Robustness Analysis of Cognitive Information Complexity Measure using Weyuker Properties

Dharmender Singh Kushwaha and A.K.Misra
Department of Computer Science and Engineering
Moti Lal Nehru National Institute Of Technology
Allahabad, India.
Email:dharkush@yahoo.com,arun_kmisra@hotmail.com

Abstract
Cognitive information complexity measure is based on cognitive informatics, which helps in comprehending the software characteristics. For any complexity measure to be robust, Weyuker properties must be satisfied to qualify as good and comprehensive one. In this paper, an attempt has also been made to evaluate cognitive information complexity measure in terms of nine Weyuker properties, through examples. It has been found that all the nine properties have been satisfied by cognitive information complexity measure and hence establishes cognitive information complexity measure based on information contained in the software as a robust and well-structured one.

Keywords: Weighted information count, cognitive information complexity unit, cognitive weight, cognitive information complexity measure, basic control structures.

1. Introduction

Many well known software complexity measures have been proposed such as McCabe's cyclomatic number [8], Halstead programming effort[5], Oviedo's data flow complexity measures[9], Basili's measure[3][4], Wang's cognitive complexity measure[11] and others[7]. All the reported complexity measures are supposed to cover the correctness, effectiveness and clarity of software and also to provide good estimate of these parameters. Out of the numerous proposed measures, selecting a particular complexity measure is again a problem, as every measure has its own advantages and disadvantages. There is an ongoing effort to find such a comprehensive complexity measure, which addresses most of the parameters of software. Weyuker[14] has suggested nine properties, which are used to determine the effectiveness of various software complexity measures. A good complexity measure should satisfy most of the Weyuker's properties. A new complexity measure based on weighted information count of a software and cognitive weights has been developed by Kushwaha and Misra[2]. In this paper an effort has been made to estimate this cognitive information complexity measure as robust and comprehensive one by evaluating this against the nine Weyuker's properties.

2. Cognitive Weights of a Software

Basic control structures [BCS] such as sequence, branch and iteration [10][13] are the basic logic building blocks of any software and the cognitive weights (W_c) of a software [11] is the extent of difficulty or relative time and effort for comprehending a given software modeled by a number of BCS's. These cognitive weights for BCS's measure the complexity of logical structures of the software. Either all the BCS's are in a linear layout or some BCS's are embedded in others. For the former case, we sum the weights of all the BCS's and for the latter, cognitive weights of inner BCS's are multiplied with the weight of external BCS's.

3. Cognitive Information Complexity Measure (CICM)

Since software represents computational information and is a mathematical entity, the amount of information contained in the software is a function of the identifiers that hold the information and the operators that perform the operations on the information i.e.
Information = f (Identifiers, Operators)

Identifiers are variable names, defined constants and other labels in a software. Therefore information contained in one line of code is the number of all operators and operands in that line of code. Thus in k_{th} line of code the Information contained is:

$$\mathbf{I_k = (Identifiers + Operands)_k}$$
$$\mathbf{= (ID_k + OP_k)\ IU}$$

Where ID = Total number of identifiers in the k_{th} LOC of software,
OP = Total number of operators in the k_{th} LOC of software,
IU is the Information Unit representing that at least any identifier or operator has one unit information in them.

Total **Information** contained in a software (ICS) is sum of information contained in each line of code i.e.

$$\mathbf{ICS = \sum_{k=1}^{LOCS} (I_k)}$$

Where I_k = Information contained in k_{th} line of code,
LOCS = Total lines of code in the software.

Thus, it is the information contained in the identifiers and the necessary operations carried out by the operators in achieving the desired goal of the software, which makes software difficult to understand.

Once we have established that software can be comprehended as information defined in information units (IU's) [2], the weighted information count is defined as :

The **Weighted Information Count of a line of code (WICL)** of a software is a function of identifiers, operands and LOC and is defined as :

$$\mathbf{WICL_k = ICS_k \;/\; [LOCS - k]}$$

Where WIC_k = Weighted Information Count for the k_{th} line,
ICS_k = Information contained in a software for the k_{th} line.

