A New Cognitive Approach to Measure the Complexity of Software’s

Amit Kumar Jakhar and Kumar Rajnish

Department of Computer Science and Engineering, Birla Institute of Technology, Mesra, Ranchi, India
amitkumar.cs08@pec.edu.in, krajnish@bitmesra.ac.in

Abstract

In software industries, the most important parameter is the exact measurement of the effort at the early phase of the software development life cycle. To estimate the effort is difficult due to dynamic behavior of the software. The basic need for this is the clarity in the software requirements like size, complexity of the project, human resource power etc. which is not correctly determined most of time. This paper presents a new cognitive complexity measures named it: New Cognitive Complexity of Program (NCCoP). First, the proposed metric is analytically evaluated against weyuker’s property for analyzing its nature and compare its result with other existing cognitive measurement evaluation results. Secondly, perform a comparative study of propose metric with existing metric and the results shows that the proposed cognitive metric do better that others metrics. And attempt has also been made to present the relationship among NCCoP, CFS, CICM, MCCM, and CPCM versus LOC to analyze the behavior of coding efficiency.

Keywords: Complexity, Cognitive informatics, Cognitive functional size, Basic control structures, Line of code

1. Introduction

According to IEEE’s definition [1], complexity is “the degree to which a system or component has a design or implementation that is difficult to understand and verify”. Software engineering community striving for some technique that can measure the complexity of software accurately, so that measuring the effort to develop and maintain the software is easy. In case of software everything is like un-measureable, because software cannot touch and visualized.

Cognitive Informatics (CI) is used in various research fields to search the solution of a given problem such as software engineering, artificial intelligence, and cognitive sciences [9], so cognitive informatics is an inter-disciplinary approach. Cognitive informatics plays a very important role to measure the software characteristics. To estimate the software system generally three types of models used i.e., Text mode, Diagrammatic model and Algorithmic model [2], all of these have its own advantages and disadvantages and may use more than one model or use separately to validate the estimation.

A common and traditional approach to measure the physical size of software system is LOC (Lines of Code) [5]; this technique is not much effective because of the flexibility of software. Albrecht [6] has developed a method that estimate the effort needed to develop software. This approach consists of five components” input, output, data files, interfaces, inquiries are rated simple, average and complex. The additional 14 characteristics has a great influence throughout the measurement, all are rated from 0 (no influence) to 5 (high influence). McCabe Cyclomatic Complexity (CC) [3] uses the flow graph to measure the
complexity with control flow without knowing the internal data objects. This uses a numeric number that is derived from graph. Basically it is designed for the testability and understandability of a module. Greater CC means high complexity i.e. hard to maintain and test and lower CC means low complexity i.e., easy to maintain and test than higher cyclomatic complexity number. Halstead’s 1977 [4] introduced a different way to measure the complexity that is based on the operators and operands within a program. This approach focuses on the internal structure of a program using number of distinct operators, number of distinct operands, total number of operators and total number of operand means considering input/output of software.

CI is a promising area from last decades in the field of research, cognitive complexity play an important role in software measurement. Measure the complexity of a software using cognitive approach find a way to fully understand the software in all aspects i.e. data objects: input, output, constant and variable, loops and branches so that it reflect difficulty for the developers to understand the software, can be used to predict the effort required to develop, test and maintain the software. The CI based on [7] found that the functional complexity of a software system is depending on three factors: input, output and architectural flow. The cognitive complexity takes both internal structure and input/output the software processing.

The main aim of the work presenting in this paper is to propose NCoP, which is based on the operands, internal behavior of the software and the data objects including I/O is taken in account by the NCoP with individual weight of Basic Control Structures (BCSs) of every LOC. Next, the proposed NCoP is analytically evaluated against Weyuker’s property along with results of other existing cognitive complexity metrics for analyzing its nature. Finally, perform the comparative study of NCoP with some existing cognitive measurements to observe the effect on program complexity and also analyze its coding efficiency by taking the relationship among proposed and other existing cognitive measurement with LOC.

