IJE TRANSACTIONS B: Applications Vol. 28, No. 11, (November 2015) 1579-1588

International Journal of Engineering

Journal Homepage: www.ije.ir

Measurement of Complexity and Comprehension of a Program Through a Cognitive Approach

A. K. Jakhar*, K. Rajnish

Department of Computer Science & Engineering, Birla Institute of Technology, Mesra, Ranchi, Jharkhand, India

PAPER INFO

Paper history:
Received 08 February 2015
Received in revised form 21 September 2015
Accepted 16 October 2015

Keywords:
Complexity
Basic Control Structure
Cognitive Informatics
Operators
Operands
Cognitive Weight

ABSTRACT

The inherent complexity of the software systems creates problems in the software engineering industry. Numerous techniques have been designed to comprehend the fundamental characteristics of software systems. To understand the software, it is necessary to know about the complexity level of the source code. Cognitive informatics perform an important role for better understanding the complexity of the software. These informatics also facilitate researchers to understand the behavior of the source code, internal as well as the control structure. This paper presents a new cognitive complexity measure (C2M) for measuring the complexity of the software system and it is tested on 50 'C' programs. The proposed C2M measure was also validated with the help of Weyuker property; eight out of nine properties were satisfied by the proposed measure. The performance of the proposed measure was also compared with other existing cognitive complexity measures. The test data was distributed among 10 undergraduate students of our institute and they were asked to understand the source code. The time taken by the individual student was recorded and the meantime of the recorded data from students was considered as the actual time required to understand the program. It was further correlated with the estimated time, which was calculated through C2M measure. From the experimental results, it was observed that proposed measure provided better quality results.

doi: *10.5829/idosi.ije.2015.28.11b.05*

1. INTRODUCTION

Software estimation is a complex process due to the inherent complexity of softwares. Till date, numerous complexity measures have been developed to measure the software systems. These measures assist researchers to analyze the behavior of the source code that can be utilized further to measure the software in some different facets. The traditional approach to measure the software is lines of code (LOC) [1] that is used to measure the project's physical size. This approach is programmer dependent because the naive users increase the program size unnecessarily as compared to sophisticated users. To overcome the aforementioned limitation, McCabe's [2] developed Cyclomatic Complexity in 1976 that is used to count the independent paths in software system and the complexity is measured with the help of the connected graph that shows the complete structure of the software. This measure only considers the control structure of the software to measure the complexity. In 1977, Halstead proposed another measure which concerns the internal structures, but disregards the control structures [3]. In that work, operators and operands are used to measure the complexity of software. In continuation of his work, Halstead proposed a lot of methods to measure the software in different aspects. Several other measures also have been reported to find the clarity of the software [4, 5]. It is observed from the literature that the software complexity depends on both the control structure and the internal structure of the software and the cognitive technique can do this job very well. Cognitive Informatics (CI) is used in various research fields to find the solution of a given problem such as software engineering, artificial intelligence, and cognitive sciences, thus, cognitive informatics is an inter-disciplinary research area [6, 7]. CI [8] is utilized

*Corresponding Author's Email: *amitjakhar69@gmail.com* (A. K. Jakhar)

Please cite this article as: A. K. Jakhar, K.Rajnish, Measurement of Complexity and Comprehension of a program through a Cognitive Approach, International Journal of Engineering (IJE), TRANSACTIONS B: Applications Vol. 28, No. 11, (November 2015) 1579-1588

to measure the software system by understanding the essential characteristics of the source code. Cognitive complexity focuses on the mental effort required to understand the software or how difficult it is to perform the tasks. In CI, it is found that the functional complexity of the software depends on three facets: input, output and internal architecture [6]. The cognitive complexity measures emphasis on all aspects of software systems. Numerous cognitive complexity measures are available for measuring the software systems. Each complexity measure has its own benefits and limitations and it is not an easy task to find a most suitable measure for the software systems. Still, it is an ongoing process which can comprehend all aspects of the software systems accurately. Whenever, a new measure is proposed, it is necessary to analyze the outcome of the measure with some existing approach for finding its effectiveness. Weyuker [9] has proposed nine properties to validate a new complexity measure. These nine properties are utilized to determine the efficiency of any complexity measure for software systems. Most of the properties should be satisfied by a good measure and eight out of nine properties are satisfied by the proposed measure.

The aim of this paper is to develop a new cognitive complexity measure for calculating the complexity and the required time to understand the software system. The proposed measure is based on some attributes of the source code like: operands, operators, data types, lines of code, and cognitive weight of the basic control structures (BCSs). The performance of the proposed measure is also compared with some existing complexity measures such as CFS [10], MCCM [11], CPCM [12], and NCCoP [13]. In other work, Adamo [14] presented an experimental design and tool for finding out the cognitive weights of the BCSs for a particular programming language. Fourteen Java programming experienced graduate students participated in the experiments. Some code snippets were given to the participants and the time taken by individual participants for understanding the snippets was recorded for both correct and incorrect responses. Another work [15] addresses the reduction of space and time complexity in the network system. In another investigation [16], the authors proposed a cognitive complexity metric to measure the code complexity of Java programs with the help of six attributes of the source code. These programs were ranked by seven experts of Java programming language. Jakhar et al. [17] also proposed a complexity measure for OO system in 2015. In this paper, the authors utilized some attributes of the program like: operands, operators and ratio of accessing similar parameters by methods of a class and the cognitive weight (W_c) to measure the complexity of OO programs. However, the proposed approach focuses on measuring the cognitive complexity of the procedural language instead of OO languages. Both approaches have different structures and features. So, we dropped these efforts to further study in our experiments. In addition to our proposed work, 10 undergraduate students of the institute were participated in an experiment and they were asked to comprehend the source code of 50 'C' programs. The time taken by the individual student was recorded and meantime of the recorded data from all the students was considered as an actual time needed to understand the code. The average time indicates the complexity level of the programs, means, if a program has higher complexity, then the time required to understand the program increases due to the complex structure of the program and vice versa. Finally, correlation is calculated between the actual time and estimated time to verify the outcome.

