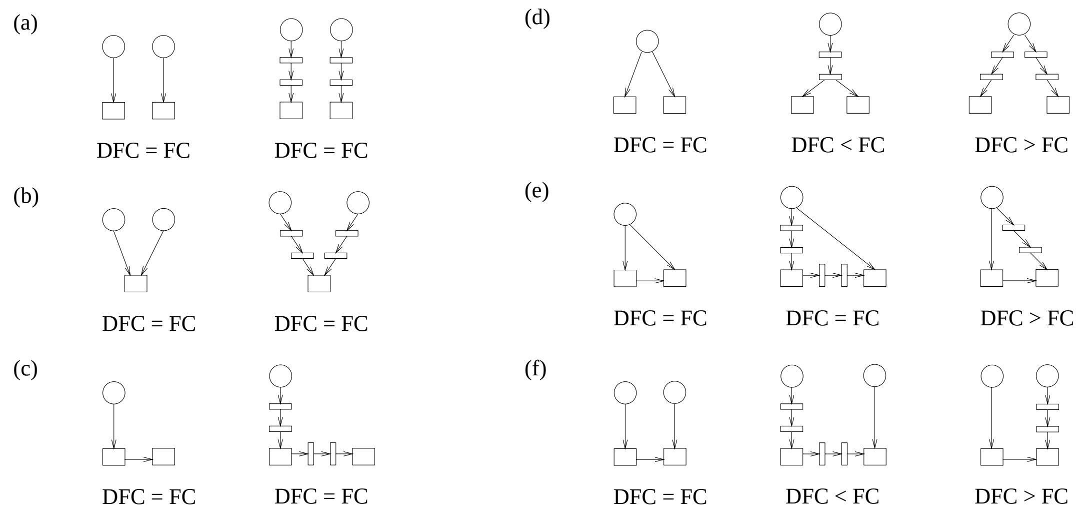


Figure 4: Comparing the DFC and FC measures.

Figure 4 (a), (b), and (c) show that a change in the number of essential or isolated data tokens in a module may not affect FC measures. All input-output components in a module are isolated for case (a), and essential for cases (b) and (c). If the FC values are 1 for a given module, the DFC values are 1, if the FC values are 0 for a given module, the DFC values are 0. If the DFC values are between 0 and 1 for a given module, the corresponding FC values depend on the relative number of isolated, non-isolated, and essential data tokens. Therefore, when FC $>$ DFC, we know that there is a greater relative number of essential data tokens than essential input-output components. When DFC $>$ FC, there is a greater relative number of isolated data tokens than isolated input-output components.

We see that the DFC and FC measures are equivalent only for some modules. There is, however, a general correspondence between the DFC and FC measures. An empirical study may confirm or refute the correspondence. Such a study can determine the distribution of isolated and essential data tokens in real software.

FC measures provide more detailed information for restructuring existing modules than DFC measures. The FC measures captures the cohesion due to internal details. For example, the second module in Figure 4(d) is more difficult to decompose into two modules than the third module in 4(d). To decompose the second module, most of data tokens need to be rewritten. However, the FC measures alone can not capture input-output relationships. For example, high values of FC measures may be due to essential input-output components or other essential data tokens. Both measures, when used together, can provide more complete information.

4 Relationship between the DLC and DFC Measures

The DLC measure is an association-based measure and the three DFC measures are slice-based measures. Both sets of measures have been defined using an intuitive understanding of cohesion based on the "relatedness" of module components. An analysis of the relationship between the DLC and DFC measures provides further evidence of how the measures correspond to the intuition of cohesion.

We investigate the effect on the measures of increases in the number of the connections between module components and increases in the number of module components. To compare the DFC measures with the DLC measure, we use the simplified IODG. The simplified IODG (without dependence labels) cannot account for the difference between 'conditional', 'iterative', and 'communicational' DLC levels. Thus, these relation levels are represented as an 'indirect' relation, and their corresponding cohesion levels are 'indirect' cohesion.

4.1 The effect of increasing the number of dependence connections.

Figure 5 shows the IODG, IODT, and DFC measures, the association level of each pair of outputs, and the DLC measures for seven module configurations. To show the effect of increasing the number of connections on the measures, we fix the number of inputs and outputs for each module. Each module in the figure has three inputs and three outputs.

The number of direct or indirect dependence connections increases from module (a) to module (g). We look at the effect of increasing the number of connections for each measure.

