A Function Oriented Software Cohesion Metrics

Timothy K. Shih
Dept. of Computer Science and Information Engineering
Tamkang University
Taiwan R.O.C
e-mail:tshih@cs.tku.edu.tw

Ming-Chi Lee
Dept. of Business Management
National Pingtung Institute of Commerce
Tai an R.O.C
e-mail:lmc@sun1.npic.edu.tw

Teh-Sheng Huang
Yeong-Huei Chen
Dept. of Computer Science and Information Engineering
Tamkang University
Taiwan R.O.C
e-mail:tehsheng@ms6.hinet.net

Abstract

Cohesion is one of the most important factors for software quality as well as maintainability, reliability and reus ability. Cohesion is defined as a quality attribute that seeks to measure the singleness of the purpose of a module. A coincidental cohesion is the lowest degree of cohesion in a module. A functional cohesion is the highest degree of cohesion in a module. For software managers and engineers, it will be inevitable to introduce a well-defined and well-examined cohesion metrics to produce desirable cohesion software. In this paper, we propose a function -oriented cohesion metrics based on the analysis of live variables and live span. They will be developed in a mathematical model, and be experimented using typical cohesion examples. As of the results of experiments, the proposed cohesion metrics not only matches the Fenton's cohesion strength spectrum but also meets nonlinear scale that is asserted by Pressman and Somerville in their literatures.

Keywords: *Live Variables, Live Span, Software Metrics, Cohesion*

1.Introduction

Without software metrics, software would be error prone, expensive and with a poor qua lity. Therefore, suitable software measurements are essential for software development. Module cohesion is defined as a quality attribute that seeks to measure 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 elements[1]. Cohesion refers to how closely the operations are related in a procedure. Ideally, a procedure should offer a single service to other procedures. Cohesion measurement can support software engineer design module with a higher cohesion. Module cohesion indicates how closely a module's internal components are related to others.

Due to the previous mentions, a software module with low cohesion will be error prone. A high cohesion module will lead to a good quality. The result implies that the quality of software can be improving by maximizing the degree of module cohesion. The software cohesion had been categorized into *functional*, *sequential*, *communicational*, *procedure*, *temporal*, *logical* and *coincidental* [9][2].

Live variables describes the extent of variable to be refereed within a module, while variable span captures the range of variable to be used in a module [4]. Generally instance variable usage is used to deriv e higher-level inference on cohesion [5]. In this paper, the proposed function-oriented cohesion metrics is based on an analysis model of live variable semantics of a module. We will examine the function -oriented cohesion metrics (FOCM) for each output fun ction, the most tightest of function-oriented cohesion metrics (MTFOCM), the least tightest of function-oriented cohesion metrics (LTFOCM), and the average tight of function-oriented cohesion metrics (ATFOCM) of the specific procedure.

In general, we always strive for high cohesion. Although the mid-range of the cohesion spectrum is often acceptable [10] and a module may exhibit more than one type of cohesion. However, we will show that the proposed metrics not only closely matches strength spectrum of Fe nton’s cohesion [3] but also meets nonlinear cohesion scale of cohesion [10].[12] Those proposed function -oriented software metrics should improve the software quality, and help to refine the procedure to approximate the desirable cohesion strength. The key contributions of this paper are that we propose the scientific basis, formal definition, and well designed empirical experiments to improve software quality and maintainability.

The rest of the paper is organized as follows. We define some essential concepts and our cohesion analysis model in section 2. We address the proposed software function-oriented cohesion metrics in section 3. Section 4 to experiment the cohesion metrics using typical cohesion implementations and. In section 5, we give our conclusions and future works.

2. Preliminaries

A module is a contiguous sequence of program statements that are bounded by boundary elements, and has an aggregate identifier [9]. However, many researchers define a module to be either a compilation unit of code or a procedure. By functionality, the variable reference in a procedure could be categorized into the following four categories

- Input variable (IV): a set composed of the variables that are the input arguments to the procedure.
- Internal variable (INV): a set of local variables of the procedure.
- Global variable (GV): a set is global variables.
- Output variable (OV): a set of variables that are the output functions of the procedure.

