Program Complexity Finder: A Tool for Finding Program Complexity in Terms of Cognitive Weight based on Complexity Measurement Algorithm

Samrat Kumar Dey
Lecturer, Dept. of CSE
Dhaka International University
Dhaka, Bangladesh

Tamim Al Mahmud
Lecturer, Dept. of CSE
Dhaka International University
Dhaka, Bangladesh

ABSTRACT
Complexity Measurement of any piece of programming problems is a key issue for Distributing Equivalent Problems among examinees. Basic Control Structure or BCS Plays an Important role to design a program and hence measuring complexity value of any piece of programming problems. Using of Cognitive weight concept of any BCS are purely based on the thinking Capacity of Human Brain. In this Research Basic control structure has been established in such a way to reduce the limitation of existing Measures. According to these cognitive data, a new software tool based on java SE language and MySQL Database has been established by using own developed algorithm. This software is structured and developed based on the outcome of research data which is capable of determining the complexity value of several programming languages. It will facilitate the instructors distributing the programming problems among the students by maintaining equivalent level of difficulty. Thus, the automatic complexity measurement application will ensure the students to obtain programming problems with equal difficulty level for evaluation.

General Terms
Software Complexity, Complexity Analysis, Software Testing, Algorithm Analysis, Human Computer Interaction

Keywords
complexity measurement; basic control structure; cognitive weight; equal distribution; software complexity;

1. INTRODUCTION
In the last several years, there has been a great deal of interest in defining appropriate ways to measure the complexity of software [23]. There are different facets of software complexity, some of which have been computed using widely accepted metrics like cyclomatic complexity, data/information flow metrics, but very less attempts have been made to measure the cognitive aspect of the complexity [5]. The human mind's efforts needed for the comprehension of the source code reflect a different dimension of complexity, which is being measured in this paper [5]. Another issue encountered in software complexity analysis is the consideration of software as a human creative artefact and the development of a suitable measure that recognizes this fundamental characteristic. The existing measures for software complexity can be classified into two categories: the macro and the micro measures of software complexity [18]. A significant issue encountered in computer science and engineering in the field of programming problem distribution is to distribute the problem equivalently among the learners. In this paper, propose a new model for distributing programming problems equivalently to the learners according to the complexity values of programming problems. Motive is to build a question bank with programming problems where designed system will access the solution codes for finding out the weight of respective question bank by applying the established method. There are methods to find out the running time of algorithms, which can be used for the equal distribution of programs by comparing two different algorithms. But, it is very difficult to measure especially when there is no defined algorithm or is too much large and complex. In Materials and Methods section, conducted a survey with various programming problems; various Cognitive Weights for various Basic Control Structures have been defined. Implementation section provided with designed algorithm and developed software which will help us to find out the complexity of programming solution code and finally in Result and Discussion section, an approach for the equal distribution of the programming problems to the students during examination has been provided.

2. METHODS AND MATERIALS
Solving programming problem is the heart of CSE courses. In the section of programing courses evaluation programming problems are need to be solved within a limited time are to be distributed among students. Variant students will be tested with miscellaneous programming problems. According to this motive, programming problems may not be equally distributed. As a result, some can find their problems easy and others can get problems which are comparatively complex. In this context, examines will not get the proper pronouncement. But it should not happen. Aim of study is to distribute the problem equally. For this reason, to minimize these problems there should have a question bank which will contain various programming problems and solution with different complexities. There are various ways to compute run time complexity of an algorithm. Different well-known algorithms have also defined complexity value based on their run time. Programming problems can be distributed consequently to these values. In a specified boundary similar and approximately similar problems will be put. Same methods will be revolved for all problems on the questions bank. Problems within a specific boundary will be deliberated as analogous kinds of problems. In this exploration, we have built a new complexity model for measure complexity values of programming problems based on their solution codes. In this case, after selecting a problem respective solution code should be developed.