The **Weighted Information Count of the Software (WICS)** is defined as :

$$\mathbf{WICS = \sum_{k=1}^{LOCS} WICL_k}$$

In order to be a complete and robust measure, the measure of complexity should also consider the internal control structure of the software. These basic control structures have also been considered as the Newton's law in software engineering [10, 11]. These are a set of fundamental and essential flow control mechanisms that are used for building the logical architectures of software.
Using the above definitions, **Cognitive Information Complexity Measure (CICM)** is defined as the product of weighted information count of software(WICS) and the cognitive weight (W_c) of the BCS in the software i.e.

$$\mathbf{CICM = WICS * W_c}$$

This complexity measure encompasses all the major parameters that have a bearing on the difficulty in comprehending software or the cognitive complexity of the software. It clearly establishes a relationship between difficulty in understanding software and its cognitive complexity. It introduces a method to measure the amount of information contained in the software thus enabling us to calculate the coding efficiency (E_I) as ICS / LOCS [2].

4. Evaluation of Cognitive Information Complexity Measure

Weyuker[14] proposed the nine properties to evaluate any software complexity measure. These properties also evaluate the weakness of a measure in a concrete way. With the help of these properties one can determine the most suitable measure among the different available complexity measures. In the following paragraphs, the cognitive information complexity measure has been evaluated against the nine Weyuker properties for establishing itself as a comprehensive measure.

Property 1: $(\exists P)(\exists Q)(|P| \neq |Q|)$ Where P and Q are program body.

This property states that a measure should not rank all programs as equally complex. Now consider the following two examples given in Fig. 1 and Fig. 2. For the program given in Fig. 1 in Appendix I, there are two control structures: a sequential and a iteration. Thus cognitive weight of these two BCS's is 1 + 3 = 4.

Weighted information count for the above program is as under:
WICS = 3/6 + 1/4 + 6/3+ 4/2 = 4.75
Hence Cognitive information complexity measure (CICM) is:
CICM = WICS * W_c = 4.75 * 4 = 19.0

For the program given in Fig. 2 in Appendix I there is only one sequential structure and hence the cognitive weight W_c is 1. WICS for the above program is 2.27. Hence CICM for the above program is 2.27 * 1 = 2.27.

From the complexity measure of the above two programs, it can be seen that the CICM is different for the two programs and hence satisfies this property.

Property 2: Let c be a non-negative number. Then there are only finitely many programs of complexity c.

Calculation of WICS depends on the number of identifiers and operators in a given program statement as well as on the number of statements remaining that very statement in a given program. Also all the programming languages consist of only finite number of BCS's. Therefore CICM cannot rank complexity of all programs as c. Hence CICM holds for this property.

Property 3: There are distinct programs P and Q such that $|P| = |Q|$.

For the program given in Fig. 3 in Appendix I, the CICM for the program is 19, which is same as that of program in Fig. 1. Therefore this property holds for CICM.

Property 4: $(\exists P)(\exists Q)\ (P \equiv Q\ \&\ |P| \neq |Q|)$

Referring to program illustrated in Fig.1, we have replaced the while loop by the formula "sum = (b+1)*b/2" and have illustrated the same in Fig.2. Both the programs are used to calculate the sum of first n integer. The CICM for both the programs is different, thus establishing this property for CICM.

Property 5: $(\forall P)(\forall Q)(|P| \leq |P;Q|$ and $|Q| \leq |P;Q|)$.

Consider the program body given in Fig.4 in Appendix I: The program body for finding out the factorial of a number consists of one sequential and one branch BCS's. Therefore W_c = 3. For the program body for finding out the prime number, there are one sequential, one iteration and two branch BCS's. Therefore W_c = 1 + 2*3*2 = 13. For the main program body for finding out the prime and factorial of the number, there are one sequential, two call and one branch BCS's. Therefore W_c = 1+5+15+2 = 23. WICS for the program is 5.1. Therefore the Cognitive Information Complexity Measure for the above program = 5.1 * 23 = 117.3.