In general, a module will contain some specific function semantics. In pr actice, 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 elements that are referenced in the statement. To address the concept of live variables, Figure 1 is used.

```
 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
```

Figure 1. A procedure example

As *SumAndProd* listed in Figure 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* at other statements other than line 4 and 7? The answer is "yes". However, software engineers constantly have to keep in mind what *sum* may be referred between statements 4 and 7 Software engineers always realize that in each iteration *sum* has a differe nt status in the "for" loop, *sum* will be incremented, although the statements do not refer to *sum* in line 5 and 6. Finally, the summation operation will exit. Similarly, *prod* has the same scenario. And, Comprehending software artifacts is an important soft ware engineering activities. A significant amount of time of a software engineer is spent in looking at the source code t discover information during testing, review and code inspection. Thus, the more data items what a programmer must keep track of when constructing a procedure, the more difficult it is to construct. Based on above mentions, live variables of a statement are not limited to the number of variable reference in that statement. In this paper, we adopt the scope of live variable is from the fi rst reference to the last reference within a procedure. More precisely, we use the mathematics method to define the live variables and use live span to visualize a live variables scope. Therefore, we denoted the lifespan of a variable to be the reference domain that begins at the first referenced and extends through the last referenced.

Definition1: The lifespan of a variable in a procedure is a set, denoted by LS(*var_nam*). The elements of the set are the name of the variable from the first (*i*-th line) to the last (*j*-th line) referenced, LS(*var_nam*)={ $var\_nam_i, var\_nam_{i+1}\ldots$ $\ldots\ldots, var\_nam_j$}. And, the size of lifespan, | LS(*var_nam*) | = j-i+1.

As an example in Figure 1, we investigate the live span of variable *sum*. The set LS(*sum*)={$sum_4, sum_5, sum_6, sum_7$} with the size of 4.

Definition2: The live variables (LV) of a specific procedure (*sp*) is the set of union of the lifespan of each variable which belongs to IV, INV, GV or OV. We denote LV(*sp*)= $\cup$ LS(var_nam_i) where $var_nam_i \in$IV $\cup$ INV $\cup$ GV $\cup$ OV and $i \in$N.

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

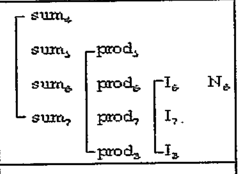
Line	LV(SumAndProd)	Count	Live Span
4	sum_4	1	
5	$sum_5, prod_5$	2	
6	$sum_6, prod_6, I_6, N_6$	4	
7	$sum_7, prod_7, I_7$	3	
8	$prod_8, I_8$	2	
		12	

Figure 2. The LV and LS of *SumAndProd*

The live span and live variables of Figure 1 are illustrated In Figure 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 line 4 and line 7, although *sum* is not real appear in statement 5 and statement 6. Therefore, from 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 visualization of the scope of live variables in a specific procedure.

3. Cohesion metrics

In this section, we propose function-oriented cohesion measurements that are based on the analysis of live variable abstraction of a procedure. In Figure 2, we know that not all variables involve the computing of output functions in a specific 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 operation. As of the result, $prod_5$ and $prod_8$ are the function-oriented live variables of output *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 $_8$ indirectly. Therefore, function-oriented live variables of the *prod* is a set, denoted {$prod_5$, I_6, N_6, $prod_8$, I_8}. However, we really need a precise definition on the function-oriented live variable for output functions.

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

For instance, variable I_7 contributes 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 perform the result of the output function via direct variables.

For instance, variable I_6, N_6 of iteration statement in line contributes 6 the value of the *sum_7* indirectly. So $_6$, N_6 is the indirect variable of the output function *sum_7*.

Definition 5. The *function-oriented live variables* of the output function in the specific procedure is a set, denotes FOLV(ov_i)={lv ∈ LV(sp) | lv are direct or indirect variables of ov_I; ov_I∈OV(sp)}

Definition 6. The *function-oriented live span* of the output function in the specific procedure is a set, denotes FOLS(ov_i)={lv ∈ LS(sp) | lv are direct or indirect variables of ov_I; ov_I∈OV(sp)}

The function-oriented live variables and function -oriented live span of procedure *SumAndProd* in Figure 1 are depicted in Figure 3,which is reduced from Figure 2.

