Measuring Design-Level Cohesion

James M. Bieman, *Senior Member*, *IEEE*, and Byung-Kyoo Kang

Abstract—Cohesion was first introduced as a software attribute that, when measured, 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. An analytical and empirical analysis shows that the design-level measures correspond closely with code-level cohesion measures. They can be used as predictors of or surrogates for the code-level measures. The design-level cohesion measures are formally defined, have been 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 Yourdon and Constantine as "how tightly bound or related its internal elements are to one another" [19, 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" [19, 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 [7].

Following the original guidelines [15], 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 [18].

Existing techniques can measure or assess the cohesion of procedural code [5], [9], [2], structured design documents [15], [16], packages [13], [12], [3], and classes [4], [14], [1]. Our objective is to develop techniques to objectively measure cohesion in terms of information available from the detailed design of modules in procedural programs. More precisely, for each procedure we assume that a detailed design includes the specification of a procedure or function interface and the dependencies between interface components. In this paper, we assume that a "module" consists of one procedure or function rather than a collection or file of procedures.

We follow two approaches that have been used to develop objective, automatable methods for measuring module cohesion. The first approach, an *association-based* approach, is used by Lakhotia [9] to formalize the notion of the associations between processing elements as a set of rules concerning data dependencies in module code. Lakhotia's methods require the analysis of code-level information; they can be adapted for use on design-level constructs.

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. The computation of functional cohesion also requires code level information.

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], [10], [11]. Method bodies are needed to apply these code-level class cohesion measures.

Cohesion is only one of many attributes that designers strive to improve. For example, designers also aim to reduce coupling. We need to be able to objectively measure such attributes in order to evaluate alternative design options. Some design goals may be in conflict—high cohesion with low coupling appear to be competing goals. We need to be able to measure these attributes so that we can study the relationships between such apparently conflicting objectives. Only then we can determine, for example, if high cohesion and low coupling are mutually exclusive.

In this paper, we show that module cohesion can be objectively assessed using only design-level information. We develop and compare a set of association-based and slice-base design-level cohesion measures, 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 [15]. SMC Cohesion is determined by inspecting the association between all pairs of a module's processing elements.

- *J.M. Bieman is with the Department of Computer Science, Colorado State University, Fort Collins, CO 80523. E-mail: bieman@cs.colostate.edu.*
- *B.-K. Kang is with the Electronics and Telecommunications Research Institute, 161 Kajong-Dong, Yusong-Gu, Taejon, 305-350 Korea. E-mail: bkkang@nice.etri.re.kr.*

Manuscript received 27 Feb. 1997; revised 9 Oct. 1997.
Recommended for acceptance by B. Littlewood.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 104031.

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 SMC Cohesion [9]. 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 dependencies 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, as originally defined, 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 [7], we used SMC Cohesion as an empirical relation system to help us to define a cohesion measure that satisfies the representation theorem of measurement [6] and 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 a Module

The input/output dependence graph (IODG), adapted from the variable dependence graph of Lakhotia [9], 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 data and control dependence informally using the notation of Lakhotia [9]; formal definitions are given in compiler texts, for example, see reference [20].

DEFINITIONS.

- A variable y has a *data dependence* on another variable x ($x \xrightarrow{d} y$) if x 'reaches' y through a path consisting of a 'definition-use' and 'use-definition' chain (from Lakhotia [9]).
- 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 (from Lakhotia [9]).
- 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 ($x \xrightarrow{cc} y$) if y has a control dependence on x, and x is used in the predicate of a decision (i.e., if-then-else) structure.
- A variable y has *iteration-control dependence* on another variable x ($x \xrightarrow{ic} y$) 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 ($x \xrightarrow{c} y$) if the dependence path between x and y contains a condition-control dependence but no iteration-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.

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$\}$.

The IODG shows the relationship between input and output components of a module. Each input contributes to one or more outputs; it is used to compute output(s), as input data, decision invariant, and/or loop invariant. The IODG is used to define a design-level cohesion measure.

2.2 Design-Level Cohesion (DLC) Measure

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

1) **Coincidental relation** (R_1):

   $R_1(o_1, o_2) = o_1 \neq o_2 \wedge \neg(o_1 \rightarrow o_2) \wedge \neg(o_2 \rightarrow o_1) \wedge \neg\exists x\,[(x \rightarrow o_1) \wedge (x \rightarrow o_2)]$

   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):

   $R_2(o_1, o_2) = o_1 \neq o_2 \wedge \exists x\,[(x \xrightarrow{c} o_1)] \wedge (x \xrightarrow{c} o_2)]$

   Two outputs are c-control dependent on a common input.

3) **Iterative relation** (R_3):

   $R_3(o_1, o_2) = o_1 \neq o_2 \wedge \exists x\,[(x \xrightarrow{i} o_1) \wedge (x \xrightarrow{i} o_2)]$

   Two outputs are i-control dependent on a common input.

4) **Communicational relation** (R_4):

   $R_4(o_1, o_2) = o_1 \neq o_2 \wedge \exists x\,[((x \xrightarrow{d} o_1) \wedge (x \xrightarrow{d} o_2) \vee ((x \xrightarrow{p} o_1) \wedge (x \xrightarrow{q} o_2))]$, where $p, q \in \{d, c, i\}$, and $p \neq q$.

   Two outputs are dependent on a common input. An input is used to compute both outputs, but as neither a condition flag to select one of two outputs nor a loop invariant to compute both outputs.

5) **Sequential relation** (R_5):

   $R_5(o_1,o_2) = o_1 \neq o_2 \wedge ((o_1 \rightarrow o_2) \vee (o_2 \rightarrow o_1))$

   One output is dependent on the other output.

6) **Functional relation** (R_6):

   $R_6(o_1,o_2) = (o_1 = o_2)$

   There is only one output in a module.

Our relations are derived from Lakhotia [9]; relations 1, 4, and 5 are essentially identical to the corresponding relations of Lakhotia. The remaining relations are defined to fit the IODG model.

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 [8].