3. COGNITIVE WEIGHT

3.1 Cognitive Weight of a Software

The objective of this section is to discuss the basis of research for measuring program complexity. Cognitive Informatics to measure the complexity of a program, which is called Cognitive complexity. Cognitive complexity, the new measure for software complexity, is a measure of the cognitive and psychological complexity of software as a human intelligence artifact. For comprehending a given program, naturally need to focus on the architecture and basic control structures (BCSs) of the software. Here, we have designed a model to find out the cognitive weights of the BCSs. BCSs are a set of essential flow control mechanisms that are used for building logical software architectures [22]–[2]. Three BCSs are commonly identified: the sequential, branch, and iteration structures [8]. Although it can be proven that an iteration may be represented by the combination of sequential and Branch structures, it is convenient to keep iteration as an independent BCS. In addition, two advanced BCSs in system model-ling, known as recursion and parallel, have been described by Hoare et al. [22]. Wang [22]–[2], [22] extended the above set of BCSs to cover Function call and interrupt. There are two other important control structures that can be found in various modern programming languages: 1. Exceptions, 2. internal exits from loops (for example expressed by the break-statement in C or Java). Note that some languages like Pascal do not allow such exits. However, there are good reasons for using this control structure under some circumstances [17]. We have used 13 Basic Control Structures: SEQUENTIAL, IF-THEN-ELSE, SWITCH, BREAK, CONTINUE, RETURN, FOR, WHILE, DO-WHILE, USER DEFINED FUNCTION (UDF), RECURSION, EXCEPTION HANDLING, and REPEAT-UNTIL. There are many researches that are based on cognitive weight to find out the complexity of a program. Some of them used extra two control structures such as PARALLEL EXECUTION and INTERRUPT shown in Table 1. Table 1 shows the cognitive weights proposed in [18] and [20] and the "inherent complexities" from [15]. [18] Says that the cognitive weights in this paper have been defined "based on empirical studies in cognitive informatics". However, the layout of the experiments (if there were any) has never been published. From a scientific point of view, this means nothing else than that these empirical studies have to be regarded as being non-existent. [15] Stresses that the inherent complexity weights used in their paper have been defined "as a starting point" as a subjective measurement. No experiments have been carried out.

Table 1. Cognitive Weights [21]

Category	Control Structure	Cognitive Weight in [Shao *et al.* 2003]	Cognitive Weight in [Wang 2006]	Inherent Complexity in [McQuaid 1997]
Sequence	Sequence	1	1	1
Branch	if-then-[else]	2	3	3
	Case	3	4	3
Iteration	for-do	3	7	2
	repeat-until	3	7	
	while-do	3	8	3
Embedded Component	Call of a user-defined function	2	7	
	Recursion	3	11	
Concurrency	Parallel Execution	4	15	
	Interrupt	4	22	

Definition 1: The cognitive weight of software is the measurement of difficulty or comparative time and effort required for understanding a given piece of software modelled by a number of BCSs. [18]

Definition 2: Total cognitive weight of a software component, Wc, is defined as the sum of the cognitive weights of its j linear blocks composed of individual BCSs. Since each block may consist of k layers of nesting BCSs, and each layer of *i* linear BCSs, the total cognitive weight, Wc, can be calculated by

$$Wc = \sum_{q=1}^{j} \left| \prod_{m=1}^{k} \sum_{n=1}^{i} Wc(q,m,n) \right|$$

If there is no embedded BCS in any of the j block i.e. k=1 then it can be simplified as

$Wc = \sum_{q=1}^{j} \sum_{n=1}^{i} Wc(q,n)$[18]

