ISSN:0975-9646

Generation of a New Complexity Dimension Scheme for Complexity Measure of Procedural Program

Nikhar Tak, Dr. Naveen Hemrajani

Department of Computer Science & Engg,
Suresh Gyan Vihar University,Jaipur

Abstract - **Software complexity measurement has been an age-long quandary in software engineering as the effort used to develop, comprehend, or retain the software depends on so many complicated factors. Measuring and controlling of complexity will have an important influence to improve productivity, quality and maintenance of software. So far, most of the researches have tried to identify and measure the complexity in design and code phase. However, when we have the code or design for software, it is too late to control complexity and leads inaccuracy publishing in the whole system. In this paper, we propose a new dimension scheme, in the first phase i.e. Requirement phase of software development which is based on software requirement specifications.**

Keyword – **Code based complexity measures, Cognitive complexity measures and new scheme for complexity measure.**

1. Introduction

Complexity is a feature of a computer program much like storage space and speed of execution. It is the main factor that can lead to defects. The problem of reliability is basically the problem of software complexity. When the complexity reaches some thresholds, the defects or faults of the software grow rapidly. The maintainability of the software is also having tight correlation with complexity.

Complexity control and management have important roles in risk management, cost control, reliability prediction, and quality improvement. The complexity can be classified in two parts: problem complexity (or intrinsic complexity) and solution complexity (also referred to as additional complexity). Solution complexity is added during the development stages following the requirements phase, mostly during the designing and coding phase.

Many researchers suppose that software complexity is made up of the following complexity:

- Problem complexity, which measures the complexity of the critical problem. This type of complexity can be traced back to the requirement phase, when the problem is defined.
- Algorithmic complexity, which reflects the complexity of the algorithm implemented to resolve the problem.
- Structural complexity reflects the complexity of the algorithm implemented to solve the problem.
- Cognitive complexity measures the effort required to understand the software.

Algorithmic complexity measured implemented algorithm to solve the problem and is based on mathematical methods. This complexity is computable as soon as an algorithm of a solution is created, usually during the design phase.

Structural complexity is composed of data flow, control flow and data structure. Some metrics are proposed to measure this type of complexity, for example McCabe cyclomatic complexity(that directly measures the number of linear independent paths within a module and considered as a correct and reliable metric), Henry and Kafura metric(measures the information flow to/from the module are measured, high value of information flow represent the lack of cohesion in the design that will cause higher complexity) and Halstead metric (which is based on the principle of count of operators and operand and their respective occurrences in the code among the primary metrics, and is the strongest indicator in determining the code complexity).

There are some metrics based on cognitive methods [8] such as KLCID complexity metric (It defines identifiers as the programmer defined variables and based on identifier density. To calculate it, the number of unique program lines is considered).

To prevent slaying helpful resources and complexity, it is better to focus on early stages of the software life cycle. Therefore, the result of identifying complexity factors is low costs and high quality in software development and particularly in maintenance stages of software. By knowing these factors, we try to prevent occurring them or establish new measure in requirement phase. [7] Requirements form the foundation of the software development process. Loose foundation brings down the whole structure and weak requirements documentation leads to project failure. Recent surveys suggest that 40% to 85% of all defects are inserted in the requirements phase. Thus, if errors are not identified in the requirements phase, it is leading to make mistakes, wrong product development and loss valuable resource.

Well-defined requirements will increase the probability of the overall success of the software project and later stages of software development rely heavily on the quality of requirements, there is a good reason to pay close attention to it.

This paper aims to assessment and point out the blemish of existing measures and propose a new measure in early phase of SDLC.

2. Code Based Complexity Measures

Code based complexity measures, as its name is indicating are based on the code of the program. Code based measures are typically depends upon the program sizes, program flow graphs, or module interfaces such as Halstead's software science metrics [2] and the most widely known measure of cyclomatic complexity developed by McCabe [3]. However, Halstead's software metrics purely calculates the number of operators and operands, but it does not consider the internal structures of modules, while McCabe's cyclomatic complexity does not consider I/O' s of a system as it is based on the flow chart of the program. It uses flow chart of the program and on the basis of nodes and edges it provides complexity of the program.

