Review of Complexity Metrics for Object Oriented Software Products

Tieng Wei Koh, Mohd Hasan Selamat, Abdul Azim Abdul Ghani, Rusli Abdullah
Faculty of Computer Science and Information Technology, University Putra Malaysia, 43400 UPM Serdang, Selangor Darul Ehsan, Malaysia.

Abstract

It is widely accepted that sizing or predicting the volumes of various kinds of software deliverable items is one of the first and most dominant aspects of software cost estimating. Most of the cost estimation model or techniques usually assume that software size or structural complexity is the integral factor that influences software development effort. Although sizing and complexity measure is a very critical due to the need of reliable size estimates in the utilization of existing software project cost estimation models and complex problem for software cost estimating, advances in sizing technology over the past 30 years have been impressive. This paper attempts to review the 12 object-oriented software metrics proposed in 90s' by Chidamber, Kemerer and Li.

1. Introduction

In 90s', several software industries have moved to the object-oriented paradigm (OOP) in the hope that with this new technology could increase their capability for programming in "large" through reusability function offered by OOP. This trend has created a new challenge especially to the management team as the conventional metrics invented for classical paradigm seemed not longer valid in supporting their project planning and resource allocation, where the OOP consider both attributes and operations to be equally important. In contrast, the classical techniques are either operation oriented or attribute.

Since the implementation of OOP, several researchers have made efforts to modify and validate the conventional metrics theoretically or empirically in order for object-oriented software production to fulfill its promise. Among the most impressive contributions are the sizing and complexity metrics which is use for effort and cost estimation in project planning.

This paper is organized as follows. Section 2 presents a very brief summary on the literature review have been done. Section 3 describes the 6 metrics proposed by Chidamber and Kemerer [11, 12]. Section 4 presents another new metric suite for object-oriented programming, also six in number proposed by Li [14] and short concluding remarks are presented in Section 5.

2. Literature review

Several approaches for predicting software size have been proposed in the literature since 70s'. Most of the researchers assume that complexity and size are strongly related to the effort value. In fact, most of object-oriented metrics are also based on this assumption.

In general, object-oriented metrics can be classified into two categories: 1) Adaptation of classical sizing metrics and 2) Object-oriented sizing and complexity metrics. We review these contributions found in the literature in the following sub-section.

2.1 Adaptation of classical sizing metrics

Laranjeira [5] proposed a software size estimation model and claimed that the model has the potential for providing more accurate size estimates than existing methods which are not yet reliable enough to be consistently used with existing cost estimation models. This method – functional model takes advantage of a characteristic of object-oriented systems, the natural correspondence between specification and implementation, in order to enable users to come up with better size estimates at early stages of the software development cycle. We notice that there is no obvious comparison between this model and the existing methods.

Another similar finding was reported by Condori-Fernandez et al. [8] described a measurement protocol to map the concepts used in the Object-Oriented Method Requirement Model onto the concepts used by the COSMIC Full Function Points (COSMIC-FFP). This protocol describes a set of measurement operations for modeling and sizing object-oriented software systems

Manuscript received November 5, 2008
Manuscript revised November 20, 2008

from requirement specifications obtained in the context of the object-oriented method. It is an extended set rules that allows estimating the functional size of object-oriented systems at an early of the development lifecycle.

Briand et al. [6] empirically investigated relationship between class size and the development effort using regression techniques. Antoniol et al. [1] defined an adaptation of traditional function points, called “Object-oriented Function Points” to enable the measurement of object-oriented analysis and design specifications. Similarly, Costagliola et al. [2] presented their class point, a function points-like approach, which was conceived to estimate the size of object-oriented products.

On the other hand, Braz and Vergilio [7] introduced two new metrics based on use case: 1) Use case size points, which if considers the internal structure if the use case and better captures its functionality. 2) Fuzzy use case size points, considers concepts of Fuzzy Set Theory to create gradual classifications that better deal with uncertainty.

Nesi and Querci proposed a new complexity and size metrics for effort evaluation and prediction are presented in [10]. These metrics were compared with respect to the most important classical metrics in their literature. They were also reported the validation of those metrics. While Sherif and Sanderson [4] reported the implementation of object-oriented metrics on two software projects developed at the Jet Propulsion Laboratory (JPL).