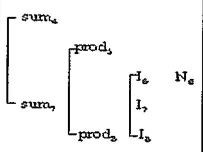
Line	FOLV(*SumAndProd*)	Count	FOLS
4	sum_4	1	 sum_4 $prod_5$ I_6 N_6 sum_7 I_7 $prod_8$ I_8
5	$prod_5$	1	
6	I_6,N_6	2	
7	sum_7,I_7	2	
8	$prod_8$,I_8	2	
		8	

Figure 3. The FOLV and FOLS of *SumAndProd*

For instance, in Figure 3, 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. A function oriented live span in Figure 3 is reduced from live span in Figure 1. More important, the live span could 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) is the number of elements which belongs to LV(sp), and influence the value of ov directly or indirectly. Then, we believe that the size of FOLV(ov), that implies how many live variables element in the specific procedure, shall contribute to the result of the output function. More precisely speaking, the proportion of |FOLV(ov)| in |LV(sp)| can be viewed as the strength of function-oriented cohesion which just restrict on one output function in a module. We have known that seven levels of the cohesion of a specific procedure are from functional t coincidental with decrement. Ideally, the specific procedure will be process in the single function. Before addressing the function-oriented cohesion measures, we now try investigate the function-oriented live variables in the specific procedure.

Definition 7. For a given specific procedure(sp), if there are more than one output functio n (i.e., |OV(sp)| =N_ov and N_ovr > 1) then the FOLV(sp) is the union of each FLOV(ov), where ov belongs to OV(sp). Similary, FOLS(sp) is the union of each FLOS(ov).

Now, based on definition 6, we the FOLV of the procedure *SumAndProd* in the Figure 3, inc luding both *sum* and *prod* in the OV(*SumAndProd*). The FOLV(*SumAndProd*) is {sum_4,$prod_5$,I_6,N_6,sum_7,I_7,$prod_7$,I_8} with a size of 8. In Figure 3, each element in FOLV(*SumAndProd*) will appear either in FOLV(*sum*) or FOLV(*prod*). Hence, the whole set of FOLV(*SumAndProd*) is the union of both FOLV(*sum*) and FOLV(*prod*). This means that some elements of FOLV(*SumAndProd*) may exist in both FOLV(*sum*) and FOLV(*prod*). From the perspective of the cohesion, the shared function -oriented live variables of the output functions of *sum* and *prod* are the most critical elements of the specific procedure. The set of the most critical elements of the specific procedure will be the intersection of FOLV(*sum*) and FOLV(*prod*). The FOLV(*sum*) and FOLV(*prod*) are shown in Figure 4.

However, the previous analysis is based on live variables semantics, the *function-oriented cohesion measurement* for each ov_i is defined as follows: FOC(ov_i)= | FOLV(ov_i)| / |LV(sp)| where for each ov_i ∈ OVR(sp). According to the adherent elements in the specific procedure, the cohesion level of a module is determined by the relation levels of output pairs[6].

Variables	FOLV(*sum*)	FOLV(*sum*)∩ FLOV(*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			$prod_5$,$prod_7$
	5	2	5

Figure 4. The FOLV of *Sum, Prod* and their intersection

We define the *most tightest function -oriented cohesion measurement* of the specific procedure as follows: MTFOC(*sp*)= | ∩FOLV(ov_i)| / |LV(*sp*)| where ov_i ∈ OVR(*sp*) and FOLV(ov_i) restrict on each ov_i In the procedure, not a ll live variables have function oriented relation with each *ov*. Obviously, the set of elements which has no functional relationship with each *ov* in live variable is denoted VL *sp*) – FOLV(*sp*). Therefore, the *least tight function-oriented cohesion measurement* is: LTFOC(*sp*)= | LV(*sp*) – ∪ FOLV(ov_i)| / |LV(*sp*)| where ov_i ∈ OVR(*sp*) We propose the *average tight function-oriented cohesion measurement* of the specific procedure as follows: ATFOC(*sp*)= | ∪ FOLV(ov_i) | / | LV(*sp*) | where ov_i∈ OVR(*sp*). Consequently, e proposed four function-oriented cohesion measures in the specific procedure. The values of the four proposed cohesion measures are in the range of between 0 and 1. Therefore, the numerical systems of the function -oriented cohesion measures are well-normalized cohesion measures.

4. Experiments

In this section, there are six typical cohesion procedure implementations to be used to make an experiment on our proposed measurements. The purpose of measuring the distinctive cohesion procedure examples is to e valuate the proposed function-oriented cohesion measurements and to estimate whether they are compatible with the cohesion strength spectrum. We believe that the experiments will not affect the completeness of the empirical study although the six implementations will not include the *temporal* one.

Coincidental cohesion (CC) is to estimate whether a module performs more than one function, and whether there are unrelated [11]. The abstract function and information flow diagrams of a *coincidental* cohesion example are depicted in Figure 5.