The rest of the paper is organized as follows: Section 2 includes the related works on existing cognitive measurements. Section 3 presents the description of proposed cognitive measurement along with its analytical evaluation against Weyuker’s property. Section 4 presents a comparative study of proposed and existing cognitive measurements. Section 5 presents conclusion and future scope respectively.

2. The Existing Cognitive Complexity Measures

This section presents the description of existing cognitive complexity measures which are as follows:

2.1. CFS (Cognitive Functional Size) of Program

CFS proposed by Wang [10] to measure the complexity of a software program. The functional size of software depends on three parameters input, output and internal control flow. The internal control flow of a program derived from BCS’s

\[ \text{CFS} = (N_i + N_o) \times W_c \]

Where \( N_i \) is the number of inputs to the program and \( N_o \) is the number of output from the program and \( W_c \) is the entire cognitive weight of all BCSs. The program is basically structured in three formats: Sequential, Branch and Iteration structures [11, 20]. Hoarse [13] modeled two more BCSs namely Recursion and Parallel. The above two BCSs was again extended by Wang [11, 12] to covers interruption and function call. Two different scenarios are possible to calculate the \( W_c \) either all BCSs are in sequential manner or it contains one into another, the latter scenario is calculated by:
In eq. 1 \( W_c \) is the sum of q linear blocks comprises of individual BCSs. Every block can consist of m nesting BCSs, and each layer contains n linear BCSs. If the q block does not contain any BCSs embedded in it, means m=1 the eq.1 simplifies as follow:

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

(1)

Weights \( W_c \) of different BCSs are shown in Table 1.

<table>
<thead>
<tr>
<th>Category</th>
<th>BCSs</th>
<th>Weight ( W_c )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence</td>
<td>Sequence (SEQ)</td>
<td>1</td>
</tr>
<tr>
<td>Branch</td>
<td>If – then – else (ITE)</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Case (CASE)</td>
<td>3</td>
</tr>
<tr>
<td>Iteration</td>
<td>For-do (R,)</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Repeat-until (R,)</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>While – do (R,)</td>
<td>3</td>
</tr>
<tr>
<td>Embedded component</td>
<td>Function call (FC)</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Recursion (REC)</td>
<td>2</td>
</tr>
<tr>
<td>Concurrency</td>
<td>Parallel (PAR)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Interrupt (INT)</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 1. BCSs with its Cognitive Weights (\( W_c \))

Implementation of CFS is easy and independent to technologies. But it excludes some important details of cognitive complexity such as information that contains in the identifier and the operators. This limitation of CFS is overcome by many other measures that will discuss further.

2.2. CICM (Cognitive Information Cognitive Measure)

CICM was proposed by A. K. Misra and Kushwaha [14], it defines as:

\[
\text{CICM} = \text{WICS} \times W_c
\]

(3)

where WICS (Weighted Information Count of Software) and \( W_c \) is the BCSs weight.

Since software is a mathematical entity and information is contained in the program is function of identifiers, and the operators are used to perform operations on information.

\[
\text{Information} = f \text{ (Identifiers, Operators)}
\]

Information is supplied to the entire program.

\[
\text{ICS} = \sum_{k=1}^{\text{LOCs}} (I_k)
\]

ICS (Information Contained in Software), LOCs are number of lines in the software.

\[
\text{WICL} = f \text{ (Identifiers, operands, LOC)}
\]

and defined as:

\[
\text{WICL}_k = \text{ICS}_k / \text{LOCs} - k
\]

where WICL (weighted Information Count of \( k_{th} \) LOC)

Moreover the WICS is defined as:
WICS = \sum_{k=1}^{LOCs} WICL_k

As shown above, this is very complicated to calculate the complexity. Weighted information is calculated of each line. In this work it is shown that the information is the function of operators and operands, but the information is only contained in the operands, and operators are just used to perform some operation on operands.

2.3. MCCM (Modified Cognitive Complexity Measure)

It was developed by Sanjay Misra in 2006 [17], MCCM is formulated as:

\[
MCCM = (N_{i1} + N_{i2}) \times W_c
\]  

where \( N_{i1} \) is the total number of operators and \( N_{i2} \) is the total number of operands. It simplifies the complication that was associated with CICM. However, the information is multiplied with the weight \( W_c \) derived from entire BCSs remains its drawback.