MC measure. The DFC MC measure always detects an increase in the number of dependence connections, and is clearly more sensitive than the LC and TC measures. Figure 5 shows that the MC values precisely correspond to changes in the number of dependence connections in each module, which is consistent with our intuition about cohesion. That is, modules with more related components are more cohesive than modules with fewer related components.

LC measure. The LC measure captures the relative number of isolated (or non-isolated) components in a module. A relatively low LC value means that there are more isolated components than non-isolated ones.

The modules in Figure 5(c) and Figure 5(d) have the same number of dependence connections and equal MC values. However module 5(d) has more isolated components than module 5(c). Module 5(c) has two input components connecting output components while module 5(d) has only one such connection. This difference between modules 5(c) and 5(d) is reflected by the LC measure.

TC measure. The TC measure detects the relative number of the components with the strongest connection. These are the essential components of the module. TC is zero when there are no components that are used to compute every output. TC equals one when all components in the module are tightly related and essential to the functionality of the module. Modules 5(a), 5(b), and 5(c) contain no essential components. All components of module 5(g) are essential and tightly related. Thus, TC is 0 for modules 5(a), 5(b), and 5(c), and 1 for module 5(g).

DLC measure. Figure 5 shows that DLC is not very sensitive to the different number of connections in the modules. In contrast to MC and LC, DLC does not distinguish between modules 5(a), 5(b), and 5(c). DLC finds the weakest connection among module components. Finding the weakest connection is important, because "for debugging, maintenance, and modification purposes, a module behaves as if it were only as strong as its weakest link" [10, p. 132]. Also, the DLC measure computed using a labeled IODG (where dependence is classified) provides more precise information about the relationship between output components than the DFC measures. For example, when an input is used by two outputs, while the DFC measures treat the input as simply an essential component for the outputs, the DLC measure classifies the relationship between the two outputs into conditional, iterative, or communicational relation using the classified dependence information.

Among MC, LC, and TC, TC is closest to DLC. In calculating DLC, the lowest cohesion level of all pairs is the cohesion of the module. The module of Figure 5(c) contains three pairs of outputs. The lowest relation level is 'coincidental', so the corresponding cohesion level of the module is coincidental. TC is 0 for the module since there are no essential components — components that connect all outputs. Whenever the DLC level for a module is 'coincidental', the TC value is 0. If there is even one pair of outputs whose relation level is 'coincidental', there can be no component that connects all outputs. The reverse is, however, not true. When a module TC is 0, the cohesion level is not always coincidental, because there may be some components that connect some portion of the outputs, and those components together connect all outputs. When all outputs are connected, the DLC cohesion level is not coincidental.

Both DLC and TC are calculated using the most extreme cases. Thus, they generally correspond to each other. This correspondence between these measures is in all modules of Figure 5. In modules (a), (b), and (c) of the figure, the DLC levels are 'coincidental' and the TC values are 0. In 5(d) and 5(e), the DLC levels are 'indirect' and the TC values are 1.6.

4.2 The effect of increasing the number of input-output components.

Figure 6 shows how the DFC measures change as the number of input or output components are increased. Each module in the figure has equal DFC measures (MC, LC, and TC) which are represented as a single

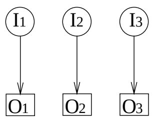
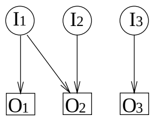
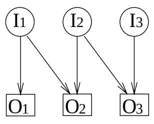
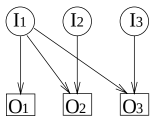
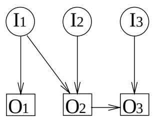
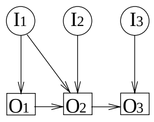
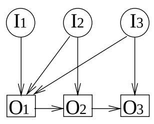
	IODG	IODT	DFC measure	Association Level	DLC measure
(a)		$\begin{array}{cccc} & O_1 & O_2 & O_3 \\ I_1 & 1 & 0 & 0 \\ I_2 & 0 & 1 & 0 \\ I_3 & 0 & 0 & 1 \\ O_1 & 1 & 0 & 0 \\ O_2 & 0 & 1 & 0 \\ O_3 & 0 & 0 & 1 \end{array}$	LC = 0 MC = 0 TC = 0	O_1, O_2 : Coincidental O_2, O_3 : Coincidental O_1, O_3 : Coincidental	Coincidental
(b)		$\begin{array}{ccc} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}$	LC = 1/6 MC = 1/12 TC = 0	O_1, O_2 : Indirect O_2, O_3 : Coincidental O_1, O_3 : Coincidental	Coincidental
(c)		$\begin{array}{ccc} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}$	LC = 2/6 MC = 2/12 TC = 0	O_1, O_2 : Indirect O_2, O_3 : Indirect O_1, O_3 : Coincidental	Coincidental
(d)		$\begin{array}{ccc} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}$	LC = 1/6 MC = 2/12 TC = 1/6	O_1, O_2 : Indirect O_2, O_3 : Indirect O_1, O_3 : Indirect	Indirect
(e)		$\begin{array}{ccc} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{array}$	LC = 4/6 MC = 5/12 TC = 1/6	O_1, O_2 : Indirect O_2, O_3 : Sequential O_1, O_3 : Indirect	Indirect
(f)		$\begin{array}{ccc} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{array}$	LC = 5/6 MC = 9/12 TC = 4/6	O_1, O_2 : Indirect Sequentail O_2, O_3 : Sequential O_1, O_3 : Indirect Sequentail	Sequential
(g)		$\begin{array}{ccc} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{array}$	LC = 1 MC = 1 TC = 1	O_1, O_2 : Indirect Sequentail O_2, O_3 : Sequential O_1, O_3 : Indirect Sequentail	Sequential