An IODG can be displayed visually in an IODG diagram. In its graphical form, the IODG visually displays the functional structure of the module. 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. Fig. 1 shows six cohesion levels for six simple modules.

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 [17]. 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 at the point of procedure exit. *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.

Fig. 2 shows functional cohesion computations for an example program. 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 one column 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 six 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.

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 dependencies. The DFC measures are analogous to the slice-based FC measures in that both are defined in terms of the connections between components and outputs. The components used to define the FC measures include all "data tokens," while only input and output components are used to define the DFC measures. Inputs and outputs are the only externally visible components, and they represent the design-level information of the module. Fig. 3a shows a graphical (IODG), and Fig. 3b shows a tabular (IODT) representation of procedure *Sum_Max_Avg* of Fig. 2.

In Fig. 3b, 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 the number of *isolated* and *essential* components, and the *connectedness* of components:

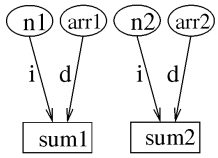
DEFINITION. *A component is* isolated *if it affects only one local functionality, i.e., it has a dependence relationship with only one output.*

For example, in Fig. 3, component 'max' is isolated since it has a dependence 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 Fig. 3, components 'n' and 'arr' are essential since they affect all outputs.

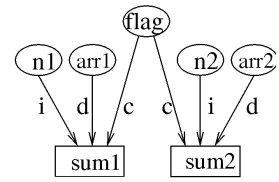
```
procedure Sum1_and_Sum2
  ( n1, n2 : integer;
    arr1, arr2 : int_array;
    var sum1,
        sum2 : integer );
var i : integer;
begin
    sum1 := 0;
    sum2 := 0;
      for i := 1 to n1 do
          sum1 := sum1 + arr1[i];
      for i := 1 to n2 do
          sum2 := sum2 + arr2[i];
end;
```



Coincidental cohesion

(a)

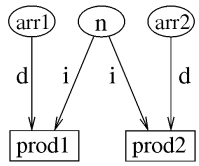
```
procedure Sum1_or_Sum2
  ( n1, n2, flag : integer;
    arr1, arr2 : int_array;
    var sum1,
        sum2 : integer );
var i : integer;
begin
    sum1 := 0;
    sum2 := 0;
    if flag = 1
      for i := 1 to n1 do
          sum1 := sum1 + arr1[i];
    else
      for i := 1 to n2 do
          sum2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



Conditional cohesion

(b)

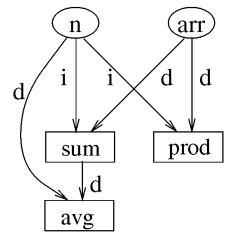
```
procedure Prod1_and_Prod2
  ( n : integer;
    arr1, arr2 : int_array;
    var prod1,
        prod2 : integer );
var i : integer;
begin
    prod1 := 1;
    prod2 := 1;
    for i := 1 to n do begin
        prod1 := prod1 * arr1[i];
        prod2 := prod2 * arr2[i];
    end;
end;
```



Iterative cohesion

(c)

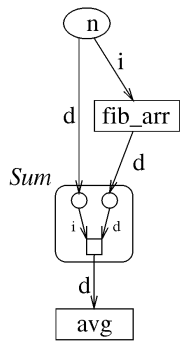
```
procedure Sum_and_Prod
  ( n : integer;
    arr : int_array;
    var sum,
        prod : integer;
    var avg : float );
var i : integer;
begin
    sum := 0;
    prod := 1;
    for i := 1 to n do begin
        sum := sum + arr[i];
        prod := prod * arr[i];
    end;
    avg := sum / n;
end;
```



Communicational cohesion

(d)

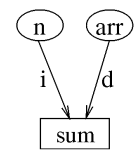
```
procedure Fibo_Avg
  ( n : integer;
    var fib_arr : int_array;
    var avg : float );
var sum : integer;
    i : integer;
begin
    fib_arr[1] := 1;
    fib_arr[2] := 2;
    for i := 3 to n
        fib_arr[i] := fib_arr[i-1]
                + fib_arr[i-2];
    Sum(n, fib_arr, sum);
    avg := sum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