2.4. CPCM (Cognitive Program Complexity Measure)

This measure was proposed by Sanjay Misra [15]. The work shows that the total number of occurrences of input and output is strongly effect of the cognitive complexity of software. The CPCM is defined as:

\[
CPCM = S_{IO} + W_c
\]  

\[
S_{IO} = N_i + N_o
\]

Where \( N_i \) is total occurrence of input variables and \( N_o \) is total occurrence of output variables.

Analytical evaluation of CPCM is done with the help of Weyuker’s property; seven out of nine properties are satisfied. Counting the number of input and output is not clear and ambiguously interpreted.


As discussed in the previous section that software is just nothing but a collection of information and the information is contained in the identifiers or variables. Till now various techniques of cognitive complexity are developed, some takes I/O; some takes total number of I/O and some takes the identifiers and operators and multiply these with the different weights of different BCSs separately. Therefore, the complexity is totally depends on the variables and the internal control structure BCSs. So, NCCoP technique is proposed to measure the cognitive complexity of a program. In this proposal, operators are not considered, just count the number of variables and constant line by line and multiply it with its BCSs weight that is shown in Table 1. Advantage of doing this is that we have the weight of each LOC and we can use this in counting the maximum weight of a module to reduce the chances of severe errors due to higher complexity of a module. The proposed NCCoP is formulated as:

\[
NCCoP = \sum_{k=1}^{LOCs} \sum_{v=1}^{LOCs} N_v \times W_c (k)
\]  

Where, the first summation is the line of code from 1 to the last LOC, \( N_v \) are the number of variables in a particular line of code, \( W_c \) is the weight that are shown in the Table 1 corresponding to the particular structure of line.
3.1. Example for Illustration of Proposed Technique

The given example shown below in Table 2 has been taken from [14]. In this example there are eight lines, and their respective complexity values of different cognitive complexity measures of executable lines is given below and the result of all measures is compared with the proposed measure.

Example 1: A program that find the sum of n numbers shown in Table 2:

- Line 2 = number of variable * $W_c$ (sequential i.e. 1) = 4 * 1 = 4 (means cognitive complexity of line 2 is 4)
- Line 4 = 1 * 1 = 1
- Line 5 = 5 * 3 = 15
- Line 6 = 3 * 1 = 3
- Line 7 = 1 * 1 = 1

Now, it is clear how to find the cognitive complexity of a program using proposed measure, in line 5 cognitive weight is 3 because of iteration (refer the Table 1).

So $NCCoP = 4 + 1 + 15 + 3 + 1 = 24$.

Hence, the cognitive complexity of the above program is 24 $NCCoP$. Measuring $NCCoP$ is very easy and can be helpful to measure the other attributes of the software like code density per LOC.

### Table 2. Find the Sum of n Numbers

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>main() {</td>
</tr>
<tr>
<td>2.</td>
<td>int i, n, sum=0; // BCS</td>
</tr>
<tr>
<td>3.</td>
<td>scanf(&quot;%d&quot;, &amp;n); // BCS</td>
</tr>
<tr>
<td>4.</td>
<td>for (i=1;i&lt;=n;i++) // BCS</td>
</tr>
<tr>
<td>5.</td>
<td>sum=sum+i; // BCS1</td>
</tr>
<tr>
<td>6.</td>
<td>printf(&quot;the sum is %d&quot;, sum); // BCS</td>
</tr>
<tr>
<td>7.</td>
<td>getch();}</td>
</tr>
</tbody>
</table>

MCCM = 14 (no. of operators and operands) * 4 (BCS weight 1 for Seq. +3 for branch) = 56. Analysis of existing cognitive complexity measures with the proposed technique is shown in the Table 3 with their cognitive complexity weight. The values shown in Table 3 for CFS, CICM, and CPCM have been calculated in [15].