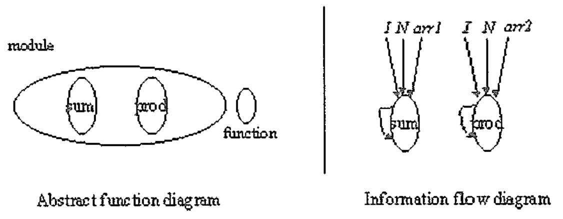


Figure 5. A function diagram of *coincidental cohesion*

In the *coincidental cohesion* example, there is n significant relationship between the elements. However, the values of function -oriented cohesion measures of coincidental cohesion example are given as follows

```
1 Procedure SumAndProd(N:integer; Var Sum;
          Prod:Integer;arr1,arr2:int_arry)
2 Var I:Integer
3 begin
 4  Sum:=0;                     |
 5  Prod:=1;                    | FOCM(sum) = 0.353
 6    For I:=1 to N do          | FOCM(prod)=0.353
 7      Sum:=Sum+arr1[I];       | MTFOCM(CC)=0.0
 8    For I:=1 to N do;         | LTFOCM(CC)=0.29
 9      Prod:=Prod*arr2[I];     | ATFOCM(CC)=0.71
 10   end                       |
 11 end
```

Figure 6. A *coincidental cohesion* implementation and the experimented results

In this example, there are no common processing elements that are used to produce both output functions. And, the value of MTFOCM of the coincidental cohesion is zero. The Coincidental cohesion type is the lowest cohesion level.

Logical cohesion (LC) is to check whether the module performs more than one function, and whether there are related logically [11]. The abstract function and information flow diagrams of a *logical* cohesion example are depicted in Figure 7.

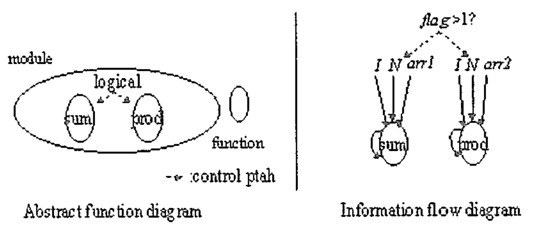


Figure 7. A function diagram of *logical cohesion*

In the *logical cohesion* example, the module performs some related functions. One or more of them are selected by calling module. In other words, there is a dynamic function selection in the module. However, values of function-oriented cohesion measures of logical cohesion example are given as follows

```
Procedure SumAndProd(N:integer; Var Sum;
            Prod:Integer;arr1,arr2:int_arry)
Var I,flag:Integer
begin
  Sum:=0;
  Prod:=1;
  If(flag>1)                    FOCM(sum) = 0.29
    for I:=1 to N do begin      FOCM(prod)=0.29
    Sum:=Sum+arr1[I];           MTFOCM(LC)=0.041
  else                          LTFOCM(LC)=0.46
    for I:=1 to N do begin      ATFOCM(LC)=0.64
    Prod:=Prod*arr2[I];
end
```

Figure 8. A *logical cohesion* implementation and experimented results

In this example, the value of MTFOCM of the logical cohesion is 0.041. This logical type cohesion is just little stronger than coincidental cohesion. From a viewpoint of cohesion strength, this *logical cohesion* is just little stronger than *coincidental cohesion*.

Procedural cohesion (PC) is to evaluate whether the module that performs more than one function, and whether the module is related to a general procedural effected by the software [11]. This means that if each function in the module needs to execute following a specific order then it has the strong procedure cohesion. The abstract function diagram and information flow diagrams of a *procedural* cohesion example are sketched in Figure 9.

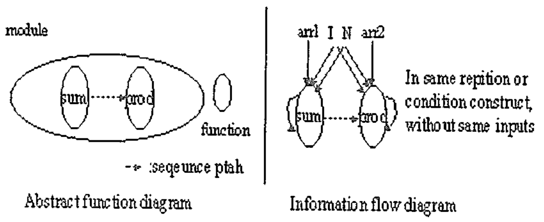

module
sum
prod
function
:seqeunce ptah
Abstract function diagram
arr1 I N arr2
sum
prod
In same repition or condition construct, without same inputs
Information flow diagram

