International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:5, No:10, 2011

Attribute Weighted Class Complexity: A New Metric for Measuring Cognitive Complexity of OO Systems

[1]Dr. L.Arockiam, [2]A.Aloysius,

***Abstract*—**In general, class complexity is measured based on any one of these factors such as Line of Codes (LOC), Functional points (FP), Number of Methods (NOM), Number of Attributes (NOA) and so on. There are several new techniques, methods and metrics with the different factors that are to be developed by the researchers for calculating the complexity of the class in Object Oriented (OO) software. Earlier, Arockiam et.al has proposed a new complexity measure namely Extended Weighted Class Complexity (EWCC) which is an extension of Weighted Class Complexity which is proposed by Mishra et.al. EWCC is the sum of cognitive weights of attributes and methods of the class and that of the classes derived. In EWCC, a cognitive weight of each attribute is considered to be 1. The main problem in EWCC metric is that, every attribute holds the same value but in general, cognitive load in understanding the different types of attributes cannot be the same. So here, we are proposing a new metric namely Attribute Weighted Class Complexity (AWCC). In AWCC, the cognitive weights have to be assigned for the attributes which are derived from the effort needed to understand their data types. The proposed metric has been proved to be a better measure of complexity of class with attributes through the case studies and experiments.

***Keywords*—**Software Complexity, Attribute Weighted Class Complexity, Weighted Class Complexity, Data Type

I. Introduction

THE term software complexity refers to the difficulty to understand, change and maintain the software. Software complexity deals with the Psychological complexity of the programs [7]. Software metrics play a vital role in the software industry to assure the quality of the software. Several software Industries have moved to object oriented paradigm in order to increase their capability through reusability function offered by OOP. The use of OOP has increased the complexity [5]. So, there is a need for introducing new complexity measures. The complexity reflects the cognitive load in programming and hence cognitive complexity plays a vital role in measuring the complexity. A new metric namely Cognitive Weighted Class Complexity (CWCC) is proposed for an OO system which is an extension of the Extended Weighted Class Complexity (EWCC) proposed by Arockiam et.al [1]. AWCC includes the cognitive complexity due to Data Type (DT) of the attributes and is a better indicator of complexity of OO systems.

A DT in a programming language is a set of data with values having predefined characteristics. Three DT's are commonly identified such as the Primary, derived and user defined data types. Integer, float, char, etc., are classified as Primary Data Types (PDT). Array is known as Derived Data Types (DDT). Structure, union, class, etc., are classified as User Defined Data Types (UDDT). It is proven that an UDDT may be represented as the combination of PDT and DDT.

Cognitive complexity of computer program can be studied with respect to many cognitive processes. One of the important cognitive processes involved in programming is program comprehension. In this paper, a new metric AWCC is defined and validated against comprehension process.

II. Literature Review

Several metrics have been proposed for OO systems by researchers. A metric suite proposed by Chidamber and Kemerer (C&K) is one of the best known suites of OO metrics. The six metrics proposed by CK are Weighted Method per Class (WMC), Depth of Inheritance Tree (DIT), Response For Class(RFC), Number Of Children(NOC), Lack of Cohesion of Methods(LOCM) and Coupling Between Objects(CBO)[4,9]. A metric for Class Inheritance Hierarchy [6] has been proposed by Rajnish K, and Bhattacherjee V. In 2008, Sanjay Mishra and Ibharam Akman have proposed object oriented complexity measure called weighted class complexity [3], which is calculated by the method complexity and the Number of Attributes in the class.

Classes are the building blocks of any object oriented program. Class is an encapsulation of attributes and methods. The attributes are used for storing and manipulating the data in the program. Attributes are one of the major factors which will affect the complexity of the class and it is clear that the use of different data type of attributes will increases the complexity of the programs. There is no Specific measure exists to calculate the complexity arising due to inheritance. Hence, a new metric [AWCC] has been proposed for object oriented system with inheritance.

