ISSN: 2277–3061 (online)
International Journal of Computers & Technology
Volume 2 No.1 February 2012

Code Comprehending Measure (CCM)

Gurdev Singh
Professor and Head
Department of Computer Science & Engineering, Adesh Institute of Engineering & Technology, Faridkot, India.

Satinderjit Singh
Associate Professor and Head
Department. of Computer Application GGNIMT, Civil Lines, Ludhiana, India

Monika Monga
Assistant Professor:
Department of Computer Science & Engineering, Adesh Institute of Engineering & Technology, Faridkot, India.

Abstract
software complexity, accurately, plays a vital role in life cycle of the software. Many metrics have been proposed in the past like LOC, McCabes' cyclomatic measure, Halstead's measures and cognitive measures. This paper proposes a new method to measure the software complexity, by not only taking into account the internal structure of the algorithm in terms of the total cognitive weights of the basic control structures but also by quantifying the flow of data between the various basic control structures and data volume factor (variables and operators) used within basic control structure. The preliminary tests show that this metrics is independent of the existing measures. Comparison with some existing measures has been done to prove the robustness of this new metrics.

Keyword
Software Complexity, Complexity Metrics, Cognitive Weights, Data Flow Factor, Data Volume Factor.

I. INTRODUCTION

It is nothing new to state that the software systems are extremely complex entities. From the last few decades it has been the endeavour of the software industry to find a good measure of the software complexity. Any measure that will predict the complexity of a software taking into account all the important factors influencing the complexity and also the human effort in understanding of the structures that make up the software, will be of great use and value to the software industry and the study of software engineering as a whole.

Earlier measures of software's complexity typically depended on program size like counting the number of lines of codes [8], then some improvement was made by taking into consideration data flow and module interfaces such as the Halstead's software metrics [6] and measure of cyclomatic complexity developed by McCabe [7] became very popular. Halstead's metrics calculates the number of operators and operands, but gives no consideration to the structural complexity of basic components, while McCabe's cyclomatic complexity does not consider data flow between components of a system.

In 2003 Yingxu Wang and Jingqiu Shao proposed a new measure of software complexity based on the cognitive weights [1]. This was a revolution of sorts as for the first time a metrics was proposed based on the human effort to understand the complexity hidden in the basic control structures that makeup the component of any software. This method had some drawbacks as shown by [2].
The brief organization of this paper is as follow. The section 2 discusses some of the popular metrics to measure the software complexity and their drawbacks. In section 3 we will discuss the Code Comprehending Measure (CCM) and section 4 compares CCM with other metrics.

SOME POPULAR SOFTWARE COMPLEXITY MEASURES

Lines of Code

'Lines of Code' is a metrics that measures the physical size of the code. This is a popular metrics to measure the software complexity as it gives a fair idea of the number of developers required to do the work.
In this measure we count the relevant lines of code and may chose to ignore the comments and blank lines. The measurement of LOC is very simple, but has some major drawbacks like it is dependent on programming languages, application areas, and programmer's skills. So this measure encourages the inefficient programming practices. Also, LOC's measure of complexity is heavily influenced by factors like difficulty of algorithms, and other functional requirements.
McCabe's Cyclomatic Complexity

Thomas McCabe in 1976 introduced the concept of cyclomatic complexity [7]. This concept was based on graph theory. If we can draw a connected graph G of the function then this metrics calculates the cyclomatic number V(G) of a graph G with n vertices, e edges, and p connected components as

$$V(G) = e - n + p.$$

This metrics counts the number of enclosed areas in the graph G. This measure gives a good idea about the structural complexity of a function. But a major limitation of this measure is that it ignores the size of a component, it also ignores the data flowing from one component to another. These two together can contribute massively to the complexity of the software and should not be ignored.

A. Halstead's Complexity Metrics

In the year 1977 Maurice Halstead proposed a set of six computational metrics. This concept counted distinct number of operators (n1) and operands (n2) and total number of operators (N1) and operands (N2). Based on these values Basic, Derived & Estimated measures were calculated.