Figure 9. A function diagram of *procedure cohesion*

In the *procedural cohesion* example, the module performs a series of functions related by a sequence of steps. Values of function-oriented cohesion measurements of procedural cohesion example are given as follows

```
Procedure SumAndProd(N:integer; Var Sum;
            Prod:Integer;arr1,arr2:int_arry)
Var I:Integer
begin
  Sum:=0;                       FOCM(sum) = 0.428
  Prod:=1;                      FOCM(prod)=0.428
   For I:=1 to N do             MTFOCM(PC)=0.142
     Sum:=Sum+arr1[I];          LTFOCM(PC)=0.288
     Prod:=Prod*arr2[I];        ATFOCM(PC)=0.71
end
```

Figure 10. A *procedure cohesion* implementation and the experimented results

In this example, the value of MTFOCM of the procedural cohesion is 0.142. From the cohesion strength perspective, *Procedural cohesion* is a little stronger than *logical cohesion*.

Communicational cohesion (CmC) is to indicate whether the module performs more than one function, and whether the module is on the same data [11]. This means that if each function in the module all operate on the same data. The abstract function and information flow diagrams of *communicational cohesion* example are sketched in Figure 11.

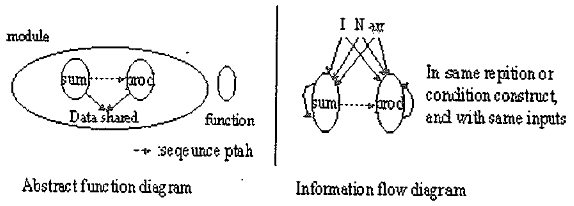

module
sum
prod
Data shared
function
:seqeunce ptah
Abstract function diagram
I N arr
sum
prod
In same repition or condition construct, and with same inputs
Information flow diagram

Figure 11. A function diagram of *communication cohesion*

In the *communicational cohesion* example, the module performs a series of functions related by the same data. Values of function -oriented cohesion measurements of communicational cohesion example are given as follows

```
Procedure SumAndProd(N:integer; Var Sum;
            Prod:Integer;arr:int_arry)
Var I:Integer
 Begin
  Sum:=0;                       FOCM(sum) = 0.428
  Prod:=1;                      FOCM(prod)=0.428
   For I:=1 to N do begin       MTFOCM(CmC)=0.142
     Sum:=Sum+arr[I];           LTFOCM(CmC)=0.288
     Prod:=Prod*arr[I];         ATFOCM(CmC)=0.71
  end
end
```

Figure 12. A *communicational cohesion* implementation and the experimented results

In this example, the value of MTFOCM of the communicational cohesion is 0.142. The value is same as procedural cohesion. From the viewpoint of cohesion strength, *Communicational cohesion* is not significant stronger than *procedural cohesion.*

Sequential cohesion (SC) is to estimate whether the module performs more than one function. Function dependency occurs in an order, which is described in the specification [11]. Generally, this means that the output data from a function is the input for the next function in a module. The abstract function diagram and flow diagram of *sequential* cohesion example are illustrated in Figure 13.

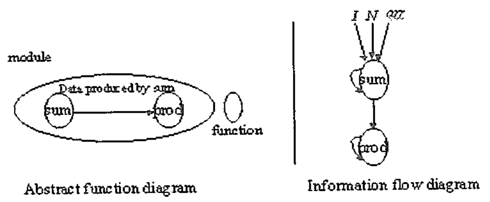


Figure 13. A function diagram of *sequential cohesion*

In the *sequential cohesion* example, the module performs a series of functions related by I/O data. Values of function oriented cohesion measurements of sequential cohesion example are given as follows

```
Procedure SumAndProd(N:integer; Var Sum;
              Prod:Integer;arr:int_arry)
Var I:Integer
Begin                        |
  Sum:=0;                    | FOCM(sum) = 0.46
  Prod:=1;                   | FOCM(prod)=0.69
    For I:=1 to N do begin   | MTFOCM(SC)=0.46
      Sum:=Sum+arr[I];       | LTFOCM(SC)=0.31
      Prod:=Prod*Sum;        | ATFOCM(SC)=0.69
    end                      |
end
```

Figure 14. A *sequential cohesion* implementation and the experimented results

In this example, the value of MTFOCM of the sequential cohesion is 0.46. The value is a little larger than the one of communicational cohesion. From a viewpoint of cohesion strength, *Sequential cohesion* is stronger than *communicational cohesion.*

Functional cohesion (FC) is to check whether the module performs on a single function [11]. This means that the module is the one on which all of the eleme nts contribute to exactly one function. The abstract function and information flow diagrams of *functional* cohesion example are illustrated in Figure 15.

