A Well-Evaluated Cohesion Metrics for Software Quality*

LEE Ming-chi[1], SHIH Kuo-chen(Timothy)[1,2], HUANG Teh-sheng[1,2], DENG Yu-kuang[1,2]

[1](*Department of Buisness Administration, Ping Tung Institute of Commerce, Ping Tung*);

[2](*Department of Computer Science and Information Engineering, Tamkang University, Ping Tung*)

E-mail: g8190070@tkgis.tku.edu.tw

Received November 6, 2000; accepted March 5, 2001

Abstract: Without software metrics, software would be error prone, expensive and with low quality. Cohesion is one of the most important factors for software quality as well as maintainability, reliability and reusability. The module of poor quality should be a serious obstacle to system quality. In order to design and maintain good quality software, software managers and engineers inevitably need to introduce cohesion metrics to measure and produce desirable software. In this paper, based on the analysis of live variables and the visualized live span, a function-oriented cohesion metrics is proposed. A series of experiments are shown to support the cohesion metrics, and a set of properties to evaluate the proposed cohesion metrics. Therefore, a well-defined, well-experimented and well-evaluated cohesion metrics is proposed to indicate software cohesion strength and thus improve software quality. Furthermore, this cohesion metrics can be easily incorporated into software CASE tool to help software engineers to ensure software quality.

Key words: software metrics; cohesion; software quality; live variables

Module cohesion is defined as a critical quality attribute, which seeks for measuring the singleness of purpose of a module. Cohesiveness is a measure of an individual module's internal strength, which is the strength of the interrelationship of its internal element[3]. Fifty percent of the highly cohesive procedures are fault free, whereas only eighteen percent of procedures with low cohesion are fault free[4]. Software production is often out of control due to the lack of measurement[2]. Cohesiveness is one of the possible measurements for program modifiability[5]. This implies that software quality can be improving by maximizing the degree of module cohesion. However, Stevens, Myers and Contantine firstly define the notion of cohesion (SMC Cohesion) on ordinal order of seven levels that describes the degree of modules. The factors are functional, sequential, communicational, procedural, temporal, logical and coincidental[3,6]. The cohesion strength of a specific procedure is increasing from coincidental to functional. A highly cohesive software module has one basic function and is indivisible[7]. And, the coincidental is the least desirable but the functional is the most desirable cohesion level.

Live variables describe the extent of variables to be referred within a module, while variable span captures the

---

* **LEE Ming-chi** was born in 1960. He received his Ph. D. degree in the computer science of Tamkang University in 1994. He is now an associate professor in the Ping Tung Institute of Commerce. His major researches include object oriented database and software testing. **SHIH Kuo-chen (Timothy)** received his Ph. D. degree in the computer engineering of Santa Clara University in 1993. His research interests include multimedia computing, software engineering, and formal specification and verification. **HUANG Teh-sheng** received his MA in the Tamkang University in 1986. He is now a doctoral candidate at Tamkang University. His research focuses on software engineering, distributed object computing and distributed multimedia system. **DENG Yu-kuang** received his M. S. degree in the computer science of Tamkang University in 1999. He is now a doctoral candidate at Tamkang University. His researches include multimedia presentation and software engineering.

range of variable to be used in a module[8]. Module cohesion aims to measure how tight between output processing elements[9~11]. In this paper, the proposed function-oriented cohesion metrics is based on an analysis model of live variable semantics. We examine the function-oriented cohesion metrics (FOCM) for each output function, the most tightest function-oriented cohesion metrics (MTFOCM), the least tightest function-oriented cohesion metrics (LTFOCM), and the average tight function-oriented cohesion metrics (ATFOCM) of a specific procedure.

In general, we always strive for a high cohesion. The mid-range of the cohesion spectrum is often acceptable[12]. And, a module may exhibit more than one type of cohesion. However, we will show that the proposed metrics not only closely match SMC Cohesion and Fenton's cohesion strength spectrum[7], but also meet nonlinear cohesion scale of cohesion that was stressed by Pressman and Sommerville[12,13]. For past twenty years, researchers constantly attempt to develop meaningful software measures, and practitioners have strong willing to introduce the measures as tools to control software quality. But, software measurement must be well-grounded in theory and be obtained through well-designed experimental works[14]. The remainder of the paper is organized as follows. We address basic works about live variables semantic and formal definitions in Section 1. We propose visualized analysis model and cohesion metrics in Section 2. Section 3 addresses an experimental evaluation and theoretical evaluation to validate the cohesion metrics. Section 4 presents related works and comparisons. We give the conclusions in Section 5.

1 Basic Definitions

A module is a contiguous sequence of program statements that are bounded by boundary elements, and has an aggregate identifier[5]. A module is usually defined as a segment of codes that is independently compilable[8]. However, many researchers define a module to be a compilation unit of code, a view of program, or a procedure. By functionality, the processing elements in a procedure could be categorized into the following four categories:

- Input variable (IV): a set of the variables that are the input arguments of a procedure.
- Internal variable (INV): a set of local variables that are declared in a procedure.
- Global variable (GV): a set of global variables of a procedure.

Output variable (OV): a set of variables that are the output functions of a procedure.

In general, variables in an individual module may be used in assignment statement, initial statement, decision predicate and output function, iteration constructor and Boolean semantics which will construct the main body of the whole procedure. A module will contain specific function semantics. In practice, it will be the key issue to develop a suitable model to construct the overall reference scenarios of the function in the procedure scope. Live variables of a statement in module are the processing elements that are referenced in the statement. For sake of explaining the concept of live variables, a procedure example is given as follows:

As SumAndProd listed in Fig. 1, *sum* in line 4 is a live variable. Both *I* and *N* are live variables in line 6. On the other hand, *sum* is only referred at two statements in the procedure. Should software engineers concern *sum* in other statements other than lines 4 and 7? The answer is "yes". However, software engineers constantly have to keep in mind where *sum* may be referred between statements 4 and 7. Software engineers always realize that in each iteration *sum* has a different status in the for_loop, *sum* will be incremented, though the statements do not refer to *sum* in lines 5 and 6. Finally, the summation operation will exit. Similarly, *prod* has the same scenario like *sum*. And, Comprehending software artifacts are important software engineering activities. A significant amount of time of a software engineer is spent in looking at the source code to discover information during testing, review and code inspection[15]. We also know that programmers must be aware of the status of a number of data items during the programming process[8]. Thus, the more data items a programmer must keep track of when constructing a procedure, the more difficult it is to construct. Based on the above issues, live variables of a statement are

not limited to the number of variable reference in that statement. In the past years, there are several possible scopes of live variables to be defined inevitably[8,16]. The first is the live variable computes from the beginning of a procedure to the end of the procedure. According to this definition, the computation is simple, but this violates the live variable abstract. The second is at a particular statement, only if variable is referenced by a certain number of statements before or after that statements. Depending on the second definition, we may calculate lifespan. But there is no agreement on what "certain number of statements" should be and no successful use has been reported[8]. The third one is from its first reference to its last reference within a procedure. However, the third definition not only meets the principles of live variable abstraction, but also can be counted algorithmically as well. We believe that the third is a suitable domain for computing module cohesion. Therefore, we will adopt the third as the definition of live variables in this paper. We use mathematics to define the live variables and use live span to visualize a live variable scope. Therefore, we denoted the life span of a variable to be the occurrence domain that begins at the first referenced and extends through the last occurrence.

```
1  Procedure SumAndProd(N:integer;Var sum;prod:Integer)
2  Var I:Integer
3  Begin
4   sum:=0;
5   prod:=1;
6    For I:=1 to N do begin
7     sum:=sum+I;
8     prod:=prod*I;
9   end
10 end
```

Fig. 1 A procedure example

Definition 1. The lifespan of a variable in a procedure is a set, denoted by LS(var_nam). The elements of the set are the names of the variables which are referenced from the first (i-th line) to the last (j-th line), LS(var_nam)={var_nam_i,var_nam_{i-1},...,var_nam_j}. And, the size of lifespan, |LS(var_nam)|=$j-i+1$.