The rest of the paper is organized as follows: Section 2 deals with the existing cognitive complexity measures. The description of proposed measure and analytical evaluation besides Weyuker property is provided in section 3. Section 4 gives a comparative analysis of the proposed and existing complexity measures. Section 5 summarizes the conclusion and future work of this paper.

2. EXISTING COGNITIVE COMPLEXITY MEASURES

Numerous complexity measures rely on the cognitive informatics. Some of them are studied and examined in the following subsections.

2. 1. Cognitive Functional Size (CFS) Wang et al. [10] proposed a cognitive functional size (CFS) measure to compute the complexity of the software. This measure relies on the cognitive informatics. The functional size of the software is computed by using a number of inputs, output and cognitive weight (W_c) of all BCSs. Generally, every software consists of three specified structures which are sequential, branch and iterative structure [18]. In CFS, the aforementioned parameters are taken into account for measuring the complexity of software. The CFS is calculated using Equation (1).

$$CFS = (N_i + N_o) \times W_c \quad (1)$$

where N_i: number of inputs; N_o: number of outputs; and W_c: overall cognitive weight of all the BCSs present in the source code and cognitive weight (W_c) of individual BCS are assigned by Wang et al.[10].

In formulation of CFS, W_c of all BCSs is calculated by using Equation (6).

As aforementioned, CFS only considers the number of input, output and cognitive weight (W_c). However, it is also observed that the complexity also depends on the

total occurrence of input, output and other attributes which are comprised in the source code. These attributes are also contributing in the complexity of the software. So, it is suggested that these parameters should also be considered while calculating the cognitive complexity of software.

2. 2. Modified Cognitive Complexity Measure (MCCM) MCCM has been proposed by Sanjay Misra in 2006 [11]. This measure utilized the total number of operands, operators and cognitive weight (W_c) of the BCSs for calculating the complexity of the software. MCCM is defined as:

$$MCCM = (N_{i1} + N_{i2}) \times W_c \quad (2)$$

where, N_{i1}, N_{i2} are total occurrences of operands and operators, respectively and W_c is cognitive weight of all the BCSs.

The resulted values of MCCM are very large, due to the multiplication of the BCSs cognitive weight with the sum of all operators and operands of the software. In large software applications, this approach can create difficulty while measuring the complexity. If one branch, loop or function is added, then cognitive complexity of the software increases rapidly, though, in reality, it is not true. So, to overcome this limitation, the attributes of the software are arranged in such a manner that the resulted value is in under control.

2. 3. Cognitive Program Complexity Measure (CPCM) CPCM has also been developed by Sanjay Misra in 2007 [12]. In continuation of his work, it is found that operands of the program affect the complexity of software. In this work, each occurrence of input and output variables are considered to measure the cognitive complexity. The cognitive weight of each BCS is added with the calculated operands of the program. The formula of CPCM is given as below in Equation (3).

$$CPCM = S_{IO} + W_c \quad (3)$$

where, $S_{IO} = N_i$ (total occurrences of input variables) + N_o (total occurrences of output variables), and W_c is the cognitive weight of all the BCSs.

To validate CPCM measure, Weyuker property has been used. Seven out of nine properties are satisfied by the CPCM measure. This measure ignore the occurrences of operators while measuring the complexity of software. The CPCM measure stated that operators do not affect the complexity of a software system whether the software is large or small. However, a software comprises of operators and operands and both have a significant contribution to calculating the complexity of software. It is to be noted that the information is manipulated with the help of operators and the manipulated information is very hard to handle and even harder to understand. So, the number of operators should be included while measuring the complexity of software or using some other techniques to compensate it.

2. 4. New Cognitive Complexity Measure of Program (NCCoP) NCCoP measure has been proposed by Jakhar et al. [13]. In their work, variables and constants along with the BCSs cognitive weight are considered for computing the cognitive complexity of the programs. This work is carried out line by line from starting to the end of the program. But, operators are excluded from the formation of NCCoP as the BCS cognitive weight is multiplied with the number of operands of each LOC. Equation (4) is used to find the complexity of software.

$$NCCoP = \sum_{k=1}^{LOCs} \sum_{V=1}^{n} N_V(k) \times W_C(k) \quad (4)$$

where, N_v and W_c are the total number of operands and the cognitive weight of LOC k, respectively. The entire complexity of the program is calculated by summation of the complexity of each LOC. For an "if" statement the complexity of the following LOC "if (a>b)" is evaluated according to NCCoP which is 4, i.e. 2×2=4, as two variables *a* and *b* are present in the statement and the cognitive weight W_c of "if" structure is 2.