2.2 Object-oriented sizing metrics

The most impressive finding related to object-oriented sizing metrics was the one proposed by Chidamber and Kemerer. Since the proposal of the six metrics [11] theoretically, other researchers have made effort to validate the metrics empirically. Li and Henry [15] conducted an empirical study on the metrics using maintenance effort data, while Basili et al. [13] validated the metrics using software defects collected from student projects. In 1994, Chidamber and Kemerer revised the original metrics which was proposed in 1991 using measurement theory and empirical data [12]. Chucher and Shepperd were having some comments on possible ambiguities in some of those metrics was reported in [9]. Evaluation on those metrics based on Kitchenham’s metric-evaluation framework has been reported and another 6 new object-oriented metrics were reported by Li [14].

3. Chidamber and Kemerer metrics

New set of software metrics for object-oriented design has developed and implemented by Chidamber and Kemerer [11, 12]. These metrics were based in measurement theory and also reflect the viewpoints of experienced object-oriented developers. In particular, this set of six proposed metrics is presented as a first attempt at development of formal metrics for object-oriented design.

The authors claimed that this proposal should lay the groundwork for a formal language with which to describe metrics for object-oriented design. The following sub-section described the metrics proposed.

3.1 Weight methods per class (WMC)

The Weighted Methods per Class (WMC) metric is defined as the sum of the complexity of a class’ local methods and intended to count the combined complexity of local methods in a given class. According to Jones [3], the weight portion of this metric is still under examination and is being actively researched. Li [14] claimed that the metric carrying two different units respectively and can be used with two intentions: 1) count of local methods, and 2) sum of the internal complexity of all local methods, but the problem is the number of local methods and the internal structural complexity of local methods are two independent attributes of a class and the dual interpretation of WMC metric might create a difficulties to the practitioner. Li [14] 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. These two metrics are present in Section 4.

3.2 Depth of Inheritance tree (DIT)

According to Li [14], the Depth of Inheritance Tree (DIT) metric is defined as “Depth of inheritance of the class is the DIT metric for the class. In cases involving multiple inheritances, the DIT will be 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 when inheritance tree with multiple roots and 2) conflicting goals stated in the definition, the theoretical basis, and the viewpoints for the DIT metric where theoretical basis and viewpoints 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 [14] proposed a new metric: Number of Ancestor Classes (NAC) as an alternative to DIT

3.3 Response for class (RFC)

The Response for Class (RFC) is the number of methods that can execute in response to a message sent to an object within this class, using to one level of nesting [3].

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 [12].

3.4 Number of children (NOC)

According to Chidamber and Kemerer [11, 12], the Number of Children (NOC) metric is defined as the number of immediate sub-classes subordinated to a class in the class hierarchy. Li [14] 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.

3.5 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 [3, 13]. The different definitions of the Lack of Cohesion of Methods (LCOM) metric in [11, 13] were noticed and discussed [14].

3.6 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 [14] 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 [14] proposed 2 new metrics: 1) Coupling Through Abstract Data Type (CTA) and 2) Coupling Through Message Passing (CTM) as an alternative metrics are presents in section 4.

4. Li metrics

The problems associated with some of the Chidamber and Kemerer metrics were discovered during the course of defining the unit definition model for the metrics. An alternative suite of object-oriented metrics that does not have the problem is proposed [14]. According to the author, the attribute is related to a specific concept in object-oriented programming and the following section presents six metrics proposed in order to overcome some limitation found in Chidamber and Kemerer metrics.

4.1 Number of ancestor classes (NAC)

The Number of Ancestor classes (NAC) metric was proposed, as an alternative to the DIT metric, to measure this attribute of a class. Li define the NAC as 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. The unit for the NAC metric is “class”, Li [14] justified that because the attribute that the NAC metric captures is the number of other classes’ environments from which the class inherits. This unit is defined with reference to a standard which is class inheritance relation in object-oriented programming.

4.2 Number of descendent classes (NDC)

The Number of Descendent Classes (NDC) metric is proposed as an alternative to the NOC metric. It defined as the total number of descendent classes (subclass) of a class. The theoretical basis and viewpoints remain the same as NOC. Li [14] reported that the attribute of a class that the NOC metric captures is the number of classes that may potentially be influenced by the class because of inheritance relations. Li claimed that the NDC metric captures the classes attribute better than NOC.

