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
kang@cs.colostate.edu, bieman@cs.colostate.edu

Abstract

Cohesion was first developed to predict properties of implementations 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.

Keywords: 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"[9, 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" [9, 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 [6], skilled engineers conduct subjective assessments of module cohesion. Such assessments are difficult to automate and use in practice [8].

There are objective, automatable methods for measuring *code-level* cohesion. Lakhotia [4] uses an *association-based* approach to formalize the notion of the associations between processing elements as a set of rules concerning data dependencies in module code. Bieman and Ott [2] use a *slice-based* approach to measure functional cohesion in terms of the connections between module output slices. 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].

We use both the association-based and slice-based approaches to develop *design-level* cohesion measures.

2 Association-based Cohesion

Stevens, Myers and Constantine define module cohesion (SMC Cohesion) on an ordinal scale including *coincidental*, *logical*, *temporal*, *procedural*, *communicational*, *sequential*, and *functional* cohesion [6]. Coincidental is the weakest and functional is strongest cohesion. SMC Cohesion is determined by the associations between all pairs of a module's processing elements.

We [3] have used SMC Cohesion as an empirical relation system to help us to derive a cohesion measure that can be applied to both the design and code of a module, and can be readily automated. We now summarize the derivation.

2.1 A Design-Level View of a Module

The input-output dependence graph (IODG) models the data and control dependence relationships between module input and output components. Input components of a module include in-parameters and referenced global variables. Output components include out-parameters, modified global variables, and 'function return' values. An array, a linked list, a record, or a file is one component rather than a group of components. We use terms based on definitions from compiler design sources [10].

IODG Preliminaries. Variable y has a *data depen-*

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

dence on variable x if x 'reaches' y through a path consisting of a 'definition-use' and 'use-definition' chain; y has a *control dependence* on x if the value of x determines whether or not the statement containing y will be performed; y is *dependent* on x when there is a path (a *dependence path*) from x to y through a sequence of data or control dependence; y has *condition-control dependence* on x if y has a control dependence on x, and x is used in the predicate of a decision (i.e., if-than-else) structure; y has *iteration-control dependence* on x if y has a control dependence on x, and x is used in the predicate of an iteration structure; y has *c-control dependence* on x if the dependence path from x to y contains a decision-control dependence; y has *i-control dependence* on x 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 digraph, $G_M =$ *(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)

We define six relations between a pair of output components based on the IODG representation:

1. **Coincidental relation (R_1):** Two module outputs have neither dependence relationship with each other, nor dependence on a common input.

2. **Conditional relation (R_2):** Two outputs are c-control dependent on a common input, or one output has c-control dependence on the input and another has i-control dependence on the input.

3. **Iterative relation (R_3):** Two outputs are i-control dependent on a common input.

4. **Communicational relation (R_4):** Two outputs are dependent on a common input. One has data dependence on the input and the other has either a control or a data dependence.

5. **Sequential relation (R_5):** One output is dependent on the other output.

6. **Functional relation (R_6):** There is only one output in a module.

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

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

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

Figure 1: Data slice profile for *Sum_Max_Avg*.

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.

The DLC measure is consistent with the ordinal scale of SMC Cohesion [3].

3 Slice-based Cohesion Measures

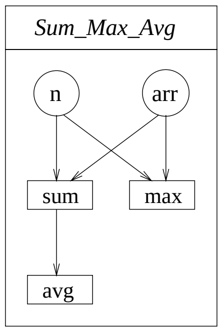
A program *slice* is the portion of the program that might affect the value of a particular identifier at a specified point in the program [7]. Slices can represent the functional components of a module.

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 based on "data slices" for each output of a procedure. The *data slice* of a variable is the sequence of data tokens which have a dependence relationship with the variable. *Glue tokens* are data tokens common to more than one data slice; *superglue tokens* are common to every data slice of a module. The adhesiveness of a data token is the number of data slices that the data token lies on.

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

Figure 1 shows example functional cohesion computations. Each column in the figure corresponds to



(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)

Figure 2: An example (a) the IODG and (b) IODT of a procedure *Sum_Max_Avg*.

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.

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 2(a) shows an IODG diagram and Figure 2(b) shows a tabular (IODT) representation of procedure *Sum_Max_Avg* of Figure 1.

In the IODG diagram of Figure 2(a), 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.

In Figure 2 (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 defined using the concepts of *isolated* and *essential* components, and component *cohesiveness*.

DFC Preliminaries: A component is *isolated* if it affects only one local functionality, i.e., it has a dependence relationship with only one output. A component is *essential* if it affects (or is affected by) all functionalities of the module — it has dependence relationships with all outputs of the module.

Component 'max' in Figure 2 is the only isolated component; 'max' has a dependence relationship with only one output, itself. If a module contains only one output, the output is the only functionality of the module. Thus, all components in the module are essential and not isolated. In Figure 2, components 'n' and 'arr' are essential since they affect all outputs.

