December 2005

A Critical Suggestive Evaluation of CK Metric

Parvinder Sandhu
Guru Nanak Dev Engineering College

Hardeep Singh
Guru Nanak Dev University

A Critical Suggestive Evaluation of CK Metric

Parvinder Singh Sandhu
Assistant Professor
Guru Nanak Dev Engineering College
Ludhiana(Punjab) - 141 006 India.
parvindersingh@gndec.ac.in

Dr. Hardeep Singh
Professor& Head(Computer Science)
Guru Nanak Dev University
Amritsar (Punjab) - 143 005 India
hardeep_gndu@rediffmail.com

Abstract

In the era of Computerization Object Oriented Paradigm is becoming more and more pronounced. This has provoked the need of high quality object oriented software, as the traditional metrics cannot be applied on the object-oriented systems. Although CK suit of metric is the widely accepted metrics but when analyzed, according to their validation criteria on which these are based, these metrics can't satisfy certain axioms. This paper gives the evaluation of CK suit of metrics and suggests the refinements and extensions to these metrics so that these metrics should reflect accurate and precise results for OO based systems.

Keywords: CK Metric, Weyukar's axioms, CBO, LCOM, DIT, NOC.

1. Introduction

The OO paradigm for the software development differs from traditional procedural counterpart so the traditional metrics can't be applied on OO software. Opponents of the use of traditional metrics within the OO paradigm argue that such metrics were originally designed to go along procedural methodologies and languages, and therefore fail to capture concepts as inheritance and polymorphism which are unique to the OO paradigm (Lorenz et. al. 1994; Li et. al. 1995). Moreover Traditional paradigm requires more effort during the coding and maintenance phases as compared to OO paradigm, which put more emphasis on the earlier stages of software development, especially on design phase (Henderson et. al. 1991). Krishnan et al. empirically show that higher up-front investment in design helps in controlling costs as well as in improving quality (Krishnan et. al. 1994). Software metrics are measures of the attributes of software products and processes. Because of the importance of knowing quality of the design phase there is the need of metrics that measures the goodness of design and it provides the designer with improved insight that leads to a higher level of quality. Chidamber and Kemerer metric & MOOD metric are considered the best OO design metrics. But The Chidamber and Kemerer (CK) metrics suite is the most criticized, perhaps due to its popularity.

Even today, there is lack of availability of metrics that measures all aspects of OO systems. Hence additional metrics are required to measure currently unexplored or partially explored dimensions of OO systems. Research on metrics for OO software development is limited and empirical evidence linking the OO methodology and project outcomes is scarce. Recent work

in the field has also addressed the need for research to better understand the determinants of software quality (Basili et. al. 1996).

The organization of the rest of the paper is as follows: In the next section, we discuss Weyukar's nine axioms and CK Suit of Metrics with the flaws and the inconsistencies in the suit that arose when validated against Weyukar's nine axioms. Section2 we also discussed about the refinements to the discrepancies in CK suit of metrics .In the In Section 3, we present the conclusion and finally the reference.

2. CK Suit of Metrics

One of the first suites of OO design measures was proposed by Chidamber and Kemerer (Chidamber et. al. 1991; Chidamber et. al. 1994). The authors of this suite of metrics claim that these measures can aid users in understanding object oriented design complexity and in predicting external software qualities such as software defects, testing, and maintenance effort. Use of the CK set of metrics and other complementary measures are gradually growing in industry acceptance. This is reflected in the increasing number of industrial software tools, such as Rational Rose®, that enable automated computation of these metrics. Even though this metric suite is widely, empirical validations of these metrics in real world software development settings are limited. Various flaws and inconsistencies have been observed in the suite of six class-based metrics as shown under.

2.1 Validation Criteria for CK Metric Suite

Weyuker establishes a standard for software measures in this seminal article. She states the conditions for a measure as follows:
"All the measures we consider depend only on the syntactic features of the program."
(Weyuker 1988)
The nine properties of measures are (Chidamber et. al. 1994):

Property 1: Non-coarseness

Given a class P and a metric μ another class Q can always be found such that: $\mu(P) \neq \mu(Q)$. This implies that not every class can have the same value for a metric; otherwise it has lost its significance as a measurement.

Property 2: Granularity

According to this property there could be a finite number of cases having the same metric value. Since the universe of discourse deals with at most a finite set of applications, each of which has a finite number of classes, this property will be met by any metric measured at the class level.

Property 3: Design details are important

Given two class designs, P and Q, which provide the same functionality, does not imply that $\mu(P) = \mu(Q)$. The specifics of the class must influence the metric value. The intuition behind Property 3 is that even though two class designs perform the same function, the details of the design matter in determining the metric value for the class.