2.1 Halstead Complexity Measure

Halstead complexity [4] metrics is established for measuring program complexity with accent on computational complexity. Halstead metric, directly measure the complexity from the source code and based on four numeric values as followed:

n1: Number of non-recurring operators
n2: Number of non- recurring operands
N1: Number of all operators
N2: Number of all operands

Further Length and Vocabulary serve as the basis for finding out Volume, Latent Volume, Difficulty, effort and finally Time by using equations (1-7):

Length $N = N1+ N2$ (1)
Vocabulary $n = n1+ n2$ (2)
Volume $V = n \log_2^n$ (3)
Latent Volume $V^* = (2 + n2) \log_2^{(2+n2)}$ (4)
Difficulty $D = V^*/V$ (5)
Effort $E = V/D$ (6)
Time $T= E/18$ (7)

The main problem with this method or we can say blockage with this is that it does not distinguish the differences among the same operators and among the same operands in a program. Moreover, it ignores the nested structure and fails to analyze the case statement when code is not accessible. Other drawback with the Halstead metric is that they are difficult to compute, especially in large programs.

2.2 Mac Cabe's Cyclometric Complexity

McCabe's cyclomatic complexity also known as conditional complexity based on control flow. It denotes the number of linearly independent paths through a program's source code [10] [3]. This measure provides a single ordinal number that can be used to measure the complexity of different programs

The metric is calculated by using equation (8):

$$CC = e - n + p \qquad (8)$$

Here,
e is the edges of graph,
n is the nodes of graph,
p is the non-connected parts of the graph.

Another formula for calculating complexity is the following:

CC = Number of Decisions +1

It can be computed early in life cycle than of Halstead's metrics but there are some difficulties with the McCabe metric [6]. Although no one would argue that the number of control paths relates to code complexity, some argue that this number is only part of the complexity picture. According to McCabe, a 5,000-line program with six IF/THEN statements is less complex than a 500-line program with seven IF/THEN statements and this shows the complexity of uncontrolled statement are ignored.

3. Cognitive Complexity Measures

In cognitive informatics, the functional complexity of software in design and comprehension is dependent on fundamental factors such as inputs, outputs, Loops/branches structure, and number of operators and operands [5].

3.1 KLCID Complexity Metrics

Klemola and Rilling proposed KLCID which defines identifiers as programmer's defined labels. It defines the use of the identifiers as programmer defined variables and identifiers (ID) when software is built up [5] [13].

ID = Total no. of identifiers/ LOC

In order to calculate KLCID, we need to find the number of unique lines of code in a module, lines that have same type and kind of operands with same arrangements of operators would be consider equal. I define KLCID as –

KLCID= No. of Identifier in the set of unique lines/ No. of unique lines containing identifier

This is a time consuming method when comparing a line of code with each line of the program. KLCID accepts that internal control structures for different software's are identical.

3.2 Coginitive Functional Size (CFS)

Wang proposed a Cognitive Functional Size (CFS) state that the complexity of software is dependent on inputs, outputs, and its internal processing [9]. As –

$$CFS = (N_i + N_o) * Wc$$

Where,

N_i = No of inputs.
N_o = No of outputs.
Wc =The total cognitive weight of software

The cognitive weight of software [11] is the degree of intricacy or relative time and attempt for comprehending given software modeled by a number of Basic control structures (BCS).

3.3 Cognitive Information Complexity Measure

Cognitive Informatics plays an important role in understanding the fundamental characteristics of software. CICM [12] is defines as the product of weighted information count of software (WICS) and the cognitive weight (W_c) of the BCS's in the software i.e,

$$CICM = WICS * W_c$$

Where, WICS is sum of weighted information count of line of code (WICL). WICL for k_{th} line of code is given by [9]:

$$WICL_k=ICS_k/ [LOCS-k]$$

Where, ICS_k information contained in software for k_{th} line of code LOCS: total lines of code. Further ICS is given by:

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

Where, I_k is the information contained in K_{th} line of code and calculated

$$I_k = (\text{Identifiers} + \text{Operands})_k$$

Note that, similar to KLCID CICM is also difficult and complex to calculate. It calculates the weighted information count of each line. In their formulation they claim that CICM is based on 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. It is difficult to understand that how they claimed that information is function of operators [5]. Operators are run time attributes and cannot be taken as information contained in the software.