If a program follows only a sequential structure then the result of CPCM [12] and NCCoP is identical, whereas NCCoP includs total number of operators as well, then outcome of the MCCM [11] and NCCoP is identical. Generally, the software does not follow the sequential structure for solving today's complex problems, so this is not the case of today's scenario. Now, we analyze the result of NCCoP by considering and ignoring the operators of the program. Consider the "for" statement that is given below:

```
for (i=0; i<10; i++)
```

When operators are taken into account:
Modified NCCoP [13]=
[5 (operands)+3 (operators)]×3 (W_c)=24
MCCM [11]=
[5 (operands)+3(operators)]×3 (W_c)=24
When operators are not taken into account:
NCCoP [13]=5 (operands)×3 (W_c)=15
CPCM [12]=5 (operands)+3 (W_c)=8

As the result indicates that the complexity value of NCCoP and MCCM measures is the same when the number of operators are considered, i.e. 24, but the complexity value of NCCoP and CPCM is not same as multiplication is used instead of addition in CPCM. If the W_c is added with the number of operands and operators instead of multiplication in NCCoP, then this

measure generates the same complexity value as CPCM, i.e. 8. The complexity of the same statement is:
Modified NCCoP [13]=
[5 (operands)+3(operators)]+3(W_c)=11
Modified NCCoP [13]=5(operands)+3(W_c)=8
The overall result of the program by NCCoP will not be same as CPCM and MCCM. Because in NCCoP, W_c of each LOC is multiplied with the number of operands of the same LOC, but in later cases the entire W_c of the program is added or multiplied with the calculated information like operands and operators.

As the above result indicates, if the operators are included in NCCoP, then the measure may generate the same complexity value as MCCM. For larger software, MCCM creates problem due to its high complexity value that is not desirable. That's why the operators are ignored and multiplication is used instead of addition as in CPCM. NCCoP cognitive complexity technique can be helpful in modular programming to measure the complexity of the individual modules. This is a hypothesis that an overly complex code due to bad structure with low cohesion is unreliable and difficult to maintain [19]. The most efficient way to deal with the large software is dividing the software into smaller modules. The smaller modules are reliable, easy to maintain and test. In order to calculate the complexity of individual modules the NCCoP measure is helpful.

3. PROPOSED COGNITIVE COMPLEXITY MEASURE (C2M)

In this section, an attempt is made to develop a new cognitive complexity measure (C2M), which is used to calculate the complexity of the software. The software is a collection of information and the information is manipulated through operators. Specific operations can be performed on a particular data type and it directly affects the complexity of the code. So, the number of data types also plays an important role while calculating the complexity of software. Hence, in this work, a new measure (C2M) is proposed which consideres five user-defined parameters to measure the cognitive complexity of the software. The proposed Cognitive Complexity Measure (C2M) depends on the followings parameters:

- The total occurrence of operands (variables and constants).
- The total occurrence of operators (only pure operators).
- The total number of data types present in the program (int, char, float, structure, pointer, etc.).
- The total executable LOCs of the program.
- The cognitive weight (W_c) of all BCSs.

In this proposed approach, a relation is formed between several parameters of the source code for calculating the complexity of software, which is given in Equation (5).

$$C2M = \sum N_{Operands}, N_{Operators}, N_{DT}, Exe_{loc}, W_c \quad (5)$$

All attributes of C2M measure are described above and the W_c is assigned to each BCS according to the effort, time and difficulty to comprehend the source code [6]. All BCSs of the source code and their respective cognitive weights are allotted on the classification of cognitive phenomenon as given by Wang et al. [10]. The structure of a program may include the sequential, iterative, branch and embedded instructions. Further, these BCSs are either linear or nested/embedded structure.

For the nested structure, total W_c of a software component is calculated as the sum of cognitive weights of its q linear blocks composed of individual BCSs. Each block may consist of m layers of nesting and n linear control structures. Therefore, the overall W_c is calculated using Equation (6).

$$W_c = \sum_{j=1}^{q}[\prod_{k=1}^{m}\sum_{i=1}^{n} W_c(j,k,i)] \quad (6)$$

If no nested structure presents in any of the q blocks, i.e. m=1, then Equation (6) can be simplified as:

$$W_c = \sum_{j=1}^{q}\sum_{i=1}^{n} W_c(j,i) \quad (7)$$

How Equation (6) is used to calculate the W_c is shown in Figure 1. W_c is calculated with nested structure of the BCSs as shown in Figure 1. The repetition of the loops are not considered here because to understand the software, repetition does not enhance the difficulty level of the source code. The W_c '3' is assigned to the iterative statements according to its difficulty level that's why number of repetitions are not considered. If all the BCSs are linear in the program, then the assigned cognitive weight of the individual BCSs are simply added. Illustration of C2M and other existing cognitive complexity measures with an example is given below.

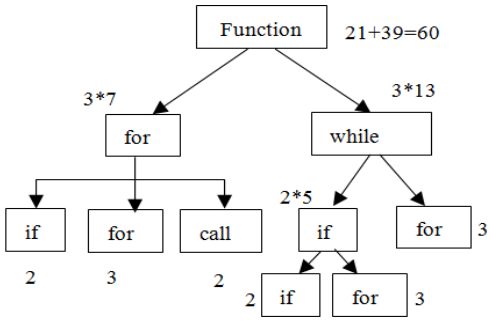

Figure 1. An example W_c calculations

The example program "sum of n numbers" is used to elaborate how the cognitive complexity measures calculate the complexity of software. The given example program consists of two BCSs: sequential and iterative. The source code of the example program is given below.

```
//A program to calculate the sum of n numbers:
main() {
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(); }
```

Here, we elaborate the proposed C2M measure to calculate the complexity of the example program by calculating its attributes separately.