As an example in Fig. 1, we investigate the live span of variable *sum*. The set LS(sum)={sum_4,sum_5,sum_6,sum_7}, with a size of 4. The LS set also represents a mapping from the references of *sum* to different location (i.e. lines 4,5,6 and 7).

Definition 2. The live variables (LV) of a specific procedure (sp) is the set union of the lifespan of each variable, which belongs to IV, INV, GV or OV. We denote LV(sp)=$\bigcup$LS(var_nam_i) where $var\_nam_i \in$ IV$\bigcup$INV$\bigcup$GV$\bigcup$OV and $i \in N$.

In Fig. 1, we could construct LV(*SumAndProd*). First, we know the processing elements are IV={N}, OV={sum,$prod$}, GV=$\varnothing$, and INV={I} respectively. Next, LS(iv)={N_6} where $iv \in$ IV, LS(ov)={sum_4,sum_5,sum_6,sum_7,$prod_5$,$prod_6$,$pord_7$,$prod_8$} where $ov \in$ OV, LS(gv)=$\varnothing$ where $gv \in$ GV, LS(inv)={I_6,I_7,I_8} where $inv \in$ INV. Finally, LV(*SumAndProcd*)=LS(iv)$\bigcup$LS(ov)$\bigcup$LS(gv)$\bigcup$LS(inv). For simplicity, we can describe the live variables of a procedure as Fig. 2.

The live span and live variables of Fig. 1 are illustrated in Fig. 2. For instance, *sum* is firstly referenced in line 4, the last reference is in line 7. According to Definition 1, the software engineer have to keep the reference of *sum* from lines 4 and 7, although *sum* does not appear in statements 5 and 6. Therefore, line 4 to line 7 are the live span of variable *sum* and we use a symbolic "[" to describe the live span of a variable. Together, a live span could clearly be viewed as a visualized scope of live variables in a specific procedure.

Line	LV(*SumAndProd*)	Count	Live Span
4	sum_4	1	sum_4
5	sum_5, $prod_5$	2	sum_5 $prod_5$
6	sum_6, $prod_6$, I_6, N_6	4	sum_6 $prod_6$ I_6 N_6
7	sum_7, $prod_7$, I_7	3	sum_7 $prod_7$ I_7
8	$prod_8$, I_8	2	$prod_8$ I_8
		12	

Fig. 2 Live variables and live span of *SumAndProd*

2 Cohesion Metrics

According to the previous section, we know that not all variables involve the computing of output functions in a specific procedure. Logically, a module is the function that processes input, internal and global variables to produce a result. The output functions in OV(*sp*) are on behalf of the result of the module. In fact, not all elements of live span of a variable contribute to the result of the output function of the procedure. For instance, the LS (*prod*) consists of $prod_5$, $prod_6$, $prod_7$, *and* $prod_8$, in which no element influences the result of output *sum*. But both $prod_5$ and $prod_8$ may involve the value of the output function *prod* through the assignment statement. As of the result, $prod_5$ and $prod_8$ are the function-oriented live variables of output function *prod*. Similarly, variables I_6, I_8, N_6 are the function-oriented live variables of *prod*. Both I_6 and N_6 may change the value of the output function and compute the multiplication via I_8 indirectly. Therefore, function-oriented live variables of *prod* is a set, denoted $\{prod_5, I_6, N_6, prod_8, I_8\}$. Therefore, the function oriented live variables (FOLV) and the function oriented life span (FOLS) of the procedure *SumAndProd* are shown in Fig. 3.

Line	FOLV(*SumAndProd*)	Count	FOLS
4	sum_4	1	sum_4
5	$prod_5$	1	$prod_5$
6	I_6, N_6	2	I_6 N_6
7	sum_7, I_7	2	sum_7 I_7
8	$prod_8$, I_8	2	$prod_8$ I_8
		8	

Fig. 3 The FOLV and FOLS of *SumAndProd*

However, it is not formal enough that we express function-oriented live variables and live span just rely on the above description. Therefore, firstly we will define direct variable and indirect variable of an output function before giving the definitions of function-oriented live variables.

Definition 3. Direct variable of an output function is an element of LV(*sp*), which may perform the result of the output function in the same statement.

For instance, variable I_7 contributes to the value of the sum_7 directly. Thus, I_7 is the direct variable of the output function sum_7. According to Definition 3, output variable surely is the direct variable by itself.

Definition 4. Indirect variable of an output function is an element of LV(*sp*), which may decide the execution of the output function or the result of the output function via direct variables.

For instance, variables I_6 and N_6 of iteration statement in line 6 contribute the value of the sum_7 indirectly. So I_6 and N_6 is the indirect variable of the output function sum_7.

Definition 5. The function-oriented live variables of an output function in the specific procedure is a set, denotes $FOLV(ov_i)=\{lv \in LV(sp) \mid lv$ is a direct or indirect variable of ov_i; $ov_i \in OV(sp)\}$.

Definition 6. The function-oriented live span of an output function in the specific procedure is a set, denotes $FOLS(ov_i)=\{lv \in LS(sp) \mid lv$ is a direct or indirect variable of ov_i; $ov_i \in OV(sp)\}$.

According to Definitions 5 and 6, the function-oriented live variables and function-oriented live span of *sum* and *prod* are depicted in Fig. 4.

Variables	FOLV(*sum*)	FOLV(*sum*)∩FOLV(*prod*)	FOLV(*prod*)
sum	sum_4, sum_7		
I	I_6, I_7	I_6	I_6, I_8
N	N_6	N_6	N_6
prod			$prod_5$, $prod_8$
	5	2	5

Fig. 4 The FOLV of *sum*, *prod* and their intersection

In Fig. 4, the FOLV of the output function *sum* is the set $\{sum_4, I_6, N_6, sum_7, I_7\}$. The size of the FOLV (*sum*) is 5. On the other hand, the FOLV of the output function *prod* is the set of $\{prod_5, I_6, N_6, prod_7, I_8\}$, the size of the FOLV(*sum*) is 5. We know each element in FOLV(*SumAndProd*) will appear either in FOLV(*sum*) or in FOLV(*prod*), which is defined by FOLV(*sum*)∩FOLV(*prod*) as shown in Fig. 3. More importantly, the function-oriented live span can help software engineer to friendly visualize the scope of function oriented live variables in the specific procedure *SumAndProd*.

In general, the size of the FOLV(ov_i) is the number of elements which belonging to LV(*sp*), and influence the value of ovi directly or indirectly. Then, the size of FOLV(ov_i) implies the number of live variables in the specific procedure, which shall contribute to the result of the output function *ov*. More precisely speaking, the proportion of |FOLV(ov_i)| in |LV(*sp*)| can be viewed as the cohesion strength which just restricts on output function ov_i. Therefore, the function-oriented cohesion measure restricts on ov_i is defined as follows:

$$\mathrm{FOCM}(ov_i) = |\mathrm{FOLV}(ov_i)| / |\mathrm{LV}(sp)| \quad \text{for each } ov_i \in \mathrm{OV}(sp).$$

From the perspective of the cohesion, module cohesion aims to measure how tight it is between output procession elements. And, the relation levels of output function pairs determine a cohesion level of a module[10]. Therefore, the common shared function-oriented live variables of *sum* and *prod* are the most critical elements. We define the most tight function-oriented cohesion measure of the specific procedure (*sp*) as follows:

$$\mathrm{MTFOCM}(sp) = |\cap \mathrm{FOLV}(ov_i)| / |\mathrm{LV}(sp)| \quad \text{for each } ov_i \in \mathrm{OV}(sp).$$