4.3 Number of local methods (NLM)

This is one of the metric proposed in Li [14] in order to measure the attributes of a class that WMC metric intends to capture. 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. 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 is the local interface of a class. This attribute is important 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 [14] stated three viewpoints for NLM metric as following:

1. The NLM metric is directly linked to a programmer’s comprehension effort when a class is reused in an OO design. The more local methods a class has, 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.

4.4 Class method complexity (CMC)

The Class Method Complexity (CMC) metric is defined as the summation of the internal structural complexity of all local methods. Regardless whether they are visible outside the class or not. This definition is essentially the same as the first definition of the WMC metric in [12]. However, the CMC metric's theoretical basis and viewpoints are significantly different from WMC metric.

The NLM and CMC metrics are fundamentally different because they capture two independent attributes of a class. However, there is some commonality in the viewpoints of the two metrics – they affect the effort required to design, implement, test and maintain a class.

4.5 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 [14]. Consider the example in Fig. 1. Class B is coupled with class A through the use of abstract data type because class B use class A in its data-attribute declaration.

```
class A {                     class B {

        int a;                        A anA;

        public:                       public:

        void A ( );                   void B ( );
    };                            };
```

Fig. 1. An example of coupling through abstract data Type

Theoretical basis according to Li [14]:
The 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.

Viewpoints according to Li [14]:
1. A software engineer needs to spend more time 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 that 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.

4.6 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 [14]. Consider the example in Fig. 2. Both classes A and B are in the same object-oriented design, and they are not related through inheritance or abstract data type as a class attribute. However, class A is coupled with class B because of A's methods sends a message to B's object.

```
class A {                     class B {

        int a;                        A anA;

        public:                       public:

        void A ( );                   void B ( );
        void A1 (B*b)
        {b->B1( );};                  void B1 ( );
    };                            };
```

Fig. 2. An example of coupling through message passing

Theoretical basis according to Li [14]:
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.

Viewpoints according to Li [14]:
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 calss 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. Discussion and concluding remark

This paper overview and compare the metrics related to object-oriented paradigm which was proposed by Chidamber and Kemerer and refine by Li. Chidamber and Kemerer were evaluated using the metric evaluation framework proposed by Kichenham and her colleagues [11]. Li and other researchers have made effort to validate the six metrics theoretically and empirically. A new suite of object-oriented metrics that does not have the noted deficiencies was proposed later.

The 6 metrics by Li is intended to complement and strengthen the 4 metrics proposed by Chidamber and Kemerer. However, we found out there is some shortfalls for the metrics that have been proposed. The studies of the metrics are discussed in the following sub section.

5.1 WMC Vs CMC and NLM

WMC proposed by Chidamber and Kemerer leave the impression of measuring two independent attributes of a class at the same time: 1) the count of local methods and 2) the sum of internal complexity of all local methods. This issue was bring out by Li and proposed the CMC and NLM metric.

The CMC metric intends to measure the internal structural complexity of a class via capturing the complexity of information hiding in the local methods. The theoretical basis and viewpoints stated by Li sounds logic. However, the unit definition model is defined by reference to conversion from another unit. This metric proposal doesn't sound convincing as the conversion rules among the complexity metrics have not well defined up to circa 2008. The measures of structural complexity mentioned in [14] are proposed for classical structural paradigm, those metrics are not theoretically nor empirically validated.

We suggested that if there is a metric proposed for class method complexity based on the structure of the class would be more practical.

The NLM, a metric that only consider local methods defined in a class, which is accessible to the other classes. The private methods in a class do not included in this metric, although it shows the properties of object and indicates the size of a class.

Due to the metric definition proposed by Li is not completely comprehensive comply to the theoretical basis, we 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.

5.3 DIT Vs NAC

The attribute of a class that DIT metric intends to capture is the number of classes that have potential influence on the class because of the inheritance relations. However, there are some ambiguous factors lie in 1) When multiple inheritance and multiple roots, classes that do not inherit from any other classes, are present at the same time, and 2) Conflicting goals stated in the definition, the theoretical basis, and the viewpoints.

NAC, this metric was proposed to overcome the ambiguities lie in DIT metric. Li [14] claimed that number of ancestor classes of a class could be the best metric to capture the class inheritance relations, regardless of the number of roots or whether multiple inheritances is present compare to the metric DIT, which is a measure of depth of inheritance of the class.