$N_{Operands}=14$
$N_{Operators}=6$
$N_{DT}=1$
$EXE_{LOC}=5$
$W_c=1+3=4$
C2M=14+6+1+5+4=30

Therefore, the cognitive complexity of the program using proposed C2M measure is '30'. Table 1 contains the calculated complexity result value of the example program with our proposed and other existing measures in the domain of cognitive complexity. The result of each measure is calculated by their own formula given in the respective subsections. Different attributes are used by the complexity measures in some different sequences, that's why each measure yields different complexity values.

The calculated value from the proposed measure is also further utilized to estimate the understand-ability factor of the software. It indicates the required time to understand the source code. When a person starts reading the source code, the simple things can be flip very easily, but when the complex concepts are introduced, then it is difficult for the person to understand it, so the required time increases. Hence, it is very helpful in the testing and maintenance phase of the software development life cycle. So, this factor is also calculated with the help of proposed measure and it is 7% of the complexity value.

3. 1. Theoretical Validation of Proposed Cognitive Complexity Measure (C2M)

Whenever, a new metric is proposed, it is necessary to validate the metric with the help of some practical and formal validation techniques. Weyuker [9] proposed nine properties in 1988, which can evaluate the strength and weaknesses of the new complexity metric.

TABLE 1. Calculated complexity values of existing and proposed cognitive complexity measures

LOC	CFS	CPCM	NCCOP	MCCM	C2M
5	12	16	24	80	30

These nine properties are used to validate the C2M measure and 50 'C' programs are taken into account from the literature [20]. Eight out of nine properties are satisfied by the proposed C2M measure. The result of the proposed and other measures as a complexity value of all 50 programs is provided in Table 3.

Weyuker property is described below one by one with proposed cognitive complexity measure.

Property 1: (∃P) (∃Q) (|P|≠|Q|), and program P and Q are the body of the program.

This Weyuker property states that a measure should not rank all the programs as equally complex. Referring to Table 3, Program 1 and program 2 are considered to analyze the first Weyuker property. The cognitive complexity of program 1 according to the proposed measure, as Equation (5), is=17+0+6+7+1=31. In program 1, the only sequential structure is incorporated. The cognitive complexity of program 2 as Equation (5) is=16+4+2+14+7=43. In program 2, sequential and embedded components are present in the program body. So, the W_c of the BCSs is used as '7' instead of '1' as stated in the evaluation of program 1. The analysis of program 1 and program 2 clearly indicates that both programs are not equally complex. Hence, the proposed cognitive complexity measure holds the first Weyuker property.

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

The software is a set of information and the information is the function of operands and operators. Each program of any language contains some finite operands, operators, control structures, executable LOCs, etc. for solving the problems. Without these parameters, a program cannot do anything. The proposed measure is built on all the above-said parameters, which are incorporated in each program of any language. Hence, the C2M holds the second Weyuker property.

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

This property states that two different programs can be denoted as equally complex. Two programs 4 and 6 are considered to check this property. Program 4 has two internal structures: sequential and iterative. According to the proposed measure, total cognitive complexity of program 4 is=14+6+1+5+4=30, and the cognitive complexity of the program 6 is=15+6+1+7+1=30, only sequential structure is

incorporated into program 6, so cognitive weight '1' is used instead of '4' as in program 4.

TABLE 3. Cognitive complexity values of 50 programs with concerned cognitive complexity measures

S. No	Col-pp.no	Loc	CFS	MCCM	CPCM	NCCoP	C2M
1	1-32	7	12	17	18	17	31
2	1-33	14	35	140	23	16	43
3	2-34	16	8	54	47	46	73
4	2-77	5	12	80	18	20	30
5	1-40	5	4	29	20	19	36
6	2-46	7	4	21	16	15	30
7	2-47	7	5	26	19	18	35
8	1-51	7	10	66	52	51	76
9	2-71	11	20	285	42	41	75
10	1-78	7	24	84	18	19	33
11	2-86	3	1	21	14	33	26
12	2-87	6	1	43	28	71	51
13	2-90	5	8	64	16	16	26
14	2-91	8	12	174	24	30	45
15	1-92	8	12	186	26	31	46
16	2-93	6	12	162	24	35	41
17	2-98	8	10	75	15	20	29
18	1-115	23	36	657	59	76	106
19	1-122	9	9	69	22	25	36
20	1-126	21	40	250	45	58	78
21	2-136	9	20	260	28	32	46
22	1-157	8	36	210	32	48	50
23	2-158	15	91	897	63	99	98
24	1-169	13	77	506	45	60	71
25	1-171	18	70	790	66	83	108
26	1-182	20	624	4641	127	165	180
27	2-183	23	624	5148	136	189	196
28	1-198	11	180	1050	65	94	98
29	1-217	38	640	5248	155	194	237
30	2-223	7	7	266	32	45	55
31	2-227	6	8	68	17	23	30
32	2-229	14	21	336	43	60	72
33	1-232	10	12	234	35	36	58
34	2-245	11	21	231	32	49	54
35	1-246	7	12	174	26	37	45
36	1-250	13	12	270	35	60	67
37	1-268	7	9	60	21	26	32
38	2-285	23	80	1056	72	85	106
39	1-313	13	49	469	43	46	89
40	2-332	10	8	92	24	28	39
41	1-355	9	28	336	32	37	50
42	2-355	13	72	702	47	46	73
43	2-359	17	56	287	33	36	71
44	2-412	172	1664	5678	482	537	826
45	1-435	34	216	1992	89	101	143
46	1-450	42	252	2912	105	118	177
47	2-89	9	16	232	28	34	47
48	1-89	7	12	156	23	30	40
49	47+48	21	42	722	58	70	98
50	4+48	11	36	540	39	42	72

Program 4 and program 6 are different programs, still the proposed measure rank these two programs as equally complex, i.e. equal complexity value. So, from the above-given description, it is clear that the proposed measure also satisfies the third property of Weyuker.