Sequential cohesion

(e)

```
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

(f)

Fig. 1. IODG and DLC levels for six simple procedures.

sum	max	avg	statement
			procedure Sum_Max_Avg
1	1	1	(n : integer; /* pre: n>0 */
1	1	1	var arr: int_array;
1		1	var 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

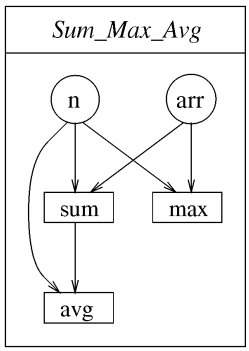
FC Measures:

WFC = 17/27 = 0.63

A = (11 * 2 + 6 * 3)/(27 * 3) = 0.49

SFC = 6/27 = 0.22

Fig. 2. Data slice profile for *Sum_Max_Avg*.



(a)

component \ 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)

Fig. 3. An example. (a) the IODG; (b) IODT of a procedure *Sum_Max_Avg*.

We define the *connectedness* of a component as the degree of "relatedness" of the component to the outputs. Connectedness provides more information than a simple classification of a component as isolated or essential. The connectedness of a component represents the relative number of outputs that the component relates together. We do not address the cases where an input does not contribute to the computation of any output, i.e., in our model, every component has a dependence relation with at least one output. Therefore, the connectedness 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 connectedness of every component in the module is 1.

DEFINITION. *For an arbitrary module, the connectedness of the* 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 i*th component has a dependence relation, and O is the total number of outputs in the IODG model of the module.*

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

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

DFC MEASURE DEFINITION.

$$\mathrm{LC}(m) = \mathrm{D}/\mathrm{T}$$

$$\mathrm{TC}(m) = \mathrm{E}/\mathrm{T}$$

$$\mathrm{MC}(m) = \frac{\sum_{i=1}^{T} C_i}{T}$$

where D, E, and C_i are the number of nonisolated components, the number of essential components, and the connectedness 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 connectedness, module cohesiveness can be expressed as:

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

The three measures for the procedure *Sum_Max_Avg* in Fig. 2 and Fig. 3 are:

$$\mathrm{LC}(Sum\_Max\_Avg) = 4/5 = 0.8$$
$$\mathrm{TC}(Sum\_Max\_Avg) = 2/5 = 0.4$$
$$\mathrm{MC}(Sum\_Max\_Avg) = (2 * 2 + 2 * 1)/(5 * 2) = 0.6$$

An isolated component has zero connectedness, a nonisolated component has connectedness of greater than 0, and essential component has connectedness of one. Thus, for a given module *m*:

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

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

$$\mathrm{TC}(m) \le \mathrm{MC}(m) \le \mathrm{LC}(m)$$

DFC measures have been derived using only interface components by following the approach used to define FC measures. Thus, we expect that each measure of DFC has some relationship with corresponding FC measures. DFC and DLC measures are both design-level measures defined in terms of program IODG information. We also determine the relationship between DFC and DLC measures. In next two sections, we investigate the relationships between the cohesion measures analytically and empirically.

4 Analytical Comparison of Cohesion Measures

4.1 Relationship Between the DFC and FC Measures

The DFC measures correspond closely to the FC measures of Bieman and Ott [2]. 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.

To be consistent with our derivation of the DFC measure, we can treat internal superglue tokens as *essential* internal tokens and internal nonglue as *isolated* internal tokens. Fig. 4 contains unlabeled IODG diagrams for different module configurations. Input, output, and selected internal data tokens are represented by circles, rectangles, and square bars, respectively. Fig. 4d 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 superglue or essential data tokens. As a result, the FC measures of the second module are higher than those of the first module. The third module contains more nonglue or isolated data tokens. As a result, the FC measures of the third module are lower than those of the first module. Fig. 4e and 4f also show that an increase in the number of essential or isolated data tokens affects the FC measures.

Fig. 4a, 4b, and 4c 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, nonisolated, 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.

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 Fig. 4d is more difficult to decompose into two modules than the third module in Fig. 4d. To decompose the second module, most of the data tokens need to be rewritten. However, the FC measures alone cannot 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.

We see that the FC and DFC measures are equivalent only for some modules. There is, however, a general correspondence between the FC and DFC measures. An empirical study can determine the distribution of isolated and essential data tokens in real software. We show, in Section 5, that there is a strong empirical relationship between FC and DFC measures.

4.2 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 demonstrate the effect on the measures of increases in the number of 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.

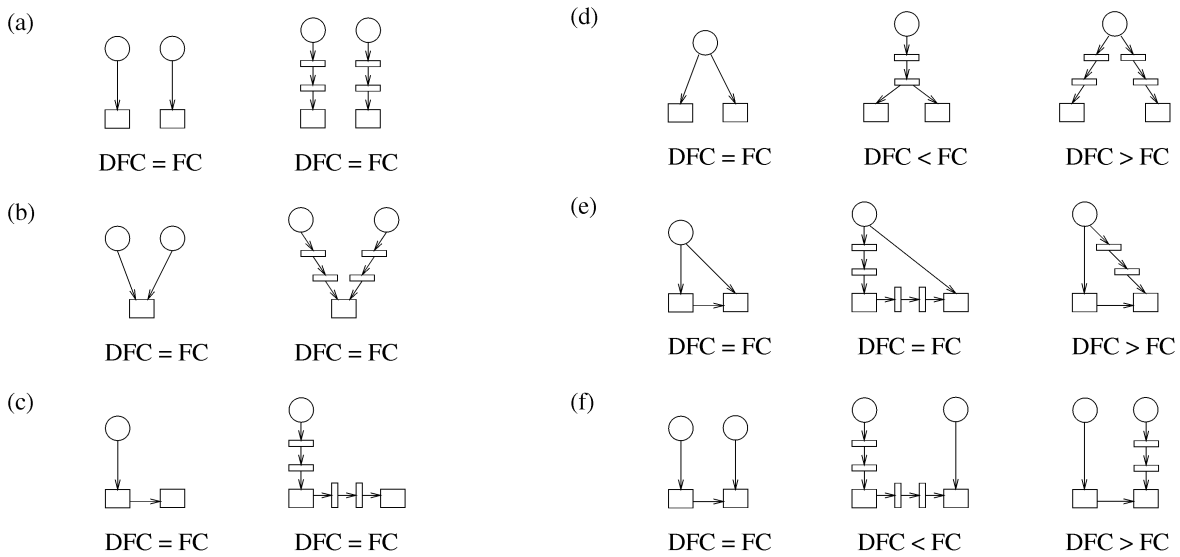


Fig. 4. Comparing the DFC and FC measures.

4.2.1 The Effect of Increasing the Number of Dependence Connections