![Figure 1. Comparison of LOC, CFS, CICM, MCCM, CPCM and NCCoP of Table 3](image)

### Table 3. Analysis Result of Various Cognitive Complexity Measures

<table>
<thead>
<tr>
<th>LOC</th>
<th>CFS</th>
<th>CICM</th>
<th>MCCM</th>
<th>CPCM</th>
<th>NCCoP</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>12</td>
<td>19.0</td>
<td>56</td>
<td>16</td>
<td>24</td>
</tr>
</tbody>
</table>
3.2. Analytical Evaluation of NCCoP against Weyuker’s Property

E. J. Weyuker’s [16] develop nine properties that are used to evaluate any complexity measure. These properties show the flaws of any proposed measure in practical manner. This is the main idea to evaluate the proposed proposal by own. Different examples have been taken from [14] to evaluate NCCoP.

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

The first Weyuker’s property states that the two programs should not have the same complexity number. The Figure 1 and Figure 2 of [14] are considered, the former contains iterations and the latter is sequential. The complexity of Figure 1 according to rows as: \( \text{NCCoP} = ((4*1) + (1*1) + (5*3) + (3*1) + (1*1)) = 24 \text{ NCCoP} \). The next Figure 2 as NCCoP = \( ((1*1) + (2*1) + (1*1) + (5*1) + (1*1)) = 10 \text{ NCCoP} \).

Now it is clear that the complexity of Figure 1 and Figure 2 are different, so this property is satisfied by the proposed measure.

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

Consider \( c \) is a non-negative integer. All the programming languages consist of input, output and some finite number of control structures. The cognitive complexity considers the variables, constant and BCSs weight to measure the complexity and the program is a finite line of code and some BCSs. Hence the NCCoP hold the second property.

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

Program that is given in Fig. 3 of [14] consists of two internal structures: sequential and iteration. Total cognitive weight of this program is: \( \text{NCCoP} = ((2*1) + (1*1) + (3*1) + (4*1) + (2*3) + (1*1) + (3*1) + (3*1) + (3*1) + (1*1)) = 27 \text{ NCCoP} \).

Consider Figure 6 of [14], three internal structures: sequential, branch and iteration. Cognitive complexity of this program is: \( \text{NCCoP} = ((2*1) + (1*1) + (2*2) + (5*3) + (3*1) + (1*1)) = 27 \text{ NCCoP} \). These examples show that the two different programs can have the same complexity i.e., 27. So the NCCoP holds the third property.

Property 4: \((\exists P) (\exists Q) \{P \equiv Q \& |P| \neq |Q|\}\).

This property states that the two programs are implementing with different algorithm should have different complexity. If we take an example of Figure 1 and Figure 2 of [14] the output of both the programs are same but the loop is replaced by the sequential formula. With this change the NCCoP of Figure 1 is 24 and for Figure 2 are 10. It is very clear that the two programs with same objects, there complexity is different. Hence the NCCoP holds this property.

Property 5: \((\forall P) (\forall Q) \{|P| \leq |P; Q| \text{ and } |Q| \leq |P; Q|\}\).

The program body of Fig. 4 is consider, this program consist of two program body, one for calculating the factorial and second for the finding the given number is prime or not. First, program contains one sequential, one branch and a function call. The second program also contains the same structure. The total cognitive weight of this complete program body is =((1*1) + (1*1) + (1*1) + (1*1) + (2*2) + (2*2) + (2*2) + (1*1) + (2*1) + (2*1) + (2*2) + (2*1) + (3*2) + (1*1) + (2*1) + (1*1) + (2*2) + (2*1) + (5*3) + (3*2) + (2*1) + (2*1) + (1*1)) = 70 \text{ NCCoP} \).

The Figure 5 and Figure 6 is calculated in the same manner as: complexity of Figure 5 is =((3*1) + (1*1) + (2*2) + (2*1) + (5*3) + (3*2) + (2*1) + (2*1) + (2*1) + (2*1) = 37 \text{ NCCoP} \). The complexity of Figure 6 is calculated in the same manner as: complexity of Figure 5 is =((2*1) + (1*1) + (1*1) + (2*2) + (5*3) + (3*2) + (2*1) + (2*1) + (2*1) + (2*1) = 37 \text{ NCCoP} \).
(3*1) + (1*1) = 27 NCCoP. Since, it is clear that the cognitive complexity of Figure 4 (P+Q) is greater than Figure 5 and Figure 6. Hence the NCCoP holds this property.