Property 4: (∃P) (∃Q) (P≡Q & |P|≠|Q|).

This property states that the two programs are implemented with different algorithms and the functionality of both programs is same, but the complexity of both implemented programs should be different from each other. Program 45 and 46 are considered to check whether this Weyuker property is satisfied by the proposed measure or not. Both programs are related to the stack implementation, the former uses an array and the later uses a linked list to implement a stack. The cognitive complexity of program 45 is=65+18+2+34+24=143. The internal structure of this program includes the sequential, branch, iteration and embedded components. Cognitive complexity of the program 46 is=77+27+3+42+28=177, and the sequential, branch, iteration, and embedded structures are incorporated. These two programs have the same functionality, but the implementation algorithm is different. According to the above-described example, it is quite clear that both programs are not equally complex, i.e. 143 # 177. Hence, C2M also holds this Weyuker property.

Property 5: (∀P) (∀Q) (|P|≤|P; Q| and |Q|≤|P; Q|).

This fifth property of Weyuker states that when the two program body are combined into a third single program body then the newly constructed program has higher complexity than both of the individual programs. Program 47, 48, and 49 are used to verify this property. Program 47 is used to find whether the number is prime or composite. Program 48 is used to find the factorial of a given number and program 49 is the combination the functionality of program 47 and 48 into a single program body. The cognitive complexity of the program 47 is=20+9+1+9+8=47, and the cognitive complexity of program 48 is=17+9+1+7+6=40. When the functionality of these two program is embedded into a single program, i.e. 49, then the cognitive complexity is=42+17+2+21+16=98. Sequential, branch and iterative statements are incorporated into the first two programs, but in the third program, one more embedded structure is incorporated. It is clear from the above-described example that, the cognitive complexity of individual programs are '47' and '40' which are less than the complexity of the combined program, i.e. (98>(47+40)). So, the fifth property is also satisfied by the proposed cognitive complexity measure.

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

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

Weyuker's sixth property declares that the two program bodies have the same cognitive complexity and these programs are separately concatenated into the third program. After that, the proposed measure should yield different complexity values of both the newly generated programs. Program 14 and 35 are taken into account to verify this property. Both programs have the equal complexity, i.e. '45'. Now, program 31 is integrated into both programs. After addition, the cognitive complexity of both programs will be '73' and '71', respectively. This is because of the new data type and one new executable line incorporated into the new generated program *R* (comprises of program 14 and 31). Hence, 73 # 71, indicates that the proposed C2M measure also hold this Weyuker property.

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|).

As aforementioned, the proposed approach utilizes the number of operands, operators, data types, executable LOCs, and cognitive weight of BCSs to measure the complexity of the software. The proposed cognitive complexity values will not change due to the permuting the order of the instruction of the source code. Thus, this property is not satisfied by the proposed measure.

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

The result of the proposed measure is always a positive numeric value, which does not depend on the file name. If the file is renamed, then there will be no effect on complexity value of the proposed measure. So, this eighth Weyuker property is obviously satisfied by the C2M.

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

The ninth Weyuker property states that, if the program size grows, then the complexity of the program should further increase. If programs 47, 48, and 49 are considered, the cognitive complexity value of the individual programs 47 and 48 is '47' and '40', respectively. Later, the functionalities of both programs are incorporated into a third program, i.e. program 49. The cognitive complexity of the newly generated program is '98' as described earlier in the fifth property, which shows that the summation of complexity values of individual program is smaller than the concatenated program, i.e. (87 (47+40)<98). This is due to the additional information and BCSs are added into the component bodies of the united program. A larger program is always harder to understand and unreliable than many similar small programs. So, this Weyuker property is clearly satisfied by the proposed C2M.

Weyuker property shows that the proposed measure is valid for measuring the complexity of the programs. These properties are necessary to satisfy by an effective complexity measure, but this does not provide sufficient condition for complete validation.

TABLE 2. Conformance of proposed and other cognitive measures to Weyuker's property

S. No	LOC	CFS	MCCM	CPCM	NCCoP	C2M
1	Y	Y	Y	Y	Y	Y
2	Y	Y	Y	Y	Y	Y
3	Y	Y	Y	Y	Y	Y
4	Y	Y	Y	Y	Y	Y
5	Y	Y	Y	Y	Y	Y
6	N	N	N	N	N	Y
7	N	Y	N	N	N	N
8	Y	Y	Y	Y	Y	Y
9	N	Y	Y	Y	Y	Y

The proposed measure is not a direct measure because it uses some different parameters in different sequences to calculate the complexity of programs. C. Karner [21] provides a more practical approach to validate the measures. The description of the practical approach to validate the proposed cognitive complexity measure is given below.