Figure 5: The effect on the DFC and DLC measures of increasing the number of dependence connections.

DFC value.

If there is only one output in a module, DFC = 1 no matter how many inputs there are. The DLC measure indicates "functional" cohesion. If there are multiple outputs and every component is isolated, the DFC measures are 0 without regard to the number of inputs and outputs in the module, which corresponds to coincidental cohesion as indicated by DLC. These correspondences are shown in Figure 6 (a) and (b).

The DFC measures are sensitive to the relative number of isolated or essential components in a module. As the relative number of isolated components in a module is increased, (more components are not related with each other) the DFC value decreases. Figure 6 (c), (d), and (g) show the decrease of DFC when the relative number of isolated components is increased. Figure 6(f) shows that when the relative number of essential components in a module is increased, the DFC value increases. In cases 6(e) and 6(h), the relative number of essential components are not changed, and the DFC values also show no change.

As we see in Figure 6, the DLC measure does not capture the differences in the relative number of cohesive components. When the number of isolated or essential components is changed, the corresponding DLC levels are not changed.

To summarize, the DFC measures MC, LC, and TC are sensitive to the relative number of dependence connections, the relative number of isolated components, and the relative number of essential components, respectively. The DLC measure is, however, not very sensitive to the relative number of connections, isolated, and essential components in a module. However, DLC measure always finds the weakest connection among module components and determines the cohesion level. DLC also provides more precise information for the relationship between output components, than the DFC measures. Among the three DFC measures, the TC measure has been found to correspond most closely to the DLC measure.

There is a fundamental difference between the DFC measure and the DLC measure. When calculating a cohesion value, the DFC measures average the cohesion values of all components, while the DLC measure finds the most weakly connected relation. This difference is intentional. The generated data from both measures should be interpreted differently.

5 Applications of Design-level Cohesion Measures

The IODG model and associated measures can be used to improve software quality during design and maintenance.

- IODG diagrams give a visual representation of module interfaces. Such visualizations can help software engineer understand the functional structure of programs during design and maintenance. For existing software, the IODG information can be generated automatically using a compiler-like tool. Without an implementation, IODG information should be part of a detailed design. IODG diagrams can be also drawn automatically based on the IODG information.

- The DLC/DFC measures can be used to locate modules that perform multiple functions having no or weak relations with each other. These modules may be poorly-designed and should be redesigned or restructured. The measures can be computed easily from the IODG information.

- The IODG diagram and associated cohesion measures can be used to redesign software during the design process, and to restructure existing software [3]. The measures are criteria for determining whether or not a given module should be redesigned or restructured. An IODG diagram can help one to decide how a selected module will be restructured.

Figure 7 shows how restructuring can be accomplished using IODG diagrams, DLC/DFC measures, and the following process:

1. IODG's of programs of interest are generated.

2. Modules with low DLC/DFC values are located. The optimal DLC/DFC value will depend on the application, the required reusability, readability, and maintainability of the software. Managers need to specify expected marginal DLC/DFC values.