3.2 Data Collection and Observation

For finding out the cognitive weight, we have selected some programs each of them represents one control structure. Every program contains nearly same number of lines. We have not used two control structures, PARALLEL and INTERRUPT individually in this observation. Because, a program that describes this control structure contains other control structures used in the observation. So by using the weights of other control structures we can find the complexity of these two types of control structures. During the survey, we gave 20 programs containing individual BCSs to different class of students at undergraduate level. Research work carried on twenty students among which thirteen students were from level 3 and 4 and seven were from level 1 and 2. After reading the problems, every student gave the result of the selected programs. They gave the result according to their thinking capacity of brain for individual BCS. All 20 students consider the initial fact of sequential statement as a sequential statement contain weight 1. Based on the weight of sequential statement selected 20 students gave the other BCS respective weight. These weight will help us to find out the average weight of individual BCS and later we will use this to find the complexity of any program. Maximum number of problems is taken from [C for contest by Tamnun E Mursalin]. Some programs are also developed by us and some are taken from the Internet. We have chosen some common programs like operator precedence, Fibonacci number generation, GPA Calculation, vowel test, factorial generation, reverse number print, and so on because the above listed programs are quite capable of defining BCSs which will use in this research. Reason for choosing such common program for making problems relatively in a similar level (simple or complex).

Following is a sample of a sequential statement program which we have used in survey.

```
#include<stdio.h>
int main (void){
int first,second,result;
printf("Enters two numbers: ");
scanf("%d %d",&first,&second);
result= first+second;
printf ("The two numbers are:%d %d\n ",first, second);
printf ("The Result is: %d", result);}
```

3.2.1 Survey Sample Questions

BCS 1: Sequential

```
#include<stdio.h>
int main (void)
{int first,second,result;
printf("Enters two numbers: ");
scanf("%d %d",&first,&second);
result=first+second;
printf("The two numbers are: %d %d\n ",first,second);
printf("The Result is: %d",result);}
```

BCS 2: Jump

Break Statement

```
#include< stdio.h>
Main()
{int x=1;
while(x<=10)
{printf("x=%d\n",x);
If(x==5)
break;
x++;}}
```

Continue Statement

```
#include<stdio.h>
main(){
int x;
for(x=0;x<=100;x++)
{if(x%2) continue;
printf("%d\n",x)}}
```

Return statement

```
#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>
int max(int num1, int num2);
int main ()
{  int a = 100;
   int b = 200;
   int ret;
   ret = max(a, b);
   printf( "Max value is : %d\n", ret );
   return 0;}
int max(int num1, int num2)
{   int result;
   if (num1 > num2)
      result = num1;
   else
   result = num2;
   return result;}
```

BCS 3: Conditional

If statement

```
#include<stdio.h>
main(){
int number
printf("Type in your number")
scanf("%d",&number);
if(number<0)
number=number*(-1);
printf("The absolute value is: %d\n",number);
}
```

3.2.2 Survey Result Analysis

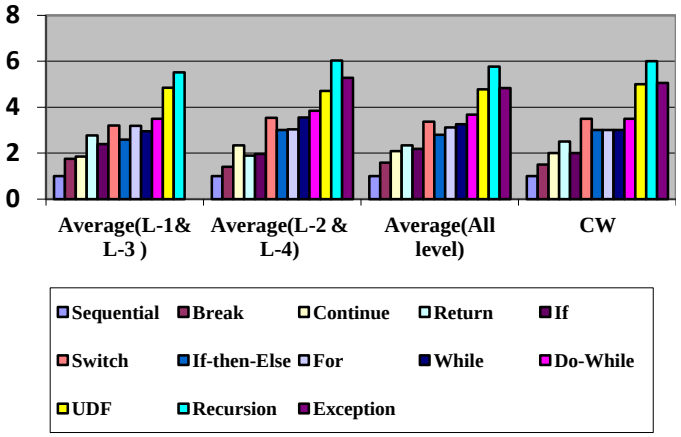

Fig 1: Graphical Representation of Derived Cognitive Weight Based on Survey Result

3.3 Representation of Complexity Model

In this section, we will show how measured cognitive weights can be used to find out total weight or complexity, W_{bcs} of a given program. There are two different architectures for calculating Wbcs [16]:

- Either all the BCS's are in a linear layout. For this case, sum of the weights of all n BCS's are added