The proposed metric AWCC is explained in section 3, the experimentation of a new metric and the case study is described in section 4, a comparative study of AWCC with WCC, WMC, EWCC and the metric calculated using a tool in section 5 and Section 6 presents the conclusion and future work.

Dr. L. Arockiam, Associate Professor, Department of Computer Science, St. Joseph's College (Autonomous), Tiruchirappalli – 620 002, Tamil Nadu, India(Mobile: 94439 05333; e-mail: larockiam@yahoo.co.in).

A. Aloysius, Assistant Professor, Department of Computer Science, St. Joseph's College (Autonomous), Tiruchirappalli – 620 002, Tamil Nadu, India(Mobile: 9443399227; e-mail: aloysius1972@gmail.com).

III. Proposed Metric: Attribute Weighted Class Complexity (AWCC)

AWCC is used to calculate the complexity of the class using the method complexity, attribute complexity of the class, and the inherited members' complexity.

If there are n attributes, m methods in a class and the class is derived from m1 number of classes then, the AWCC of that class can be calculated using the Equation (1).

$$\mathrm{AWCC} = \sum_{i=1}^{n} \mathrm{AC_i} + \sum_{j=1}^{m} \mathrm{MC_j} + \sum_{k=1}^{m_1} \mathrm{ICC_k} \qquad (1)$$

Where

AC is the attribute complexity,
MC is the method complexity,
ICC is the inherited class complexity.

Attribute complexity (AC) is used to calculate the complexity of the attribute in the class, by using the Equation (2).

$$AC = (PDT * W_b) + (DDT * W_d) + (UDDT * W_u) \qquad (2)$$

Where

PDT is the number of Primary Data Type attributes
DDT is the number of Derived Data Type attributes
UDDT is the number of User Defined Data Type attributes
Wb is the Cognitive Weights of the PDT attributes
Wd is the Cognitive Weights of the DDT attributes
Wu is the Cognitive Weights of the UDDT attributes

The weighting factor of attribute is based on the classification of cognitive phenomenon as described by Wang[11], is as follows

	Weights
Sub-Conscious Cognitive Attribute (PDT)	1
Meta Cognitive Attribute (DDT)	2
Higher Cognitive Attribute (UDDT)	3

The Method Complexity (MC) is calculated by assigning the cognitive weights proposed by Wang et.al, to the control structures in the method. Wang[7] has proposed cognitive weights 1, 2, 3, and 2 to the sequence, branch, iteration and call structures respectively. J.Charles et. al [2] has also validated the weights proposed by Wang. ICC can be calculated using the Equation (3)

$$\mathrm{ICC} = (\mathrm{DIT} * \mathrm{C_L}) * \sum_{k=1}^{s} \mathrm{RMC_k} + \mathrm{RN_a} \qquad (3)$$

Where

s is the number of inherited methods
RNa is the total number of Reused attributes
RMC is the Reused Method Complexity
IC is the Inherited Complexity
DIT is the Depth of Inheritance Tree
CL is the Cognitive Load of Lth level

CL is the cognitive Load of Lth level which will differ from person to person according to the cognitive maturity level [5]. Here, the value of CL is assumed to be 1 for simplicity.

IV. Experimentation and A Case Study

The proposed complexity metric given by equation 1 is evaluated with the following three programs namely PROGRAM 1, PROGRAM 2 and PROGRAM 3.

Program 1(with Primary data type attributes):

```
#include <iostream>
using namespace std;

class BaseClass {
protected:
  int i, j;
public:
  void set(int a, int b) {
    i = a;
    j = b;
  }
  void show() {
    cout << i << " " << j << endl;
  }
};

// i and j inherited as protected.
class DerivedClass1 : public BaseClass {
  int k;
public:
  void setk() {
    k = i*j;
  }
  void showk() {
    cout << k << endl;
  }
};

class DerivedClass2 : public DerivedClass1 {
  int m;                // i and j inherited indirectly through DerivedClass1.
public:
  void setm() {
    m = i-j;
  }
  void showm() {
    cout << m << endl;
  }
};
int main()
{
  DerivedClass1 object1;
  DerivedClass2 object2;

  object1.set(2, 3);
```

```
    object1.show();
    object1.setk();
    object1.showk();

    object2.set(3, 4);
    object2.show();
    object2.setk();
    object2.setm();
    object2.showk();
    object2.showm();
    return 0;
}
```

