Comparative Study of Cognitive Complexity Measures

Sanjay Misra, Ibrahim akman
Department of Computer Engineering, Atilim University, Ankara, Turkey
Email: smisra@atilim.edu.tr; akman@atilim.edu.tr

Abstract

Complexity metrics are used to predict critical information about reliability and maintainability of software systems. Cognitive complexity measure based on cognitive informatics, plays an important role in understanding the fundamental characteristics of software, therefore directly affects the understandability and maintainability of software systems. In this paper, we compared available cognitive complexity measures and evaluated cognitive weight complexity measure in terms of Weyuker's properties.

***Keywords*:** Software metrics, cognitive weights, basic control structures, Weyuker's Properties

1. Introduction

Software metrics have always been important for software engineers to assure software quality because they provide approaches to the quantification of quality aspects of software. However, absolute measures are uncommon in software engineering [9]. Instead, software engineers attempt to derive a set of indirect measures that lead to metrics, which provide an indication of quality of some representation of software. The quality objectives may be listed as performance, reliability, availability and maintainability and are closely related to software complexity [12].

The complexity measures based on cognitive informatics [10] are still in the developing phase. Cognitive complexity measures are the human effort needed to perform a task or difficulty in understanding the software. Numbers of researchers have proposed metrics based on cognitive informatics [1], [2], [4] and [11]. In this paper, an attempt has been made to analyze and compare the measures based on it. Further, we suggested to adopt a simple method for calculating the complexity of code in terms of cognitive weights. The cognitive weight of software [12] is the extent of difficulty or relative time and effort for comprehending given software. Further, IEEE defines Complexity and quality metrics as "The degree to which a system or component has a design or implementation that is difficult to understand and verify" and "a function whose input are software data and whose output is a single numerical value that can be interpreted as the degree to which the software possesses a given quality attribute" respectively. In other words, cognitive weights are nothing but itself represents the complexity value because it measures the difficulty in understanding the software. The inputs of the cognitive weights are weights of BSC's, which represents the logical structure of the software and output is a positive number, which represents the structural complexity of the program. This complexity value gives the direct indication of understandability and therefore the maintainability of the software. Further, there is no need for addition or modification in the measure of cognitive weight. We proposed the cognitive weights as a measure of complexity [7], which is the more suitable due to not only its simplicity but also it, provides the sufficient information about the internal structure of software. The comparative study with similar measures (see section 3) proved our claim. It is also worth mentioning that concept of cognitive weight is not new, but in all the previous measures, none of them use cognitive weights as a complexity measure. In the present paper, we evaluated Cognitive Weight Complexity Measure (CWCM) in terms of Weyuker's properties and compared with other similar measures.

In section two we discussed the relation between cognitive weights, and cognitive informatics. The different measures based on cognitive informatics are analyzed and compared in section 3. We evaluated CWCM through Weyuker's properties in section 4. The conclusion is given in section 5.

2. Cognitive Weights and Cognitive Informatics

In cognitive informatics, it is found that the functional complexity of software in design and comprehension is dependent on the internal architecture of the software. The internal architecture is represented by Basic control structures (BCS) and sequence, branch and iteration [10, 11] are the basic logic building blocks of software. The cognitive weight (W_{bcs}) of software [11] is the extent of difficulty or relative time and effort for comprehending given software modeled by a number of BCS's. There are two different architectures for calculating W_{bcs}: either all the BCS's are in a linear layout or some BCS's are embedded in others. For the former case, sum of the weights of all n BCS; are added and for the latter, cognitive weights of inner BCS's are multiplied with the weights of external BCS's. The cognitive weights for different Basic Control Structures are as given in [11]. Actually, these weights are assigned

978-1-4244-2881-6/08/$25.00 ©2008 IEEE

on the classification of cognitive phenomenon as discussed by Wang [11]. He proved and assigned the weights according to the complexity of the functions. For example, weight for lowest cognitive function is assigned as 1 for sequence structure of the program and weights for concurrency and interrupts are assigned as 4, the most complex structure of the programs. These weights also show the structured ness of the program and can be easily represented by the graph. We refer to authors to read Wang's paper [11] for the details of the cognitive weights.