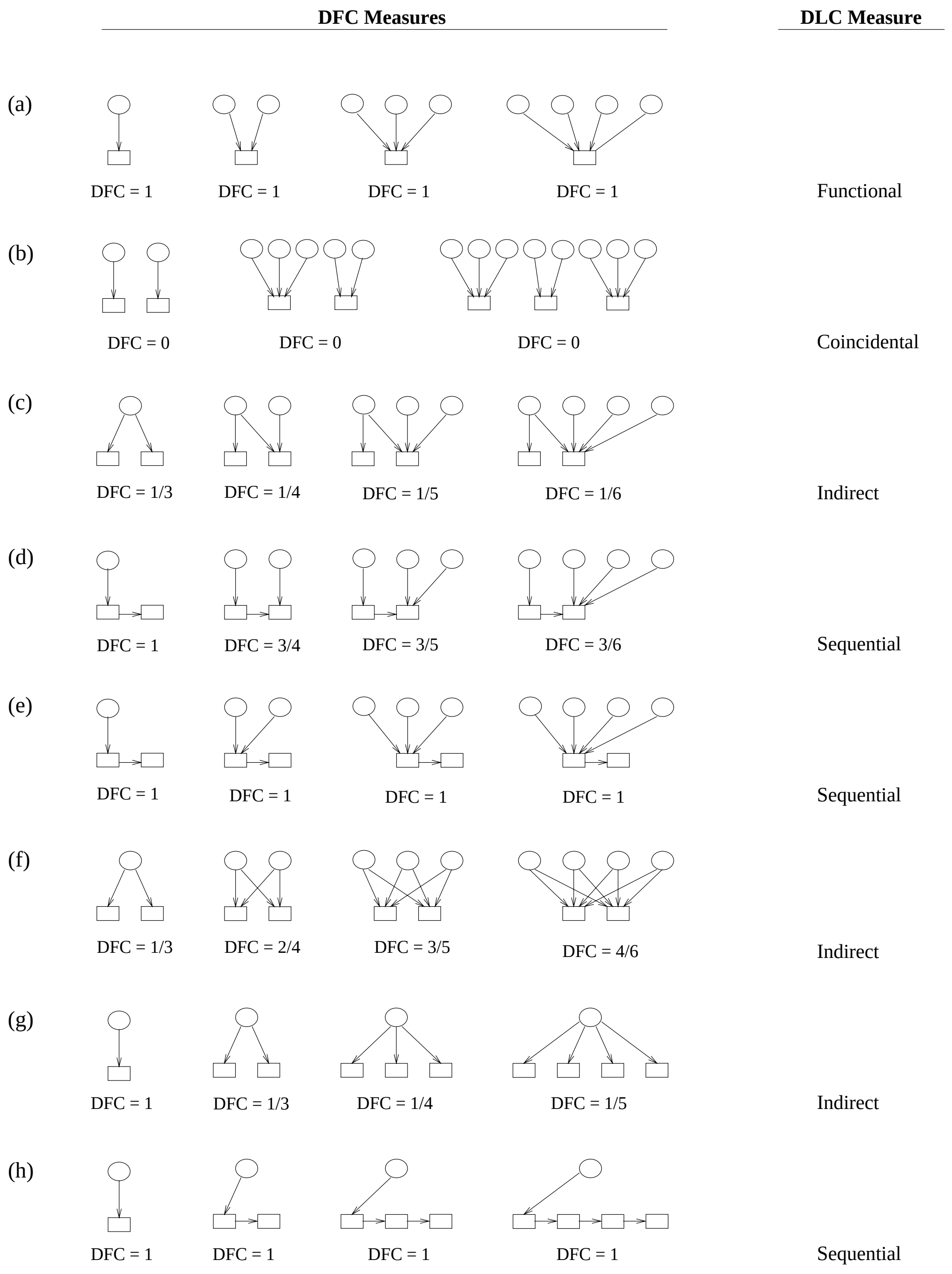


Figure 6: The effect on the DFC and DLC measures of increasing the number of input-output components.

