Limitations of an Object-Oriented Metric : Weighted Complexity Measure

D. I. De Silva and N. Kodagoda
Department of Information Technology
Faculty of Computing
Sri Lanka Institute of Information Technology
New Kandy Road, Malabe, Sri Lanka
{dilshan.i & nuwan.k}@sliit.lk

S. R. Kodituwakku and A. J. Pinidiyaarachchi
Department of Statistics and Computer Science
Faculty of Science
University of Peradeniya
Peradeniya, Sri Lanka
{salukak and ajp}@pdn.ac.lk

Abstract—**Many computer science practitioners and software developers believes that the complexity of a program could be controlled more effectively by using object-oriented programming concepts. In addition to controlling complexity, the object-oriented approach allows faster development, reduction in costs, higher quality, easier maintenance, increased scalability, better information structures, and increased adaptability. As such, more and more programs are written using the object-oriented programming approach rather than using the traditional functional approach. This demand has spurred the provision for a number of object-oriented metrics. Out of them, Chidamber and Kemerers' metrics suite is one of the most prominent object-oriented metrics that has been proposed. It has been widely validated and has been accepted as a useful predictor of object-oriented design complexity. But it does not consider the complexities that occur due to factors such as the nesting level and type of control structures, and the size of the program. Thus, Chhillar and Bhasins' introduced the weighted complexity measure to address these issues. It is the only metric which considers the complexities that occur due to inheritance level of statements, nesting level and type of control structures, and the size of the program. However, weighted complexity measure also has some limitations. This paper attempts to draw the readers' attention to those limitations, with the hope that it will be further improved by addressing them.**

Keywords- software complexity; object-oriented metrics; weighted complexity measure

I. INTRODUCTION

Unlike traditional approaches which mainly focused on a function-oriented view that separates procedures and data, the concept of object-orientation revolves around modeling the real world problems in terms objects. In object oriented programming (OOP) an object is simply an instance of a class. By making use of attributes and operations a class describes an object. A program written to solve a real world problem might consist of many classes. OOP provides five main concepts: localization, inheritance, encapsulation, polymorphism (information hiding) and abstraction to properly manage the classes in a program and reduce the complexity it. Many computer science practitioners and software developers believed that the complexity of a program could be controlled more effectively by using these OOP concepts. In addition to controlling complexity, the object-oriented (OO) approach allowed faster development, reduction in costs, higher quality, easier maintenance, increased scalability, better information structures, and increased adaptability [1], [19]. As such, more and more programs were written using the OOP approach rather than using the traditional functional approach.

This made way for researchers to start introducing complexity metrics for OO programs. Out of the proposed OO metrics, the suite of six metrics that was introduced by Chidamber and Kemerer is widely accepted as a useful predictor of OO design complexity. Basili, Briand and Melo proved the validity of this metrics suite by empirically investigating them using eight medium-sized information management systems written in C++ [2]. But this metrics suite does not consider the complexities that occur due to factors such as the nesting level and type of control structures, and the size of the program. Thus, Chhillar and Bhasin introduced weighted complexity measure by considering the complexities that occurred due to the inheritance level of statements, nesting level and type of control structures, and the size of a program. Although weighted complexity measure is a useful metric, it also has some limitations.

The main intent of this paper is to draw the readers' attention to the limitations of the Chhillar and Bhasins' weighted complexity measure so that it can be further improved by addressing them. The rest of the paper is organized as follows. Section II discusses the proposed OO complexity metrics. Section III presents an overview of the weighted complexity measure. Section IV describes the limitations of the weighted complexity measure. Finally section V concludes the paper.

II. OBJECT-ORIENTED COMPLEXITY METRICS

With OOP becoming one of the most prominent developments in computer programming, proposing metrics to identify the complexities that occur in programs written using OO concepts became increasingly popular. This prompted researchers in proposing complexity metrics to measure the complexity of OO programs.

In 1989, Moreau and Dominck suggested three metrics for OO graphical information systems [3]. Pfleeger introduced a cost estimation model for OO development in 1990 [4]. One of the significant developments for OO metrics investigation was

978-1-4799-8353-7 /15/$31.00 ©2015 IEEE

Chidamber and Kemerers' 1991 metric suite. It consisted of six metrics: weighted methods per class (WMC), depth of inheritance tree (DIT), number of children (NOC), coupling between object (CBO), response for a class (RFC), and lack of cohesion in methods (LCOM), describing the design and complexity of object-oriented software [5]. In 1992 and 1993, Rajaraman and Lyu [9] and Li and Henry [10] tested and measured these metrics for applications developed by university students. In 1994, Chidamber and Kemerer revised these six metrics using measurement theory and empirical data [11]. Later in 1996 and 1999, they were validated by Basili [2] and Tang in [6].