4. Proposed Complexity Measure

Code based complexity measures such as Halstead Complexity Measure and Mc Cabe's Cyclomatic Complexity Measure are based on the source code of the procedural programs. On the other hand Cognitive based complexity measures such as Kinds of Lines of Code Identifier Density (KLCID), Cognitive Functional Size (CFS) and Cognitive Information Complexity Measure (CICM) depend on the internal architecture of the procedural programs [1]. Thus both the methods will wait for the source code of the program and take more time to get implemented.

It will be more beneficial if we can calculate the complexity of the procedural programs in the earlier phases of the software development life cycle at the time of preliminary assessment that is requirement analysis.

New dimension scheme consists of some of the attributes that must be studied at the time of software requirement specification on the basis of which procedural program is to be developed. So the merit of this approach is that it is able to estimate the software complexity in early phases of software development life cycle, even before analysis and design is carried out. Due to this fact this is a cost effective and less time consuming.

It can be implemented by considering the various attributes such as:

4.1 Key In – Out (KIO)

KIO can be define as –

KIO = No. of Inputs + No. of outputs + No. of files + No of interfaces

4.2 Functional Requirement (FR)

Functional requirements should define the elementary trial that must take place. This can be defined as –

$$FR = \text{No. of Functions} * \sum_{i=1}^{n} SPF_i$$

Here, SPF is Sub Process or Sub-functions available after decomposition.

4.3 Non Functional Requirement (NFR)

It refers to the system qualitative requirements and not satisfying those leads to customer's dissatisfaction. This can be represented as –

$$NFR = \sum_{i=1}^{n} Count_j$$

Table - 4.1 Different Types of Non Functional Requirement

Type	Parameters	Count
Optional Requirement Type	For Loop	1
	While Loop	
	Do While Loop	
	If Then Else	
Must be Type	Printf/scanf just after optional type	2
Very Important Type	Any other information handled after Must be Type	3

4.4 Obligatory Complexity Measure (OC)

This can be calculated by the sum of all functional and its decomposition into sub-functions and non functional requirements –

$$OC = FR + NFR$$

4.5 Special Complexity Attributes (SCA)

This is referred to as the Cost Driver Attributes of unique Category from COCOMO Intermediate model proposed by Berry Boehm. Mathematically defined as –

$$SCA = \sum_{i=1}^{5} MF$$

Here, MF is a Multiplying Factor.

4.6 Design Constraints Obligatory (DCO)

It refers to the number of constraints that are to be considered during development of software.

Represented as –

$$DCO = \sum_{i=0}^{n} C_i$$

Where C_i is Number of Constraints and value of C_i will vary from 0 to n.

$C_i = 0$	If Blind Development.
C_i = Non-Zero	If Constraints exists.

4.7 Interface Complexity (IFC)

This parameter is used to define number of external interfaces to the proposed program.

$$IFC = \sum_{i=0}^{n} I_f$$

Here I_f is Number of External Interfaces and value of I_f will vary from 0 to n.

$I_f = 0$	No External Interface.
$I_f =$ Non-Zero	If External Interface exists

4.8 Users / Location Complexity (ULC)

This parameter discuss the number of users for accessing the program and locations (Single or Multiple) use. This can be symbolized as –

ULC = No. of User * No. of Location

4.9 Program Feature Complexity (PFC)

If advancement of the program is to be done then some features are added and this parameter shows the program feature complexity by multiplying all the features that have been added into it. Thus mathematical representation is as follows –

$$PFC = (Feature_1 * Feature_2 * □□□□. * Feature_n)$$

Now by considering all these parameter and defining a new measure that is “SRS oriented complexity measure.”