In a specific procedure, some elements of live variables may not really involve in influencing the result of output functions. This implies that not all of live variables have function-oriented relation with each *ov*. Obviously, the live variable LV(*sp*) excluding the function-oriented live variable FOLV(*sp*) is a subset of live variables, which is defined as LV(*sp*)-FOLV(*sp*). The subset is the least function-oriented relation with each output function. Therefore, we define the least tight function-oriented cohesion measurement as:

$$\mathrm{LTFOCM}(sp) = |\mathrm{LV}(sp) - (\cup \mathrm{FOLV}(ov_i))| / |\mathrm{LV}(sp)| \quad \text{for each } ov_i \in \mathrm{OV}(sp).$$

However, it is also an important reference point for software measures that the average strength of all live variables involves the results of output functions. We propose the average tight function-oriented cohesion measurement of the specific procedure as follows:

$$\mathrm{ATFOCM}(sp) = |\cup \mathrm{FOLV}(ov_i)| / |\mathrm{LV}(sp)| \quad \text{for each } ov_i \in \mathrm{OV}(sp).$$

Consequently, we proposed four function-oriented cohesion measures in a specific procedure. The values of the four proposed cohesion measures are in the range between 0 and 1. Therefore, the numerical system of the function-oriented cohesion measures is a well normalized cohesion.

3 Experimental and Theoretical Evaluations

In general, there are three validation approaches to validate a new metrics, which are experimental evaluation, theoretical evaluation and empirical evaluation. In this paper, we will adopt experimental evaluation and theoretical evaluation to validate the cohesion metrics.

3.1 Experimental evaluation

In this section, there are six typical cohesion procedure implementations to be given to make an experimental works using our proposed cohesion measures. The purpose of measuring the distinctive cohesion examples is to check the results of experiment of the proposed function-oriented cohesion measurements and to estimate whether they comply with the cohesion strength spectrum and are non-linear scale. Six specific procedure implementations are coincidental, logical, procedural, communicational, sequential, and functional cohesion examples. We believe that the experiments will not affect the completeness of the empirical study though the six implementations will not include the temporal one. A system initialization module can be considered as a typical temporal cohesion.

In general, there are many definitions of the seven levels of cohesion in Refs. [7,17]. A functional cohesion is the strongest cohesion in the cohesion strength spectrum. Therefore, the functional cohesion is the most desirable.

Coincidental cohesion is to estimate whether a module performs more than one function, and whether there are unrelated[7]. In the coincidental cohesion example, there is no significant relationship between the output elements. Similarly, it is hard to describe module purpose. However, a typical coincidental cohesion example and its abstract function diagram are given in Fig. 5.

```
01  Procedure SumAndProd ( N:integer; var sum, prod: integer; arr1,arr2: int array);
02  var I:integer;
03  begin
04    sum:=0;
05    prod:=0;
06    for I:=1 to N do
07      sum:=sum+arr1[I]
08    for I:=1 to N do
09      prod:=prod*arr2[I]
10  end
```

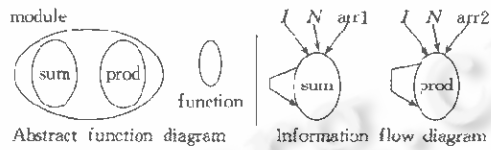


Fig. 5 A typical coincidental cohesion implementation

By definitions 1 and 2, we know LV(*SumAndProd*) = {sum_4, sum_5, $prod_5$, sum_6, $prod_6$, I_6, N_6, sum_7, $prod_7$, $arr1_7$, I_7, $prod_8$, I_8, N_8, $prod_9$, $arr2_9$, I_9}, and |LV(*SumAndProd*)|=17. We express LV and LS of the coincidental cohesion example in Fig. 6. On the other hand, FOLV(*sum*) = {sum_4, I_6, N_6, sum_7, $arr1_7$, I_7}, FOLV(*prod*) = {$prod_5$, I_8, N_8, $prod_9$, $arr2_9$, I_9}. They are shown in Fig. 7. The FOLV of *sum*, *prod* and their interseection are illustrated in Fig. 8.

Line	LV(*SumAndProd*)	Count	LS
4	sum_4	1	
5	sum_5, $prod_5$	2	
6	sum_6, $prod_6$, I_6, N_6	4	
7	sum_7, $prod_7$, $arr1_7$, I_7	4	
8	$prod_8$, I_8, N_8	3	
9	$prod_9$, $arr2_9$, I_9	3	
		17	

Fig. 6 The LV, LS of a coincidental cohesion example

Line	FLOV(*SumAndProd*)	Count	FOLS
4	sum_4	1	sum_4
5	$prod_5$	1	$prod_5$
6	I_6, N_6	2	sum_7 I_6 N_6
7	$sum_7, arr1_7, I_7$	3	I_7 $arr1_7$
8	I_8, N_8	2	I_8 N_8
9	$prod_9, arr2_9, I_9$	3	$prod_9$ I_9 $arr2_9$
		12	

Fig. 7 The FOLV and FOLS of a coincidental cohesion example

Variables	FOLV(*sum*)	FOLV(*sum*)∩FOLV(*prod*)	FOLV(*prod*)
sum	sum_4, sum_7		
I	I_6, I_7		I_8, I_9
N	N_6		N_8
arr1	$arr1_7$		
arr2			$arr2_9$
prod			$prod_5, prod_9$
	6	0	6

Fig. 8 The FOLV of *sum*, *prod* and their intersection

According to the function-oriented cohesion measures as expressed in Section 2, we know FOCM(*sum*) =0.353, FOCM(*prod*) = 0.353, MTFOCM(*SumAndProd*) = 0, LTFOCM(*SumAndProd*) = 0.29, and AVFOCM(*SumAndProd*)=0.71. In this example, there are no common processing elements used to produce both output functions. And, the value of MTFOCM of the coincidental cohesion is equal to zero. The Coincidental cohesion type is the lowest cohesion level.

Logical cohesion is to check whether the module performs more than one function, and whether there are related logically[7]. In a logical cohesion example, the module performs some related functions. One or more of them are selected by calling module. However, a typical logical cohesion example and its abstract function diagram are shown as follow.

```
01  Procedure SumAndProd ( N:integer; var sum, prod: integer; arr1,arr2: int_array);
02  var I, J ,flag:integer;
03  begin
04    sum:=0;
05    prod:=1;
06    if (flag>1);
07      for I :=1 to N do begin
08      sum:=sum+arr1[ I]
09    else
10      for I :=1 to N do begin
11      prod:=prod* arr2[ I]
12  end
```

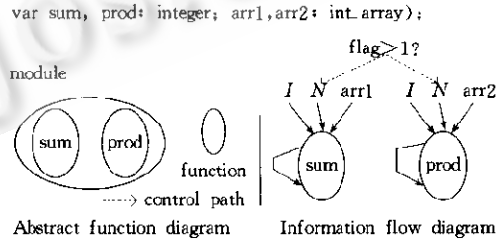


Fig. 9 A typical logical cohesion implementation

By definitions 1 and 2, LV(*SumAndProd*)={$sum_4, sum_5, prod_5, sum_6, prod_5, flag_6, sum_7, prod_7, I_7, N_7, sum_8, prod_8, arr1_8, I_8, N_8, prod_9, I_9, N_9, prod_{10}, I_{10}, N_{10}, prod_{11}, arr2_{11}, I_{11}$}, and |LV(*SumAndProd*)|=24, which are diagramed in Fig. 10. In other way, we may have FOLV(*sum*)={$sum_4, flag_6, I_7, N_7, sum_8, arr1_8, I_8$} and |FOLV(*sum*)|=7, FOLV(*prod*)={$prod_5, flag_6, I_9, N_9, prod_{10}, arr1_{10}, I_{10}$} and |FOLV(*prod*)|=7, and they are shown in Fig. 11. The FOLV of *sum*, *prod* and their intersection are illustrated in Fig. 12.