In this complexity metrics a major drawback is that, this does not take into account structural complexity of any component. It also does not consider the flow of data from one component to another. It is also said that some of these measures are not relevant and can be argued upon.

B. Wang's Cognitive Complexity Measure

In the year 2003 Yingxu Wang and Jingqiu Shao introduced the concept of cognitive functional complexity of soft wares. In this

metrics the different Basic Control Structures (BCS) are assigned different cognitive weights. BCS are the set of fundamental and essential flow control mechanisms that are used for building logical architecture of software [1]. In this metrics the total cognitive weight of a component is measured by either adding the weights of two or more BCS if they are in sequence or the cognitive weight of a BCS is multiplied with the weight of another BCS if it is embedded in the other BCS. Then the Cognitive Functional Size (CFS) is calculated by the following formula [1].

$$= (Ni + No) \cdot \{ \sum_{j=1}^{q} [\prod_{k=1}^{m} \sum_{i=1}^{n} W_c(j,k,i)] \}$$

Where Ni is the number of inputs and No is the number of outputs.

In this metrics the different BCS are assigned the weights as shown in table 1. These weights are based on the human effort in comprehending these BCS. In Code Comprehending Measure (CCM), as we discuss later, these weights are also used as they have been proposed by Wang and Shao in [1].

TABLE 1

COGNITIVE WEIGHTS OF DIFFERENT BCS.

BCS	Cognitive Weight
Sequence	1
Branch If–Then-Else	2
Case	3
Iterations	3
Function Call	2
Recursion	3
Parallel	4
Interrupt	4

This was a revolutionary new concept in software metrics. But there are some shortcomings. First of all this metrics does not fulfill all the properties as proposed by E J Weyuker in [5].
Wang's measure does not take into consideration the data flow complexity of a component which is not embedded in one another [2]. Also this metrics does not take into consideration the internal data volume complexity of any BCS. This is explained with the following two examples.

TABLE II
EXAMPLE 1

```
for(j=2; j<i; j++)
{
        if(i%j= =0)
                break;
}

if(i = = j)
        printf("\t%d",i);
```

In table 2 example 1 Wang's measure considers the 'for' and the 'if ' structures independently. But the two structures cannot be considered independently, as data flows from one structure to the other, and in doing so it carries with it some complexity. This is true because we cannot understand the 'if' structure independently without considering the preceding 'for' structure.
Another shortcoming in Wang's CFS method is that it considers all the similar BCS as same regardless of any data complexity where as in reality two similar BCS may not be exactly same. This is because of the fact that the number of variables and operators which makeup the internal complexity of a BCS may be different in different BCS. Consider the following example.
TABLE III
EXAMPLE 2

```
for(a,b,c  ;a>=b+c;  a--,b++,c++)
{
      printf("\n%d",a);
}
```

2(a)

```
for(a  ;  a<10  ;  a++)
{
      printf("\n%d",a);
}
```

2(b)

There are two loops (2a & 2b) in the above example, it is clear that the first for loop is more difficult to comprehend in comparison to the second for loop. This can be attributed to the number of variables and operators in any BCS which makeup its internal data volume. Wang's measure fails to consider the internal data volume complexity of a BCS. As invariably it considers all iterations as having the same cognitive complexity and so is true for other BCS.

CODE COMPREHENDING MEASURE (CCM)

The Code Comprehending Measure derives the complexity of a function from the following three factors:

Data Volume Factor (DVF)
Structural Complexity Factor
Data Flow Factor (DFF)

Data Volume Factor (DVF) primarily calculates the total number of distinct variables and operators used in any BCS and it calculates the total number of occurrences of the variables and operators used in that BCS. It is quite reasonable to state that more the number of variables and operators in any BCS, more is the complexity of the BCS.
Structural Complexity Factor takes into consideration the cognitive weights (Wc) of any BCS. These are the same as those given by Wang in [1], with the same calculating method.