Measure's purpose: the main purpose of the measure is calculating the cognitive complexity of the program and on the basis of calculated complexity value the developers can self-analyze whether the complexity of the software is legitimate or not. If they feel any problem in the program behavior, further action can be accommodated to overcome the problem before it becomes critical at later stages.

Measure's scope: measure can be used after the development of the source code but not at earlier stages of the software development life cycle. This may be applied to earlier stages if some attributes of the source code can be predicted before the development of the code.

Measure's instruments: measurement of the proposed approach can be done manually or by using some automated tools.

Instrument natural variability while measurement: the proposed measure is simple and straightforward, so there is no variability while measuring the attributes of the proposed cognitive complexity measure.

Relation between parameters and the metric value: a direct relation exists between the parameters and the metric value because when the C2M value increases it means, the complexity increases and the quality of the product decreases with respect to time and space. The proposed metric is a quality indicator but not unique.

The effect of the automated instrument: once an automated tool is developed, there is no further

personnel required for calculation of attributes of software and only the automated tool cost will be imposed on the company. After analysis of the above discussion, it is found that the C2M measure has good capability to measure the complexity of the software.

4. RESULT EVALUATION OF C2M AND OTHER COGNITIVE COMPLEXITY MEASURES

This section deals with the analysis of the proposed and other cognitive complexity measures. For analysis of the cognitive complexity measures, 50 programs are collected from "Programming in C" [20]. Results of our approach and other approaches like: CFS [10], MCCM [11], CPCM [12], and NCCoP [13] are provided in Table 3.

In addition to it, another experiment is also conducted with the help of 10 undergraduate students of our institute. The programs are distributed among all the students to understand the program. The time taken by the individual student is recorded and the meantime of all the students is expected as the actual time required to understand the source code. The actual time (the meantime of the recorded data from students) and estimated time (calculated with the help of proposed measure) of all concerned programs is provided in Table 4. Actual time to understand (ATU) and estimated time to understand (ETU) the source code are quality parameters that can be utilized in testing as well as in maintenance phase.

Coding efficiency of all 50 programs is also calculated using proposed and other existing cognitive complexity measures. Equation (8) is used to calculate coding efficiency of the programs.

$$CE = Cognitive\_complexity\_value / LOCs \quad (8)$$

High value of CE indicates that more information is located into a small program, means, the complicated statements present in source code. With the help of CE factor, the productivity of the individual programmer can be estimated that can be useful to measure the development effort and time of the program.

Cognitive Functional Size (CFS) includes the input, output and the W_c of all BCSs to calculate the complexity of the software. The problem with CFS is that, this measure does not consider the local information, which is not a part of input and output. Another issue with CFS is that, if only one branch or iteration statement increases in the program, then the cognitive complexity according to CFS can be twice or thrice, respectively. The same problem also happens with MCCM, it can be verified from Table 3.

TABLE 4. Calculated coding efficiency, ATU and ETU of 50 programs

S. no.	CFS	MCCM	CPCM	NCCoP	C2M	ATU	ETU
1	1.7	2.4	2.6	2.4	4.4	1.5	2.2
2	2.5	10.0	1.6	1.1	3.1	3.6	3.0
3	0.5	3.4	2.9	2.9	4.6	3.6	5.1
4	2.4	16.0	3.6	4.0	6.0	2.3	2.1
5	0.8	5.8	4.0	3.8	7.2	2.0	2.5
6	0.6	3.0	2.3	2.1	4.3	1.8	2.1
7	0.7	3.7	2.7	2.6	5.0	2.0	2.5
8	1.4	9.4	7.4	7.3	10.9	3.7	5.3
9	1.8	25.9	3.8	3.7	6.8	4.6	5.3
10	3.4	12.0	2.6	2.7	4.7	2.4	2.3
11	0.3	7.0	4.7	11.0	8.7	1.6	1.8
12	0.2	7.2	4.7	11.8	8.5	2.6	3.6
13	1.6	12.8	3.2	3.2	5.2	2.1	1.8
14	1.5	21.8	3.0	3.8	5.5	3.3	3.1
15	1.5	23.3	3.3	3.9	5.8	3.5	3.2
16	2.0	27.0	4.0	5.8	6.8	3.2	2.9
17	1.3	9.4	1.9	2.5	3.6	2.2	2.0
18	1.6	28.6	2.6	3.3	4.6	6.6	7.4
19	1.0	7.7	2.4	2.8	4.0	2.8	2.5
20	1.9	11.9	2.1	2.8	3.7	5.4	5.5
21	2.2	28.9	3.1	3.6	5.1	3.9	3.2
22	4.5	26.3	4.0	6.0	6.3	4.1	3.5
23	6.1	59.8	4.2	6.6	6.5	8.2	6.9
24	5.9	38.9	3.5	4.6	5.5	5.6	5.0
25	3.9	43.9	3.7	4.6	6.0	8.0	7.6
26	31.2	232.1	6.4	8.3	9.0	13.8	12.6
27	27.1	223.8	5.9	8.2	8.5	13.7	13.7
28	16.4	95.5	5.9	8.5	8.9	7.9	6.9
29	16.8	138.1	4.1	5.1	6.2	27.7	16.6
30	1.0	38.0	4.6	6.4	7.9	3.9	3.9
31	1.3	11.3	2.8	3.8	5.0	2.2	2.1
32	1.5	24.0	3.1	4.3	5.1	4.8	5.0
33	1.2	23.4	3.5	3.6	5.8	4.2	4.1
34	1.9	21.0	2.9	4.5	4.9	3.8	3.8
35	1.7	24.9	3.7	5.3	6.4	3.4	3.2
36	0.9	20.8	2.7	4.6	5.2	4.6	4.7
37	1.3	8.6	3.0	3.7	4.6	2.6	2.2
38	3.5	45.9	3.1	3.7	4.6	8.8	7.4
39	3.8	36.1	3.3	3.5	6.8	5.3	6.2
40	0.8	9.2	2.4	2.8	3.9	2.8	2.7
41	3.1	37.3	3.6	4.1	5.6	4.6	3.5
42	5.5	54.0	3.6	3.5	5.6	6.4	5.1
43	3.3	16.9	1.9	2.1	4.2	4.9	5.0
44	9.7	330.1	2.8	3.1	4.8	57.5	57.8
45	6.4	58.6	2.6	3.0	4.2	11.1	10.0
46	6.0	69.3	2.5	2.8	4.2	13.6	12.4
47	1.8	25.8	3.1	3.8	5.2	3.6	3.3
48	1.7	22.3	3.3	4.3	5.7	3.1	2.8
49	2.0	34.4	2.8	3.3	4.7	7.8	6.9
50	1.8	18.7	2.8	3.2	4.2	4.3	3.9

