Comparing Complexity in Accordance with Object Oriented Metrics

Dr. Rakesh Kumar
Reader, Department of Computer Science and Applications, Kurukshetra University Kurukshetra

Gurvinder Kaur
Lecturer, Guru Nanak Khalsa Institute of Technology and Management Studies, Yamuna Nagar

ABSTRACT
Metrics are essential in software engineering for measuring software complexity, quality, estimating size and project effort. The major techniques for software cost estimation are sizing or predication of various kinds of software deliverable items. The cost estimation techniques consists of various categories like tools and methods for estimating and measuring software size, function points, lines of code, and object points. This paper highlights the object-oriented software metrics proposed in 90s' by Chidamber, Kemerer and several studies were conducted to validate the metrics and discovered several deficiencies. Further new object oriented metrics were proposed by Li. Chidamber, Kemerer proposed six software metrics as Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC) , Coupling Between Object classes (CBO), Response For a Class (RFC), Lack of Cohesion in Methods (LCOM). A new metrics suite for object-oriented programming proposed by Li includes Number of Ancestor Classes (NAC), Number of Local Methods (NLM), Class Method Complexity (CMC), Number of Descendent Classes (NDC), Coupling Through Abstract Data Type (CTA), and Coupling Through Message Passing (CTM) as an alternatives to Chidamber and Kemerer metrics. Here the comparisons have been made between the metrics proposed by Chidamber, Kemerer and Li.

Keywords
Software metrics, object-oriented metrics, sizing, software complexity, software measurement.

1. INTRODUCTION
The software engineers were of the view that the isolating objects makes their software easier to manage but many of them were of reverse views that software becomes more complex to maintain and document, or even to engineer from the start. This made the move towards the object-oriented paradigm (OOP) as it could increase the capability of programming through its reusability function. Researchers studied ways to maintain software quality and developed object-oriented programming in part to address common problems by strongly emphasizing discrete, reusable units of programming logic. By the implementation of OOP the researchers modified and validated the conventional metrics theoretically or empirically. Sizing and complexity metrics were the most impressive contributions for effort and cost estimation in project planning.

The OO approaches control complexity of a system by supporting hierarchical decomposition through both data and procedural abstraction [BOO91]. According to Brooks "The complexity of software is an essential property, not an accidental one"[BRO87]. The OO decomposition process helps to control the inherent complexity of the problem; it does not reduce or eliminate the complexity. Software complexity being one of the major contributing factors to the cost of developing and maintaining software [GRA92]. Software complexity measurement contributes in making the cost trade-offs in two ways. These are 1) To provide a quantitative method for predicting how difficult it will be to design, implement, and maintain the system. 2) To provide a basis for making the cost trade-offs necessary to reduce costs over the lifetime of the system.

Since 70's several approaches for predicting the software size were proposed. It was found that the complexity and size are strongly related to the effort value and also most of object-oriented metrics are based on this assumption. The object-oriented software metrics considers the measure items such as the number of lines in the code, the number of attributes [BRI95][CHI94] or the complexity of methods [CAB76][NEJ88]. Since these metrics are correlated to fault-proneness, they require an advanced state of development like the implementation stage.

2. OBJECT-ORIENTED METRICS
Chidamber and Kemerer [CHI94] proposed six Object-Oriented design measures which were considered as the foundation of Object-Oriented metrics. These metrics are: 1) Weighted Methods per Class (WMC): the weighted sum of all methods in a class. 2) Depth of Inheritance Tree (DIT): maximum length from the class to the root in the inheritance tree. 3) Number of Children (NOC): number of directly inherited classes. 4) Coupling Between Object classes (CBO): count of the number of other classes coupled to the considered class. 5) Response For a Class (RFC): number of methods that can be invoked by a message received by an object of the considered class. 6) Lack of Cohesion in Methods (LCOM): number of methods using the same set of attributes minus the number of methods using a different set of attributes.The object-oriented metrics can be classified into two categories: 1) Adaptation of classical sizing metrics and 2) Object-oriented sizing and complexity metrics.