Property 6(a): \( \exists P \) \( \exists Q \) \( \exists R \) \(|P| = |Q| \) & \(|P;R| \neq |Q;R|\)

Let P and Q are the programs that are shown in Figure 3 and Figure 6 respectively. The NCCoP of P and Q is 27. Let R is another program that is illustrated in Figure 1. Program R is appending to program P, the result is shown in Figure 7.

Cognitive complexity of the program 7 is calculated i.e., 42 NCCoP. Similarly the program R is appending to program Q, complexity of these both together is 51 NCCoP. And 42 \( \neq \) 51, hence property 6 (a) is satisfied by the NCCoP.

Property 6(b): \( \exists P \) \( \exists Q \) \( \exists R \) (|P| = |Q|) & (|R;P| \neq |R:Q|)

If any numbers of statements are added into program P and program Q the complexity will not be changed. So, NCCoP does not hold this 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| \neq |Q|).

The complexity is measured according to data no matter how the data is spread into the program, so the NCCoP will not hold this property.

Property 8: If P is renaming of Q, then \(|P| = |Q|\)

The measure gives the numeric value so renaming the program will not affect the complexity of a program. Hence this property is clearly satisfied by NCCoP.

Table 4. Conformance of Measuring Techniques with Weyuker’s Property [16-19] (\(\sqrt{\quad}\): Means Which Satisfies Properties and \(X\): Which did not Satisfies Properties)

<table>
<thead>
<tr>
<th>S. No</th>
<th>LOC</th>
<th>McCabe’s Cyclomatic</th>
<th>Halstead’s effort</th>
<th>Data flow complexity</th>
<th>CFS</th>
<th>MCC</th>
<th>CPCM</th>
<th>NCCoP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(\sqrt{\quad})</td>
<td>(X)</td>
<td>(\sqrt{\quad})</td>
<td>(X)</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(X)</td>
<td>(X)</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>(X)</td>
<td>(X)</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(X)</td>
<td>(X)</td>
<td>(X)</td>
</tr>
<tr>
<td>8</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>(X)</td>
<td>(X)</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td>(\sqrt{\quad})</td>
<td></td>
</tr>
</tbody>
</table>

Property 9: \( \exists P \) \( \exists Q \) \(|P| + |Q| < (|P;Q|)\)

Additional complexity is introduced when a new a component of a program grows. If we consider the Figure 4-6 from [14], where the cognitive complexity of individual are 37 and 27, and the combination of both the programs into one program then the complexity is 70. This states that the new measure hold this property.
4. Comparative Studies between Proposed and Existing Cognitive Measures

4.1. Collection of Data and Flow Chart of Proposed Technique

This section describes the collection of experimental data. The source code for C programs has been collected from [14]. The flow of proposed model mention below in Figure 2.

![Flow Chart of Proposed Model](chart.png)

**Figure 2. Flow Chart of Proposed Model**

4.2. Empirical Data

The cognitive complexity values for different existing cognitive measures and propose measure are shown in Table 5 and the coding efficiency for individual programs for different cognitive measures along with proposed one with LOC are shown in Table 6. The graphs for comparison between existing cognitive measures and proposed measure are shown in Figure 3, Figure 4, Figure 5, Figure 6 and Figure 7.
Table 5. Cognitive Complexity Values of CFS, CICM, MCCM, CPCM and NCCoP