Data Flow Factor (DFF) considers the complexity arising due to flow of data from one BCS to another BCS. If two BCS are linearly arranged then we cannot always consider them in independence as data may flow from one such structure to another. When data flows from one BCS to another it also takes along with it some inherent complexity. This fact is tackled by the DFF

Definition 1: Code Comprehending unit (CCU) is the unit of the CCM. One CCU is defined as the complexity of the software component having only one sequence BCS and no data variable and operators and no data flowing from other BCS.
In CCM complexity of a function is calculated by finding out the DVF, DFF and Wc for each of the BCS trees which are linearly arranged to form the function. Therefore for i BCS trees each BCS

tree having j nested BCS and each nested BCS having k number of linearly arranged BCS [1], the CCM can be calculated as

$$\sum_{i=1}^{n}[DVF_i * Wc_i * DFF_i]$$

This gives us the CCM for i BCS trees which are linearly arranged to form the function.

Data Volume Factor

The DVF is calculated by finding out the total number of distinct variables and operators used in a BCS and their total number of occurrences, only arithmetic, logical, comparison operators are considered for this purpose. By using the following equation DVF is calculated

[1+ {N*log10(1+n)} ½]

N is the total number of variables and operators used in a BCS tree.
n is the number of distinct operators and variables occurring in that BCS tree.

Data Volume Factor

The Data Volume Factor is the amount of complexity inbuilt in a BCS as result of number of variables and operators in it.
It is worth noting that we do not consider the operators like comma, parenthesis, array's indexes etc which do not add to the complexity.

Structural Complexity Factor

The Structural Complexity Factor of CCM calculates that part of the total complexity of any function which is due to the architecture of the function. This is done by calculating the cognitive weights of the linearly arranged BCS trees that form the function. Wang and others calculate the cognitive weight of the function in [1] and [9].

Data Flow Factor

When the data flows from one BCS into another BCS it invariably takes along with it the cognitive complexity of the block in which it was last modified. This is because of the fact that the BCS into which the data has flown into cannot be completely understood without understanding the BCS from which the data is coming. To understand the complexity of any code this important factor cannot be ignored. This factor of flow of data is calculated in CCM as follows

$$\prod_{j=1}^{m}\left(W_j^{1/4}\right)$$

If there are m numbers of variables which are not initialized or declared in this BCS or its direct parent lineage, then it is carrying along with it some complexity. This is DFF which is calculated by the above equation. Here we take the fourth root of Wc of the BCS in which this variable was last modified. The product of all such fourth roots makes up the DFF for this BCS tree.

Issues to be considered for DFF

We calculate data flow factor for a BCS by identifying the variables coming from outside the BCS and taking the fourth root of the cognitive weight of BCS in which that variable has been last modified. However there are some issues related to the calculation of data flow factor which need to be kept in mind.

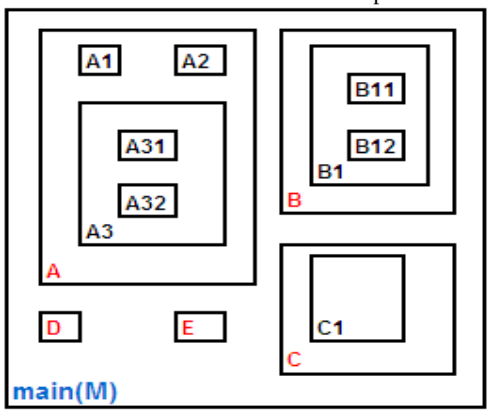

Fig. 1 Block diagram with various BCS

Those variables initialized or modified in the parent BCS do not carry any DFF complexity to the child BCS. This is because data flow factor for these variables is already taken into account while calculating the cognitive weight of that BCS tree. It is only in the case of data flow between BCS which are not in the same branch of the tree that the variables carry data flow complexity and we need to take fourth root of cognitive weight of BCS from where data is flowing. For example in the diagram shown in table 4 we need not take data flow factor for any variable flowing between A3 and A32, or A & A31. However we have to consider DFF in case of variable flowing from A1 to A32.