Sheetz, Tegarden and Monarchi also proposed a set of measures which measured the complexity of an OO system at the variable, method, object, and application levels in 1991[7]. Complexity of these levels was presented as function of measurable characteristics such as fan-in, fan-out, number of input/output variables, fan-up, fan-down, and polymorphism. In 1992, Lake and Cook proposed metrics to measure the inheritance of C++ programs [8]. In April 1993, Chen and Lui introduced eight metrics: operation complexity metric (OXM), operation argument complexity metric (OACM), attribute complexity metric (ACM), operation coupling metric (OCM), class coupling metric (CCM), cohesion metric (CM), class hierarchy metric (CHM) and, reuse metric (RM) to measure the complexity and reusability of object-oriented software [14]. In October 1994, metrics for object-oriented design (MOOD) was proposed by Abreu et al. These MOOD metrics: method inheritance factor (MIF), attribute inheritance factor (AIF), method hiding factor (MHF), attribute hiding factor (AHF), polymorphism factor (PF), and coupling factor (CF) were able to measure OO mechanisms such as inheritance, encapsulation, polymorphism, message passing.

In 1998, W. Li proposed an alternative suite of OO metrics to overcome some of the limitations that were present in Chidamber and Kemerers' metric suite [12]. Li's metric suite also consisted of six metrics : number of ancestor classes (NAC), number of descendent classes (NDC), number of local methods (NLM), class method complexity (CMC), coupling through abstract data type (CTA), and coupling through message passing (CTM). Out of these metrics NAC and NDC metrics were proposed as alternatives for DIT and NOC.In January 1999, C. R. Douce, P. L. Layzell, and J. Buckley introduced a set of spatial measures of software complexity for both the procedural and object-oriented codes [15].

In 2002, Bansiya and Davis proposed quality model for object oriented design (QMOOD). This QMOOD metrics: design size in classes (DSC), number of hierarchies (NOH), average number of ancestors (ANA), data access metric (DAM), direct class coupling (DCC), class interface size (CIS), measure of aggregation (MOA), cohesion among method of class (CAM), measure of functional abstraction (MFA), number of polymorphic methods (NOP), number of methods (NOM) were able to assess quality attributes such as reusability, functionality, effectiveness, understandability, extendibility and flexibility[13].

In 2007, Mishra proposed class complexity (CC) metric which measured the complexity of a class from a method level [16]. Yadav and Khans' 2009 metric calculated the overall complexity of design hierarchy by considering the inherited methods [17]. By taking into account the inheritance level of statements in classes, types of control structures, and nesting level of control structures, Chhillar and Bhasin proposed a weighted composite complexity measure for OO systems in 2011 [18].

III. Weighted Complexity Measure [18]

Chhillar and Bhasin believed that it is not just a single factor that contributes to the complexity of software; rather it is a combination of several factors. Taking this into consideration, they proposed the weighted complexity measure based on four prominent complexity factors. The four factors are as follows:

Inheritance level of statements in classes (Wi): The degree of understanding a statement increases with the level of inheritance of classes. Taking this into account a weight of zero is assigned to executable statements in the bass class, one for the executable statements which are at the first derived class, two for the statements at the next derived class and so on.

Type of control structures in program (Wc): The complexity added to a program by a control structure varies depending on its type. Thus, a weight of zero is assigned to sequential statements, one for conditional control structures such as if-else, if-elseif conditions, two for iterative control structures such as for, while and do-while loops, and n for switch-case statements with n cases.

Nesting level of control Structures (Wn): With the level of nesting of control structures the understandability of the statements which are inside them increases and consequently adds more complexity to the program. Thus, to consider the complexity added by nesting level of control structures, a weight of zero is assigned for sequential statements, one for statements which are at the outer most level, two for statements which are at the next inner level of nesting and so on.

Size of a program: The complexity of a program increases along with the size of it. The size of a particular executable statement is calculated in terms of the operators, operands, methods, and strings in that statement.