2.1 Classical Sizing Metrics
Software size estimation model by Laranjeira [LAR90] provided a method that helped for sizing object-oriented systems based on successive estimations of refinements of the system objects. Confidence in size estimates increases as the system becomes more and more refined. He used various statistical techniques for determining the rate of convergence to the actual estimate.

Minkiewicz [MIN09] considered the value of various measures of size, lines of code and function points. The model [FRA06] estimated size, measured by function points [ALB83] directly from a conceptual model of the system being built. A model proposed by Tan et al. estimated lines of code based on the counts of entities, relationships, and attributes from the conceptual data model [TAN06]. A model related to the early information on use cases into a size estimate, measured in function points was given by Diev [DIE06].

Briand et al. [BRI01] empirically quoted the relationship between class size and the development effort by using regression techniques. Antoniol et al. [ANT99] followed adaptation of traditional function points, called "Object oriented Function Points" for enabling the measurement of object-oriented analysis and design specifications. . First the constructs were identified in object-oriented systems (e.g., classes and methods) for using them as parameters for OOFPs, and then a flexible model was built to estimate system size. Similarly, Costagliola et al. [COS05] presented their class point, a function points-like approach, for estimating the size of object-oriented products.

Two new metrics by Braz and Vergilio [BRA06] based on use case were introduced: 1) Use case size points for the internal structure to captures the functionality. 2) Fuzzy use case size points, on Fuzzy Set Theory for creation of gradual classifications dealing with uncertainty. Nesi and Querci also proposed a new complexity and size metrics for effort evaluation and prediction [NES98].

2.2 Object-Oriented Sizing Metrics

Object-oriented sizing metrics by Chidamber and Kemerer theoretically proposed six metrics [CHI94]. An empirical study on the metrics by Li and Henry was conducted using maintenance effort data, while Basili et al. [BAS96] validated the metrics using software defects. In 1994, Chidamber and Kemerer revised the original metrics of 1991 using measurement theory and empirical data [CHI91]. Chucher and Shepperd also commented on possible ambiguities in some of those metrics was reported in [CHU95].

3. CHIDAMBER AND KEMERER METRICS

Chidamber and Kemerer [CHI94] [CHI91] gave a new set of 6 proposed software metrics for object-oriented design. The metrics proposed are described as:

a) Weight methods per class (WMC) - The Weighted Methods per Class (WMC) metric is the sum of the complexity of methods and count of the combined complexity of methods in a given class. This assigns a complexity value of 1 to each method, and therefore the value of the WMC is equal to the number of methods in the class. The number of methods and the complexity of the methods is a predictor of how much time and effort is required to develop and maintain the class. The larger the number of methods in a class, the greater the potential impact on children. Churcher and Shepperd's (C&S) criticized on the ambiguity of WMC. They pointed out that there are two factors for C++ methods which C&K didn't specify; whether constructor/ destructor methods were all counted and whether operators are included as methods.

Li [LI98] emphasized that the metric can be used with two intentions: 1) count of methods, and 2) sum of the internal complexity of all methods, but the problem was that the number of methods and the internal structural complexity of methods are two independent attributes of a class and the dual interpretation of WMC metric might create a difficulties to the practitioner. Li [LI98] then proposed two new metrics: 1) Number of Local Methods (NLM) and 2) Class Method Complexity (CMC) to measure the two attributes that the WMC intends to capture.

b) Depth of Inheritance tree (DIT) - The Depth of Inheritance Tree (DIT) metric is "the maximum length from the node to the root of the tree". Li found there are two ambiguous points in this definition: 1) maximum length from node to root becomes unclear with multiple roots and 2) conflicting goals stated in the definition and theoretical basis for the DIT metric. It indicate that the DIT metric measure the number of ancestor class of a class, but the definition of DIT stated that it should measure the length of the path in the inheritance tree, which is the distance between two nodes in a graph. Li [LI98] proposed a new metric: Number of Ancestor Classes (NAC) as an alternative to DIT.