Line	LV(*SumAndProd*)	Count	LS
4	sum_4	1	sum_4
5	sum_5, $prod_5$	2	sum_5 $prod_5$
6	sum_6, $prod_6$, $flag_6$	3	sum_6 $prod_6$ $flag_6$
7	sum_7, $prod_7$, I_7, N_7	4	sum_7 $prod_7$ I_7 N_7
8	sum_8, $prod_8$, $arr1_8$, I_8, N_8	5	sum_8 $prod_8$ I_8 $arr1_8$ N_8
9	$prod_9$, I_9, N_9	3	$prod_9$ I_9 N_9
10	$prod_{10}$, I_{10}, N_{10}	3	$prod_{10}$ I_{10} N_{10}
11	$prod_{11}$, $arr2_{11}$, I_{11}	3	$prod_{11}$ I_{11} $arr2_{11}$
		24	

Fig. 10 The LV and LS of a logical cohesion example

Line	FOLV(*SumAndProd*)	Count	FOLS
5	sum_4	1	sum_4
6	$prod_5$	1	$prod_5$
7	$flag_6$	1	$flag_6$
8	I_7, N_7	2	I_7 N_7
9	sum_8, $arr1_8$, I_8	3	sum_8 I_8 $arr1_8$
10		0	
11	I_{10}, N_{10}	2	I_{10} N_{10}
12	$prod_{11}$, $arr2_{11}$, I_{11}	3	$prod_{11}$ I_{11} $arr2_{11}$
		13	

Fig. 11 The FOLV and FOLS of a logical cohesion example

Variables	FOLV(*sum*)	FOLV(*sum*)∩FOLV(*prod*)	FOLV(*prod*)
sum	sum_4, sum_8		
I	I_7, I_8		I_{10}, I_{11}
N	N_7		N_{10}
flag	$flag_6$	$flag_6$	$flag_6$
arr1	$arr1_8$		
arr2			$arr2_{11}$
prod			$prod_5$, $prod_{11}$
	7	1	7

Fig. 12 The FOLV of *sum*, *prod* and their intersection

According to the definitions of function-oriented cohesion measures, the values of function-oriented cohesion measures of logical cohesion example are FOCM(*sum*)=0.29, FOCM(*prod*)=0.29, MTFOCM(*SumAndProd*)=0.041, LTFOCM(*SumAndProd*)=0.46, and ATFOCM(*SumAndProd*)=0.64. In this example, there are some common elements used to decide which function to be requested logically. The value of MTFOCM of the logical cohesion is equal to 0.041. From the viewpoint of cohesion strength, this logical cohesion is just a little stronger than coincidental cohesion.

Procedural cohesion is to evaluate whether the module performs more than one function, and whether the module is related to a general procedural affected by the software[7]. This means that if each function in the module needs to execute following in a specific order, it has the strong procedure cohesion. Explicitly, both functions are elements of some iteration or decision operations[18]. However, a typical procedural cohesion example and its abstract function diagram are given as follows:

By definitions 1 and 2, we know LV(*SumAndProd*) = {sum_4, sum_5, $prod_5$, sum_6, $prod_6$, I_6, N_6, sum_7, $arr1_7$, $prod_7$, I_7, $prod_8$, $arr2_8$, I_8} and |LV(*SumAndProd*)|=14, we express LV and LS of the procedural cohesion

```
01  Procedure SumAndProd ( N:integer; var sum, prod: integer; arr1,arr2: int_array);
02  var I :integer;
03  begin
04    sum:=0;
05    prod:=1;
06    for I:=1 to N do
07      sum:=sum+arr1[I]
08      prod:=prod*arr2[I]
09  end
```

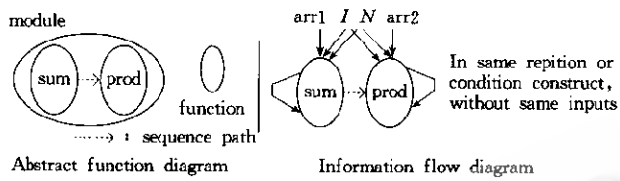


Fig. 13 A typical procedural cohesion implementation

example in Fig. 14. And, we may have FOLV(*sum*)={$sum_4, I_6, N_6, sum_7, arr1_7, I_7$} with size of 6, FOLV(*prod*)={$prod_5, I_6, N_6, prod_8, arr2_8, I_8$} with size of 6, and their visualized FOLS is displayed in Fig. 15. The FOLV of *sum*, *prod* and their intersection are presented in Fig. 16.

Line	LV(*SumAndProd*)	Count	LS
4	sum_4	1	sum_4
5	$sum_5, prod_5$	2	sum_5 $prod_5$
6	$sum_6, prod_6, I_6, N_6$	4	sum_6 $prod_6$ I_6 N_6
7	$sum_7, prod_7, arr1_7, I_7$	4	sum_7 $prod_7$ I_7 $arr1_7$
8	$prod_8, arr2_8, I_8$	3	$prod_8$ I_8 $arr2_8$
		14	

Fig. 14 The LV and LS of a procedure cohesion example

Line	FOLV(*SumAndProd*)	Count	FOLS
4	sum_4	1	sum_4
5	$prod_5$	1	$prod_5$
6	I_6, N_6	2	I_6 N_6
7	$sum_7, arr1_7, I_7$	3	sum_7 I_7 $arr1_7$
8	$prod_8, arr2_8, I_8$	3	$prod_8$ I_8 $arr2_8$
		10	

Fig. 15 The FOLV and FOLS of a procedure cohesion example

By the definition of function-oriented cohesion measures, we can derive the values of function-oriented cohesion measures of coincidental cohesion example, they are FOCM(*sum*)=0.428, FOCM(*prod*)=0.428, MTFOCM(*SumAndProd*)=0.143, LTFOCM(*SumAndProd*)=0.288, and ATFOCM(*SumAndProd*)=0.712. In this example, a number of elements are involved in different activities. But the activities are sequential. The value of MTFOCM of the procedural cohesion is equal to 0.142. From the cohesion strength perspective, procedural cohesion is a little stronger than logical cohesion.

Variables	FOLV(*sum*)	FOLV(*sum*)∩FOLV(*prod*)	FOLV(*prod*)
sum	sum_4, sum_7		
I	I_6, I_6	I_6	I_6, I_8
N	N_6	N_6	N_6
arr1	$arr1_7$		
arr2			$arr2_8$
prod			$prod_5, prod_8$
	6	2	6

Fig. 16 The FOLV of *sum*, *prod* and their intersection

Communicational cohesion is to indicate whether the module performs more than one function, and whether the module is on the same data[7]. This means that whether every function in the module operates on the same data. However, a typical communicational cohesion example and its abstract function diagram are given as follows:

```
01 Procedure SumAndProd ( N:integer; var sum, prod: integer; arr1: int_array);
02 var I:integer;
03 begin
04   sum:=0;
05   prod:=1;
06   for I:=1 to N do begin
07     sum:=sum+arr1[I]
08     prod:=prod*arr1[I]
09   end
10 end
```

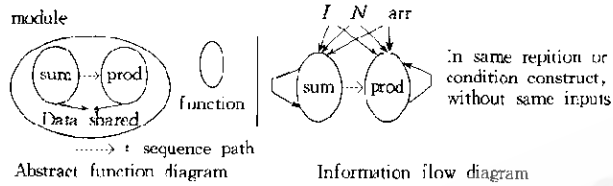


Fig. 17 A typical communicational cohesion implementation

According to definitions 1 and 2, we know LV(*SumAndProd*) = {sum_4, sum_5, $prod_5$, sum_6, $prod_6$, I_6, N_6, sum_7, $prod_7$, $arr1_7$, I_7, $prod_8$, $arr1_8$, I_8} and |LV(*SumAndProd*)| = 14, which are depicted in Fig. 18. Again, FOLV(*sum*) = {sum_4, I_6, N_6, sum_7, $arr1_7$, I_7} with size of 6, FOLV(*prod*) = {$prod_5$, I_6, N_6, $prod_8$, $arr1_8$, I_8} with size of 6, and their visualized FOLS is sketched in Fig. 19. The FOLV of *sum*, *prod* and their intersection are illustrated in Fig. 20.