It can be mathematically shown as –

SRSO = ((KIO + OC) * SCA + (DCI + IFC + PFC))* ULC

The SRS Based Complexity will be higher for the programs, which have higher functionality to be performed and more quality attributes which is to be retained. All above measure have been illustrated with the help of an example below –

Example –1:

Develop a procedural program to enter 5 numbers at a time & display them in reverse order of the input.

Consider this aim to upon going through the SRS; we are able to extract the following parameters –

Number of Inputs 05(Numbers)

Number of Outputs 05(Reverse sequence of numbers)

Number of Interfaces 01(User Interface)

Number of Files 01(Storage of values)

KIO = 5+5+1+1=12

Number of function = 0

FR = 0

Number of Non Functional Requirement (NFR) = 06

Obligatory Complexity (OC) = FR + NFR

OC = 06

Special Complexity Attribute (SCA) = 0.90 (Suppose Programmer Capability = High)

Design Constraints Imposed (DCI) = 00 (No directives)

DCI= 0

Interface Complexity (IFC) = 0

Since this program is not to be further connected with any external interface Number of User / Location (ULC) = 1 * 1= 01

Program Feature Complexity (PFC) = 0

Now,

OCM = ((KIO + OC) * SCA + (DCI + IFC + PFC))* ULC

SRS Oriented Complexity Measure = 16.20

The complexity measured by SRS Oriented Complexity Measure for the given SRS is 16.20, now program code is illustrated in Program – 1. Based on the above code we compute the complexity of the other proposed measures as shown in Table – 4.2 and 4.3.

Program – 1:

Program to Enter 5 Number at a time & Display them in Reverse Order of the input.