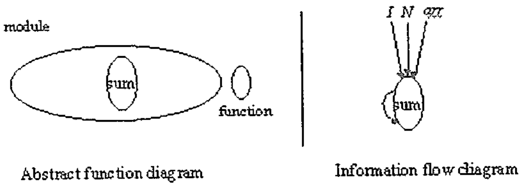


Figure 15. A function diagram of *functional cohesion*

In this example, all elements involved in a singl e activity. The value of MTFOCM of the functional cohesion is 0.86. Functional cohesion is stronger cohesion strength than sequential cohesion.

In the *functional cohesion* example, the module achieves exactly one goal. Values of function -oriented cohesion measurements of functional cohesion example are given as follows

```
Procedure SumAndProd(N:integer; Var Sum;
              Prod:Integer;arr:int_arry)
Var I:Integer
Begin                        | FOCM(sum) = 0.86
  Sum:=0;                    | MTFOCM(FC)=0.86
  For I:=1 to N do begin     | LTFOCM(FC)=0.143
  Sum:=Sum+arr[I];           | ATFOCM(FC)=0.857
end
```

Figure 16. A *functional cohesion* implementation and the experimented results

In this example, the value of MTFOCM of the functional cohesion is 0.86. Functional cohesion is stronger than sequential cohesion. The *functional cohesion* type is the highest cohesion level.

According to the empirical experiments, we know that the MTFOCM not only closely matches the Fenton' cohesion strength spectrum, but also meets nonlinear cohesion scale of Pressman's cohesion. More important, there are some paradigmatic characteristics about the proposed LTFOCM, ATFOCM and MTFOCM as follows: (1)MTFOCM(*sp*) ≤FOCM(ov_i) for each ov_i ≤ATFOCM(*sp*), (2)LTFOCM(*sp*)+ ATFOCM(*sp*)=1.0, (3) normalization.

5.Conclusions

Cohesion is an attribute that can be predict properties of implementations such as "ease of debugging, ease of maintenance, and ease of modification"[9]. Cohesion can be characterized as module strength. In this paper, we proposed the function-oriented cohesion metrics, which use live variables semantics to analyze the specific procedure, and to derive the FOCM, MTFOCM, LTFOCM and ATFOCM based measurement theory. They have been examined by six typical cohesion implementation examples. Together, the result of MTFOCM consists with the Fenton' cohesion spectrum, and the nonlinear scale that is stressed by Pressman and Somerville in their literatures. The major contributions of this paper are that we proposed a formal definition, scientific basis, well-designed experiments and easy measure algorithmically function-oriented cohesion metrics to improve software quality and maintainability. Our future works is to extend the current works t investigate the cohesion issues in object oriented paradigm.

Reference

[1] J. Martin,C. McClure, Software Maintenance the problem and its solutions, Prentice-Hall, INC.

[2] W. Stevens, G. Myers, and L. Constantine, "Structured Design", IBM Systems Journal,Vol. 13, No 2(1974).

[3] N. E. Fenton, "Software Measurement: A Necessar Scientific Basis", IEEE Transactions on Software Engineering,Vol. 20. No.3, March 1994.

[4] S. D. Conte, H. E. Dunsmore, V.Y. Shen, "Software Engineering Metrics and Models", TheBenjamin/Cummings Publishing Company, Inc.

[5] http://www.hatteras.com/metr_dis.html

[6] J. E. Bieman, Byung-Kyoo Kang, "Measureing Design-Level Cohesion", IEEE Transactions on Software Engineering, Vol. 24. No. 2, Feb. 1998.

[7] Dunsmore, Gannon,"Data referencing: an empirical investigation",IEEE Comp., pp50-59,Dec. 1979.

[8] Bill Curtis, "Human Factors in Software Development", pp. 170-179, Silver Spring, Comp. Soc. Press, 1981.

[9] E. Yourdon, L.L. Constantine, "Structured Design", Prentice Hall, 1979.

[10] Roger S. Pressman, "Software Engineering: Practitioners Approach", McGraw-Hill International Editions.

[11] N. E. Fenton,"Software Metrics, a Rigorous Approach", Chapman& Hall, London,1991.

[12] I. Sommerville, "Software Engineering", 5th ed. Addison-Wesley, 1996.