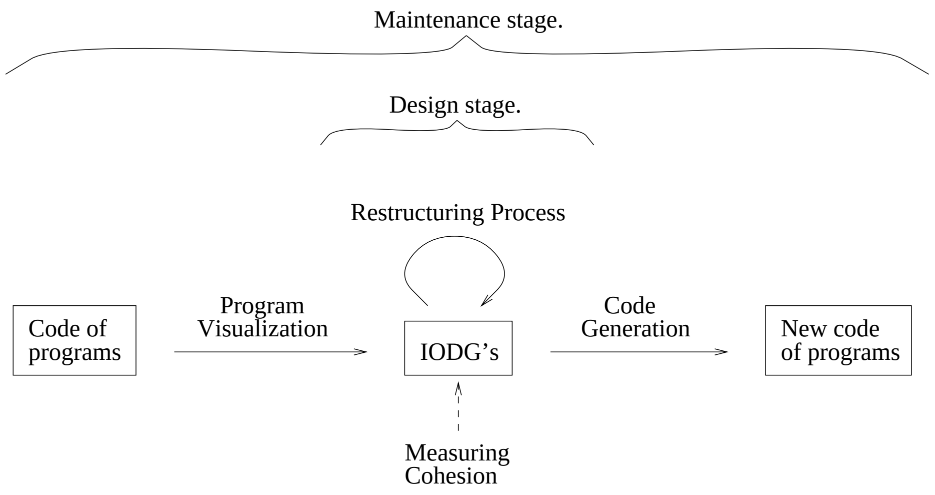


Figure 7: Software Restructuring Process

3. Each module that has been identified as poorly-designed is decomposed: (1) Partition the output components of the IODG so that when decomposed according to the partition, each resulting IODG has higher DLC level. (2) Decompose each IODG according to its partition. Each resulting IODG includes input-output components that have dependence relations with the partitioned outputs. This step is repeated until the cohesion values for each resulting module are acceptable.

4. Unnecessarily decomposed modules are located and composed. To locate overmodularized modules, a practitioner can use other quality measures such as coupling, size, and/or reuse measures. The IODG can help identify candidates for composition through visualization of the module structure.

5. Finally, new program code is generated. The information of data tokens and their dependence from the initial programs obtained during step 1 is used for this process.

The DLC/DFC measures use the IODG information to determine the module cohesion. Thus, one needs to manipulate only the IODG's and need not deal with implementation details. Compiler-like tool can automate the generation of IODG diagrams, DLC/DFC measures, and restructuring process. We are developing such tools for C programs.

6 Conclusions

We have formalized the concept of design cohesion using a graph model of a procedure interface, the IODG. The IODG models dependencies between externally visible module components and can be generated from design-level information.

The IODG forms the basis for a set of cohesion measures that can be applied prior to implementation. The behavior of these cohesion measures matches the original intuitive, informal definition of software cohesion [7]. Our design-level cohesion measures also generally correspond to several existing code-level cohesion measures.

We derived these cohesion measures using an association-based approach similar to that used by Stevens et al [7] and the slice-based approach used to derive code-level functional cohesion measures [2]. Each of our measures quantify different attributes of the notion of cohesion. Three slice based measures are sensitive to the number of connections, the number of isolated components, or the number of essential components (components connected with all procedure outputs). One association-based cohesion measure is sensitive to the weakest connection between module components.

The IODG model can be used to visualize the functional structure of programs and provides support for program understanding. The design-level cohesion measures can be used to locate poorly designed modules. Our model and measures can help to restructure software during design and maintenance. We are now developing tools to partially automate a restructuring process based on the IODG model and associated measures.

References

[1] J. Bieman and B-K Kang. Cohesion and reuse in an object-oriented system. *Proc. ACM Symp. Software Reusability. (SSR'94)*, pages 259–262, April 1995. Reprinted in *ACM Software Engineering Notes*, Aug. 1995.

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

[3] B-K Kang and J. Bieman. Using design cohesion to visualize, quantify, and restructure software. Technical Report CS-96-103, Colorado State University, January 1996.

[4] A. Lakhotia. Rule-based approach to computing module cohesion. *Proceedings. 15th International Conference on Software Engineering*, pages 35–44, 1993.

[5] Bindu Mehra. Measuring data cohesion in the object-oriented paradigm. Master's thesis, Department of Computer Science, Michigan Technological University, in preparation.

[6] L. Ott, J. Bieman, B-K. Kang, and B. Mehra. Developing measures of class cohesion for object-oriented software. *Proc. Annual Oregon Workshop on Software Metrics (AOWSM'95)*, June 1995.

[7] W. Stevens, G. Myers, and L. Constantine. Structured design. *IBM Systems J.*, 13(2):115–139, 1974.

[8] M. Weiser. Program slicing. *IEEE Trans. Software Engineering*, SE-10(4):352–357, 1984.

[9] M. Woodward. Difficulties using cohesion and coupling as quality indicators. *Software Quality J.*, 2(2):109–127, June 1993.

[10] E. Yourdon and L. Constantine. *Structured Design.* Prentice-Hall, Englewood Cliffs, NJ, 1979.

[11] H. Zima and B. Chapman. *Supercompilers for Parallel and Vector Computers.* Addison-Wesley, 1991.