$W_{linear}=\sum_{q=1}^{j} Wc(q)$ (1)

Here, q = 1, 2,…,j

j = Number of linear BCSs

For example, we can consider the following Java code. Numbers in the right hand side indicates the weights of the respective BCSs.

```
1. public class SST{
2. public static void main (String args []){
   a.   Integer value_1 = 20;          1
   b.   Integer value_2 =10;           1
   c.   value_1 = value_1+ 5;          1        += 4
   d. System.out.println (value_1);    1
3.          }
4.          }
```

In the above program, line number 1 and 2 are common in Java code. Within the brackets, there are four (a-d) linear sequential statements. They are all independent i.e. they are not embedded in other BCSs. From the previous section, we have obtained the weights of various control structures, where the weight for sequential statements is 1. Therefore, total weight of this program will be, $W_{linear}=1+1+1+1=4$

- Or some BCS's are embedded in others. For this case, cognitive weights of inner BCS's are multiplied with the weights of external BCS's.

$W_{embedded}=\prod_{i=n}^{1} Wc(i)$…….. (2)

Here, i = n,……., 2, 1

n = Total number of BCSs in which the current BCS is embedded + 1

1 = Own weight

2 = Respective parent BCS of 1

3 = Respective parent BCS of 2

……..

n = Respective parent BCS of (n-1)

Parent Control Structure means the structure which encloses another control structure. So, by examining the total weight or complexity of the above Java program, we can conclude the calculation of total complexity by the following equation:

$W_{bcs}=\sum_{q=1}^{j} Wc(q) + \sum_{l=1}^{p} \prod_{i=n}^{1} W_{l,i}$……. (3)

Here, l = 1, 2… p

p = Total number of embedded BCSs

For making the complexity measure language independent, first two lines of the above source code are not considered for measuring complexity. They are common for the Java codes such as main () or void main () used in C

4. SOFTWARE IMPLEMENTATION

In this section we will show how designed system scan the programs and then give the weight of the programs. We have built own method to measure the weight of programming problems. Implementing this task in real life is not so much easy. It's a hard task and done it by own developed program. Initial step is to measure the complexity of programs which not contain any embedded structure. Embedded structure means one BCS contain another BCS inside his structure. We are quite successful to do this by the help of survey result which represent the cognitive weight of the respective BCS. Main goal is to access the database which is basically a question bank containing programs and then pick the set of problems from database for measuring the complexity of respective problems. Following figure showing how designed system works in real life from set of input problems to complexity value of problems as output