Fig. 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. Fig. 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 nonisolated) components in a module. A relatively low LC value means that there are more isolated components than nonisolated ones.

The modules in Fig. 5c and 5d 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. Fig. 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" [19, 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, consider a module with an input that is used by two outputs. The DFC measures simply treat the input as an essential component for the outputs, while the DLC measure classifies the relationship between the two outputs into conditional, iterative, or communicational relation using the classified dependence information.

Among the MC, LC, and TC measures, TC is closest to DLC. In calculating DLC, the lowest cohesion level of all pairs is the cohesion of the module. The module in Fig. 5c 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 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 Fig. 5. In modules (a), (b), and (c) of Fig. 5, the DLC levels are 'coincidental' and the TC values are 0. In Fig. 5d and 5e, the DLC levels are 'indirect' and the TC values are 1/6.

4.2.2 The Effect of Increasing the Number of Input/Output Components

Fig. 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 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. In this case, The DLC indicates *coincidental* cohesion. These correspondences between DFC and DLC are shown in Fig. 6a and 6b.

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. Fig. 6c, 6d, and 6g show that DFC decreases when the relative number of isolated components is increased. Fig. 6f shows that when the relative number of essential components in a module is increased, the DFC value increases. In cases Fig. 6e and 6h, the relative number of essential components are not changed, and the DFC values also show no change.

The IODG's in rows (d), (e), and (h) of Fig. 6 exhibit *sequential* DLC. The IODG's in rows (e) and (h) all have DFC = 1, while those in row (d) with more than 1 output show lower DFC values. All components in the IODG's with DFC = 1 are linked to all outputs; the second output is computed only in terms of the first output, which is computed using all inputs. The IODG's with lower DFC values have one or more components linked to only one output. The second output makes use of input components that are independent of the first output. The differences in DFC values reflect the relative connectedness of the input components. The DLC measure does not show this difference, because it is determined only by the weakest relation, the sequential relation between the outputs.

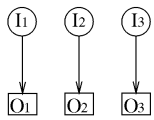
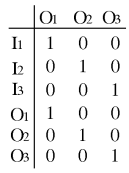
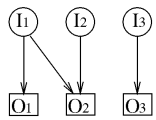
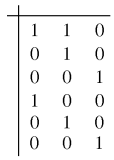
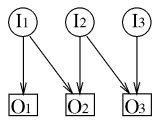
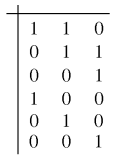
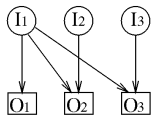
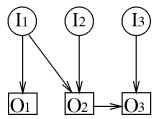
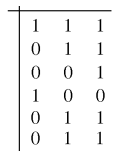
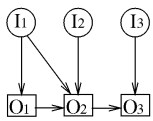
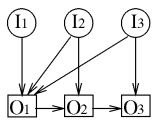
IODG	IODT	DFC Measure	Association Level	DLC Measure
		LC = 0 MC = 0 TC = 0	O_1, O_2 : Coincidental O_2, O_3 : Coincidental O_1, O_3 : Coincidental	Coincidental
(a)				
		LC = 1/6 MC = 1/12 TC = 0	O_1, O_2 : Indirect O_2, O_3 : Coincidental O_1, O_3 : Coincidental	Coincidental
(b)				
		LC = 2/6 MC = 2/12 TC = 0	O_1, O_2 : Indirect O_2, O_3 : Indirect O_1, O_3 : Coincidental	Coincidental
(c)				
		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
(d)				
		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
(e)				
		LC = 5/6 MC = 9/12 TC = 4/6	O_1, O_2 : Indirect Sequential O_2, O_3 : Sequential O_1, O_3 : Indirect Sequential	Sequential
(f)				
		LC = 1 MC = 1 TC = 1	O_1, O_2 : Indirect Sequential O_2, O_3 : Sequential O_1, O_3 : Indirect Sequential	Sequential
(g)				

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

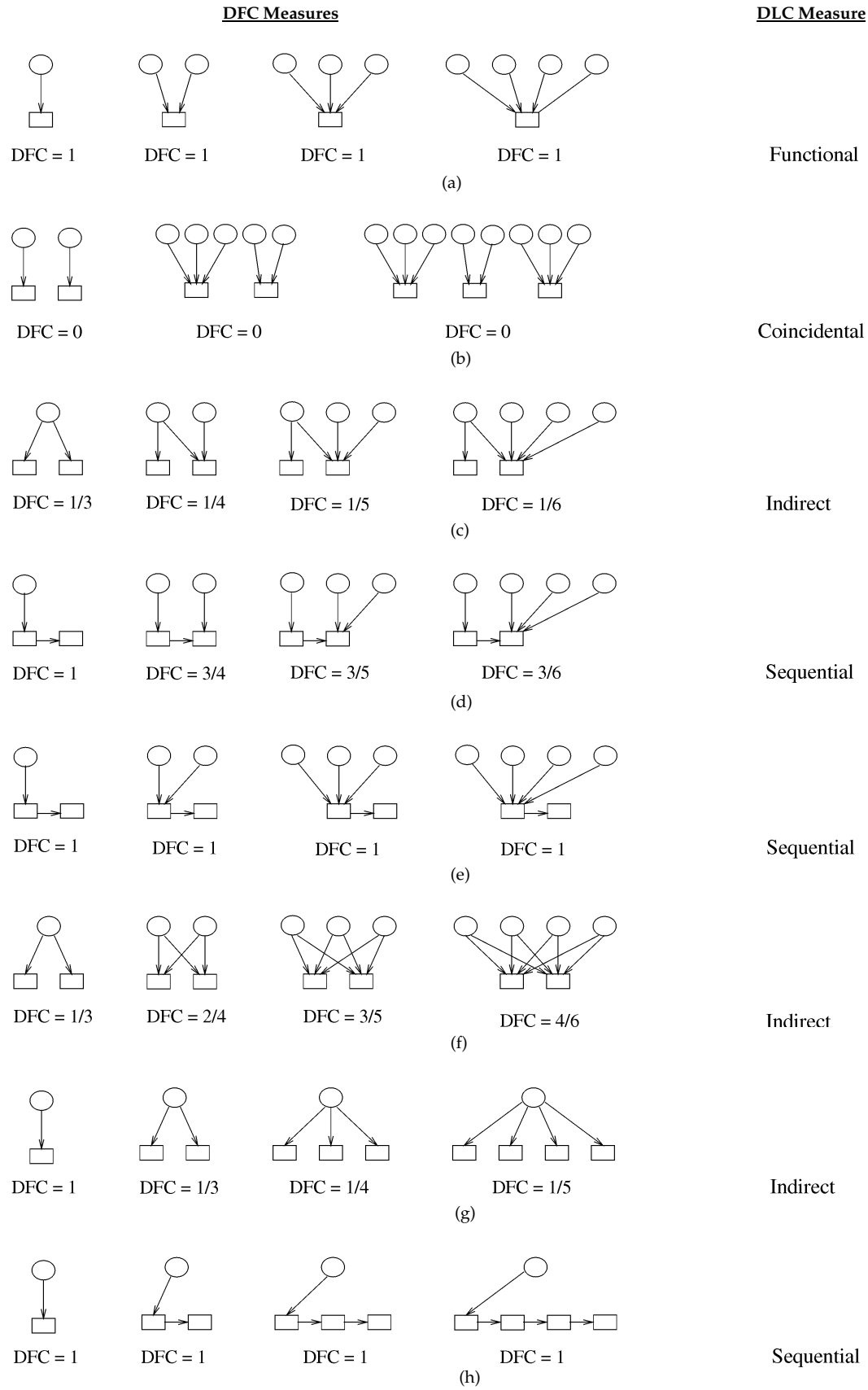


Fig. 6. The effect on the DFC and DLC measures of increasing the number of input/output components.

Except for the cases with *sequential* DLC, the most notable differences in DFC values are for one output IODG's versus two output IODG's. For example, see the first two IODG's in row (g). In the IODG displayed in the first column one input is used to compute one output, while the second column shows an IODG with one input that is used to compute two outputs. The two outputs may involve completely independent computation on the input, and from the interface alone, we cannot tell if the two outputs really belong in one procedure. Implementation details are needed to determine the actual degree of independence between the three outputs. Thus, as shown in Fig. 5d, the FC values may be higher or lower than the DFC values.

As we see in Fig. 6, the DLC measure does not capture 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, the DLC measure always reflects the weakest connection among module components and this weakest connection determines the cohesion level. DLC also provides more precise information characterizing the relationship between output components than the DFC measures. Among the three DFC measures, the TC measure corresponds 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 Empirical Comparison of Cohesion Measures