Line	LV(*SumAndProd*)	Count	LS
4	sum_4	1	sum_4
5	sum_5, $prod_5$	2	sum_5 $prod_5$
6	sum_6, $prod_6$, I_6, N_6	4	sum_6 $prod_6$ I_6 N_6
7	sum_7, $prod_7$, $arr1_7$, I_7	4	sum_7 $prod_7$ I_7 $arr1_7$
8	$prod_8$, $arr1_8$, I_8	3	$prod_8$ I_8 $arr1_8$
		6	

Fig. 18 The LV and LS of a communicational cohesion example

Line	FLOV(*SumAndProd*)	Count	FOLS
4	sum_4	0	sum_4
5	$prod_5$	1	$prod_5$ I_6 N_6
6	I_6, N_6	2	sum_7 I_7 $arr1_7$
7	sum_7, $arr1_7$, I_7	0	$prod_8$ I_8 $arr1_8$
8	$prod_8$, $arr1_8$, I_8	3	
6		6	

Fig. 19 The FOLV and FOLS of communicational cohesion example

Variables	FOLV(sum)	FOLV(*sum*)∩FOLV(*prod*)	FOLV(*prod*)
sum	sum_4, sum_7		
I	I_6, I_7	I_6	I_6, I_8
N	N_6	N_6	N_6
arr1	$arr1_7$		$arr1_8$
prod			$prod_5$, $prod_8$
	6	2	6

Fig. 20 The FOLV of *sum*, *prod* and their intersection

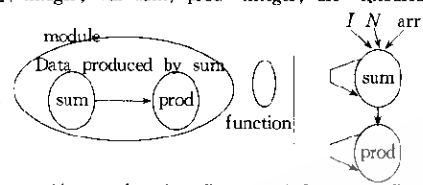
However, the values of function-oriented cohesion measures of coincidental cohesion example are FOCM(*sum*) = 0.428, FOCM(*prod*) = 0.428, MTFOCM(*SumAndProd*) = 0.142, LTFOCM(*SumAndProd*) = 0.288, and AVFOCM(*SumAndProd*) = 0.712.

In this example, a number of elements are involved in different activities, but the activities are sequential. The value of MTFOCM of communicational cohesion is equal to 0.142. That is not significantly stronger cohesion strength than procedural cohesion.

Sequential cohesion is to estimate whether the module performs more than one function. Function dependency occurs in an order, which is described in the specification[7]. Generally, the output data from a function is the input

for the next function in a module. However, a typical sequential cohesion example and its abstract function diagram are given as follows:

```
01  Procedure SumAndProd ( N:integer; var sum, prod: integer; arr: int_array);
02  var I :integer;
03  begin
04    sum:=0;
05    prod:=1;
06    for I:=1 to N do begin
07      sum:=sum+arr[I];
08      prod:=prod*sum;
09  end
```



Abstract function diagram Information flow diagram

Fig. 21 A typical sequential cohesion implementation

By the definition of live variables and function oriented live variables, we have LV(*SumAndProd*) = {sum_4, sum_5, $prod_5$, sum_6, $prod_6$, I_6, N_6, sum_7, $prod_7$, arr_7, I_7, $prod_8$, sum_8} and |LV(*SumAndProd*)| = 13, We express LV and LS of the sequential cohesion example in Fig. 22. FOLV(*sum*) = {sum_4, I_6, N_6, sum_7, arr_7, I_7} with size of 6, FOLV(*prod*) = {sum_4, $prod_5$, I_6, N_6, sum_7, I_7, $prod_8$, sum_8} with size of 9, and their visualized FOLS are depicted in Fig. 23. The FOLV of *sum*, *prod* and their intersection are illustrated in Fig. 24.

Line	LV(*SumAndProd*)	Count	LS
4	sum_4	1	sum_4
5	sum_5, $prod_5$	2	sum_5 $prod_5$
6	sum_6, $prod_6$, I_6, N_6	4	sum_6 $prod_6$ I_6 N_6
7	sum_7, $prod_7$, arr_7, I_7	4	sum_7 $prod_7$ I_7 arr_7
8	$prod_8$, sum_8	2	sum_8 $prod_8$
		13	

Fig. 22 The LV and LS of a sequential cohesion example

Line	FOLV(*SumAndProd*)	Count	FOLS
4	sum_4	1	sum_4
5	$prod_5$	1	$prod_5$
6	I_6, N_6	2	I_6 N_6
7	sum_7, arr_7, I_7	3	sum_7 I_7 arr_7
8	$prod_8$, sum_8	2	sum_8 $prod_8$
6		6	

Fig. 23 The FOLV and FOLS of sequential cohesion example

Variables	FOLV(*sum*)	FOLV(*sum*) ∩ FOLV(*prod*)	FOLV(*prod*)
sum	sum_4, sum_7	sum_4, sum_7	sum_4, sum_7, sum_8
I	I_6, I_7	I_6, I_7	I_5, I_7
N	N_6	N_6	N_6
arr	arr_7	arr_7	arr_7
prod			$prod_5$, $prod_8$
	6	6	9

Fig. 24 The FOLV of *sum*, *prod* and their intersection

According to the definitions of function-oriented cohesion measures, the values of function-oriented cohesion measures of Sequential cohesion example are FOCM (*sum*) = 0.46, FOCM (*prod*) = 0.69, MTFOCM (*SumAndProd*) = 0.46, LTFOCM (*SumAndProd*) = 0.31, and AVFOCM (*SumAndProd*) = 0.69. In this example, some elements involved in different activities. But the activities are sequential. The value of MTFOCM of the sequential cohesion is equal to 0.46. The values show that sequential cohesion is a little stronger than communicational cohesion.

Functional cohesion is to check whether the module performs on a single function[7]. The module is the one on which all of the elements contribute to exactly one function. However, a typical functional cohesion example and its abstract function diagram are given as follows:

```
01  Procedure Sum ( N:integer; var sum: integer; arr: int-array);
02  var I:integer;
03  begin
04  sum:=0;
05  for I:=1 to N do begin
06  sum:=sum+arr[I]
07  end
```

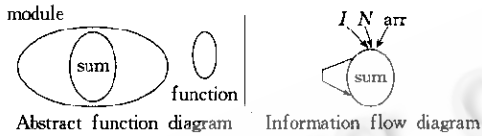


Fig. 25 A typical functional cohesion implementation

Rely on definitions 1 and 2, it is easy to know that LV(*sum*)={sum_4, sum_5, I_5, N_6, sum_5, arr_5, I_6} with size of 7 and FOLV(*sum*)={sum_4, I_5, N_5, sum_6, arr_6, I_6} with size of 6. Their visualized LS and FOLS are developed in Fig. 26 and Fig. 27 respectively. However, there is a single output function in the functional cohesion implementation. In Fig. 28, the FOLV of the single output function *sum* are depicted.