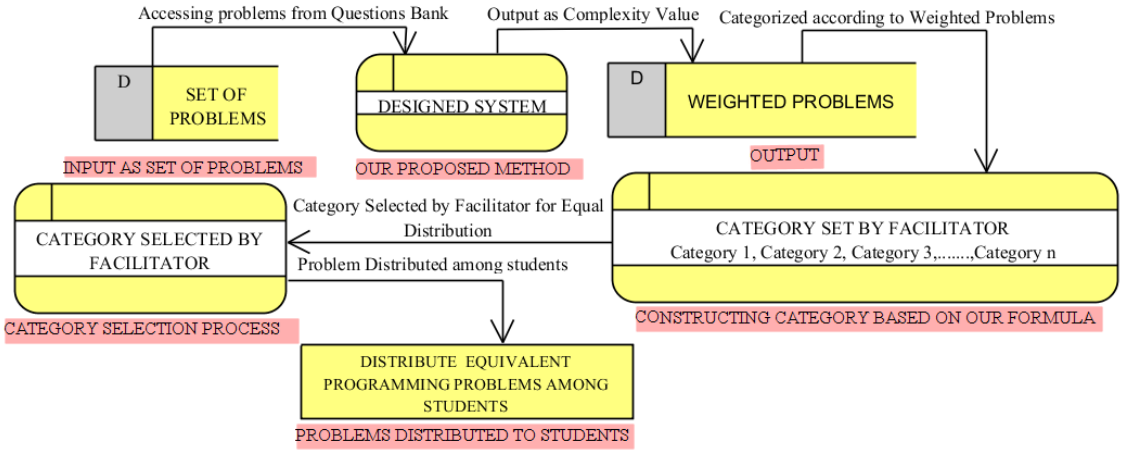

Fig 2: Proposed model for distributing equivalent problems

4.1 Developed Algorithm

With the help of the research data of Cognitive Weight we have built own Algorithm which is Capable of Determining the Exact complexity value of any programming problems as we found in Manual Calculation. The following snippet of pseudo code of program will indicate how developed system execute for finding programs complexity values.

```
Input: A set of programs
Output: Calculated complexity of programs
For each input character
IF (input stream==for)    THEN
    Call class for ()
Complexity          for-check ()
    Index           return index
IF (input stream==if)     THEN
    Call class IF ()
IF (input stream==SEQUENCE)    THEN
    Complexity          SEQUENCE complexity
 END FOR
   PRINT complexity
```

4.2 Developed Software for measuring Complexity Value

For measuring complexity value of any programming problems we have built a software system with the help of Java language and MySQL Database. Basically this software is capable of measuring complexity value by using developed algorithm which is based on Cognitive Weight of Basic Control Structure (BCS). To work with it first of all we need to build a question bank which contain Programming Problems in MySQL Database. Programming problems can be put into several Question bank. So we can have a bank or a set of problems for different source, books or collection from where problems could be select for distribution. Need to browse the problems from Database for complexity measurement. The value of programs complexity is shown in following Fig. 7. The Measured Value will also store in database. From the stored value software systems categorized the problems for further distribution among the learners according to own formula of measurement. However, this software is fully compatible with any operating systems supporting Java. Hopefully, further improvement of this software will surely produce a great dimension in near future.