<table>
<thead>
<tr>
<th>Figure No. [14]</th>
<th>LOC</th>
<th>CFS</th>
<th>CICM</th>
<th>MCCM</th>
<th>CPCM</th>
<th>NCCoP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>12</td>
<td>19</td>
<td>56</td>
<td>16</td>
<td>24</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>2</td>
<td>3</td>
<td>10</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>16</td>
<td>19</td>
<td>88</td>
<td>22</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>34</td>
<td>42</td>
<td>97</td>
<td>722</td>
<td>68</td>
<td>70</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>30</td>
<td>19</td>
<td>187</td>
<td>29</td>
<td>37</td>
</tr>
<tr>
<td>6</td>
<td>12</td>
<td>14</td>
<td>30</td>
<td>90</td>
<td>19</td>
<td>27</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
<td>24</td>
<td>83</td>
<td>243</td>
<td>35</td>
<td>42</td>
</tr>
<tr>
<td>8</td>
<td>11</td>
<td>16</td>
<td>22</td>
<td>84</td>
<td>22</td>
<td>31</td>
</tr>
<tr>
<td>9</td>
<td>14</td>
<td>9</td>
<td>10</td>
<td>27</td>
<td>14</td>
<td>14</td>
</tr>
</tbody>
</table>

Table 6. CE (Coding Efficiency) of Individual Program of all Measures with LOC

<table>
<thead>
<tr>
<th>Program No. [14]</th>
<th>CFS</th>
<th>CICM</th>
<th>MCCM</th>
<th>CPCM</th>
<th>NCCoP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.5</td>
<td>2.4</td>
<td>7</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>0.3</td>
<td>0.4</td>
<td>1.4</td>
<td>1.1</td>
<td>1.4</td>
</tr>
<tr>
<td>3</td>
<td>1.5</td>
<td>1.7</td>
<td>8.0</td>
<td>2.0</td>
<td>2.5</td>
</tr>
<tr>
<td>4</td>
<td>1.2</td>
<td>2.9</td>
<td>21.2</td>
<td>2.0</td>
<td>2.1</td>
</tr>
<tr>
<td>5</td>
<td>1.5</td>
<td>1.0</td>
<td>9.4</td>
<td>1.5</td>
<td>1.9</td>
</tr>
<tr>
<td>6</td>
<td>1.2</td>
<td>2.5</td>
<td>7.5</td>
<td>1.6</td>
<td>2.3</td>
</tr>
<tr>
<td>7</td>
<td>1.5</td>
<td>5.2</td>
<td>15.2</td>
<td>2.2</td>
<td>2.6</td>
</tr>
<tr>
<td>8</td>
<td>1.5</td>
<td>2.0</td>
<td>7.6</td>
<td>2.0</td>
<td>2.8</td>
</tr>
<tr>
<td>9</td>
<td>0.6</td>
<td>0.7</td>
<td>1.9</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Figure 3. Comparison of CFS, CICM, CPCM, MCCM and NCCoP
Figure 4. Comparison of CFS, CICM, CPCM and NCCoP

Figure 5. Plot of Coding Efficiency of CFS, CICM, MCCM, CPCM, and NCCoP

Figure 6. Plot of Average Coding Efficiency of all 9 Programs
4.3. Discussion

The experimental result of the proposed technique is discussed in this section. The performance of the proposed technique is validated on the nine C programs which is taken from [14] and compared with LOC, CFS, CICM, MCCM and CPCM. The experimental results of the proposed technique and other techniques which have been used for comparisons are shown in Table 5.

Figure 3 shows the behavior of MCCM, it obtains very high value that is not expected and not useable. Thus due to the above reason, the MCCM is excluded in Figure 4. On the other side, LOC is used to count the physical size of program with disregard of complexity inside the program. If a program exhibit the sequential structure then CFS, CICM, MCCM and CPCM consider the whole weight is 1, because the weight of the sequential BCS is 1. But every line it has its own weight, since these measures did not consider the weight for consecutive sequential LOC. So, in the proposed technique NCCoP consider the weight of every line according to its structure.

CICM obtains high value of programs 4 and 7 of Table 5 due to high information contents at the end of the program. Programs 2 and 5 of Table 5 is complex according to NCCoP because it uses a sequential formula instead of iteration, but the remaining measures have no capability to measure such complexity. The same can show again in program 8.