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int i, x[5];
 clrscr();
for (i=0;i<5;i++)
   {
   printf("Enter the number\n");
   scanf("%d",&x[i]);
   }
 printf("\nThe reverse order is\n");
 for (i=4;i>=0;i--)
   {
   printf("\n%d",x[i]);
   }
   getch();
}
```

KLCID		CFS		CICM	
ID	6	N_i	5	LOC	18
LOC	18	N_o	5	Identifier	6
ID	0.34	BCS (Seq.)	1	SBCS	7
No. of exceptional Lines having Identifier	6	BCS (For Loop)	3		
		BCS (For Loop)	3		
No. of Identifier in the set of exceptional Lines	12	W_c	7	WICS	2.49
KLCID	0.50	CFS	70	CICM	17.43

Table 4.2 – Calculation for code and cognitive complexity measure

Table 4.3 – Calculation for code and cognitive complexity measure

Mc Cabe		Halstead	
Nodes	9	n_1	17
		n_2	05
Edges	10	N_1	36
		N_2	07
Predicate Node	2	Vocabulary	22
		Program Length	42
Regions	2	Effort	6648.14
		Time	369.34
V(G)	3	Halstead Difficulty	18.38

5. Comparision Between Various Complexity Measures

SRSO is applied on a program developed in C language and used to enter 5 numbers at a time & display them in reverse order of the input.In order to analyze the validity of the result; the SRSO is calculated on the basis of SRS and further compared with other established measures which are based on Code and Cognitive complexity. In order to show comparison result a chart is considered.

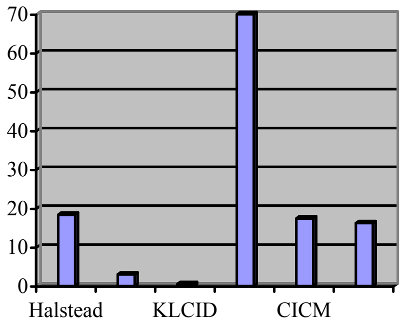

Figure – 5.1 Comparison among various complexity measures

This graph consist both code and cognitive based complexity measures along with the SRS oriented complexity measure for a procedural program for display reverse of 5 numbers. In other methods that are code and cognitive complexity measure the source code of the program is needed but in SRS oriented complexity measure time in waiting for coding is saved as it gauge the complexity at the time of very first phase of software development life cycle that is requirement analysis phase in which software requirement specification is built up and this also improves the quality of the program as Design Constraints Obligatory (DCO), Interface Complexity (IFC), Users / Location Complexity (ULC), Program Feature Complexity (PFC) are different parameters that are also calculated for the program and this other information is gathered in SRS phase so quality of program is increased as most of the parameters are very well known and calculated earlier. While other complexity measures will wait for the source code of the program and even then there is no way to calculate design constraints, interface needed, location of accessing the program or number of users who can access program and any program features calculating parameter.

CONCLUSION

In this paper, the drawbacks of existing software complexity measures were analyzed such as Halstead measure the main problem with this is that it does not distinguish the differences among the same operators and among the same operands in a program. Other drawback with the Halstead metric is that they are difficult to compute, especially in large programs; a difficulty with the McCabe metric is it ignored the complexity of uncontrolled statement.

Likewise when we analyze the cognitive complexity measures such as KLCID complexity metrics we found that it is very time consuming; similar to KLCID Cognitive Information complexity measure (CICM) is also difficult and complex to calculate because it calculates the weighted information count of each line. Further CICM claimed that information is a function of identifiers and operators. It is difficult to understand that how they claimed that information is function of operators. Operators are run time attributes and cannot be taken as information contained in the software.

Thus we presented a new qualitative method to measure the complexity of procedural programs which is applicable before coding phase i.e, SRS Oriented Complexity Measure.

It is useful for procedural programs as it is implemented at requirement phase of software development life cycle. So, it is not depended on the coding phase to be completed and developing cost and resources will be saved too. At the time of coding, programmer will be having complexity so it will help him/her to keep limit on the complexity of the code is to be generated. This measure is computationally simple and will aid the developer and practitioner in evaluating the software complexity in early phases which otherwise is very tedious to carry out as an integral part of the software planning. Since entire approach is based on SRS document so it is for sure that an SRS must have all the characteristics, content and functionality to make this estimation precise and perfect. This measure is simple to understand, easy to calculate and less time consuming i.e. it satisfy most feature of a good measure.

References

[1] Weyuker E., Evaluating software complexity measures. IEEE Transactions on Software Engineering, 1988, 14 (9):1357-1365

[2] Halstead, M.H., Elements of Software Science, Elsevier North, New York, 1977.

[3] Mc Cabe, T.H., A Complexity measure, IEEE Transactions on Software Engineering, SE-2,6, pp. 308- 320, 1976.

[4] Halstead M.H, "Element of Software Science", Amsterdam: Elsevier, 1977.

[5] T.Klemola and J.Rilling, "A Cognitive complexity metric based on Category learning" Proceedings of EEEE (ICCI'03), pp.103-108, 2003.

[6] Elaine J. Weyuker, "Evaluating Software Complexity Measures", IEEE Transactions on software engineering, Vol. 14, No. 9, September 1988.

[7]IEEE Computer Society: IEEE Recommended Practice for Software Requirement Specifications, New York, 1994.

[8] Wang. Y and Shao,J., "On Cognitive informatics", 1st IEEE International Conference on Cognitive Informatics, Pages 34-42, August 2002.
[9] Wang, Y. and Shao, J., "Measurement of the Cognitive Functional Complexity of Software", 3 rd IEEE International Conference on Cognitive Informatics (ICCI'04).
[10]Y.Wang. and J. Shao. "Measurement of the
Cognitive Functional Complexity of Software," Cognitive weights", Proceedings of IEEE (ICCI'03). 69-74, 2003.
[11] Wang.Y and Shao J. (2003). A new measure of software complexity based on coginitive weights, Can.J.Elect. Comput. Eng., 28, 2, 69-74.
[12]D.S.Kushwaha and A.XK Misra,"Robustness Analysis of Cognitive Information Complexity\Measure using Weyuker Properties," ACM SIGSOFT SEN 31, 1, 2006.
[13]Anurag Bhatnagar,Nikhar Tak and Shweta Shukla, "A literature survey on various software complexity measures", IJASCSE Volume 1 Issue 1 2012.