Now consider the program given in Fig.5 in Appendix I to check for prime. There is one sequential, one iteration and three branch BCS's. Therefore W_c = 1 + 2*3*2 + 2 = 15. WICS = 1.85. So CICM = 1.85 * 15 = 27.79.
For the program given in Fig.6 in Appendix I, there is one sequential, one iteration and one branch BCS's .
Wc for this program is 7 and WICS is .5.11. Hence CICM = WICS * W_c = 5.11 * 7 = 35.77.

It is clear from the above example that if we take the two-program body, one for calculating the factorial and another for checking for prime whose CICM are 27.79 and 35.77 that are less than 117.3. So property 5 also holds for CICM.

Property 6(a) : (∃P)(∃Q)(∃R)(|P| = |Q|) & (|P;R| ≠ |Q;R|)

Let P be the program illustrated in Fig.1 and Q is the program illustrated in Fig.3. The CICM of both the programs is 19. Let R be the program illustrated in Fig.6. Appending R to P we have the program illustrated in Fig.7 in Appendix I.

Cognitive weight for the above program is 9 and WICS is 8.3. Therefore CICM = 8.3*9=74.7.

Similarly appending R to Q we have W_c = 9 and WICS = 8.925. Therefore CICM = 8.925*9 = 80.325 and 74.7 ≠ 80.325. This proves that Property 6(a) holds for CICM.

*Property 6(b)***:** (∃P)(∃Q)(∃R)(|P| = |Q|) & (|R;P| ≠ |R:Q|)

To illustrate the above property let us arbitrarily append three program statements in the programs given in Fig.1, we have the program given in Fig.8 in Appendix I. There is only one sequential and one iteration BCS. Hence cognitive weight is 1 + 3 = 4. There is only one sequential and one iteration BCS. Hence cognitive weight is 1 + 3 = 4 and WICS = 5.58. So CICM = 5.58 * 4 = 22.32.

Similarly appending the same three statements to program in Fig.3 we again have cognitive weights = 4 and WICS = 5.29. Therefore CICM = 21.16 ≠ 22.32. Hence this property also holds for CICM.

Property 7: There are program bodies P and Q such that Q is formed by permuting the order of the statement of P and (|P| ≠ |Q|).

Since WICS is dependent on the number of operators and operands in a given program statement and the number of statements remaining after this very program statement, hence permuting the order of statement in any program will change the value of WICS. Also cognitive weights of BCS's depend on the sequence of the statement[1]. Hence CICM will be different for the two programs. Thus CICM holds for this property also.

Property 8 : If P is renaming of Q, then |P| = |Q|.

CICM is measured in numeric and naming or renaming of any program has no impact on CICM. Hence CICM holds for this property also.

Property 9: (∃P)(∃Q)(|P| + |Q|) < (|P;Q|)
OR
(∃P)(∃Q)(∃R)(|P| + |Q| + |R|) < (|P;Q;R|)

For the program illustrated in Fig.4, if we separate the main program body P by segregating Q (prime check) and R (factorial), we have the program illustrated in Fig.9 as shown in Appendix I. The above program has one sequential and one branch BCS. Thus cognitive weight is 4 and WICS is 1.475. Therefore CICM = 4.425. Hence 4.425 + 27.79 + 35.77 < 117.3. This proves that CICM also holds for this property.

5. Comparative Study of Cognitive Information Complexity Measure and Other Measures in Terms of Weyuker Properties

In this section cognitive information complexity measure has been compared with other complexity measures in terms of all nine Weyuker's properties.