3. Cognitive Complexity Measures: A Comparative Study

The first cognitive complexity measure to calculate the functional size of a program is developed by Wang [11]. In cognitive informatics, the functional complexity of software depends on three factors: internal architecture, input, and output. This means, in cognitive informatics, the functional size depends on input, output and basic control structures (BCS's). Accordingly, Cognitive functional Size (CFS) of software is defined as:

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

where, N_i = Number of inputs, N_o = Number of outputs, W_c = Total cognitive weight of the software.

Actually, CFS satisfies the rules of cognitive informatics up to a limit. In the formulation of CFS, only distinct number of input and output has been considered. However, total numbers of occurrences of input and output are equally important as the total occurrences of BCS's [4]. The occurrences of inputs and outputs directly affect the internal structure as well as cognitive complexity of software. In addition, total number of inputs and outputs represents the information contents of the software, and directly associated with design information. Further, in the formulation of CFS: distinct number of input and output are multiplied with the cognitive weights, which is also not justified (why multiply). The same approach has been applied in MCCM [6] i.e. total occurrences of input and output are multiplied by cognitive weights.

Klemola and Rilling [2] have proposed a complexity measure based on Category learning. It defines identifiers as programmer's defined labels. Based on this, Identifier Density (ID) can be calculated as the number of identifiers divided by Line of Code (LOC). Further, for calculating the Kind of Line of Code Identifier Density (KLCID), it finds number of unique lines of code and lines that have same type of operands with same arrangement of operators, are considered equal. Hence $(a = b + c) \approx (d = e + f)$ when a, b, c, d, e and f are line of codes of the same type. It then defines KLCID as:

KLCID = No. of unique lines of code / No. of identifiers in the set of unique lines (2)

It is worth mentioning that this is difficult to calculate and time consuming when comparing a line of code with each line of the program. Besides, LOC measures are programming language dependent and therefore may penalize well-designed but shorter programs [9]. Additionally, they may not be easily used for nonprocedural languages. This implies that their use in estimation requires a level of detail that may be difficult to achieve [9].

Misra and Kushwaha [1] proposed Cognitive Information Complexity Measure (CICM). The CICM is defined as the product of weighted information count (WIC) and the cognitive weight (W_c) of the BCS's in the software i.e.

$$CICM=WICS*W_c \quad (3)$$

where WICS is sum of Weighted Information Count of Line of Code (WICL) given for kth line and

$$WICL_k=ICS_k/[LOCS–k] \quad (4)$$

where $WICL_k$ & ICS_k are weighted information count and information contained for k_{th} line.

Note that, similar to KLCID, CICM is also difficult and complex to calculate. It calculates the weighted information count of each line. There is problem in their formulation. They claim that CICM is based on cognitive informatics, but in cognitive informatics the functional complexity of software only depend on input, output and internal architecture, not on the operators. Further, they claimed that information is a function of identifiers and operators. However, we do not agree with this claim of Kushwaha and Misra [1]. It is difficult to understand how they claimed that information is function of operators. Operators are run time attributes and cannot be taken as information contained in the software.

Misra [3] proposed modification in cognitive functional size approach by taking consideration of metric due to operators and operands. He argued that operators and operands are equally important in design consideration as the basic control structures. Therefore, the cognitive complexity should also depend on operators and operands instead of input and output. Once operators and operands have been considered, the number of input and output are automatically included. The occurrence of operators and operands directly affect the internal structure and the cognitive complexity of software, which has not taken into consideration in the cognitive functional size approach. Based on this, modified cognitive complexity measure (MCCM) is given by,

$$MCCM = S_{oo}*W_c. \quad (5)$$

where, Soo is the sum of operators and operands and Wc is the cognitive weights of the basic control structure.

This is a good measure for complexity but it gives complexity value very high (in number). It is due to two factors: the total occurrences of operators and operands in program are normally too high and again it is multiplied by total cognitive weights. As a result, total complexity increases very high. It also does not depend

on the laws of cognitive informatics, because it includes the contribution of operators. An improved form of CFS [6]; named as cognitive program complexity measure was presented in ICCI'2007 [7], it follows the rules of cognitive informatics and depends on total occurrences of inputs, outputs and cognitive weights. Further, these measures were improved [8] by including other factors responsible for complexity and validated [9] through measurement theory. It is worth mentioning that all these measures give high complexity values for real projects. The high complexity values are undesirable for good complexity measure.