The *cohesiveness* of a component is its degree of "relatedness" to the outputs. The cohesiveness of a component represents the relative number of outputs that the component relates together. In our model, every component has a dependence relation with at least one output. The cohesiveness of a component is the relative number of the other output(s) with which the component has a dependence relation. The cohesiveness of i'th component of a module 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 in a dependence relation with the ith component, and O is the number of outputs in the module's IODG.

The cohesiveness of an isolated component is 0 and the cohesiveness of an essential one is 1. In Figure 2(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.

DFC Measure Definition. *Loose Cohesiveness* (LC), *Tight Cohesiveness* (TC) and *Module Cohesiveness* (MC) are the relative number of non-isolated components, the relative number of essential components, and the average cohesiveness of the components of the model, respectively:

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

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 1 and 2 are

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

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 DFC vs. FC measures

Figure 3 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 3(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, and the FC measures of the second module are higher than those of the first module. The third module contains more isolated data tokens. Thus, the FC measures of the third module are lower than those of the first module. Figure 3 (e) and (f) also show that an increase in the number of essential or isolated data tokens affects the FC measures.

Figure 3 (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.

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 3(d) is more difficult to decompose into two modules than the third module in 3(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 DLC vs. 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 a simplified IODG (without dependence labels). The simplified IODG cannot distinguish between 'conditional', 'iterative', and 'communicational' DLC levels, so these three levels are denoted as 'indirect' relations with 'indirect' cohesion.

4.1 The effect of increasing the number of dependence connections.

To see the effect of increasing the number of connections on the measures, we assume a fixed number of inputs and outputs for a set of modules. 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. 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

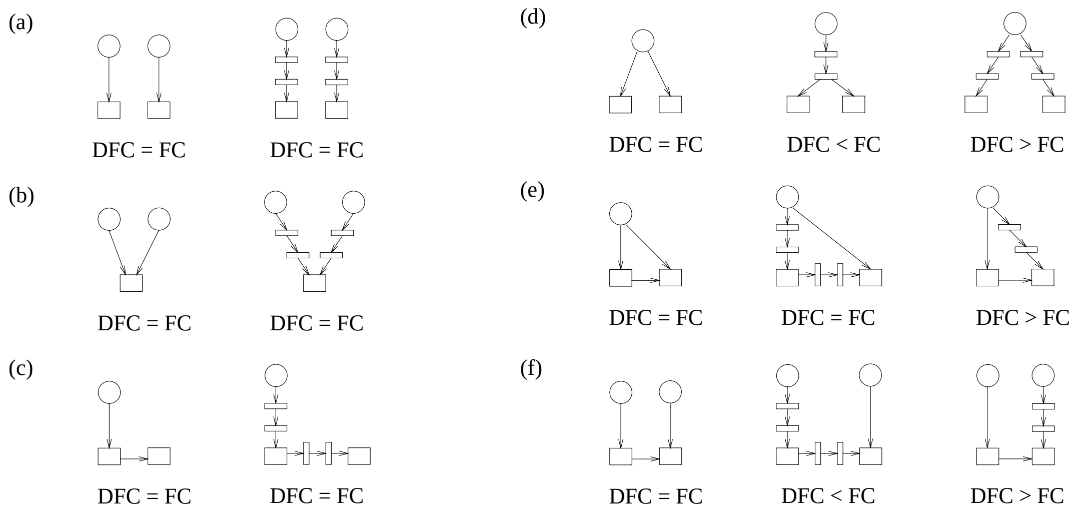


Figure 3: Comparing the DFC and FC measures.

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.

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.

DLC measure. 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 with some unconnected components. 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" [9, p. 132].

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. TC is 0 for a module when 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.

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

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.

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. When the relative number of essential components in a module is increased, the DFC value increases. If the relative number of essential components are not changed, the DFC values do not change.

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, the DLC measure always finds the weakest connection among module components to determine 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 corresponds 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

The IODG model and associated measures can 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. For existing software, the IODG information can be generated automatically using a compiler-like tool. Without an implementation, IODG information can be part of a detailed design. IODG diagrams can be generated from IODG information.

- The DLC/DFC measures can identify 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 support software redesign and restructuring [3]. The measures are criteria for determining whether or not a given module should be redesigned or restructured. An IODG diagram can help engineers decide how to restructure selected modules.

6 Conclusions

We formalize 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 [6], and generally correspond to several existing code-level cohesion measures.

We derived these measures using the association-based [6] and slice-based [2] approaches. Each measure quantifies 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 help visualize the functional structure of programs and provides support for program understanding. The design-level cohesion measures can identify poorly designed modules. The 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)*, pp. 259–262, 1995.

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

[3] B-K Kang and J. Bieman. Using design cohesion to visualize, quantify, and restructure software. *Proc. 8th Int. Conf. Software Engineering and Knowledge Engineering (SEKE'96)*, June 1996.

[4] A. Lakhotia. Rule-based approach to computing module cohesion. *Proc. 15th Int. Conf. Software Engineering*, pp. 35–44, 1993.

[5] 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)*, 1995.

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

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

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

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

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