Line	LV(*SumAndProd*)	Count	LS
4	sum_4	1	sum_4
5	sum_5, I_5, N_5	3	sum_5 [I_5 N_5]
6	sum_6, arr_6, I_6	3	sum_6 [I_6 arr_6]
		7	

Fig. 26 The LV of a functional cohesion

Line	FOLV(*SumAndProd*)	Count	FOLS
4	sum_4	1	sum_4
5	I_5, N_5	2	sum_6 [I_5 N_5]
6	sum_6, arr_6, I_6	3	[I_6 arr_6]
		6	

Fig. 27 The FOLV of functional cohesion

Variables	FOLV(sum)
sum	sum_4, sum_6
I	I_5, I_6
N	N_5
arr	arr_6
	6

Fig. 28 The FOLV of an output function *sum*

However, the values of function-oriented cohesion measures of functional cohesion example are FOCM(*sum*) = 0.86, MTFOCM(*sum*) = 0.86, LTFOCM(*sum*) = 0.143, and ATFOCM(*sum*) = 0.857. The value of MTFOCM of the functional cohesion is 0.86. Functional cohesion is a stronger cohesion strength than sequential cohesion. The Functional cohesion is the highest cohesion level.

3.2 Analysis of experimental results

Cohesion of a module measures the strength of the relationship between the elements within the module; good modules should be highly bound[17]. Cohesion can be evaluated on a one-dimensional scale[7]. A scale of several points can identify different types of cohesion and give their associated strength[17]. According to SMC Cohesion level and Fenton's cohesion spectrum, we know that the cohesion strength spectrum can be illustrated as follows:

In the previous section, we know the values of the cohesion of all experiments are in [0,1]. This also implies that the proposed cohesion measures are well-normalized. In particular, the values of MTFOCM on the six typical cohesion examples from coincidental to functional are increasing between 0 and 1. In the other words, the cohesion strength of coincidental is the weakest type, but the functional cohesion strength is the strongest type. According to the above experiments, there is not significant difference between the cohesion strengths of procedure cohesion and communication cohesion. More importantly, MTFOCM does match the SMC Cohesion and Fenton's cohesion strength spectrum. The quantitative values of the levels of cohesion in the MTFOCM experiments are sketched in Fig. 30.

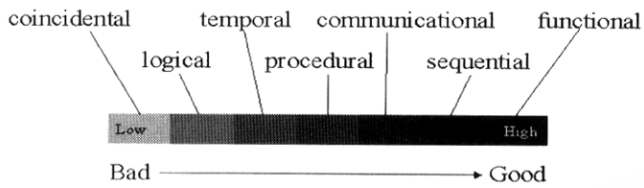


Fig. 29 Cohesion strength spectrum

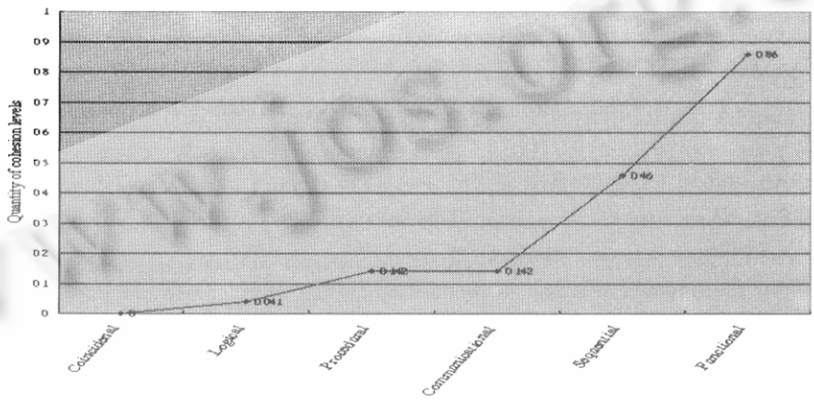


Fig. 30 The non-linear curve of cohesion levels

However, the curve of the quantitative values of MTFCOM shows that low-end cohesiveness is worse than middle-range. The value of MTFOCM of logical cohesion is just 0.041. This implies that both coincidental and logical cohesion may be undesirable cohesion levels. The value of MTFOCM of procedural or communication cohesion is 0.142. And, the curve of cohesion strength rises slowly from logical type to procedural or communication type. This means that procedural and communication are low-end cohesion levels. The curve of cohesion strength climbs up sharply at the point of procedural or communication. This means that sequential cohesion is a high-end cohesion type. The cohesion strength soars from sequential to functional. Therefore, the functional cohesion should be the most desirable cohesion type. Furthermore, the nonlinear scale from coincidental cohesion to functional cohesion consists with the assertions of Pressman and Somerville in their software engineering articles.

According to the empirical experiments, the proposed cohesion measures have some paradigmatic characteristics: (1) $MTFOCM(sp) \leqslant FOCM(ov_i) \leqslant ATFOCM(sp)$ for each ov_i, (2) $LTFOCM(sp)+ATFOCM(sp)=1.0$.

From the viewpoint of software quality, reusability, and maintainability, coincidental and logical cohesion are hard to describe the module purpose. Logical cohesion has more than one action to be intertwined logically, which is not only difficult to understand but also difficult to maintain and reuse. Procedural cohesion performs a series of sequential actions with weak connection. Communicational cohesion is a similar procedural cohesion when performing on the same data, and may be a little better than procedural cohesion for software quality. The cohesion strength of sequential cohesion is much stronger than communicational or procedural cohesion. However, functional cohesion accomplishes a single specific action, which can more readily be reused in a variety of situations. This type of cohesion is easier to understand and maintain. In practice, software engineer may not concern the cohesion categories in a specific procedure. Rather, the cohesion concepts should be realized and low-end cohesion should be avoided when software is designed.

3.3 Theoretical evaluation

Validation of a software measure is "the process of ensuring that the measure is a proper numerical characterization of the claimed attribute"[19]. Again, a software measurement should be based on an essential scientific basis[20]. In this section, we adopt Property Cohesion as a theoretical evaluation framework, which is proposed by Brians *et al.*[1], and the Property Cohesion usually can guide the development of new metrics. Therefore, it is necessary that we take the proposed cohesion measures to validate against Property Cohesion. The Property Cohesion is considered as follows:

• Property Cohesion. 1: Non-negativity and Normalization

This means that the cohesion of a module belongs to a specified interval, and normalization allows meaningful comparisons between the cohesion of different modules.

• Property Cohesion. 2: Null value

This means that if there is no intra-module relationship among the elements of a module, the module cohesion is null.

• Property Cohesion. 3: Monotonicity

This means that adding intra-module relationships does not decrease module cohesion.

• Property Cohesion. 4: Cohesion Modules

This means that the cohesion of a module obtained by putting together two unrelated modules is not greater than the maximum cohesion of the two original modules.

The Property Cohesion characterizes and formalizes intuitively cohesion measurement concept. However, it is essential to validate a new proposed module cohesion measurement by verifying the four properties, we will take the proposed function-oriented cohesion measurements to be evaluated against them. The Property Cohesion will make the proposed cohesion measures more rigorous. Additional, the Property Cohesion provides necessary but not sufficient conditions for cohesion measures[19].

Proposition 1. All of the proposed function oriented cohesion measurements are Non-negativity and Normalization.

Proof. For a given specific procedure (sp), the set of the output variables of the sp is OV. By definitions 2 and 5, $FOLV(ov_i)\subseteq LV(sp)$ hold for each $ov_i\subseteq OV$. At first, $FOCM(ov_i)=|FOLV(ov_i)|/|LV(sp)|\in[0,1]$ since $|FOLV(ov_i)|\leqslant|LV(sp)|$. Next, since $\bigcap FOLV(ov_i)\subseteq FOLV(ov_i)\subseteq LV(sp)$ for each $ov_i\in OV$. Hence $MTFOCM(sp)=|\bigcup FOLV(ov_i)|/|LV(sp)|\in[0,1]$. And, since $\bigcup FOLV(ov_i)\subseteq LV(sp)$ for each $ov_i\in OV$. Hence $ATFOCM(sp)=|\bigcup FOLV(ov_i)|/|LV(sp)|\in[0,1]$. Finally, $LV(sp)\ \bigcup FOLV(ov_i)\subseteq LV(sp)$ hold for each $ov_i\in OV$. Hence $LTFOCM(sp)=|LV(sp)-\bigcup FOLV(ov_i)|/|LV(sp)|\in[0,1]$. Therefore, all of the proposed function-oriented cohesion measurements are Non-negativity and Normalization. Especially, the tightest function-oriented cohesion of the specific procedure is Non-negativity and Normalization.