The Table 6 shows the coding efficiency of all measures with LOC. The coding efficiency is derived as:

\[ C_E = \frac{\text{Value of measure}}{\text{LOC}} \]  

The individual coding efficiency and average coding efficiency of 9 programs are shown in Figure 5 and Figure 6 respectively. The average coding efficiency of the proposed measure is ranging from 1.0 to 3 per LOC throughout the programs. In other words 1.0 LOC = 2.1 NCCoP (shown in Figure 6) on an average.

The correlation coefficient is a statistical measure that measures the relationship between two variables, if one variable is changing its value then the value of second variable can be predicted. The positive correlation exists when a high value of one variable is associated with high value of another variable and the negative correlation exists when a high value is
associated with low value. The value of correlation is varying from +1 to -1. The value is close to +1 means positive correlation exists and -1 means negative correlation exists and 0 means there is no correlation at all. The Figure 7 clearly shows that the relationship between NCCoP with the existing cognitive measures technique is near to +1. Since, there exists a high degree of correlation. Pearson and spearman correlation coefficient are used to find the correlation between proposed measure and other cognitive measures.

The new proposed NCCoP and other existing cognitive complexity measures and their result are shown in the Section 4. Program 1 has 8 and program 9 has 14 LOC but the NCCoP is 24 and 14 respectively, hence line of code increases it doesn’t mean to increase in complexity. Program 1 and program 2 have LOC is 8 and 7 respectively, CFS, CICM, and MCCM complexity values are changing drastically except the CPCM and NCCoP. But when the program 3 and 8 are considered then the complexity is same in case of CPCM, but NCCoP gives the more accurate result. Again, the program 1 and program 9 have complexity 16 and 14 according to CPCM respectively which is almost similar, but in reality the difference between these two is much more, that is indicated by the proposed technique very clearly. Later, coding efficiency of every measure is calculated that is shown in Figure 5 (individual coding efficiency of every program) and Figure 6 (Average coding efficiency of every program). At the last the Pearson and Spearman correlation of all existing measures are calculated with NCCoP shown in Figure 7.

5. Conclusion and Future Scope

In this paper an attempt has been made to propose new cognitive complexity measure called NCCoP (New Cognitive Complexity of Program), which consider the weight of different BCSs are considered to measure the complexity. The paper also presented different existing cognitive complexity measure techniques which are analyzed and discussed the limitation of existing measures. It is found that the proposed technique is more suitable measure when it is compared with other existing measures. The proposed technique multiplies the cognitive weight of individual LOC structure with number of operand of that LOC.

The result shows that the proposed technique exhibit the complexity of program very clearly and accurate than other existing cognitive measures. The analytically, the proposed measure is evaluated through the most famous Weyuker’s property, seven out of nine properties are hold by the proposed measure. Pearson and Spearman correlation coefficient methods are used to find the relationship between the different existing cognitive complexity measures with proposed measure, the experimental result shown in the Section 4.

The future scope includes some fundamental issues:

(a) The proposed work can extended to estimate the effort and development time required to develop the project.

(b) This work can also be extended for measuring the effort required to test the software in testing phase and in maintenance phase.

References

Authors

Mr. Amit Kumar Jakhar, he is student in the Department of Information Technology at Birla Institute of Technology, Mesra, Ranchi, Jharkhand, India. He received his M.E. in Computer Science & Engineering from PEC University of Technology, Chandigarh, India in the year of 2010. He received his B.E. (Honours) from MDU, Rohtak, Haryana, India in the year of 2008. His research area is software engineering.

Kumar Rajnish, he is an Assistant Professor in the Department of Information Technology at Birla Institute of Technology, Mesra, Ranchi, Jharkhand, India. He received his PhD in Engineering from BIT Mesra, Ranchi, Jharkhand, India in the year of 2009. He received his MCA Degree from MMM Engineering College, Gorakhpur, State of Uttar Pradesh, India. He received his B.Sc. Mathematics (Honours) from Ranchi College Ranchi, India in the year 1998. He has 30 International and National Research Publications. His Research area is Object-Oriented Metrics, Object-Oriented Software Engineering, Software Quality Metrics, Programming Languages, and Software Estimation.