Property 4: Monotonicity
For all classes P and Q, the following must hold: $\mu(P) \leq \mu(P+Q)$ and $\mu(Q) \leq \mu(P+Q)$ where P + Q implies combination of P and Q. This implies that the metric value for the combination of two classes can never be less than the metric for either of the component classes.

Property 5: Non-equivalence of interaction
$\exists$ P, $\exists$ Q, $\exists$ R, such that $\mu(P) = \mu(Q)$ does not imply that $\mu(P+R) = \mu(Q+R)$.
This suggests that interaction between P and R can be different than interaction between Q and R resulting in different complexity values for P+R and Q+R.

Property 6: Non-uniqueness (notion of equivalence)
There can exist distinct classes P and Q such that $\mu(P) = \mu(Q)$. This implies that two classes can have the same metric value, i.e., the two classes can be equally complex.

Property7: Permutation of elements
Permutation of elements within the item being measured can change the metric value. The intent is to ensure that metric values change due to permutation of program statements.

Property 8: Renaming
It requires that when the name of the measured entity changes, the metric should remain unchanged.

Property 9: Interaction increases complexity
$\exists$ P and $\exists$ Q such that: $\mu(P) + \mu(Q) < \mu(P+Q)$
The principle behind this property is that when two classes are combined, the interaction between classes can increase the complexity metric value.

2.2 Inconsistencies in CK metric suit and their possible solutions

The CK metrics were validated against Weyukar's nine axioms and none of the metrics was found to comply to either of the property 7 about "Permutation of elements" or property 9 about "Interaction increases complexity" of the above Weyuker list of axioms. In support of non-compliance to property 7 Chidamber & Kermer suggested that "Permutation of elements" property is meaningful in traditional program design, where the ordering of if-then-else blocks could alter the program logic (and consequent complexity). In OOD, a class is an abstraction of the problem space, and the order of statements within the class definition has no impact on eventual execution or use. For example, changing the order in which methods are declared does not affect the order in which they are executed, since methods are triggered by the receipt of different messages from other objects (Chidamber et. al. 1994). Failure to meet the property 9 suggest that it is probably not applicable to object –oriented systems where interaction might in fact decrease complexity by rendering classes closer to the abstractions they are supposed to portray. There are six metrics proposed:

2.2.1 Metric 1: Weighted methods per class (WMC)

According to this metric if a Class C, has n methods and c_1, c_2 …c_n be the complexity of the methods, then WMC(C)= $c_1 + c_2 + \ldots + c_n$. The specific complexity metric that is chosen should be normalized so that nominal complexity for a method takes on a value of 1.0. If all method complexities are considered to be unity, then WMC = n, the number of methods. WMC break an elementary rule of measurement theory that a measure should be concerned with a single attribute (Chidamber et. al. 1994). This is also not clear whether the inherited method is to be counted in base class (which defines it), in derived classes or in both.

2.2.1 Metric 2: Depth of inheritance tree (DIT)

According to this metric Depth of inheritance of a class is "the maximum length from the node to the root of the tree"[Pressman 2001]. The definition of DIT is ambiguous when multiple inheritance and multiple roots are present. Consider the class inheritance tree with multiple roots in figure 1.

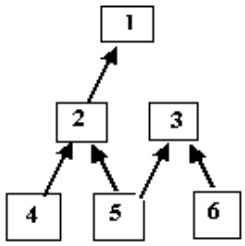

Figure 1: A Class Inheritance Tree with Multiple Root

In figure 1 the maximum length from node 5 becomes unclear, as there are two roots in the design. As the maximum length of class 5 from root 3 is 1 and maximum length of class 5 from root 1 is 2 (Sheldon et. al. 2002).

There is the inconsistency in the theoretical basis and definition of the metric in case of multiple inheritance. The theoretical basis states that DIT of a class is a measure of how many ancestor classes can potentially affect that class. The definition measures the maximum ancestor classes from the class-node to the root of the inheritance tree. The above conflict is shown in figure 2, where DIT(4)=DIT(5)=2 (Sheldon et. al. 2002).

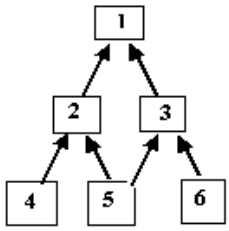

Figure 2: A Class inheritance Tree with Single Root

As the class 5 inheritance more classes than class 4, so the DIT value of class 5 should be greater than that of class 4 but according to present definition of DIT metric the value of DIT for both classes is same. Hence there is the need to rectify the definition of DIT. The alternative length of the path is not being considered in case of multiple inheritance. If we add all the ancestor classes coming in common path to the ancestor classes coming in alternative paths then that will be the true representation of the theoretical basis of the DIT metric.