We notice that DIT and NAC both serve the same objective – measure the inheritance relations among the classes as both metrics having the same theoretical basis and viewpoints. Proposed of NAC is not necessary, since we can revise the definition of DIT in order to solve the ambiguities. On the other hand, we believe that detailed out the DIT or NAC metric would provide helpful information in complexity measure for the class design in object-oriented paradigm.

5.4 NOC Vs NDC

NOC is defined as the number of immediate subclass subordinated to a class in the class hierarchy. However, Li [14] claimed that the metrics should measures the number of immediate and non-immediate subclasses as a class has influence over all its subclasses.

The NDC metric refined the shortfall of NOC by considering both immediate and non-immediate subclasses. We agreed with this point. However, we argue that by revise the definition of NOC would be good enough.

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. If we detailed out these two attributes for the metric, it might increase the accuracy of complexity measure cause by the inheritance relations among the classes.

5.5 CBO Vs CTA and CTM

CBO measures the number of non-inheritance related couples with other classes in general. Li[14] claimed that the unit measure -class, is hard to justify the measure coupling. Li proposed 2 new metrics based on different form of class coupling unit definition model: abstract data type and message passing

CTA defines as the total number of classes that are used as abstract data types in the data attribute declaration of a class. Li [14] proposed this metric by breaking down the type of class coupled: inheritance, abstract type and message passing.

We found out 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.

CTM measures 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.

The issue for this metric is similar to the CTA. In addition, we notice that the theoretical basis for this metric indicate that the purpose of this metric is to measure the services of other classes are needed to fulfill the class' own functionality. We believe that by classifying or measuring the services provided for one classes might be more practical than CTM.

References

[1] G. Antoniol, 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.

[2] G. Costagliola, 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

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

[4] J. S. Sherif and P. Sanderson, "Metrics for Object-oriented Software Projects", The Journal of System and Software 44, 1998, pp. 147-154.

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

[6] L. C. Briand 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.

[7] M. R. Braz 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.

[8] N. Condori-Fernandez, S. Abrahao, and O. Pastor, "Towards a Functional Size Measure for Object-Oriented Systems from Requirements Specifications", Proceedings of the Fourth International Conference on Quality Software (QSIC '04), IEEE Computer Society, September 2004, pp. 94-101

[9] N.I. Chucher 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] P. Nesi 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.

[11] S. R. Chidamber 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] S. R. Chidamber 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] V. L. Basili, 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.

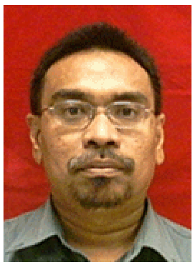
[14] W. Li, "Another Metric Suite for Object-oriented Programming", The Journal of System and Software, Vol. 44, Issue 2, December 1998, pp. 155-162.

[15] W. Li and S. Henry, "Object-oriented Metrics which Predict Maintainability", The Journal of Systems and Software, Vol. 23, Issue 2, November 1993, pp. 111-122.

Koh Tieng Wei received the first degree in computer science in 2003 and the M.S. degree in software engineering in 2006 from University Putra Malaysia. Currently, he is pursuing a PhD degree with his research work related to object-oriented software sizing measure.

Mohd Hasan bin Selamat received his M. Sc in Computer Science from Essex University in 1981 and M. Phil in Information System form East Anglia University in 1989. His research interest includes component-based software development, knowledge management, software costing, and strategic information system planning. He is now an Associate Professor in The Department of Information System, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia. He has published a number of papers related to this area.

Abdul Azim bin Abdul Ghani is an Associate Professor cum the Dean of the Faculty of Computer Science and Information Technology, University Putra Malaysia. He obtained his PhD in computer science from University of Strathclyde, Scotland. His research interest is software engineering and software measurement.



Rusli bin Haji Abdullah is a senior lecturer in Information System Department, Faculty of Computer Science and Information Technology of University Putra Malaysia. He holds a B.Sc in Computer Science from University Putra Malaysia (1988), M.Sc in Computer Science from University Putra Malaysia (1996), and PhD in Knowledge Management field at Faculty of Computer Science and Information System at University Tecknologi Malaysia (2005). He has more 12 years of teaching experience and with about 8 years of system development experience as a system analyst at higher learning institution.