Proposition 2. If there is no intra module relationship among the elements of a module, the function-oriented cohesion is null.

Proof. For a given specific procedure (*sp*), let OV be the set of the output variables of the procedure and $|OV(sp)|=n$. Suppose there is no intra-module relationship among the elements of a module, this implies that there are no processing elements to involve in the computing of each output function. By definitions 2 and 5, the FOLV of each output variable is empty and the intersection of any two FOLV is empty. By induction, if $n=1$ then $|FOLV(ov)|=0$ (i.e. $FOLV(ov)=\varnothing$) hold for $ov\in OV$. If $n=k$ and $|FOLV(ov_i)|=0$ (i.e. $FOLV(ov_i)=\varnothing$) for each $ov_i\in OV$ where $1\leqslant i\leqslant k$ then $|\bigcap_{i=1..k}FOLV(ov_i)|=0$ and $|\bigcup_{i=1..k}FOLV(ov_i)|=0$ hold. Now, suppose $n=k+1$ and $|FOLV(ov_i)|=0$ (i.e. $FOLV(ov_i)=\varnothing$) for each $ov_i\in OV$ where $1\leqslant i\leqslant k+1$, to show $|\bigcap_{i=1..k+1}$

$FOLV(ov_i)|=0$ and $|\bigcup_{i=1..k+1} FOLV(ov_i)|=0$ hold. Since $\bigcap_{i=1..k+1} FOLV(ov_i)=(\bigcap_{i=1..k} FOLV(ov_i))\cap(FOLV(ov_{k+1}))$ $=\varnothing\cap\varnothing=\varnothing$, we know $|\bigcap_{i=1..k-1} FOLV(ov_i)|-0$. On the other hand, $|\bigcup_{i=1..k+1} FOLV(ov_i)|=|(\bigcup_{i=1..k} FOLV(ov_i))$ $\cup(FOLV(ov_{k+1}))|=|(\bigcup_{i=1..k} FOLV(ov_i))|+|(FOLV(ov_{k+1}))|-|(\bigcup_{i=1..k} FOLV(ov_i))\cup(FOLV(ov_{k+1}))|=0$. Therefore, the values of FOCM, MTFOCM and ATFOCM are zero. This implies that function-oriented cohesion FOCM, MTFOCM and ATFOCM are null values.

Proposition 3. Adding intra-module relationships does not decrease module function-oriented cohesion.

Proof. Let sp with n output variables be a specific procedure and $|LV(sp)|$ be the size of live variable of sp. Suppose $L=|\bigcap FOLV(ov_i)|$, $J_i=|FOLV(ov_i)|$, $M=|LV(sp)|$, $M\geqslant L$ and $M\geqslant J_i$.

Case 1: There are k elements from original $LV(sp)$ which contribute to the computing of ov_i. It is trivial that $(L)/(M)\leqslant(L+k)/(M)$ and $(J_i)/(M)\leqslant(J_i+k)/(M)$ where $L=|\bigcap FOLV(ov_i)|$ or $J_i=|FOLV(ov_i)|$, $M=|LV(sp)|$ and $M\geqslant L$.

Case 2: There are k elements other than original LV(sp) which contribute to the computing of ov_i. To show that $(L)/(M)\leqslant(L+k)/(M+k)$. We know $(L+k)/(M+k)-(L)/(M)=((ML+Mk)-(ML+Lk))/((M+k)M)=((Mk)-(Lk))/((M+k)M)=((M-L)k)/((M+k)M)\geqslant 0$. Thus $(L)/(M)\leqslant(L+k)/(M+k)$. Similarly, we have $(J_i)/(M)\leqslant(J_i+k)/(M+k)$. Thus, FOCM, MTFOCM and ATFOCM are monotonicity.

Proposition 4. The cohesion of a module obtained by putting together two unrelated modules is not greater than the maximum cohesion of the two original modules.

Proof. Let OV_1 and OV_2 be output set of two unrelated specific procedure sp_1 and sp_2 respectively, and $OV_1\cap OV_2=\varnothing$. Let $sp_1\oplus sp_2$ be the procedure with output set $OV_1\cup OV_2$, that is obtained by putting together two unrelated module sp_1 and sp_2. Suppose $|LV(sp_1)|=M_1$, $|LV(sp_2)|=M_2$, $L_1=|\bigcap FOLV(ov_{1i})|=FOLV(sp_1)$ and $L_2=|\bigcap FOLV(ov_{2j})|=FOLV(sp_2)$. We know $MTFOCM(sp_1)=L_1/M_1$ and $MTFOCM(sp_2)=L_2/M_2$. Since some processing elements of $FOLV(sp_1)$ and $FOLV(sp_2)$ may commonly involved the computing of output function in $OV_1\cup OV_2$. So the intersection of $FOLV(sp_1)$ and $FOLV(sp_2)$ may be not empty. This can imply that $|FOLV(sp_1\oplus sp_2)|=|\bigcap FOLV(ov_k)|=|FOLV(sp_1)\cap FOLV(sp_2)|=n$, $0\leqslant n\leqslant L_1\leqslant M_1$ and $0\leqslant n\leqslant L_2\leqslant M_2$ for each $ov_k\in OV_1\cup OV_2$. To show that $MTFOCM(sp_1\cap sp_2)\leqslant\max(MTFOCM(sp_1),MTFOCM(sp_2))$. Since $MTFOCM(sp_1\oplus sp_2)=|FOLV(sp_1\oplus sp_2)|/|LV(sp_1\oplus sp_2)|=(n)/(M_1+M_2-n)\leqslant(n)/(M_1)\leqslant(L_1)/(M_1)$. Similarly, $(n)/(M_1+M_2-n)\leqslant(n)/(M_2)\leqslant(L_2)/(M_2)$. Thus $MTFOCM(sp_1\oplus sp_2)\leqslant\max(MTFOCM(sp_1),MTFOCM(sp_2))$. On the other hand, since $OV_1\cap OV_2=\varnothing$, $FOCM(ov_k)=|FOLV(ov_k)|/|LV(sp_1\oplus sp_2)|=|FOLV(ov_k)|/(M_1+M_2-n)\leqslant|FOLV(ov_k)/(M_1)\leqslant|FOLV(ov_k)|/|LV(sp_1)|$ for each $ov_k\in OV_1$, $\notin OV_2$. Likewise, $FOCM(ov_k)=|FOLV(ov_k)|/|LV(sp_1\oplus sp_2)|=|FOLV(ov_k)|/(M_1+M_2-n)\leqslant|FOLV(ov_k)|/(M_2)\leqslant|FOLV(ov_k)/|LV(sp_2)|$ for each $ov_k\in OV_2$, $\notin OV_1$. Thus, it is clear that $FOCM(ov_k)\leqslant\max(FOCM(ov_k),FOCM(ov_k))$ for each $ov_k\in OV_1$ or $\notin OV_2$. Therefore, FOCM and MTFOCM hold the Cohesion Modules property.

The proposed function-oriented measures have been shown to preserve the Property Cohesion. Especially, The MTFOCM restrictedly hold Properties Cohesion. 1 ~ Cohesion. 4. Therefore, there is no contradiction between the proposed function-oriented cohesion measurements and the definition of the cohesion measures on a ratio scale.