c) Response for class (RFC) - The response set of a class (RFC) is defined as set of methods that can be potentially executed in response to a message received by an object of that class. [JON07]. No ambiguity or inadequacy is reported for this metric. The instrumentation model for the RFC is the means to calculate the RFC metric stated in [CHI91].

d) Number of children (NOC) - According to Chidamber and Kemerer [CHI94] [CHI91], the Number of Children (NOC) metric is defined as the number of immediate sub-classes subordinated to a class in the class hierarchy. The theoretical points came out as NOC is a measure of how many subclasses are going to inherit the methods of the parent class. The viewpoints were 1) The greater the number of children, the greater the potential for reuse, since inheritance is a form of reuse. 2) The greater the number of children, the greater the likelihood of improper abstraction of the parent class. 3) The number of children gives an idea of the potential influence a class has on the design. Li [LI98] proposed a new metric: Number of Descendent Classes (NDC) as an alternative to the NOC metric to remedy the insufficiency of immediate sub-class counting in NOC.

e) Lack of cohesion of methods (LCOM) - This metric is a count of the number of disjoint method pairs minus the number of similar method pairs. The disjoint methods have no common instance variables, while the similar methods have at least one common instance variable [JON07][BAS96].

f) Coupling between objects (CBO) - The coupling Between Object Classes (CBO) metric is defined as "CBO for a class is a count of the number of non-inheritance related couples with classes". Li [LI98] claimed that the unit of "class" used in this metric is difficult to justify, and suggested different forms of class coupling: inheritance, abstract data type and message passing which are available in object-oriented programming. Li [LI98] proposed 2 new metrics: 1) Coupling Through Abstract Data Type (CTA) and 2) Coupling Through Message Passing (CTM) as an alternative metrics.

4. LI METRICS

Li discovered some metrics as he discovered problems with Chidamber and Kemerer metrics during the course of defining the unit definition model for the metrics. An alternative suite of object-oriented metrics was proposed by Li [LI98]. Six metrics, Number of Ancestor Classes (NAC), Number of Local Methods (NLM), Class Method Complexity (CMC), Number of Descendent Classes (NDC), Coupling Through Abstract Data Type (CTA), and Coupling Through Message Passing (CTM) were proposed in order to overcome some limitations found in Chidamber and Kemerer metrics.

a) Number of ancestor classes (NAC) - The Number of Ancestor classes (NAC) metric proposed as an alternative to the DIT metric measures the total number of ancestor classes from which a class inherits in the class inheritance hierarchy. The theoretical basis and viewpoints both are same as the DIT metric. In this the unit for the NAC metric is "class", Li [LI98] justified that because the attribute that the NAC metric captures is the number of other classes' environments from which the class inherits.

b) Number of local methods (NLM) - The Number of Local Methods metric (NLM) is defined as the number of the local methods defined in a class which are accessible outside the class. It measures the attributes of a class that WMC metric intends to capture. The theoretical basis and viewpoints are different from the WMC metric. The theoretical basis describes the attribute of a class that the NLM metric captures. This attribute is for the usage of the class in an object-oriented design because it indicates the size of a class's local interface through which other classes can use the class. Li [LI98] stated three viewpoints for NLM metric as following: 1) The NLM metric is directly linked to a programmer's effort when a class is reused in an OO design. More the local methods in a class, the more effort is required to comprehend the class behavior. 2) The larger the local interface of a class, the more effort is needed to design, implement, test, and maintain the class. 3) The larger the local interface of a class, the more influence the class has on its descendent classes.

c) Class method complexity (CMC) - The Class Method Complexity (CMC) metric is defined as the summation of the internal structural complexity of all local methods. The CMC metric's theoretical basis and viewpoints are significantly different from WMC metric. The NLM and CMC metrics are fundamentally different as they capture two independent attributes of a class. These two metrics affect the effort required to design, implement, test and maintain a class.

d) Number of descendent classes (NDC) - The Number of Descendent Classes (NDC) metric as an alternative to NOC is defined as the total number of descendent classes (subclass) of a class. The stated theoretical basis and viewpoints indicate that NOC metric measures the scope of influence of the class on its sub classes because of inheritance. Li claimed that the NDC metric captures the classes attribute better than NOC.