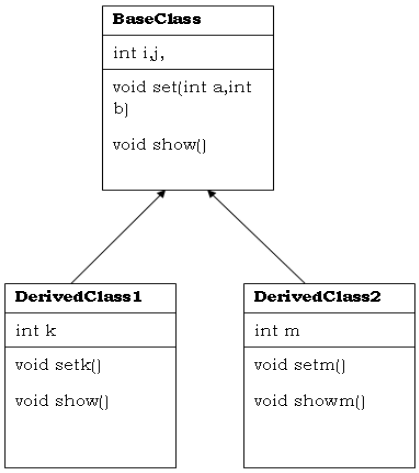

Fig. 1 An example of an object oriented system with Primary data type of attributes

BaseClass

$$AWCC_{bc} = \sum_{i=1}^{n} AC_i + \sum_{j=1}^{m} MC_j + \sum_{k=1}^{m1} ICC_k$$

AC= (2*1)+(0*2)+(0*3)= 2
MC=2
ICC=0
AWCC= 2+2+0 = 4

DerivedClass1

$$AWCC_{dc1} = \sum_{i=1}^{n} AC_i + \sum_{j=1}^{m} MC_j + \sum_{k=1}^{m1} ICC_k$$

AC= (1*1)+(0*2)+(0*3)= 1
MC=2
ICC=3
AWCC= 1+2+3= 6

DerivedClass2

$$AWCC_{dc2} = \sum_{i=1}^{n} AC_i + \sum_{j=1}^{m} MC_j + \sum_{k=1}^{m1} ICC_k$$

AC= (1*1)+(0*2)+(0*3)= 1
MC=2
ICC=3
AWCC= 1+2+3= 6

Total Attribute Weighted Class Complexity of the above object oriented code is given by;

$AWCC = AWCC_{bc} + AWCC_{dc1} + AWCC_{dc2}$

AWCC= 4+6+6 =16

Program 2(with derived data type attributes):

```
#include <iostream>
using namespace std;

class BaseClass {
protected:
  int i, j;
  int a[10],b[10];
public:
  void get() {
    cout << "Enter the first array elements" << endl;
    for(i=0;i<10;i++) {
     cout << "Element " << i+1 << ":" ;
            cin>>a[i];
    }
    cout << "Enter the second array elements" << endl;
    for(j=0;j<10;i++) {
     cout << "Element " << j+1 << ":" ;
            cin>>b[j];
    }
  }
  void show() {
    cout << "First array elements:" << endl;
    for(i=0;i<10;i++) {
     cout << "Element " << i+1 << ":" << a[i];
    }
    cout << "Second array elements:" << endl;
    for(j=0;j<10;i++) {
     cout << "Element " << j+1 << ":" << b[j];
    }
  }
};

// i and j inherited as protected.
class DerivedClass1 : public BaseClass {
  int k;
  int c[10];
public:
  void setk() {
    for(k=0;k<10;k++) {
     c[k]=a[k]*b[k];
    }
  }
  void showk() {
    cout << "Result1:" << endl;
    for(k=0;k<10;k++) {
     cout << "Element " << k+1 << ":" << c[k];
    }

  }
};
class DerivedClass2 : public DerivedClass1 {
  int m;            // i and j inherited indirectly through DerivedC
lass1.
  int d[10];
public:
  void setm() {
    for(m=0;m<10;m++) {
     d[m]=a[m]-b[m];
    }
  }
```

```
 void showm() {
   cout << "Result2:" << endl;
   for(m=0;m<10;m++) {
    cout << "Element " << m+1 << ":" << d[m];
   }
 }
};

int main()
{
 DerivedClass1 object1;
 DerivedClass2 object2;
 object1.set();
 object1.show();
 object1.setk();
 object1.showk();
 object2.set();
 object2.show();
 object2.setk();
 object2.setm();
 object2.showk();
 object2.showm();
 return 0;
}
```