LOC measure totally depends on the programmer. Skilled programmers can reduce the lines of code by using the smart techniques and some programmers increase the LOCs unnecessarily. Consider, the programs 1, 6, 7, and 8. All programs have seven executable LOCs. According to LOC, all programs are equally complex. But the cognitive complexity of these programs by different cognitive complexity measures, especially of program 8 is '10' (CFS), '66' (MCCM), '52' (CPCM), '51' (NCCoP), and '76' (C2M). CFS yields minimum complexity value among all cognitive complexity measures, it is due to the sequential structure of the program (since there is no branch or iteration in the program). From the aforementioned discussion, it is concluded that a small program may be more complex than a larger program.

In CPCM and NCCoP measure, the operators are ignored. Only the operands are utilized to measure the cognitive complexity of programs. If the internal information of the program is similar, then CPCM and NCCoP may generate the same result as a complexity value due to the sequential structure of the program.

Program 45 implements a stack using an array and program 46 implements the equivalent using a linked list. In both programs, internal information is almost identical and both measures rank the programs with almost equal complexity. But, it is far from reality because the implementation of any data structure using linked list is a challenging task than array implementation. We also test our proposed approach on same programs 45 and 46. The results of our proposed approach for programs 45 and 46 are '143' and '177', respectively. The difference indicates that the later program is very much complex than former program. So, the proposed work of this paper calculates the complexity of programs more accurately.

Table 4 lists the result of the coding efficiency of all cognitive measures along with the actual and the estimated time to understand the programs. The meantime of 10 students is assumed as an actual time required to understand the source code. On the basis of recorded actual time, the authors calculate a constant factor to measure the necessary time required to understand the source code. After analysis of the proposed cognitive complexity measure (C2M), it is found that the understand-ability factor is approximately 7% of the C2M value.

The correlation is also calculated to verify the result. The correlation between actual time and estimated time to understand the code is 0.98. This shows that the proposed approach has good capability to measure the understand-ability factor. This analysis of the cognitive complexity measures indicate the effectiveness of the C2M in several aspects and satisfies most of the features of a good measure.

5. CONCLUSION AND FUTURE WORK

In this paper, an attempt was made to present a new cognitive complexity measure which covers the different characteristics of the source code. The results of the proposed measure were compared with the other existing cognitive complexity measures. To compare the performance of the different cognitive complexity measures, 50 'C' language programs were utilized. The results of all measures about cognitive complexity, coding efficiency and ATU and ETU are provided in Tables 3 and 4. Weyuker property and a practical framework were used to validate the proposed measure. After analysis of Weyuker property, eight out of nine properties were met by the proposed cognitive complexity measure. The measure also satisfied several parameters required by the practical framework. Another important quality parameter "understand-ability" was also calculated with the help of 10 undergraduate students of our institute. The meantime was taken as the actual time required to understand the source code. It was found that the time required to understand the code is approximately 7% of the C2M value and the correlation factor between actual and estimated time is 0.98. The proposed measure is computationally simple and helps the developers and practitioners in evaluating the cognitive complexity of the software. It also establishes a relationship between the C2M value and the required time to understand the program. This measure satisfies most of functionality that a good measure should be satisfied for qualifying a worthy measure. Therefore, all these statistics reveal that, the proposed cognitive complexity measure (C2M) effectively calculates the complexity of software using different attributes of the software. This work can be extended in certain aspects like: estimating the time required to test the software, the time needed to debug and may be some other products metrics.

6. REFERENCES

1. Kearney, J.P., Sedlmeyer, R.L., Thompson, W.B., Gray, M.A. and Adler, M.A., "Software complexity measurement", ***Communications of the ACM***, Vol. 29, No. 11, (1986), 1044-1050.
2. McCabe, T.J., "A complexity measure", ***Software Engineering, IEEE Transactions on***, No. 4, (1976), 308-320.
3. Halstead, M.H., "Elements of software science (operating and programming systems series), Elsevier Science Inc., (1977).
4. Basili, V., "Qualitative software complexity models: A summary", Tutorial on models and methods for software management and engineering, (1980).
5. Oviedo, E.I., "Control flow, data flow and program complexity", in Software engineering metrics I, McGraw-Hill, Inc. (1993), 52-65.

