New Proposed Inheritance Metrics to Measure the Software Complexity

Preeti Gulia
Department of Computer Science & Applications
M. D. University, Rohtak-124 001, Haryana, INDIA

Rajender S. Chillar
Department of Computer Science & Applications
M. D. University, Rohtak-124 001, Haryana, INDIA

ABSTRACT

Inheritance is an important aspect of object-oriented paradigm during software development. Inheritance supports the class hierarchy design and relation between classes and inheritance also has an impact on the complexity of software. Complexity of software increases the testing and maintenance efforts. So researchers and developer always try to reduce the software complexity because low software complexity reduce testing and maintainability. In this study, we propose two new inheritance metrics based on level of methods like CCDIT (Class Complexity due to Depth of Inheritance Tree) and CCNOC (Class Complexity due to Number of Children) to measure the complexity of methods in classes. Firstly we present the Chidamber & Kemerer (C & K) metrics for class inheritance and related work. Secondly we measure and investigate the software complexity by generating UML diagram of software. Lastly we present comparison of newly proposed metrics with other inheritance metrics proposed by other researchers.

General Terms:

Design, Measurement and Experimentation.

Keywords:

Complexity, software metrics, inheritance, NOC, DIT, Object Oriented system

1. INTRODUCTION

Software measurement is the fundamental aspect of any process or product for its success and metrics are the unit of measurement, which are used to describe the product, process and people of software engineering. From last two decades, object-oriented technologies come into existence for fast development of software to reduce the time and cost. The recent drive towards object-oriented technology forces the growth of object-oriented software metrics [1]. Several metrics have been proposed by researchers and practitioners like C & K metrics suite, MOOD (Metrics for Object Oriented Design) metrics, Lorenz and Kidd metrics etc. [2, 3, 4, 5, and 6]. The metric suite proposed by C & K is one of the best known object-oriented metrics that is used in measuring the complexity of the software. Software complexity measures serve both as an analyzer and a predictor in quantitative software engineering. To develop better quality software, it is necessary to identify the complexity at module, method and class level. Coupling, cohesion and inheritance have an effect on complexity [7]. In this paper, our main focus is to measure the software complexity through inheritance by proposing two new metrics for the class inheritance hierarchy.

2. RELATED WORK

Inheritance is the key feature of object-oriented technology as it increases the reusability. Many studies [8, 9, and 10] have found that use of inheritance reduces the software maintenance and testing efforts. The reuse of software by inheritance is claimed to make maintainable, understandable and reliable software [11, 12, and 13]. Misra et al. [14] proposed metrics to calculate the complexity at method level by using inheritance. They also measure complexity of a class due to method and attribute in terms of cognitive weight. Deepti et al. [15] measure the complexity with the help of CCI (Class Complexity due to Inheritance) and ACI (Average Complexity of a program due to inheritance) metrics. But Harrison, Counsell and Nithi [12] contradict through experimental assessment that systems without inheritance are easier to understand and modify. The main reason is that to inherit a new class, it is necessary to understand the implementation of parent as well as any of the parent's ancestors. Rather than this, inheritance within object-oriented system is a way to increase the readability and reusability.

C & K Metrics

C & K inheritance metrics proposed by Chidamber & Kemerer are following [1]:

2.1 DIT (Depth of Inheritance Tree)

DIT is the maximum distance from the node to the root. DIT metric measures how many ancestor classes are affected by the node. The deeper a class is in hierarchy, the higher the degree of method inheritance, which makes a class more complex to predict its behavior. Deeper trees have greater design complexity, since more methods and classes are involved.

2.2 NOC (Number of Children)

NOC is the measure of immediate subclasses to a class in the class hierarchy. It means it is the measure of how many subclasses are going to inherit the methods of parent class. Greater the number of children indicates greater the reuse, since inheritance is a form of reuse. The number of children gives the idea regarding the effect of class on the overall design.

3. PROPOSED METRICS

A class is composed of attributes and methods. In this proposal, inheritance is to be measured in terms of sum of methods at each level or for each class. For CCDIT, it is the sum of methods for maximum length from the node to the root of the tree. This measures the complexity depth wise. For CCNOC, this is the sum of methods of children including the methods of parent class. We propose two metrics for inheritance namely CCDIT (Class Complexity due to Depth of Inheritance Tree) and CCNOC (Class Complexity due to Number of Children). With the help of these metrics, we can measure the complexity according to depth and breadth. The definitions of these metrics are as follows:

3.1 CCDIT (Class Complexity Due To Depth of Inheritance Tree)

It is the sum of methods for maximum length from the node to the root of the tree.

$$\sum_{i=n \text{ to } i=P(i)}^{-1} \qquad \sum_{j=i}^{M} \text{Node (i)}$$

Where n is maximum distance or length node entry

M is number of methods in node I or class

3.2 CCNOC (Class Complexity Due To Number of Children)

It is the sum of methods of children including the methods of parent class.

$$Mi + \sum_{j=1}^{n} M_{i+1,j}$$

Where Mi is method of node at level i

Mi+1 = number of method at level (i + 1)

These metrics are calculated with the help of an example for different classes. In this example figure of shapes is taken by us that contains the name of the class and methods of that class. With the help of this, we can calculate the complexity of different classes.