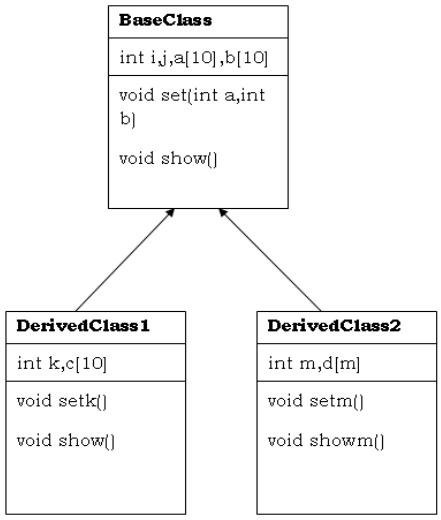

Fig. 2 An example of an object oriented system with derived data type of attributes

BaseClass

$$AWCC_{bc} = \sum_{i=1}^{n} AC_i + \sum_{j=1}^{m} MC_j + \sum_{k=1}^{m1} ICC_k$$

AC= (2*1)+(2*2)+(0*3)= 2 + 4 = 6
MC=6
ICC=0
AWCC= 6+6+0 = 12

DerivedClass1

$$AWCC_{dc1} = \sum_{i=1}^{n} AC_i + \sum_{j=1}^{m} MC_j + \sum_{k=1}^{m1} ICC_k$$

AC= (1*1)+(1*2)+(0*3)= 3
MC=6
ICC=3
AWCC= 3+6+3= 12

DerivedClass2

$$AWCC_{dc2} = \sum_{i=1}^{n} AC_i + \sum_{j=1}^{m} MC_j + \sum_{k=1}^{m1} ICC_k$$

AC= (1*1)+(1*2)+(0*3)= 3
MC=6
ICC=3
AWCC= 3+6+3= 12

Total Attribute Weighted Class Complexity of the above object oriented code is given by;
$AWCC = AWCC_{bc} + AWCC_{dc1} + AWCC_{dc2}$
AWCC= 12+12+12 =36

Program 3(with user defined data type attribute):

```
#include <iostream>
using namespace std;

class BaseClass {
protected:
 int i, j;
 int a[10],b[10];
 Struct sample {
 char name[20];
 int dno;
};
public:
 void get() {
   cout << "Enter the first array elements" << endl;
   for(i=0;i<10;i++) {
    cout << "Element " << i+1 << ":" ;
          cin>>a[i];
   }
   cout << "Enter the second array elements" << endl;
   for(j=0;j<10;i++) {
    cout << "Element " << j+1 << ":" ;
          cin>>b[j];
   }
 }
 void show() {
   cout << "First array elements:" << endl;
   for(i=0;i<10;i++) {
    cout << "Element " << i+1 << ":" << a[i];
   }
   cout << "Second array elements:" << endl;
   for(j=0;j<10;i++) {
    cout << "Element " << j+1 << ":" << b[j];
   }
 }
};

// i and j inherited as protected.
class DerivedClass1 : public BaseClass {
 int k;
 int c[10];
public:
 void setk() {
   for(k=0;k<10;k++) {
    c[k]=a[k]*b[k];
   }
```

```
  }
  void showk() {
    cout << "Result1:" << endl;
    for(k=0;k<10;k++) {
     cout << "Element " << k+1 << ":" << c[k];
    }

  }
};
class DerivedClass2 : public DerivedClass1 {
  int m;            // i and j inherited indirectly through DerivedC
lass1.
  int d[10];
public:
  void setm() {
    for(m=0;m<10;m++) {
     d[m]=a[m]-b[m];
    }
  }
  void showm() {
    cout << "Result2:" << endl;
    for(m=0;m<10;m++) {
     cout << "Element " << m+1 << ":" << d[m];
    }
  }
};
void main()
{
  sample s1;
  cout << "Enter the name of the student";
  cin >> s1.name;
  cout << "Enter the department number of the student";
  cin >> s1.dno;
  DerivedClass1 object1;
  DerivedClass2 object2;

  object1.set();
  cout << "Name: \t" << s1.name << endl << "Dno: \t" <<
s1.dno << endl;
  object1.show();
  object1.setk();
  object1.showk();

  object2.set();
  object2.show();
  object2.setk();
  object2.setm();
  object2.showk();
  object2.showm();

  return 0;
}
```