In case of variable flowing from a BCS embedded inside a BCS tree to a BCS which is outside this tree, cognitive weight of not only the inner BCS is taken into account but cognitive weight of the whole branch of that BCS is considered for DFF. The branch continues till we reach a level where either there is no parent or the receiving BCS has the same ancestor as that of the branch considered so far. For example if the data is following from A31 to A2 we need to take the weight of A31 and A3 only and takes the fourth root of (A31*A3) to calculate data flow factor. We need to stop at A3 because recipient A2 is sibling of A3.

In case of data flowing from A32 to C1, we need to take the cognitive weights of A32, A3 and A. We need to end at A because A is sibling of C which in turn is the parent of C1.

Data flow factor is not commutative in nature i.e. data flowing from X to Y is not the same as that flowing from Y to X. In above table 4 data flowing from A31 to A2 through one variable is calculated by taking the fourth root of cognitive weight of A31 and A3. On the other hand consider the case of data flowing from A2 to A31 through one variable. Here data flow factor is calculated by taking the fourth root of cognitive weights of A2 only. This is done so because A2 is already sibling of A3- parent of A31 (sender).

In case where the particular variable seems to be coming from more than one BCS , in that case data flow factor is calculated by adding the cognitive values from all the BCS concerned and then taking the fourth root of the combined value. The essence for this is that whenever the variables can come from more than one BCS then the data flow complexity is enhanced. For example if a variable in C1 is possible coming out of A1, B12 and D, Then the data flow factor of

the variable is calculated by taking the fourth root of (A1*A + B12*B1*B + D)
A variable already considered for DFF calculation will not henceforth be considered in its subsequent use within the same BCS tree.
Total CCM of the Function

The total Code Comprehending of a function is calculated as the sum of CCM of each of the BCS trees which are linearly arranged to form the function. This is shown in the following equation.

$$\sum_{i=1}^{m}\left[\left[1+\{N*\log_{10}(1+n)\}^{1/2}\right]*Wc_i*\left[\prod_{j=1}^{x}(w_j^{1/4})\right]\right]$$

For m such BCS trees which are linearly arranged to form a function. The above equation gives the CCM of the function.

F. CCM with Example

The program given in table 5 below is a simple program of LINEAR SEARCH. There are 4 BCS trees present. First there is a sequence, then there is a for loop, followed by another for loop, and this is followed by an if BCS. We need to calculate the DVF, WC, and the DFF for these 4 BCS.

TABLE IV
LINEAR SEARCH

```
void main()
{
  int a[100],sz, num, i ;
  clrscr();
  printf("Enter the size of array : ");
  scanf("%d",&sz);

  for(i=0;i<sz;i++)  {
     printf("\n\nEnter element no %d :",i+1);
     scanf("%d",&a[i]);
}
printf("\n\nEnter the no to search :");
scanf("%d",&num);
for(i=0;i<sz;i++){
   if(a[i]==num){
     printf("The number is at position %d",i+1);
        break;
   }
}

if(i==sz){
     printf("Number is not present");  }
}
```

The total CCW of the program is 58.453. This has been calculated based on the values shown in the figure 1 which shows the block diagram of the code in table 5.