In this section, we empirically test the relationship between the design-level functional cohesion (DFC) measures and the functional cohesion (FC) measures, and the relationship between the DFC measures and the design-level cohesion (DLC) measure.

We developed tools to automate the cohesion measures and collected a set of C programs. Using the measurement tools, the collected programs have been processed to generate the cohesion data. The data are tested statistically and analyzed to find relationships between cohesion measures. The empirical study included the following tasks:

1) **Cohesion Tool Development**. We developed tools to measure FC, DFC, and DLC for C programs using lex and yacc from the gcc compiler. This work included the conversion of scalar analysis problems such as the reaching definition problem into monotone data flow systems, *MDS*s. We implemented the iterative algorithm for each MDS [20]. As a simplification, alias problems were ignored, since they are rare in actual programs and thus have little affect on the cohesion measurements. The tools process C programs in a Unix workstation environment. They have been installed and tested for Sun SPARCstatations running SUN-OS and IBM RS6000 systems running AIX.
2) **Input Program Collection**. We collected programs from two sites: 1) a collection of student programs, and 2) Unix system software. A total of 607 C functions have been collected: 390 C functions have been collected from five graduate students who major in computer science, and 217 C functions from three Unix system programs (FTP, PASSWD, and TALK).
3) **Cohesion Data Generation**. We generated the three FC measures (WFC, ADH, and SFC), three DFC measures (LC, MC, and TC), and the DLC measure for the collected input programs.
4) **Correlation Test and Analysis**. We use the cohesion data to find relationships between FC and DFC measures and between DFC and DLC measures. We test correlations between the related measures by generating correlation coefficient values and significance values of each pair of corresponding measures for each partition of programs, i.e., student and system programs, small, medium, and large programs, and the entire set of programs.

   A correlation coefficient is always a number between –1.0, a perfect negative correlation and 1.0, a perfect positive correlation. A high positive or low negative value of the correlation coefficient for two variables means that they have a strong association. The significance value is the probability of obtaining a particular sample result given the null hypothesis. In our correlation test, the null hypothesis is that two corresponding measures are not correlated, i.e., they are independent from each other. By convention, if the significance value is less than 0.05, we call it statistically significant.

5.1 Cohesion Measurement Data

Table 1 shows average cohesion values of the seven cohesion measures for all 607 C functions. We find that 343 of the functions have only one output. Their DLC levels are 6 (functional cohesion) and their FC and DFC values are 1. (For a given program, when its DLC is 6, its three FC measures and three DFC measures are always 1.) To avoid this "ceiling effect," we remove the cases where the DLC level is 6 and use the remaining 264 functions to find the relationship between cohesion measures. Table 2 shows the average values for seven cohesion measures after removing the functions with only one output.

Table 2 shows that the WFC, ADH, and SFC values and LC, MC, and TC are ordered. This ordering is consistent with ordering demonstrated analytically in Section 3: For a given program, $\mathtt{WFC} \leq \mathtt{ADH} \leq \mathtt{SFC}$ and $\mathtt{TC} \leq \mathtt{MC} \leq \mathtt{LC}$.

Table 2 also shows that for all 264 programs, the following relationships hold: $\mathtt{WFC} < \mathtt{LC}$, $\mathtt{ADH} < \mathtt{MC}$, and $\mathtt{SFC} < \mathtt{TC}$. These relations hold also for all partitioned groups, the student programs and system programs, three groups of programs by program size.

TABLE 1
MEAN AND MEDIAN OF FC, DFC, AND DLC MEASURES FOR 607 C FUNCTIONS

Average	No. of data tokens	No. of input/outputs	FC			DFC			DLC
			WFC	ADH	SFC	LC	MC	TC	
Mean	57.36	6.35	0.75	0.72	0.68	0.78	0.74	0.71	4.76
Median	30.00	4	1	1	1	1	1	1	1

TABLE 2
MEAN AND MEDIAN OF FC, DFC, AND DLC MEASURES FOR 264 C FUNCTIONS AFTER REMOVING FUNCTIONS HAVING ONLY ONE OUTPUT

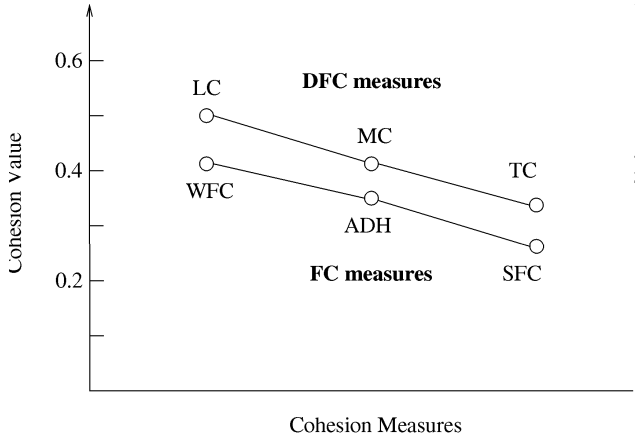
Average	No. of data tokens	No. of input/outputs	FC			DFC			DLC
			WFC	ADH	SFC	LC	MC	TC	
Mean	83.77	8.98	0.42	0.36	0.27	0.50	0.41	0.33	3.16
Median	52	8	0.42	0.31	0.14	0.50	0.33	0.20	4

The above relations occur because, on average, the distribution of data tokens are more 'isolated' and less 'essential' when compared to the distribution of interface components. In other words, programs include more isolated data tokens than isolated interface components. Fig. 7 shows graphically the relationship between the six measures, represented by Table 2.

We find very few programs exhibiting the third DLC level, iterative cohesion. This does not mean there are few iterative associations between output components. Actually, we find many iterative associations in programs, however, when DLC cohesion level is determined, iterative associations are hidden by other stronger or weaker associations.

5.2 Relationship Between FC and DFC Measures