4 Related works and Comparisons

In this section, we describe the related works and bring the proposed metrics to compare with them. We provide a list of criteria to establish an objective and a meaningful comparison of cohesion measures. A number of important criteria of comparison are explained below.

• Quantitative definition: indicate if the original definition of the measure is able to quantify software cohesion

or not.

- Well normalized: check if the measure is well normalized.
- Language independence: the measure is not language specific, and is applicable to other language.
- Experimental evaluation: whether the measure to be checked using well designed experimental examples.
- SMC Cohesion: indicate if the experimental result preserves the SMC Cohesion spectrum.
- Non-linear scale: show whether the curve of the quantity of the experimental result is a non-linear scale.
- Theoretic evaluation: whether there is a solid theoretic evaluation to validate the measures and hold Property Cohesion.

According to the previous sections, we have proposed the well-normalized function-oriented cohesion metrics with quantitative definition. The experimental evaluation is to give typical and well-designed cohesion examples to examine the metrics. And the experimental result shows that the metrics we proposed does not only comply with SMC Cohesion, but is non-linear cohesion scale. More importantly, the theoretical evaluation shows that the function-oriented cohesion metrics hold the Property Cohesion.

In the past decade, a few researchers work on cohesion measures. Ott and Thuss's approach based on slices profile and processing element flow graph to express four types of module cohesion[9]. Therefore, this approach does not consist with SMC Cohesion. And, this method does not provide a quantitative definition to perform module cohesion. In Ref. [11], Biemen and Ott proposed a program slicing based cohesion measures. The metrics provides a numerical system, and there is a set of well-designed examples to be provided and to be experimented. The mechanism is a quantitative definition, language independence and well normalized. But the approach seems not to be checked against Property Cohesion with a theoretical evaluation. For complex program functionality, the program slicing method can produce slices that are either too large to understand or too simple but not significant to comprehend[21]. Lakhotia developed a method to investigate cohesion types based on logical analysis[18]. The approach is language independent. Although the result of the experimental evaluation of the logical analysis meets the SMC Cohesion levels logically, does not provide a quantitative definition to compute module cohesion strength and theoretically check against Property Cohesion. Basili *et al.*[23] proposes a high-level cohesion metrics based on Data declaration-Data declaration (DD) interaction graph of Ada. The metrics is a quantitative definition, is well normalized, and provides a theoretical evaluation to check against Property Cohesion, but this approach is language dependent and does not provide an experimental evaluation.

5 Conclusions

Cohesion is an important attribute corresponding to the quality of software captured by the module under consideration. Researchers constantly attempt to develop meaningful software measures, and practitioners have strong willing to introduce suitable software metrics as tools to control software quality. But, software measurement must be well-grounded in theory and be validated through well-evaluated works. In this paper, we proposed function-oriented cohesion measures based on the function-oriented live variables and the visualized live span semantic analysis models. We gave experimental and theoretical evaluations to validate the cohesion metrics. The experimental evaluation is to examine the metrics using a set of well-designed cohesion examples. The theoretical evaluation is to check the cohesion metrics against Property Cohesion carefully. According to the result of the validation system, the proposed metrics does not only meet SMC Cohesion and match a nonlinear scale of cohesion, but also preserve the Property Cohesion.

The major contribution of this paper is that we have proposed objective, robust, well-normalized, well-evaluated, language independent, easy measure and algorithmically function oriented cohesion metrics to improve software quality. Moreover, it is note worthy that the cohesion metrics can be easily incorporated into software

CASE tool to enhance software development and quality. Furthermore, it is one of several possible applications to introduce the proposed cohesion metrics into the reverse engineering paradigm to extract higher quality software components from legacy systems.

References:

[1] Brians, L., Moeasc, S., Basili, V. R. Property-Based software engineering measurement. IEEE Transactions on Software Engineering, 1996,22(1).

[2] DeMarco, T. Controlling Software Projects. New York: Prentice Hall, 1982.

[3] Martin, J., McClure, C. Software Maintenance the Problem and Its Solutions. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1983.

[4] http//osiris. sunderland. ac. uk/~cs0hed/campbell/chap~7. html.

[5] Yourdon, E., Constantinc, L. L. Structured Design. Prentice Hall, 1979.

[6] Stevens, W. G., Myers, Constantine, L. Structured design. IBM Systems Journal, 1974,13(2):115~139.

[7] Fenton, N. E. Software Metrics: A Rigorous Approach. London: Chapman & Hall, 1991.

[8] Conte, S. D., Dunsmore, H. E., Shen, V. Y. Software Engineering Metrics and Models. The Benjamin/Cummings Publishing Company, Inc., 1986.

[9] Ott, L. M., Thuss, J. J. The relationship between slices and modules cohesion. In: Proceedings of the 12th International Conference on Software Engineering (ICSE-12). 1989.

[10] Bieman, J. M. Kang, B. K. Measureing design-level cohesion. IEEE Transactions on Software Engineering, 1998,24(2).

[11] Bieman, J. M., Ott, L. M. Measuring functional cohesion. IEEE Transactions on Software Engineering, 1994.

[12] Pressman, R. S. Software Engineering: A Practitioners Approach. McGraw-Hill International Editions, 1988.

[13] Sommerville. Software Engineering. 5th ed. Addison-Wesley, 1996.

[14] Baker, A. L., Bieman, J. M., Fenton, N., *et al*. A philosophy for software measurement. Journal of System and Software, 1990,12:277~281.

[15] Canfora, G., Lakhotia, A. Program comprehension. Journal of Systems and Software, 1999,44:79~80.

[16] Curtis, B. Human Factors in Software Development. Silver Spring, MD: Computer Society Press, 1981. 170~179.

[17] http://www.dis. unimelb. edu. au/staff/jacob/lectures/csdl/modularity/tsld013. html.

[18] Lakhotia, A. Rule-based approach to computing module cohesion. In: Proceedings of the 15th International Conference Software Engineering (ICSE-15). 1993.

[19] Gupta, B. S. A critique of cohesion measures in the object-oriented paradigm[MS Thesis]. Michigam Technological University, 1997.

[20] Fenton, N. E. Software measurement: a necessary scientific basis. IEEE Transactions on Software Engineering, 1994,20(3).

[21] Lucia, A. D., Fasolino, A. R. Understanding function behaviors through program slicing. In: Proceedings of the 4th IEEE Workshop on Program Comprehension. Berlin, Germany: IEEE Computer Society Press, 1996. 9~18.

[23] Morasca L. C. S., Basili, V. R. Defining and validating measures for object-based high-level design. IEEE Transactions on Software Engineering, 1999.

完善验证的凝聚力测量方法以提升软件品质

李明奇[1], 施国琛[1,2], 黄德胜[1,2], 邓有光[1,2]

[1](屏东商业技术学院 企业管理学系,屏东);

[2](淡江大学 资讯工程研究所,屏东)

摘要: 无适当使用软件测量将可能引起软件低品质且高成本的窘态.凝聚力是软件品质重要因子之一如同维护度,可靠度和再利用度.软件模组品质的优劣必影响整体系统之品质的优劣.为了设计和维护高品质软件,软件专案经理人和软件工程师无可避免需引用软件凝聚力测量以衡量和产生高品质软件.提出以活路跃变量及视觉化变量纵距为分析基础之功能导向凝聚力测量方法.进而,以一系列实际案例来作实验验证,并以一组性质来作理论辨证所提的测量方法.因此一经完善定义,完善实验和完善辩证之凝聚力测量方法被提出用于当软件凝聚力强度的指标和因此增进软件品质.这凝聚力测量方法能容易嵌入 CASE 以帮助软件工程师确保软件品质.

关键词: 软件测量;凝聚力;软件品质;活跃变量

中图法分类号: TP311 **文献标识码:** A