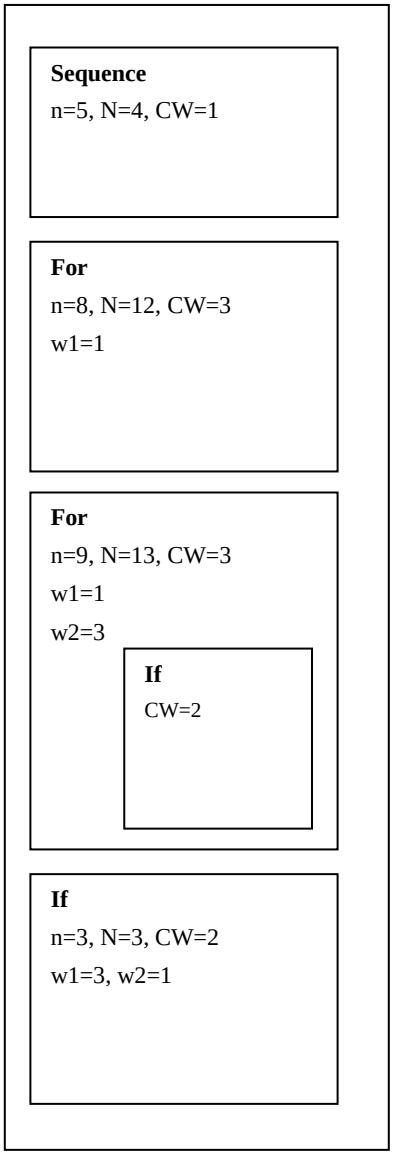

Fig. 2 Block Diagram of Linear Search

COMPARISON WITH OTHER METRICS

Complexity Values for different Programs

In order to prove the effectiveness of CCM measure we calculated its value for a set of 15 programs and compared its value with some existing metrics. The code of these 15 programs can be seen at the following URL, created by the authors http://ComprehensiveComplexityMeasure.blogspot.com
Some intresting observation are made when comparing CCM with other measures.

Using LOC we could not distinguish between program (4 & 5) and also between program (10 & 11) and programs (12 & 13) in term of the complexity. But CCM not only breaks the tie but also gave fair idea of how much one program is more complex. For example program 13 is more compex than program 12 by 1.28 times.
The issue is more grim in case of McCabe's Values, where 4 programs(P.No 1,5,6,15) are tied with value 2 and 4 programs(P.No 3,8,9,10) tied with value 5 and 2 program (P.No 12,14) tied with value 6. The CCM not only breaks the tie with exact values but also for example tells that progran 12(with MV of 6) is less complex to comprehend than program 9 (with MV of 5).
The result of the comparison is tabulated in Table below:
TABLE V
COMPLEXITY VALUES OF DIFFERENT MEASURES

No.	Description	LOC	M.V	CFS	CCM
1	Rev. triangle pattern	13	2	26	56.91
2	Palindrome	14	1	3	7.38
3	Prime No's in Range	15	5	25	174
4	LCM OF 3 No's	17	4	40	57.99
5	Fibonacci series	17	2	8	55.42
6	Under root of 3i	18	2	10	59.51
7	Tower of Hanoi	19	1	51	77.06
8	Linear search	27	5	48	58.45
9	Insertion sort	29	5	38	187.7
10	Bubble sort	31	5	100	266.6
11	Mtrx multiplication	31	8	111	319
12	Binary search	34	6	63	154.6
13	Fighter-Bomber	34	3	32	197.5
14	Selection sort	38	6	102	570.9
15	value of Pi	39	2	30	75.14

With CFS although there is no tied values but because of the fact that it does not consider data volume and data flow factor into consideration the CFS values for programs may appear too close than they actually are in term of complexity. For example programs 13 and 15 has CFS values of 32 and 30. However their CCM values are 197.5 and 75.14 respectively, reflecting the fact that data volume factor is relativly much higher in program 13 than program 15. This factor is totally ignored by CFS.
Corelation of CCM with other Metrics

The following table VI shows the correlation coefficient of CCM with 3 other measures. It clearly shows that this metrics is not closely related to any of these.
TABLE VI
CORRELATION COEFFICENT OF CCM WITH OTHER METRICES

LOC	0.6139708
MV	0.6809707
CFS	0.7910151

Ranking of 15 Programs for Different Measures

The table VIIshows the ranking of these set of 15 programs for different metrics. In this table Higher the value more complex is the program.
TABLE VII
RANKING OF 15 PROGRAMS FOR DIFFERENT MEASURES