P.N.- Property Number, S.C.- Statement Count, C N. - Cyclomatic Number, E.M.- Effort Measure, D.C.-Dataflow Complexity, C.C.M. - Cognitive Complexity Measure, CICM – Cognitive Information Complexity Measure, Y- Yes, N – NO

P N	SC	CN	EM	DC	CCM	CICM.
1	Y	Y	Y	Y	Y	Y
2	Y	N	Y	N	Y	Y
3	Y	Y	Y	Y	Y	Y
4	Y	Y	Y	Y	Y	Y
5	Y	Y	N	N	Y	Y
6	N	N	Y	Y	N	Y
7	N	N	N	Y	Y	Y
8	Y	Y	Y	Y	Y	Y
9	N	N	Y	Y	Y	Y

Table 1: Comparison of complexity measures with Weyuker properties.

It may be observed from the table 1 that complexity of a program using effort measure, data flow measure and Cognitive Information Complexity Measure depend directly on the placement of statement and therefore all these measures hold for property 6 also. All the complexity measure intend to rank all the programs differently

6. Conclusion

Software complexity measures serves both as an analyzer and a predicator in quantitative software engineering. Software quality is defined as the completeness, correctness, consistency, no misinterpretation, and no ambiguity, feasible verifiable in both specification and implementation. For a good complexity measure it is very necessary that the particular complexity measure not only satisfies the above-mentioned property of software quality but also satisfies the nine Weyuker properties. The software complexity in terms of cognitive information complexity measure thus has been established as a well- structured complexity measure.

References:

[1] Misra,S and Misra,A.K.(2005): Evaluating Cognitive Complexity measure with Weyuker Properties, *Proceeding of the3 rd IEEE International Conference on Cognitive Informatics(ICCI'04).*

[2] Kushwaha,D.S and Misra,A.K.(2005): A Modified Cognitive Information Complexity Measure of Software, *Proceeding of the 7th International Conference on Cognitive Systems(ICCS'05)* (accepted for presentation)

[3] Basili,V.R. and Phillips(1983): T.Y,Metric analysis and data validation across fortran projection *.IEEE Trans.software Eng.,*SE –9(6):652-663,1983.

[4] Basili,V.R.(1980): Qualitative software complexity model: A summary in tutorial on models and method for software management and engineering *.IEEE Computer Society Press ,Los Alamitos*,CA,1980
.
[5] Halstead,M.(1977): Elements of software science,*Elsevier North Holland,New York.1997.*

[6] Klemola, T. and Rilling, J.(2003): A Cognitive Complexity Metric Based on Category Learning, , *Proceeding of the 2nd IEEE International Conference on Cognitive Informatics(ICCI'03).*

[7] Kearney, J.K., Sedlmeyer, R. L., Thompson, W.B., Gray, M. A. and Adler, M. A.(1986): Software complexity measurement. *ACM Press, Newyork*, 28:1044-1050,1986

[8] McCabe, T.A.(1976): Complexity measure. *IEEE trans.software engg*.,(se-2,6):308-320,1976.

[9] Oviedo, E.(1980): Control flow, data and program complexity .in Proc. *IEEE COMPSAC, Chicago, lL, pages* 146-152,November 1980.

[10] Wang. and Shao,J.(2002): On cognitive informatics, keynote lecture, proceeding of the *.1st IEEE International Conference on Cognitive Informatics,* pages 34-42,August 2002.

[11] Wang ,Y .and Shao,J.(2004): Measurement of the Cognitive Functional Complexity of Software, *Proceeding of the 3 rd IEEE International Conference on Cognitive Informatics(ICCI'04)*

[12] Wang,Y .and Shao,J.(2003): On cognitive informatics, *Proceeding of the 2nd IEEE International Conference on Cognitive Informatics(ICCI'03),London,England,IEEE CS Press*, pages 67-71,August 2003 .

[13] Wang, Y.(2002): The real time process algebra (rtpa*). Annals of Software Engineering, an international journal*, and 14:235-247,2002.

[14] Weyuker, E.(1988): Evaluating software complexity measure. *IEEE Transaction on Software Complexity Measure,* 14(9): 1357-1365,september1988.