All above mentioned measures can be used to calculate the complexity of program from different perspectives; On the other hand, one can also find the structural complexity only by the consideration of cognitive weights [7]. Cognitive weights itself provide the sufficient information about the information contained in the software in terms of BCS's. Cognitive weight complexity measure (CWCM) depends upon the *cognitive weights of basic control structures*.

i.e. $CWCM=W_c$ (6)

where W_c is the cognitive weight of the basic control structure. They are basic building blocks of software. The standard weights for different control structures are given in [11]. The total cognitive weight of a software component W_c is defined as the sum of cognitive weight of its q linear blocks composed in individuals BCS'S. Since each block may consists of m layers of nesting BCS's, and each layer with n linear BCS's, the total cognitive weight, W_c can be calculated by:

$$W_c = \sum_{j=1}^{q}\left[\prod_{k=1}^{m}\sum_{i=1}^{n} W_c(j,k,i)\right] \quad (7)$$

When CWCM is compared with other related measures, [7], CWCM gives lower complexity values. It can be easily seen that CWCM already includes the considerations of information contained in terms of cognitive weights. It is also worth mentioning that lower complexity value in terms of number is considered to be better in comparison of measures, which gives higher complexity value. A plot for CWCM, CICM and CFS [7] shows the trends of complexity values for a sample of ten different programs. It is seen that the trends of each graph is almost similar, if the complexity value is high for some programs, then it reflects in all the graphs.

This comparative study proves the similarity between all the complexity measures. Once we are getting the appropriate information by a small number and by simple calculation, there is no need to adopt the complex method for the same information

4. Theoretical Validation of CWCM

Weyuker's [8] proposed the nine properties to evaluate any software complexity measure. These properties evaluate the weaknesses of a proposed measure in a practical way. By the help of them, one may obtain an idea about the quality of his/her own proposal. In the following paragraphs, the CWCM has been evaluated against nine Weyuker's properties in order to observe the soundness of the proposal. For this purpose, nine examples given in [1] are considered.

Property 1: (∃P) (∃Q)(| P |≠| Q |).Where P and Q are program body

This property states that a measure should not rank all programs as equally complex. The first of the two examples considered in [1] have two internal structures: a sequence and iteration. Total cognitive weight of this algorithm is:

$W_c = W_1 + W_2 = 4$, Hence, CWCM= W_c = 4 CWU.

For the next example in figure 2, here is only one internal structure: a sequential. The total cognitive weight of the BCS's is: W=1, Hence, CWCM=1 CWU.

Clearly, from the above two examples where cognitive weight complexity measure is different, this property is satisfied by the proposed measure.

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

All programming language consists of BCS's. For cognitive weight complexity measure, it only depends on cognitive weights, which are all finite in number for any finite length program and therefore there is only finite number of program bodies for a given complexity value. Hence, CWCM holds for this property.

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

Example 3 considered in [1] has two internal structures: a sequential and an iteration .Total cognitive weight of this algorithm is:

$W_c = W_1 + W_2 = 4$. Hence, CWCM= W_c=4= 4 CWU.

For the example in fig.1 [1], CWCM is 4 CWU.

It is clear that for two distinct programs the cognitive complexity measure is the same i.e. 4. Therefore, this property is satisfied by the cognitive complexity measure.

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

This property states that even though two programs compute the same function, the program complexity is determined by the implementation details. If in example figure 1 [1], for loop is replaced by the formula s= (b+1) b/2 (in figure 2.) the output is not affected. For this changed program CWCM is 1 CWU, which is different for CWCM value for example figure 1(CWCM=4 CWU). For measuring the sum of first n integer by two different algorithms, it is observed that the cognitive weight complexity measure for the two algorithms for same object is different. Therefore, this property is also satisfied by the given measure.

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

Cognitive weight complexity measure is an integer and the set of integers with operator holds the

following property: (∀P) (∀Q) (P ≤ P + Q) and (Q ≤ P + Q). Since this equation and Weyuker's property 5, are analogous and such property 5 is satisfied by modified cognitive complexity measure.