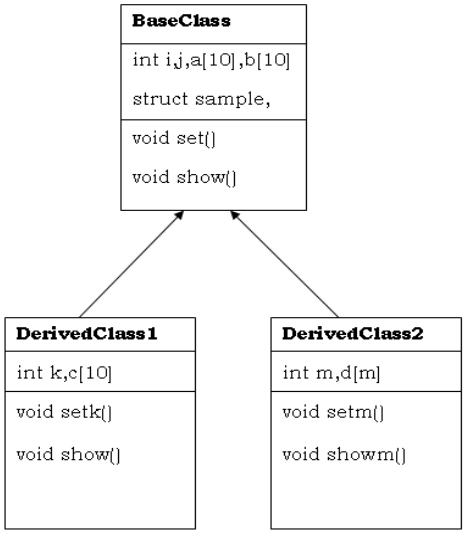

Fig. 3 An example of an object oriented system with user defined data type of attributes

BaseClass

$$AWCC_{bc} = \sum_{i=1}^{n} AC_i + \sum_{j=1}^{m} MC_j + \sum_{k=1}^{m1} ICC_k$$

AC= (2*1)+(2*2)+(2*3)= 2 + 4 + 6 = 12

MC=6

ICC=0

AWCC= 12+6+0 = 18

DerivedClass1

$$AWCC_{dc1} = \sum_{i=1}^{n} AC_i + \sum_{j=1}^{m} MC_j + \sum_{k=1}^{m1} ICC_k$$

AC= (1*1)+(1*2)+(0*3)= 3

MC=6

ICC=3

AWCC= 3+6+3= 12

DerivedClass2

$$AWCC_{dc2} = \sum_{i=1}^{n} AC_i + \sum_{j=1}^{m} MC_j + \sum_{k=1}^{m1} ICC_k$$

AC= (1*1)+(1*2)+(0*3)= 3

MC=6

ICC=3

AWCC= 3+6+3= 12

Total Attribute Weighted Class Complexity of the above object oriented code is given by;

$AWCC = AWCC_{bc} + AWCC_{dc1} + AWCC_{dc2}$

AWCC= 18+12+12 =42

V. Comparison With Other Measures

A comparative study has been made with most widely accepted CK metric suite [9] and found that none of the CK metrics provides the total complexity of the class by considering the complexity due to internal architecture of the code (methods and attributes). This differentiates our metric from the CK metrics. Mishra et.al suggested that one can calculate the complexity of the class by using cognitive weights of the methods and attributes. In our earlier paper [1], we introduced a method for measuring the complexity of a class with the inheritance. The current metric is one step ahead of EWCC. It also considers the complexity that arises due to the data type of attributes. Another advantage of our metric is that, it takes cognitive weights into consideration. In the following Table III, a comparison has been demonstrated with EWCC and AWCC.

We calculated the weight of each class by calculating Attribute Complexity (AC), Method Complexity (MC), and Inherited Class Complexity (ICC) which is better indicator than the EWCC. The weight of each method is calculated by using cognitive weights and the approach suggested by Chidamber et al. We found that the resulting value of AWCC is higher than the EWCC. This is because, in EWCC, the weight of each attribute is assumed to be one. However, including cognitive weights for calculation of the attribute complexity (AWCC) is more realistic because it provides for the complexity of the internal architecture of attribute. A tool was developed and used to measure EWCC and AWCC of three different OO programs. The results are tabulated in Table III.