Appendix I

```
/*Calculate the sum of first n integer*/
main() {
int i, n, sum=0;
printf("enter the number");                    //BCS1
scanf("%d" , &n);
for (i=1;i<=n;i++)                              //BCS2
sum=sum+i;
printf("the sum is %d" ,sumssss);
getch();}
```

Fig. 1 : Source code of the sum of first n integers.

```
main()
{
 int b;
int sum = 0;
Printf("Enter the Number");
Scanf("%d", &n);
Sum = (b+1)*b/2;
Printf("The sum is %d",sum);
getch();
}
```

Fig. 2 : Source code to calculate sum of first n integers.

```
# define N 10
main( )
{
int count
float, sum,average,number;
sum = count =0;
while (count < N )
{
scanf (" %f",& number);
sum = sum+ number;
count = count+1;
}
average = sum / N;
printf ("Average =%f",average);
}
```

Fig. 3 : Source code to calculate the average of a set of N numbers.

```
#include< stdio.h >
#include< stdlib.h >
int main() {
long fact(int n);
int isprime(int n);
int n;
long int temp;
clrscr();
printf("\n input the number");                 //BCS11
scanf("%d",&n);
temp=fact(n);                                   //BCS12
{printf("\n is prime");;}
int flag1=isprime(n);                           //BCS13
if (flag1= =1)                                  //BCS14
else
{printf("\n is not prime")};
printf("\n factorial(n)=%d",temp);
getch();
long fact(int n) {
long int facto=1;                               //BCS21
if (n= =0)                                      //BCS22
facto=1;else
facto=n*fact(n-1);
return(facto); }
int isprime(int n)
{ int flag;                                     //BCS31
if (n= =2)
flag=1;                                         //BCS32
else
for (int i=2;i<n;i++)                           //BCS33
{ if (n%i= =0)                                  //BCS34
{ flag=0; Therefore Wc = 3


break; }
else {
flag=1 ;}}
return (flag);}}
```

Fig. 4: Source code to check prime number and to calculate factorial of the number

```
#include< stdio.h >
#include< stdlib.h >
#include< conio.h >
int main() {                                    //BCS1
int flag = 1,n;
clrscr();
printf("\ n enter the number");
scanf("%d",&n);
if (n= =2)
flag=1;                                         //BCS21
else
{for (int i=2;i<n;i++)                          //BCS22
if (n%i= =0)                                    //BCS23
{ flag=0;
break;}
else{
flag=1;
continue;} }
if(flag)                                        //BCS3
printf("the number is prime");
else
printf("the number is not prime");
grtch();}
```

Fig.5 : Source code for checking prime number

```
#include< stdio.h >
#include< stdlib.h >
#include< conio.h >
int main () {
long int fact=1;
int n;
clrscr();
printf("\ input the number");        //BCS1
scanf("%d",&n);
if (n==0)                            //BCS21
else
for(int i=n;i>1;i--)                 //BCS22
fact=fact*i;
printf("\n factorial(n)=%1d",fact);
getch();}
```

Fig.6 : Source code for calculating factorial of a number

```
Int main() {
long fact(int n);
int i, n, sum=0;
printf("enter the number");
scanf("%d" , &n);
temp = fact(n);
for (i=1;i<=n;i++)
sum=sum+i;
printf("the sum is %d" ,sum);
getch();
long fact(int n){
long int facto = 1;
if (n == 0)
facto = 1 else
facto = n*fact(n-1);
return(facto);}}
```

Fig.7: Source code of sum of first n integer and factorial of n.

```
main() {
int a,b,result;
result = a/b;
printf(the result is %d",result);
int i, n, sum=0;
printf("enter the number");
scanf("%d" , &n);
for (i=1;i<=n;i++)
sum=sum+i;
printf("the sum is %d" ,sum);
getch();}
```

Fig. 8 : Source code of division and the sum of first n integers.

```
int main(){
int n;
long int temp;
clrscr();
printf("\n input the number");
scanf("%d",&n);
temp = fact(n);
{printf("\ is prime");}
int flag1 = isprime(n);
if (flag1 == 1)
else
{printf("\n is not prime)};
printf("\n factorial(n) = %d",temp);
getch();}
```

Fig.9 : Source code of main program body of program in Fig.4