Since each of DFC measures (LC, MC, and TC) was derived using an approach that parallels each of the FC measures (WFC, ADH, and SFC), we expect that the DFC measures will correspond closely to the FC measures. We showed analytically in Section 4.1 that a general correspondence between the FC and DFC measures is expected. We also showed empirically the LC, MC, and TC measures are greater than the WFC, ADH, and SFC measures, respectively. In order to know how closely the measures are related, we test correlations between WFC and LC measures, between ADH and MC measures, and between SFC and TC measures.



We also consider the effects of program development environment and program size which are possible extraneous variables. The entire set of C functions is partitioned into student programs and system programs, and small size programs (data token count ≤ 50), medium size programs (50 < data token count ≤ 100), and large size programs (100 < data token count). Table 3 shows correlation coefficient values of each pair of corresponding measures for each partition of programs. The significance value for each pair is 0.0001.

We observe that:

1) The FC and DFC measures are strongly correlated. The correlations between WFC and LC, between ADH and MC, and between SFC and TC are all strong. There is a general correspondence between the FC and DFC measures; if a program has a high MC value, then it is very likely that the program has a high ADH value too. It follows that we can predict FC values of a program from its DFC values with an expected error range. In another words, cohesion of program code can be predicted during design.
2) The correlation between the FC and DFC measures for the small programs is slightly stronger than that for the large programs. This difference between small and large programs is due to the definitions of FC and DFC measures. The FC measures are defined using module body information while the DFC measures are defined using only module interface. The possible difference between a programs FC values and DFC values increase as the number of data tokens in a program increases.
3) The correlation between the FC and DFC measures for the system programs is slightly stronger than that for the student programs. The difference is due to outliers. Fig. 8 and Fig. 9 show scatter plots of ADH and MC for student programs, and of ADH and MC for system programs, respectively. From the figures, we find more outliers for the student programs than for the system programs. A small number of outliers can cause a large correlation coefficient difference.

TABLE 3
CORRELATION COEFFICIENTS OF WFC AND LC, ADH AND MC, AND SFC AND TC FOR EACH PARTITION OF COLLECTED PROGRAMS

Site/Size	No. of functions	Mean and Median data tokens		Correlation Coefficient		
				WFC-LC	ADH-MC	SFC-TC
Student Programs	141	95.36	53	0.8412	0.8437	0.8526
System Programs	123	70.47	51	0.9285	0.9418	0.9556
data token cnt ≤ 50	129	25.51	24	0.9535	0.9486	0.9565
50 < data token cnt ≤ 100	69	73.61	73	0.8286	0.8337	0.8720
100 < data token cnt	66	208.24	172	0.8065	0.8242	0.8208
Total	264	83.77	52	0.8896	0.8939	0.9043

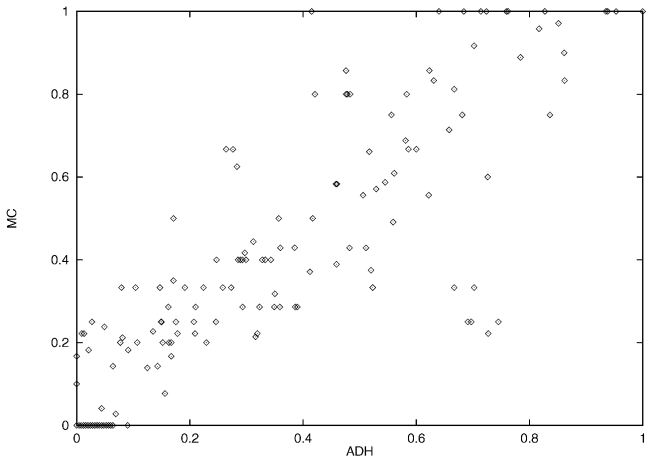


Fig. 8. Scatter plotting of ADH and MC for student programs in Table 3.

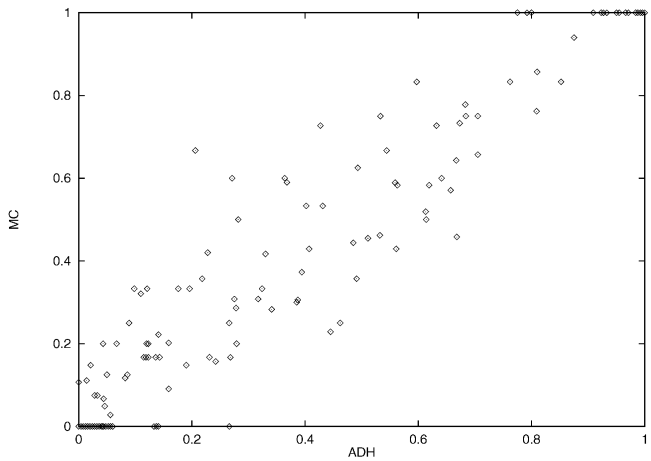


Fig. 9. Scatter plotting of ADH and MC for system programs in Table 3.

5.3 Relationship Between DFC and DLC Measures

The DLC measure finds the weakest connection among module interface components, while the DFC measure is defined as the degree of connectivity between module interface components. We showed analytically there is fundamental difference between those measures though there is some correspondence between them.

In order to know how closely the measures are related with each other, we test correlations between DLC and LC measures, between DLC and MC measures, and between DLC and TC measures. We also consider the effects of program development environment and interface size which are possible extraneous variables. The set of C functions is partitioned into student programs and system programs, and small interface programs (input/output count ≤ 5), medium interface programs (5 < input/output count ≤ 9), and large interface programs (9 < input/output count).

Since DLC is an ordinal scale measure, we use Spearman's rank correlation coefficient. The test uses the ranks of the values of variables rather than the values themselves. Table 4 shows the result of the correlation test, where a significance value corresponding to each correlation coefficient value is 0.0001.

We make the following observations from Table 4:

1) The DFC and DLC measures are correlated. The relationship between DLC and TC is stronger than the relationship between DLC and MC, and the relationship between DLC and LC is the weakest. This result matches our analytical study. 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.
2) The correlation between the DFC and DLC measures for the programs with small interfaces is stronger than that of the programs with larger interfaces. This result is consistent with the analytical study—DFC measures are sensitive to the number of interface components and the number of connections between them, while DLC is not. Thus, as the number of interface components increase, the difference between DFC values and DLC values will increase.