TABLE III
COMPLEXITY VALUES FOR DIFFERENT CLASSES FOR THE CHOSEN METRICS

Metrics / Programs	WMC	WCC	EWCC	AWCC
1	6	10	16	16
2	6	26	30	36
3	6	27	29	42

The WMC, WCC, EWCC and AWCC values were compared and found that AWCC measure was larger. According to Mishra et.al, WCC is a better indicator of complexity than WMC because it shows a higher value of complexity for a given class. Arockiam et.al has proven that, EWCC has a greater complexity than WCC because it shows an accurate value of complexity for a given class with inheritance. From the table 3, it is observed that AWCC value is larger than EWCC value which concludes that AWCC is a better indicator of complexity of the classes with inheritance because of the consideration of attribute complexity.

VI. Conclusion and Future Work

An Attribute Weighted Class Complexity (AWCC) metric for measuring the class level complexity has been formulated. The complexity of the class includes the internal complexity of the class and the inherited classes' complexity. AWCC includes the cognitive complexity due to internal architecture of the attributes, methods and the inherited complexity. AWCC has proven that, complexity of the class getting affected, which is based on the cognitive weights of the different attributes. The metric is evaluated through a case study and a comparative study, and proved to be a better indicator of the class level complexity. A tool was developed to calculate the AWCC value and to compare it with other metrics. The proposed metric focuses only on the data type. Further, it may be evaluated with the detailed data types available in the 3 categories like PDT, DDT, and UDDT. Newer metrics may also be proposed and validated for assessing the cognitive complexity of other object oriented features.

References

[1] Arockiam. L, Aloysius. A,Charles selvaraj. J "Extended Weighted Class Complexity: A new measure of software complexity for objected oriented systems", Proceedings of International Conference on Semantic E-business and Enterprise computing (SEEC), 2009, pp. 77-80.

[2] Charles Selvaraj. J, Aloysius. A, and Arockiam. L , "A Comparision of Proposed Cognitive weights for control structures and object oriented programming languages", Proceedings of International Conference on Advanced Computing (ICAC09), 2009, pp. 380-385.

[3] Sanjay Misra and k. Ibrahim Akman, "Weighted Class Complexity: A Measure of Complexity for Object Oriented System," Journal of Information Science and Engineering 24, 2008, pp. 1689-1708.

[4] Mc Quillan. J. A and Power. J. F, "On the application of software metrics to UML model," Lecture Notes in Computer Science, Vol. 4364, 2007, pp. 217-226.

[5] Ranjeeth. S, Ramu Naidoo "An Investigation Into The Relationship Between The Level Of Cognitive Maturity And The Types Of Errors Made By Students In A Computer Programming" College Teaching Methods & Style Journal-Second Quarter, 2007, pp. 31-40.

[6] Rajnish. K, Bhattacherjee. V," A New Metric for Class Inheritance Hierarchy: An Illustration", proceedings of National Conference on Emerging Principles and Practices of Computer Science & Information Technology", GNDEC, Ludhiana, 2006, pp. 321-325.

[7] Wang. Y and Shao. J, "A new measure of software complexity based on cognitive Weights." IEEE Canadian Journal of Electrical and Computer Engineering, 2003, pp. 69-74.

[8] Basili. VR, Briand. L. C, Melo. WL, "A validation of object oriented design metrics as quality indicators", Technical report,University of Maryland, Department of Computer Science,1995, pp. 1-24.

[9] Chidamber. S. R and Kemerer. C. F, "A Metric Suite for Object-Oriented Design", IEEE Trans. on Software Engineering, 1994, 476-493.

[10] Harrison. R, Counsell. SJ, Nithi. RV, "An evaluation of the MOOD set of Object-oriented software metrics", IEEE Trans.On Software Engineering, 1998, pp. 491- 496.

[11] Wang. Y, "On Cognitive Informatics." IEEE International Conference on Cognitive Informatics, 2002, pp. 69-74.