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

This property asserts that we can find two program bodies of equal complexity which when separately concatenated to a same third program yields the programs of different complexity. Our measure depends only upon cognitive weights that are the fixed for all programs and joining program R with P and Q adds the same amount of complexity. Hence, this property is not satisfied.

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

Since CWCM depends on cognitive weights of basic control structures, which are fixed for any program, hence permuting the order of statement in any program will not change the value of CWCM. Hence, the CWCM will not change for the two programs. Thus, CWCM does not hold for this property.

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

Cognitive weight complexity measure gives an integer so renaming of a program cannot change cognitive weight complexity and as such this property is clearly satisfied by the given complexity measure.

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

This property allows for the possibility that as program grows from its component program bodies, additional complexity is introduced. If we take the examples 4, 5 and 6 [1], where cognitive complexity measure for two-program bodies are 15 and 7 and for single (combination of both) is 29. This example proves that this complexity measure satisfies property 9.

In this section, it is proved that CWCM satisfies seven of nine Weyuker's property. When we compared these results with other measures, it is observed that CFS, CWCM and MCCM [3] are satisfied by 7 out of nine, and CICM is satisfied by all 9 properties It is worth mentioning that some of the Weyuker's properties are itself contradictory, and therefore it is not supposed to satisfy all Weyuker's properties. To satisfy seven properties by a newly proposed measure is a good indicator of a valid metric. In the case of CICM, which is satisfied by all Weyuker's properties, there is a doubt about the correctness of the evaluation process. Satisfactions of the seven properties by CWCM prove that the CWCM follows the same trends for being a valid measure. The additional advantage of CWCM is that it is the most simple and gives the structural complexity in terms of small numbers, when compared with other similar measures. Further, CWCM is a language independent complexity measure, because it depends on basic control structures, which are same for all programming languages, except declarative and functional languages

5. Conclusions

Comparative studies between the cognitive complexity measures have been done. It is found that cognitive weight complexity measure is comparatively more suitable measure, when it is compared with other similar measures. The important features of this measure are: it is simple, easy to understand, easy to calculate, less time consuming, gives the structural complexity value in terms of small number, and language independent i.e. it satisfy most of the properties of a good measure. Further, CWCM also gives valuable indication for the design quality of the codes. High complexity values indicate that understandability and maintainability of the code is weak. Ultimately, it helps the software developer for better design. For example, the developer, who can satisfy the user requirements through the usage of a lesser number of branching and looping primitives, is assumed to be more skilful.

References

1. D.S. Kushwaha, and A.K Misra, "Robustness Analysis of Cognitive Information Complexity Measure using Weyuker Properties," ACM SIGSOFT SEN, 31, 1, pp.1-6, 2006.
2. T. Klemola, & J. Rilling, "A Cognitive complexity metric based on Category learning," Proceedings of IEEE (ICCI'03), pp.103-108, 2003.
3. S.Misra, "Modified Cognitive Complexity Measure," LNCS 4263, pp.1050-1059, 2006.
4. S.Misra,"Cognitive Program Complexity Measure," IEEE Int. Conf. on Cognitive Informatics, pp.120-125, 2007.
5. S.Misra and I.Akman, "A Model for Measuring Cognitive Complexity of Software," Lecture Notes in Computer Science, LNAI 5178, pp.879-886, 2008.
6. S.Misra and I.Akman, "An Unique Complexity Metric," LNCS, 5073, pp.641-651, 2008.
7. S.Misra, "A Complexity Measure based on Cognitive Weights," Int. Jour. Of Theoretical and Applied Computer Sciences Vol.1, pp.1-7, 2006.
8. E.J.Weyuker. "Evaluating Software Complexity Measure. IEEE Trans. on SE, 14, pp.1357-1365, 1988.
9. R.S. Pressman, *Software Engineering: A Practitioner's approach,* Fifth edition, McGraw Hill, 2001,
10. Y. Wang and J.Shao, "On cognitive informatics", Keynote Lecture. Proc of 1st IEEE International Conference on Cognitive Informatics, pp.34–42, 2002.
11. Y. Wang and J.Shao, "A new measure of software complexity based on cognitive weights," Can. J. Electrical. Computing. Eng., 28, 2, pp. 69-74, 2003.
12. I. Somerville, Software Engineering, Sixth Edition, 2001, Addison-Wesley