3) DLC is closest to TC among the three DFC measures (MC, LC, and TC). Again, this result matches our analytical study. Both DLC and TC are calculated using the most extreme cases: DLC finds the weakest connection between interface components and TC captures only essential interface components.

5.4 Relationship Between FC and DLC Measures

We also have performed the correlation test between FC and DLC measures. From the test, we find that the relationship between FC and DLC is very similar to that between DFC and DLC except that the correlation between FC and DLC is weaker than that between DFC and DLC measures. For 264 C functions with more than one output, the

TABLE 4
CORRELATION COEFFICIENTS OF DLC AND LC, DLC AND MC, AND DLC AND TC FOR EACH PARTITION OF COLLECTED PROGRAMS

Site/Size	No. of functions	Mean and Median data tokens		Correlation Coefficient		
				DLC-LC	DLC-MC	DLC-TC
Student Programs	141	8.18	6	0.6122	0.7593	0.8635
System Programs	123	9.89	8	0.6753	0.8289	0.8715
1 < I/O cnt ≤ 5	67	4.31	4	0.8751	0.9325	0.9259
5 < I/O cnt ≤ 9	112	7.11	7	0.5597	0.7385	0.8814
9 < I/O cnt	85	15.12	13	0.4845	0.6926	0.7852
Total	264	8.98	7	0.6345	0.7884	0.8760

correlation coefficients of WFC-DLC, ADH-DLC, and SFC-DLC are 0.6007, 0.7035, 0.8202, respectively, with a significance value of 0.0001 for every case.

We can expect this result from the definitions of FC, DFC, and DLC measures: DFC measures have been derived from FC measures, and the comparison between DFC and DLC is one between design measures while the comparison between FC and DLC is one between design and code measures.

6 DISCUSSION

The results from our empirical study support the analytically developed relations between measures. We find a general correspondence between the design-level functional cohesion (DFC) measures and the code-level functional cohesion (FC) measures. Each code-level measure tends to exhibit lower cohesion values than the corresponding design-level measure. However, each design-level measure correlates with the corresponding code-level measure at the 0.0001 significance level.

The strongest correlations were between DFC and FC of the sample of systems programs. Each of the corresponding design/code-level pairs of measures exhibit correlations of more than 0.92. The correlation of DFC to FC for student programs are less than that of systems programmers. However, all of these correlations are at least 0.84.

Our results support the use of design-level cohesion measures as surrogates for code-level measures. The design-level measures can be obtained before code is written, and thus can be used to predict code-level cohesion values.

All of the cohesion measures can be used to help identify poorly designed modules. These modules may perform multiple functions that are disjoint or only weakly connected. Such poorly designed modules are candidates for redesign and restructuring. The design-level DLC measure and the code-level FC-TC measure can identify modules that are easy to decompose. They indicate the modules whose components exhibit the greatest independence. The measurement of structural attributes such as cohesion provides a mechanism for quantifying design improvements, and are, thus, a mechanism for use in restructuring designs and implementations.

Restructuring can be accomplished through a series of functional restructuring (decomposition and composition) operations based on the IODG model and a set of objective criteria [7], [8]. In addition to cohesion measures, restructuring criteria can include coupling and other information. 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.

When displayed as a diagram, the IODG model provides a visual representation that complements the quantitative information provided by the measures. The measures can help select candidate modules for restructuring. Then, an engineer can view an IODG diagram to determine if and how a candidate module should be restructured.

7 CONCLUSIONS

We have formalized the concept of design cohesion based on a graph model of a procedure interface, the input/output dependence graph (IODG). We derived a design-level cohesion (DLC) measure using an association-based approach similar to that used by Stevens et al. [15], and design-level functional cohesion (DFC) measures using the slice-based approach used to derive code-level functional cohesion (FC) measures [2]. All of these measures have been implemented. We compared the cohesion measures both analytically and empirically and evaluated potential applications of the IODG model and cohesion measures.

We find that:

1) Cohesion can be objectively defined and measured in terms of design-level entities. Our cohesion measures are consistent with intuitive notions to satisfy the representation theorem of measurement [6].
2) Design-level cohesion measures correspond closely with code-level cohesion measures. Thus, design-level cohesion measures can be used to predict cohesion at the code-level. Also design-level measures can be used as surrogates for code level measures.
3) The design-level measures can be used to help locate poorly-designed modules, especially modules that need to be restructured. The DLC measure always finds the weakest connection among module interface components while the DFC measures detect the the degree to which input/output components are connected.