By taking the four factors mentioned above, a weighted complexity measure for an object-oriented program P is suggested as:

$$C_w(P) = \sum_{j-1}^{n} (S_j) * (W_t)_j$$

Where:

$C_W(P)$ = Proposed weighted complexity measure of program P

S_j = Size of j^{th} executable statement in terms of tokens count

n = Total number of executable statements in program P

j = Index variable

$W_t = W_n + W_i + W_c$

A calculation of the weighted complexity measure for the program in Table 1 can be found in Table 2.

IV. Limitations of the Weigthed Complexity Measure

Out of the main OOP concepts, the weighted complexity measure only considers the complexities that arise due to inheritance. However, the complexities that arise due to encapsulation, polymorphism (information hiding) and abstraction should also be considered to improve the accuracy of the metric. In addition to those object-oriented concepts the factors discussed below can also be considered to improve the accuracy of the metric.

Conditional statements can be divided into two types: simple and compound. Chhillar and Bhasin only allocated a weight of one for all conditional statements (if-else and if-else if) without considering the type of it. But the accuracy of the complexity calculation can be further improved if the weight allocation differs depending on the type of the conditional statement. In the program that is shown in Table 1 the weight assigned for both the conditional statements in line number 27 and 51 is one. But the complexity of the compound conditional statement in line number 27 should be more than the simple conditional statement in line number 51.

Chhillar and Bhasin have assigned a weight of two for iterative control structures (for, while and do-while loops). However, the complexity calculation of the weighted complexity measure can be further improved by considering the complexity that occurs due to the number of iterations.

A program can consist of normal and recursive functions. In most times a problem that can be solved with a recursion can also be solved by using a 'for' or 'while' loop as well. Using a loop may be much easier to understand than a recursive function. But there may be times where a recursion function is the optimal solution. This measure calculates the complexities that arise due to a functional call and recursive functional call in a similar manner. However, the complexity of a functional call should be less than a recursive functional call.

This measure does not take into consideration the complexities that occur due to class, array, and object declarations. Thus, line numbers 3, 13, 19, 25, 33, 48, 55, and 56 in the program that is shown in Table 1 will not lead to any complexity. However, the complexity associated with a program should change depending on the number of class, array, and object declarations in it.

The number of methods and the complexity of the methods is a clear indicator of the time and effort required to implement and maintain a program [5]. This has been addressed by the weighted complexity measure. In addition to that the complexities that occur due to method calling have also been considered up to some extent. However, according to the proposed measure the complexities that occur due to method calling would not differ depending on the content of the method that is called. For example, the statements at line number 59 and 61 in Table 1 have the same complexity values. But out of the two statements, the statement at line number 59 should have a higher complexity value.

TABLE I A Sample Program Q

1	# include <iostream>
2	using namespace std;
3	class Polygon {
4	protected:
5	int width, height;
6	public:
7	void setValues(int a, int b);
8	};
9	void Polygon :: setValues(int a, int b){
10	width=a;
11	height=b; }
12	
13	class Output {
14	public:
15	void print( );
16	};
17	void Output :: print( )
18	{cout << "Area is : "; }
19	class Rectangle: public Polygon {
20	public:
21	int area( );
22	int perimeter( );
23	};
24	int Rectangle :: area( ){
25	Output out;
26	out.print( );
27	if(width >0 && height >0)
28	return width * height;
29	else
30	return 0; }
31	int Rectangle :: perimeter( )
32	{ return (width + height) * 2; }
33	class Triangle: public Polygon {
34	int ans;
35	public:
36	int area( );
37	};
47	int Triangle :: area( ){
48	Output out;
49	out.print( );
50	ans = (width * height)/2;
51	if(ans > 0)
52	return ans;
53	}
54	int main ( ){
55	Triangle tri;
56	Rectangle rec;
57	tri.setValues (4,5);
58	rec.setValues (4,5);
59	cout << rec.area( ) << '\n';
60	cout << tri.area( ) << '\n';
61	cout << rec.perimeter( )<< '\n';
62	return 0;
63	}

TABLE II CALCULATION OF WEIGHTED COMPLEXITY MEASURE FOR PROGRAM Q

Line No	S_j	W_n	W_i	W_c	W_t	$S_j*(W_t)_j$
9	4	0	1	0	1	4
10	3	0	1	0	1	3
11	3	0	1	0	1	3
17	4	0	1	0	1	4
18	3	0	1	0	1	3
24	4	0	2	0	2	8
26	3	0	2	0	2	6
27	8	1	2	1	4	32
28	4	1	2	0	3	12
30	2	1	2	0	3	6
31	4	0	2	0	2	8
32	6	0	2	0	2	12
47	4	0	2	0	2	8
49	3	0	2	0	2	6
50	7	0	2	0	2	14
51	4	1	2	1	4	16
52	2	1	2	0	3	6
54	2	0	0	0	0	0
57	6	0	1	0	1	6
58	6	0	1	0	1	6
59	7	0	2	0	2	14
60	7	0	2	0	2	14
61	7	0	2	0	2	14
62	2	0	0	0	0	0
$C_w(P)$						**205**