Fig 3: Creation of Problem Bank

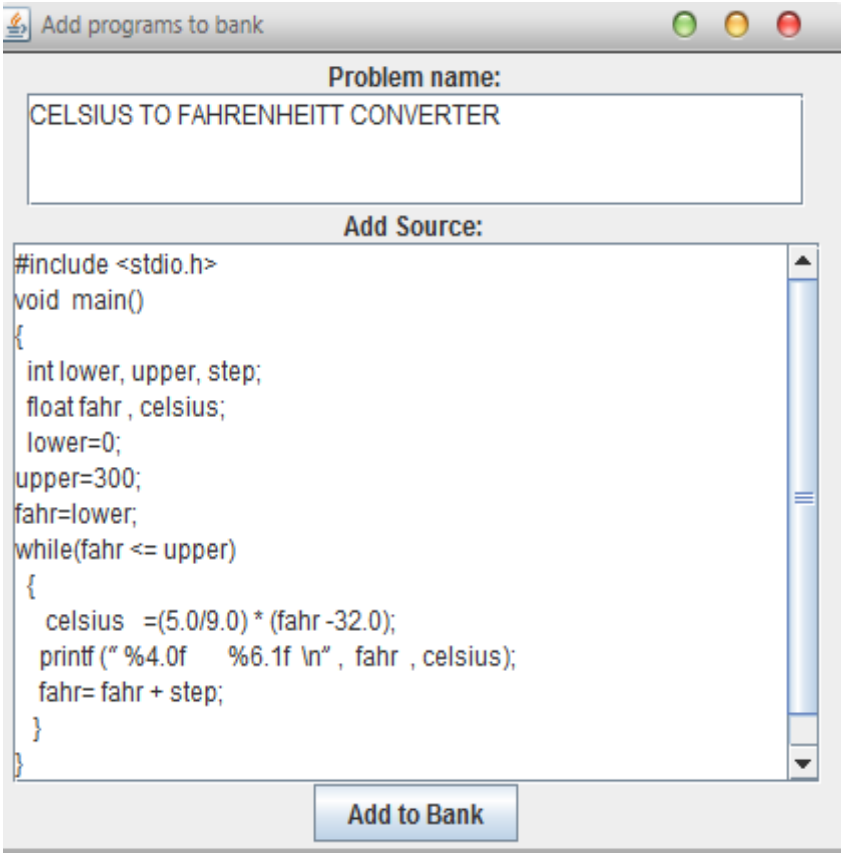

Fig 4: Inserting Problems into Question Bank

Fig 5: Measuring Complexity Value of Inserted Problems from Database

5. RESULT AND DISCUSSION

After the measurement of the complexity of some problems, we have concluded the result. We have got the complexity value of some problems. From the observation of the complexity values, we can say that, 1. If the number of lines is the same, but one program contains BCSs which are more embedded, then this program will definitely result in high complexity value. 2. If the number of linear BCSs is the same or approximately same, but one program contains BCSs of relatively higher weight, then this program will result in comparatively higher complexity value. 3. For programs with increasing number of lines, the complexity increases gradually (without any exceptions).

5.1 Problems Distribution based on Complexity Value

In order to distribute similar types of problems among the learners in a competitive manner in the examination we have categorized the values of programs complexity according following formula where

λ=Lowest Weight from a set of problems
η= Highest Weight from a set of problems
δ=Difference between η & λ
ψ=Desire Number of Category for Facilitator
χ=Number of Category

$\varphi = \delta/\psi$
$\chi_1 = \lambda + \varphi$
$\chi_2 = \chi_1 + \varphi$
$\chi_3 = \chi_2 + \varphi$
Category ή=Category (ή-1) + φ

5.2 Future Works

In this research we have developed a software based complexity measurement systems. Complexity Measurement is based on the research data of cognitive weight. But at this moment we are capable of finding out the complexity of program only for linear structure with the help of designed systems. Moreover, still we are working on how easily we can distribute the programming problems directly from software systems. Hopefully we will also include the BCSs PARALLEL and INTERRUPT into designed systems in later work.

6. CONCLUSION

Main aim is to help facilitator to distribute programming problems equally through a software system. Therefore, we have proposed a model for distributing Equivalent problems among learners. Also we built a software system to measure complexity value of problems based on cognitive Weight of Basic Control Structures (BCS). Important features of this measure are that it is easy to calculate, less time consuming, simple to understand. It also satisfies most of the properties of

a good measurement of complexity. We believe future improvement on this research will produce a great dimension in the area of complexity measurement of programming problems and also ensure the equal distribution of programming problems.

7. REFERENCES

[1] Baker, A. L. and Zweben, S. H. 1980. "A comparison of Measures of control flow Complexity," IEEE Transaction on Software Engineering, No.6, pp506-511.

[2] Barbier F. 2002. "Component-based software measurement," chap. 14 in Business Component-Based Software Engineering, ed. F. Barbier, Boston: Kluwer Academic Publishers, pp. 247–262.

[3] Bashir, G. M. M. Dey, S. K. Tariq, S. S. M. Islam, M. S. Dec. 2014. "Complexity measurement: A new approach to ensure equal distribution of programming problems for evaluation," in Proc. IEEE Int. Conf. ICECE, pp. 780 – 783.

[4] Basili, V. R. 1980. "Qualitative Software Complexity Models: A Summary in Tutorial on Models and Methods for Software Management and Engineering," Los Alamitos, Calif.: IEEE Computer Society Press.

[5] Chhabra, J. K. July 6 - 8, 2011. "Code Cognitive Complexity: A New Measure," Proceedings of the World Congress on Engineering 2011, Vol II WCE 2011, London, U.K.

[6] Halstead, M. H. 1997. "Elements of Software Science," New York: Elsevier North-Holland Inc.

[7] Henry, S. and Kafura, D. 1981. "Software structure metrics based on information flow," IEEE Transactions on Software Engineering, 7(5): 510-518.