No	Description	LOC	MV	CFS	CCM
1	Rev. triangle pattern	1	3	5	3
2	Palindrome	2	1	1	1
3	Prime Nos	3	9	4	10
4	LCM OF 3 No's	4	8	9	4
5	Fibonacci series	4	3	2	2
6	Under root of 3	6	3	3	6
7	Tower of Hanoi.	7	1	11	8
8	Linear search	8	9	10	5
9	Insertion sort.	9	9	7	11
10	Bubble sort	10	9	13	13
11	Mtrx multiplication	10	15	15	14
12	Binary search	12	13	12	9
13	Fighter-Bomber	12	7	6	12
14	Selection sort	14	13	14	15
15	value of Pi	15	3	8	7

II. CONCLUSION

In this paper we have tried to propose a new software complexity metric. This metrics "Code Comprehending Measure" tries to measure three aspects of the complexity of a function, which are Data Volume, Structural Complexity and Data Flow complexity. We believe that software are analogous to human beings which have a bone structure, muscular weight and blood flowing in the veins. The bone structure in case of software component is the cognitive weight of the component, muscle part is the data volume factor and blood flow part is the data flow factor. CCM combines the effect of these three factors influencing the software complexity into one metric. This paper tries to show how the CCM of a function is calculated. Lastly this paper compares Code Comprehending with three other popular software measures, and the results are also shown.

III. REFERENCES

[1] Yingxu Wang and Jingqiu Shao, "Measurement of the Cognitive Functional Complexity of Software", proceedings of the 2nd IEEE International Conference on Cognitive Informatics, .67, August 18-20, 2003

[2] Underlying Cognitive Complexity Measure Computation with Combinatorial Rules By Benjapol Auprasert, and Yachai Limpiyakorn World Academy of Science, Engineering and Technology Vol 35 November 2008

[3] Misra, S. and Misra, A.K "Evaluating cognitive complexity measure with Weyuker properties", cognitive Informatics, 2004. Proceedings of the Third IEEE International Conference on, 16-17 Aug. 2004

[4] Cognitive Complexity Metrics and its Impact on Software Reliability Based on Cognitive Software Development Model By Dharmender Singh Kushwaha and A.K.Misra

[5] E. J. Weyuker, "Evaluating Software Complexity Measures, IEEE Transactions on software Engineering", v.14 n.9, p.1357-1365, September 1988

[6] M.H. Halstead, Elements of Software Science, North Holland, N.Y.: Elsevier, 1977.

[7] T.H. McCabe, "A complexity measure," IEEE Trans. Software Eng., vol. 2, no. 4, Dec. 1976, pp 308–320

[8] Albrecht, A.J. and J.E. Gaffney (1983), Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation, IEEE Transactions on Software Engineering, Vol.9.

[9] Jitender Kumar Chhabra , Code Cognitive Complexity: A New Measure, Proceedings of the World Congress on Engineering 2011 Vol II London, U.K..

[10] A.Aloysius, L. Arockiam , A Survey on Metric of Software Cognitive Complexity for OO design, World Academy of Science, Engineering and Technology 58 2011.

[11] Sanjay Misra and Ibrahim Akman, A New Complexity Metric Based on Cognitive Informatics, Proceedings of 3rd International Conference on Rough Sets and Knowledge Technology, 2008, pp.620–627.

[12] Deepti Mishra and Alok Mishra, Object-Oriented Inheritance Metrics: Cognitive Complexity Perspective, Proceedings of the 4th International Conference on Rough Sets and Knowledge Technology, 2009, pp. 452–460.

[13] Ghazal Keshavarz, Nasser Modiri , Mirmohsen Pedram , Metric for Early Measurement of Software Complexity , International Journal on Computer Science and Engineering , Vol. 3 No. 6 ,June 2011.

[14] Ashish Sharma, D.S. Kushwaha, , A Complexity measure based on Requirement Engineering Document, journal of computer science and engineering, volume 1, issue 1, may 2010.

[15] Manik Sharma , Gurdev Singh , Analysis of Static and Dynamic Metrics for Productivity and Time Complexity, International Journal of Computer Applications, Vol.30 No. 1,7-13, September 2011.

[16] Gurdev Singh, Dilbag Singh et. al "A Study of Software Metrics" International Journal of Computational Engineering and Management. vol. 11. 2230-7893