e) Coupling through abstract data type (CTA) - The Coupling through Abstract Data Type (CTA) is defined as the total number of classes that are used as abstract data types in the data-attribute declaration of a class. Two classes are coupled when one class uses the other class as an abstract data type [LI98]. According to Li [LI98] the theoretical view was that **t**he CTA metric relates to the notion of class coupling through the use of abstract data types. This metric gives the scope of how many other classes' services a class needs in order to provide its own service to others. The Viewpoints were: 1) More time is required by the software engineer to spend in understanding the interfaces of the used classes in order to create the design for a high CTA class than a low one. 2) For a test engineer, more effort is needed to design test cases and perform testing for high CTA class than a low one because the behaviors of the used classes also need to be tested. 3) For a maintenance engineer, it takes more time to understand a high CTA class than a low one because a high CTA class uses more class whose behaviors may compliance the class.

f) Coupling through message passing (CTM) - The Coupling through Message Passing (CTM) defined as the number of different messages sent out from a class to other classes excluding the messages sent to the objects created as local objects in the local methods of the class. Two classes can be coupled because one class sends a message to an object of another class, without involving the two classes through inheritance or abstract data type [LI98]. Theoretical view given was that the CTM metric relates to the notion of message passing in object-oriented programming. The metric gives an indication of how many methods of other classes are needed to fulfill the class' own functionality. The Viewpoints were 1) A class designer needs to spend more effort in understanding the services provided by other classes in a high CTM class than in a low CTM class because the outgoing message are directly related to the services other classes provide. 2) A test engineer needs to spend more effort and design more test cases for high CTM class than for a low CTM class because a high CTM value means more other classes' methods are involved in the logical paths of the class. 3) For a maintenance engineer, the higher the CTM metric value, the more specific methods in other classes the engineer needs to understand in order to diagnose and fix problems, or to perform other types of maintenance.

5. CONCLUSION

This paper compares the metrics related to object-oriented paradigm proposed by Chidamber and Kemerer and then refined by Li. Chidamber and Kemerer metrics were evaluated using a framework called as the metric evaluation framework by Kichenham and her colleagues. Li and other researchers also made an effort to validate the six metrics theoretically and empirically. A new suite of object-oriented programming metrics were proposed later. Some researchers also highlighted that if a metric is proposed for class method complexity based on the structure of the class that would be more practical. They also suggested that NLM should be further divided into two more comprehensive metrics 1) Number of private methods and 2) Number of public methods with appropriate weight allocation through empirical validation. They gave their view about the DIT or NAC metric that provide helpful information in complexity measure for the class design in object oriented paradigm. For NOC and NDC they argued that the more constructive works should focus on the type of inheritance which comes in two forms: data attributes and methods that are inheritable from the class by its subclass. This might increase the accuracy of complexity measure cause by the inheritance relations among the classes. In case of CBO, CTA and CTM they found that the definition and theoretical of the metric doesn't exclude the non-inheritance related couples. This might

create double counting when the class is having the different inheritance relations which are capturing the same attributes.

6. REFERENCES

[1] [ALB83] Albrecht A. and J. Gaffney, *"Software function, source lines of code and development effort prediction,"* IEEE Transactions on Software Engineering, Vol. 9, 1983, pp.639-648.

[2] [ANT99] Antoniol G., C. Lokan, G. Caldiera and R. Fiutem, *"A Function Point-Like Measure for Object-Oriented Software",* Empirical Software Engineering, Vol. 4, Issue 3, September 1999, pp. 263-287.

[3] [BRI01] Briand L. C. and J. Wust, *"Modeling Development Effort in Object-Oriented Systems Using Design Properties",* IEEE Transactions on Software Engineering, Vol. 27, No. 11, November 2001, pp. 963-986.

[4] [BRA06] Braz M. R. and S. R. Vergilio, *"Software Effort Estimation Based on Use Cases",* Proceedings of 30th Annual International Computer Software and Applications Conference (COMPASAC '06), IEEE Computer Society, September 2006, pp. 221-228.