6. Wang, Y., "On cognitive informatics", in Cognitive Informatics, 2002. Proceedings. First IEEE International Conference on, IEEE., (2002), 34-42.
7. Salehi, S., Taghiyareh, F., Saffar, M. and Badie, K., "A context-aware architecture for mental model sharing through semantic movement in intelligent agents", ***International Journal of Engineering-Transactions B: Applications***, Vol. 25, No. 3, (2012), 233-241.
8. Wang, Y., "On the cognitive informatics foundations of software engineering", in Cognitive Informatics, 2004. Proceedings of the Third IEEE International Conference on, IEEE., (2004), 22-31.
9. Weyuker, E.J., "Evaluating software complexity measures", ***Software Engineering, IEEE Transactions on***, Vol. 14, No. 9, (1988), 1357-1365.
10. Wang, Y. and Shao, J., "Measurement of the cognitive functional complexity of software", in Cognitive Informatics, 2003. Proceedings. The Second IEEE International Conference on, IEEE., (2003), 67-74.
11. Misra, S., *Modified cognitive complexity measure*, in *Computer and information sciences–iscis*., Springer.(2006) 1050-1059.
12. Misra, S., "Cognitive program complexity measure", in Cognitive Informatics, 6th IEEE International Conference on, IEEE., (2007), 120-125.
13. Jakhar, A.K. and Rajnish, K., "A new cognitive approach to measure the complexity of software's", ***International Journal of Software Engineering & Its Applications***, Vol. 8, No. 7, (2014).
14. Adamo Jr, D., "An experiment to measure the cognitive weights of code control structures", (2014).
15. Ghasemzadeh, M., "Complexity reduction in finite state automata explosion of networked system diagnosis", (2014).
16. Shehab, M.A., Tashtoush, Y.M., Hussien, W.A., Alandoli, M.N. and Jararweh, Y., "An accumulated cognitive approach to measure software complexity", ***Journal of Advances in Information Technology***, Vol. 6, No. 1, (2015) 145-161.
17. Jakhar, A.K. and Rajnish, K., "Measure of complexity for object-oriented programs: A cognitive approach", in Proceedings of 3rd International Conference on Advanced Computing, Networking and Informatics, Springer., (2016), 397-406.
18. Wang, Y., "The real-time process algebra (rtpa)", ***Annals of Software Engineering***, Vol. 14, No. 1-4, (2002), 235-274.
19. Harrison, W., "An entropy-based measure of software complexity", ***Software Engineering, IEEE Transactions on***, Vol. 18, No. 11, (1992), 1025-1029.
20. Thareja, R., "Programming in c", Oxford University Press, (2011).
21. Kaner, C., "Software engineering metrics: What do they measure and how do we know?", in In METRICS. IEEE CS, Citeseer., (2004).

Measurement of Complexity and Comprehension of a Program Through a Cognitive Approach

A. K. Jakhar, K.Rajnish

Department of Computer Science & Engineering, Birla Institute of Technology, Mesra, Ranchi, Jharkhand, India

P A P E R I N F O

Paper history:
Received 08 February 2015
Received in revised form 21 September 2015
Accepted 16 October 2015

Keywords:
Complexity
Basic Control Structure
Cognitive Informatics
Operators
Operands
Cognitive Weight

چکیده

پیچیدگی ذاتی سیستم های نرم افزاری مشکلاتی را در صنعت مهندسی نرم افزار ایجاد می‌کند. تکنیک های متعددی برای درک ویژگی های اساسی سیستم های نرم افزاری طراحی شده اند. برای درک نرم افزار، باید از سطح پیچیدگی کد منبع آگاه بود. انفورماتیک شناختی نقش مهمی برای فهم بهتر پیچیدگی های نرم افزار دارد. این انفورماتیک همچنین درک محققان از رفتار کد منبع، ساختار داخلی و کنترل را تسهیل می‌کند. در این مقاله یک اقدام جدید پیچیدگی شناختی (C2M) برای اندازه گیری پیچیدگی سیستم نرم افزار ارائه می‌شود که در ۵۰ برنامه 'C' تست شده است. اندازه گیری C2M ارائه شده است همچنین با کمک Weyuker ارزیابی می‌شود؛ هشت خاصیت از نه خصوصیت با معیار پیشنهادی مطابقت دارد. عملکرد معیار ارائه شده همچنین با دیگر معیارهای موجود پیچیدگی شناختی مقایسه شده است. داده های تست در میان ۱۰ دانشجو در مقطع کارشناسی از موسسه ما توزیع شده و از آنها خواسته شد تا کد منبع را بشناسند. زمان گرفته شده توسط هر دانش آموز ثبت شد و زمان متوسط داده های ثبت شده دانش آموزان به عنوان زمان واقعی مورد نیاز برای شناخت برنامه لحاظ گردید. در ادامه با زمان تخمین زده شده که از طریق اندازه گیری C2M محاسبه می‌شد مرتبط شد. از نتایج تجربی، مشاهده شد که معیار پیشنهادی نتایج با کیفیت بهتر ارائه می‌دهد

doi:*10.5829/idosi.ije.2015.28.11b.05*