[8] Hoare, C. A. R. Hayes, I. He, J. J. Morgan, C.C. Roscoe, A. W. Sanders, J. W. Sorensen, I. H. Spivey, J. M. and Sufrin, B. A. Aug. 1987. "Laws of programming," Comm. ACM, vol. 30, no. 8, pp. 672–686.

[9] Kan, S. H. 2002. "Metrics and Models in Software Quality Engineering" (2nd Edition). Boston: Addison-Wesley Professional.

[10] Kearney, J. K. Sedlmeyer, R. L. Thompson, W. B. Gary, M. A. and Adler, M. A. 1986. "Software Complexity Measurement," Vol. 28, New York: ACM Press, pp. 1044–1050.

[11] Kearney, Joseph K. Sedlmeyer, Robert L. Thompson, William B. Gray, Michael A. And Adler, Michael A. November 1986. " SOFTWARE COMPLEXITY MEASUREMENT", Communications of the ACM, Volume 29, Number 11.

[12] Klemola, T. and Rilling, J. 2004. " A Cognitive Complexity Metric based on Category Learning," IEEE International Conference on Cognitive Informatics.

[13] Kushwaha, D. S. and Misra, A. K. January 2006. "A Modified Cognitive Information Complexity Measure of Software," ACM SIGSOFT Software Engineering Notes, Vol. 31, No.1.

[14] McCabe, T. H. Dec. 1976. "A Complexity Measure," IEEE Transaction on Software Engineering, vol. 2, no. 4, pp. 308-320.

[15] McQuaid, P. A. 1997. "The profile metric and software quality," International Conference on Software Quality, October 6-8 1997, Montgomery, pages 245–252.

[16] Mishra, S. 2006. " A Complexity Measure based on Cognitive Weights." International Journal of Theoretical and Applied Computer Science, vol 1, no 1, pp. 1-10.

[17] Roberts, E. S. 1995. "Loop exits and structured programming: reopening the debate," In SIGCSE '95:Proceedings of the twenty-sixth SIGCSE technical symposium on Computer science education, pages 268–272, New York, NY, USA, ACM Press.

[18] Shao, J. and Wang, Y. 2003. " A new measure of software complexity based on cognitive weights," IEEE Canadian Journal of Electrical and Computer Engineering, 28(2):69–74.

[19] Uddin, Md. R. February 2013. " Equivalence of Problems in Problem Based e-Learning of Database," Unpublished.

[20] Wang, Y. 2006. "On the Informatics Laws and Deductive Semantics of Software," IEEE Trans on Systems, Man, nad Cybernetics (C), 36(2), March, pp. 161-171.

[21] Wang, Y. Aug. 2002. "On cognitive informatics: Keynote lecture," In Proc. 1st IEEE Int. Conf. Cognitive Informatics (ICCI'02), Calgary, Alta., pp. 34–42.

[22] Wang, Y. Oct. 2002. "The real-time process algebra (RTPA)," Annals of Software Engineering, vol. 14, pp. 235–274.

[23] Weyuker, E. J. September 1988. "Evaluating software complexity measure," IEEE Transaction on Software Engineering, Vol. 14(9): 1357-1365.

[24] Yanming, CHU. and Shiyi, XU. July 2007. "Exploration of Complexity in Software Reliability" Tsinghua Science And Technology, ISSN 1007-0214 48/49 pp266-269, Volume 12, Number S1.

[25] Yin, M. L. Peterson, J. Arellano, R. R. 2004. "Software Complexity factor in Software reliability assessment,In Reliability and Maintainability," 2004 Annual Symposium-RAMS, pp190-194.

[26] Yindun, S. and Shiyi, X. July 2007. "A New Method for Measurement and Reduction of Software Complexity," Tsinghua Science And Technology, 1007-021438/49, Volume 12, Number S1, pp.212-216.

[27] Yindun, S. and Shiyi, X. July 2007. "Exploration of Complexity in Software Reliability," *Tsinghua Science And Technology*, Volume 12, Number S1, pp.266-269.