[5] [BRI95] Brito F. e Abreu *"Toward the Design Quality Evaluation of Object-Oriented Software Systems"*. In Proceedings of the 5th International Conference on Software Quality, Austin, Texas, USA, Oct 1995.

[6] [BAS96] Basili V. L., L. Briand and W. L. Melo, *"A validation of object-oriented Metrics as Quality Indicators",* IEEE Transaction Software Engineering. Vol. 22, No. 10, 1996, pp. 751-761.

[7] [BOO91] Booch G., *"Object-Oriented Design with Applications"* (The Benjamin/Cummings Publishing Company,Redwood City, CA , 1991; ISBN: 0-8053-0091-0).

[8] [BRO87] Brooks F.P., No Silver Bullets: Essence and Accidents of Software Engineering, Computer, Vol. 20, No. 4 (Apr 1987) pp. 10-19.

[9] [CHU95] Chucher N.I. and M.J. Shepperd, *"Comments on a metrics Suite for Object-oriented Design"* IEEE Transaction on Software Engineering, Vol. 21, No.3, 1995, pp. 263-265.

[10] [COS05] Costagliola G., F. Ferrucci, G. Tortora, and G. Vitiello, "Class Point: An Approach for the Size Estimation of Object- Oriented Systems", IEEE Transaction on Software engineering, Vol. 31, No. 1, January 2005, pp. 52-74

[11] [CHI94] Chidamber S. R. and C. F. Kemerer, *"A Metrics Suite for Object Oriented Design",* IEEE Transactions on Software Engineering, Vol. 20, No. 6, June 1994, pp. 476-493.

[12] [CHI91] Chidamber S. R. and C. F. Kemerer, *"Towards a Metrics Suite for Object Oriented Design",* Proceeding on Object Oriented Programming Systems, Languages and Applications Conference (OOPSLA'91), ACM, Vol. 26, Issue 11, Nov 1991, pp. 197-211.

[13] [CAB76] McCabe T. *"A Complexity Measure"*. IEEE Transactions on Software Engineering, SE-2(4), Dec 1976, pp. 308-320.

[14] [DIE06] Diev S., *"Software estimation in the maintenance context,"* ACM Software Engineering Notes, Vol. 31, No. 2, 2006, pp. 1-8.

[15] [FRA06] Fraternali P., M. Tisi, and A. Bongio, *"Automating function point analysis with model driven development,"* Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research, Toronto, Canada, ACM Press, New York, 2006, pp. 1-12.

[16] [GRA92] Grady R.B., *"Practical Software Metrics for Project Management and Process Improvement"* (Prentice Hall, Englewood Cliffs, NJ, 1992; ISBN: 0-13-720384-5).

[17] [JON07] Jones C.,"*Estimating Software Costs: Bringing Realism to Estimating",* 2nd Edition, Mc Graw Hill, New York, 2007.

[18] [LAR90] Laranjeira L. A., *"Software Size Estimation of Object- Oriented Systems",* IEEE Transaction on Software Engineering, Vol. 16, No. 5, May 1990, pp. 510-522.

[19] [LI98] Li W., *"Another Metric Suite for Object-oriented Programming",* The Journal of System and Software, Vol. 44, Issue 2, December 1998, pp. 155-162.

[20] [MIN09] Minkiewicz A., *"The evolution of software size: A search for value,"* CROSSTALK, Vol. 22, No. 3, 2009 pp. 23-26.

[21] [NES98] Nesi P. and T. Querci, *"Effort Estimation and Prediction of Object-oriented Systems",* The Journal of Systems and Software, Vol. 42, Issue 1, July 1998, pp. 89-102.

[22] [NEJ88] Nejmeh B. A., "*A measure of execution path complexity and its applications".* Communications of the ACM, 31(2), Feb 1988, pp. 188-200.

[23] [TAN06] Tan H. B. K., Y. Zhao, and H. Zhang, *"Estimating LOC for information systems from their conceptual data models,"* Proceedings of the 28th International Conference on Software Engineering, Shanghai, China, ACM Press, New York, 2006, pp. 321-330.