4) The IODG model provides a flexible tool for a quantitative and qualitative characterization of a software design. The IODG is the basis for all of the design-level measures. IODG models can also be readily displayed as diagrams to provide a form of program visualization showing the functional structure of a system. IODG can be generated from design information, or, during maintenance, they can be generated directly from program code.

Design-level measures can be used to improve software quality by providing quantitative criteria for comparing design alternatives. We plan to further evaluate the effectiveness of design measures for use in software restructuring.

The generation of design information, such as IODGs, from code is a form of reverse engineering. A long term objective is to learn how to generate software architectural structures from program code. Generating IODGs from program code is a first step. Our focus now is on generating higher-level design structures from IODGs. The software maintenance and software evolution community can clearly benefit from such reverse engineering technology.

ACKNOWLEDGMENT

This work was partially supported by the NASA Langley Research Center under Grant No. NAG1-1461.

REFERENCES

[1] J. Bieman and B.-K. Kang, "Cohesion and Reuse in an Object-Oriented System," *Proc. ACM Symp. Software Reusability, SSR'95*, pp. 259–262, Apr. 1995. Reprinted in *ACM Software Eng. Notes*, Aug. 1995.

[2] J. Bieman and L. Ott, "Measuring Functional Cohesion," *IEEE Trans. Software Eng.*, vol. 20, no. 8, pp 644–657, Aug. 1994.

[3] L. Briand, S. Morasca, and V. Basili, "Measuring and Assessing Maintainability at the End of High Level Design," *Proc. Conf. Software Maintenance*, pp. 88–95, Sept. 1993.

[4] S. Chidamber and C. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Trans. Software Eng.*, vol. 20, no. 6, pp. 476–493, June 1994.

[5] T.J. Emerson, "A Discriminant Metric for Module Cohesion," *Proc. Seventh Int'l Conf. Software Eng., ICSE-7*, pp. 294–303, 1984.

[6] N. Fenton, "Software Measurement: A Necessary Scientific Basis," *IEEE Trans. Software Eng.*, vol. 20, no. 3, pp. 199–206, Mar. 1994.

[7] B.-K. Kang and J. Bieman, "Using Design Cohesion to Visualize, Quantify, and Restructure Software," *Eighth Int'l Conf. Software Eng. and Knowledge Eng., SEKE '96*, June 1996.

[8] B.-K. Kang and J. Bieman, "Using Design Abstractions to Visualize, Quantify, and Restructure Software," *J. Systems and Software*, to appear.

[9] A. Lakhotia, "Rule-Based Approach to Computing Module Cohesion," *Proc. 15th Int'l Conf. Software Eng.*, pp. 35–44, 1993.

[10] B. Mehra, "Measuring Data Cohesion in the Object-Oriented Paradigm," master's thesis, Dept. of Computer Science, Michigan Technological Univ., 1997.

[11] L. Ott, J. Bieman, B.-K. Kang, and B. Mehra, "Developing Measures of Class Cohesion for Object-Oriented Software," *Proc. Ann. Oregon Workshop Software Metrics, AOWSM'95*, June 1995.

[12] S. Patel, W. Chu, and R. Baxter, "A Measure for Composite Module Cohesion," *Int'l Conf. Software Eng.*, pp. 38–48, May 1992.

[13] L. Rising and F. Calliss, "Problems with Determining Package Cohesion and Coupling," *Software-Practice and Experience*, vol. 22, no. 7, pp. 553–571, July 1992.

[14] M.H. Samadzadeh and S.J. Khan, "Stability, Coupling, and Cohesion of Object-Oriented Software Systems," *Proc. 22nd Ann. ACM Computer Science Conf.*, pp. 312–319, 1994.

[15] W. Stevens, G. Myers, and L. Constantine, "Structured Design," *IBM Systems J.*, vol. 13, no. 2, pp. 115–139, 1974.

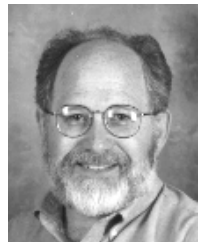
[16] D. Troy and S. Zweben, "Measuring the Quality of Structured Designs," *J. Systems and Software*, vol. 2, pp. 113–120, 1981.

[17] M. Weiser, "Program Slicing," *IEEE Trans. Software Eng.*, vol. 10, no. 4, pp. 352–357, 1984.

[18] M. Woodward, "Difficulties Using Cohesion and Coupling as Quality Indicators," *Software Quality J.*, vol. 2, no. 2, pp. 109–127, June 1993.

[19] E. Yourdon and L. Constantine, *Structured Design*. Englewood Cliffs, N.J.: Prentice Hall, 1979.

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



James M. Bieman is an associate professor in the Computer Science Department at Colorado State University. Dr. Bieman's research is focused on software measurement and automated software testing. He is developing methods to quantify software reuse, cohesion, and coupling, with a focus on attributes of object-oriented software. He is evaluating the use of executable specifications and software probes as practical software testing oracles. Dr. Bieman has conducted collaborative research projects with engineers at Reliable Software Technologies, Storage Technology, Hewlett-Packard, Martin-Marietta, CTA, Micro-Motion, and other companies. He is a senior member of the IEEE, a member of the ACM, and was general chair of *Metrics'93* and *Metrics'97*.

Byung-Kyoo Kang received a BS degree in computer science from Western Illinois University; an MS degree in computer science from Washington University in St. Louis; and the PhD degree in computer science from Colorado State University in 1997, where he developed a quantitative framework for restructuring software. While at Colorado State, he implemented the Funco tool which measures functional and design-level cohesion in C programs.Dr. Kang is a senior member of technical staff in the Software Engineering Division of the Electronics and Telecommunications Research Institute (ETRI), Korea.