We can calculate the metrics values according to the given definitions. For example: The *figure 4.1* consists of the hierarchy of shapes. It consists of classes like figure, zero dimensional, one dimensional, two dimensional, point, line arc, spline, polygon, circle etc. and the classes also contain their own functions or methods such as move(), select(), rotate() and display() for the class figure. According to the given definition, the values for different classes for the CCDIT are as follows:

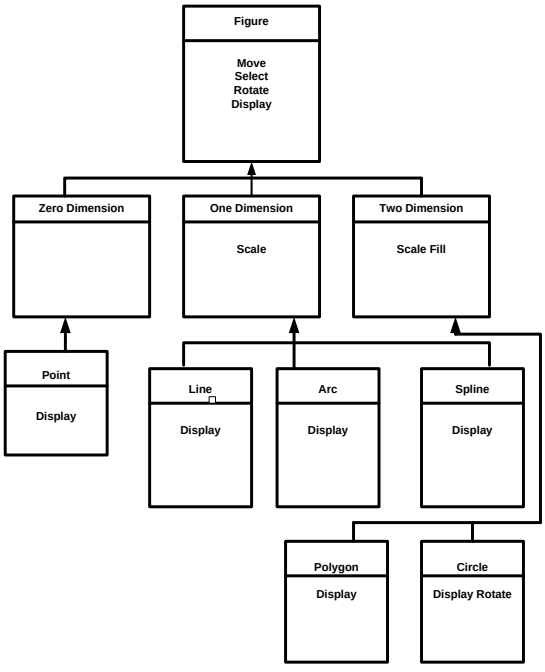

Figure 4.1 hierarchy of Shape

For the Class Figure as shown in *Figure 4.1*: the length of this class is zero and it consists of 4 methods, therefore, the value for the metric is 4, because CCDIT is the sum of methods for the maximum length from the node to root of the tree. Like this, the values for different classes are as follows:

Zero dimensional = 0+4 =4.

One dimensional = 1+4 =5.

Two dimensional = 2+4 =6.

Point = 1+0+4 = 5.

Line = 1+1+4 = 6.

Arc = 1+1+4 = 6.

Spline = 1+1+4 = 6.

Polygon = 1+2+4 = 7.

Circle = 2+2+4 = 8.

Now, we calculate the values for different classes for CCNOC according to the definition i.e. the sum of methods of children including the methods of parent class. The calculation is as follows:

For the class figure as shown in *Figure 4.1*: class figure consists of three children or subclasses, therefore the value for this class is the sum of method of classes figure, zero dimensional, one dimensional and two dimensional i.e. CCNOC for class figure is = 4+0+1+2 = 7.

Zero dimensional = 0+1 = 1.

One dimensional = 1+1+1+1 =4.

Two dimensional = 2+1+2 =5.

Point = 1

Line = 1

Arc = 1

Spline = 1

Polygon = 1.

Circle = 2.

In the last six classes i.e. point, line, arc, spline, polygon, circle, the values are taken from single class because these classes do not comprise any child or subclass. The values or complexity of different classes is shown in Table 3.1.

Table 3.1 Complexity Values of Proposed Metrics

Name of the Class	CCDIT	CCNOC
Figure	4	7
Zero Dimen.	4	1
One Dimen.	5	4
Two Dimen.	6	5
Point	5	1
Line	6	1
Arc	6	1
Spline	6	1

Polygon	7	1
Circle	8	2

4. COMPARISON WITH OTHER INHERITANCE METRICS:

The proposed metrics are compared with some other inheritance metrics. Following are some inheritance metrics with which comparison is shown:

4.1 Inheritance Metrics by C & K [8]:

Depth of inheritance tree (DIT): This is the depth of the inheritance

Number of Children (NOC): This is the number of immediate subclasses of a super class.

4.2 Inheritance Metrics by Li [17]:

Number of Ancestor Classes (NAC): This is the total number of ancestor classes from which a class inherits.

Number of Descendent Classes (NDC): This specifies total number of Descendent classes (subclasses) of a class.