V. CONCLUSION

A number of OO complexity metrics have been proposed. Out of them Chhillar and Bhasins' weighted complexity measure is the only metric that considers the complexity that arise due to inheritance level of statements, type of control structures, nesting level of control structures and size of the program. However, there exist some other factors which the weighted complexity measure has not considered. This paper puts forward such factors in an attempt to further improve the accuracy of the weighted complexity measure.

REFERENCES

[1] D.A. Taylor, "Object-Oriented Technology: A Manager's Guide," Addison-Wesley, Reading, MA, 1990; ISBN: 0-201-56358-4.

[2] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of objectoriented design metrics as quality indicators," IEEE Transactions on Software Engineering, vol. 22, pp. 751–761, 1996.

[3] D. R. Moreau, W. D. Dominick, "Object-oriented graphical information systems: research plan and evaluation metrics," J. Syst. and Software, vol. 10, pp.23-28, 1989.

[4] S. L. Pfleeger and J. D. Palmer, "Software estimation for object_oriented systems," Int. Function Point Users Group Fall Conf., San Antonia, TX, pp. 181-196,1990.

[5] S. R. Chidamber and C.F. Kemerer, "Towards a metrics suite for Object Oriented Design," in Proc. OOPSLA '91 Conference proceedings on Object-oriented programming systems, languages, and applications, New York, USA, vol. 26, pp.197- 211, Nov. 1991.

[6] A. Kamandi, "Object-oritented metrics, Sharif University of Technology," Spring 2007.

[7] S. D. Sheetzq, D.P. Tegarden and D.E. Monarchi, "Measuring object-oriented system complexity," Working Paper, Univ. of Colorado, 1992.

[8] A. Lake and C. Cook, "A software complexity metric for C++," Tech. Rep. 92-60-03, Oregon State Univ., 1992.

[9] C. Rajaraman and M. R. Lyu, "Some coupling measures for C++ programs," TOOLS USA 92, vol. 6, pp. 225-234, 1992.

[10] W. Li and S. Henry, "Maintence metrics for the object-oriented paradigm," in First Int, Software Metrics Symp., Baltimore, MD, pp. 52-60, 1993.

[11] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," IEEE Transactions on Software Engineering, vol. 20, pp. 476-493, June 1994.

[12] W. Li, "Another metric suite for object-oriented programming," The Journal of System and Software, vol. 44, pp. 155-162, December 1998.

[13] J. Bansiya, C. G. Davis, "A hierarchical model for object-oriented design quality assessment," IEEE Transactions on Software Engineering, pp. 4-17, 2002.

[14] Y. J. Chen, J. F. Lui, "A new metric for object-oriented design, Infromation of Software Technology," vol. 35, pp.232-240, April 1993.

[15] C. R. Douce, P. J. Layzell, and J. Buckley, "Spatial measures of software complexity," in Proc. 11th Meeting of Psychology of Programming Interest Group, Leeds. Limerick, Ireland, pp 1-8, Jan. 1999.

[16] S. Misra, "An Object Oriented Complexity Metric Based on Cognitive Weights," 6th IEEE International Conference on Cognitive Informatics (ICCI 07), pp. 134-139, 2007.

[17] A. Yadav, R. A. Khan, "Measuring Design Complexity - An Inherited Method Perspective," SIGSOFT Software Engineering Notes, vol. 34, July 2009.

[18] U. Chhillar, S. Bhasin, "A new weighted composite complexity for object-oriented systems", International Journal of Information and Communication Technology Research, vol. 1, pp.101-108, July 2011.

[19] L.C. Briand, C Bunse, J. Wust, C. Differding, "An experimental comparison of the maintainability of object-oriented and structured design documents," Empirical Software Engineering, vol. 2, pp. 291-312, 1997.

[20] F. Brito e Abreu, "The MOOD Metrics Set," Proc. ECOOP'95, Workshop on Metrics, 1995.