Values of different metrics including proposed metrics for different classes shown in figure 4.1 are presented in the table 4.1 that is given below

Table 4.1 Comparison of different Metrics

Class	DIT	NOC	NAC	NDC	CCDIT	CCNOC
Figure	0	3	0	9	4	7
Zero dimen.	1	1	1	1	4	1
One dimen.	1	3	1	3	5	4
Two dimen.	1	2	1	2	6	5
Point	2	0	2	0	5	1
Line	2	0	2	0	6	1
Arc	2	0	2	0	6	1
Spline	2	0	2	0	6	1
Polygon	2	0	2	0	7	1
Circle	2	0	2	0	8	2

It is evident that at the lower level in the inheritance tree classes are more complex. Because understanding of these classes require understanding of the implementation of parent as well as the ancestors of the parents. To determine the complexity of a class according to the depth, Depth of Inheritance Tree (DIT), Number of Ancestor Classes (NAC), and our proposed metric Class Complexity due to Depth of Inheritance tree (CCDIT) are more suitable since the metric values are higher for the lower level classes. To determine the complexity according to breadth, Number of Children (NOC), Number of Descendent Class (NDC) and proposed metric Class Complexity due to Number of Children are suitable. The thing which makes our proposed metrics different from other inheritance metrics is that the proposed metrics measure the complexity according to number of methods present in a class. And on the basis of present methods in a class, complexity can be determined in better manner.

5. CONCLUSION

Complexity measure can be used to check the quality of software system. Complexity of the class depends on methods and attributes. Testing efforts and maintenance can be reduced with less complexity. In this paper, we calculate the complexity with the methods. The proposed metrics are defined and calculated with the help of example and compared with other inheritance metrics. To prove that why proposed metrics Class Complexity due to Inheritance Tree (CCDIT) and Class Complexity due to Number of Children (CCNOC) are better than other metrics, reasons are also given.

6. REFERENCES

[1] Booch. G. 1991. Object-Oriented Design and Application, Benjamin/Cummings, Mento Park, CA.

[2] Bieman. J. M and B.K. Kang 1995. "Cohesion and Reuse in an Object-Oriented System", in Proc. Symp. Software Reliability, 259-26.

[3] Briand. L. C, Daly. J. W and Wust. J. K. 1998. "A Unified Framework for Cohesion Measurement in Object-Oriented Systems", Empirical Software Eng., 1, 1, 65-117.

[4] Brotoeabreu. F. 1995. D Metrics Set", in Proc. ECOOP'95 Workshop Metrics.

[5] Chae, H.S, Kwon. Y. R and Bae.D.H. 2000. Cohesion Measures for Object-Oriented Classes", Software practice and Experiences, 30, 12, 1405-1431.

[6] Churcher. N. I and Sheppered. M. J. 1995. Comments on "A Metric Suite for Object-Oriented Design", IEEE Trans. on Software Engineering. 21, 263-265.

[7] El-Emam, K. 2002. Object-oriented metrics: A review of theory and practice. In: Erdogmus, H., Tanir, O. (eds.) Advances in Software Engineering, pp. 23–50. Springer, New York.

[8] Chidamber, S.R. and Kemerer, C.F. 1994. A metrics suite for object oriented design. IEEE Transactions on Software Engineering 20(6), 476-493.

[9] Basili, V.R. 1990. Viewing maintenance as reuse oriented software development. IEEE software 7(1), 19-25.

[10] Cartwright, M. and Shepperd, M. 1996. An empirical analysis of object oriented software in industry. In: Bournemouth Metrics Workshop, Bournemouth, UK.

[11] Basili, V.R., Briand, L.C. and Melo, W.L. 1996. A validation of object-oriented design metrics as quality indicators. IEEE Transactions on Software Engineering, 22(10), 751-761.

[12] Basili, V.R., Briand, L.C. and Melo, W.L. 1996. How reuse influences productivity in object oriented systems. Commun. ACM 39(10), 104-116.

[13] Briand, L., Bunse, L., Daly, J. and Differding, C. 1997.

An experimental comparison of the maintainability of object-oriented and structured design documents. In: Proceedings of Empirical Assessment in Software Engineering (EASE), Keele, UK.

[14] Misra, S. and Akman, I. 2008. "Weighted Class Complexity: A Measure of Complexity for Object Oriented System", JISE.

[15] Mishra, D. and Mishra, A. 2009. "Object Oriented Inheritance Metrics: Cognitive Complexity Perspective", Springer Verlag,

[16] Harrison, R., Counsell, S. and Nithi, R. 2000. Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems. Journal of Systems and Software, 52, 173-179.

[17] Li, W. 1998. "Another Metric Suite for Object-Oriented Programming" Journal of System and Software, 44, 155-162.

[18] Blaha, M., Rambaugh 2005. "Object Oriented Analysis and Design with UML2"