As $DIT_{path1}(5)$ is the value of the maximum length of path of class 5 to root using path 5-3-1 , which comes out to be 2. Other alternative path of class 5 to the root is 5-2-1; the value of DIT in that case can be depicted by $DIT_{path2}(5)= 2$. So the total DIT value for class 5 can be calculated as:

$$DIT_{total}(5) = DIT_{path1}(5) + DIT_{path2}(5)= 2 + 2 = 4,$$

which is the true representation of the theoretical basis DIT metric. On the other hand in case of class 4 there is only one path to the root hence the $DIT_{total}(4)=2$. DIT metric states that as Depth of Inheritance grows, it is likely that the lower-level classes will inherit many methods, that leads to greater design complexity and potential difficulties when attempting to predict the behavior of a class. Ultimately deeper Inheritance produces hindrances in maintenance. On the other hand it states that it is better to have Depth than breadth in the Inheritance Hierarchy Hence there is contradiction in the statements of DIT metric.

2.2.3 Metric 3: Number of Children (NOC)

According to this metric Number of children (NOC) of a class is the number of immediate sub-classes subordinated to a class in the class hierarchy. Theoretical basis of NOC metric relates to the notion of scope of properties. It is a measure of how many sub-classes are going to inherit the methods of the parent class (Chidamber et. al. 1994). The definition of NOC metric gives the distorted view of the system as it counts only the immediate sub-classes instead of all the descendants of the class as illustrated by the figure 3:

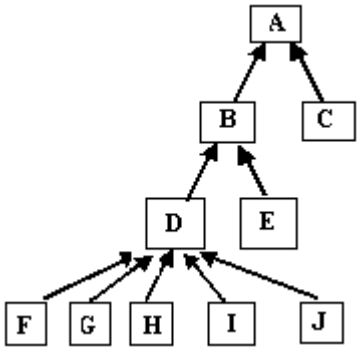

Figure 3: Example showing the distorted view of NOC metric

where Both A and B classes have NOC value of two, but there are nine classes that inherits the properties of class A and a total of seven classes inherit class B's properties. So the NOC value of a class should reflect all the subclasses that share the properties of that class.

$$\mathbf{NOC(Class) = Number\ of\ Immediate\ subclasses} + \sum_{i}^{\text{All Subclasses}} \mathbf{NOC(i)}$$

2.2.4 Metric 4: Coupling Between Object Classes (CBO)

According to this metric "Coupling Between Object Classes" (CBO) for a class is a count of the number of other classes to which it is coupled. Theoretical basis of CBO relates to the notion that an object is coupled to another object if one of them acts on the other, i .e. methods of one use methods or instance variables of another (Chidamber et. al. 1994). As Coupling between Object classes increases, reusability decreases and it becomes harder to modify and test the software system. But for most authors coupling is reuse, which raises ambiguity. So there is the need to find out the coupling level that implies the goodness of design.

Chidamber and Kermerer state that their definition of coupling also applies to coupling due to inheritance, but do not make it clear if all ancestors are involuntarily coupled or if the measured class has to explicitly access a field or method in an ancestor class for it to count. In general the definition of the CBO is ambiguous, and makes its application tough (Mayer et. al. 1999).

2.2.5 Metric 5: Response For a Class (RFC)

According to this metric Response For a Class (RFC) can be defined as | RS |, where RS is the response set for the class. The response set for the class can be expressed as:

$$RS = \{ M \} \cup \text{all i } \{ R_i \}$$

where $\{ R_i \}$ = set of methods called by method *i* and $\{ M \}$ = set of all methods in the class. The response set of a class is a set of methods that can potentially be executed in response to a message received by an object of that class. (Henderson et. al. 1991) But here the point to be noted is that because of practical considerations, Chidamber and Kermerer recommended only one level of nesting during the collection of data for calculating RFC. This gives incomplete and ambiguous approach as in real programming practice there exists "Deeply nested call-backs" that are not considered here. If CK intend the metric to be a measure of the methods in a class plus the methods called then the definition should be redefined to reflect this (Mayer et. al. 1999).

As the cardinality of response set is a measure of RFC for that class, hence RFC measures both internal and external communication by counting the number of methods, internal as well as external, available to a class. It is able to discriminate between two messages sent to the same method but from different parts of the class (Henderson et. al. 1996).

2.2.6 Metric 6: Lack of Cohesion in Methods (LCOM)

Consider a Class C_1 with *n* methods M_1 , M_2 ..., M_n . Let $\{I_j\}$ = set of instance variables used by method M_i .There are *n* such sets $\{I_1\},\{I_2\}... \{I_n\}$. Let $P = \{ (I_i ,I_j) \mid I_i \cap I_j = \varnothing \}$ and $Q = \{ (I_i ,I_j) \mid I_i \cap I_j \neq \varnothing \}$. If all n sets $\{I_1\},\{I_2\}... \{I_n\}$. are $\varnothing$ then let $P = \varnothing$ [4]. Lack of Cohesion in Methods (LCOM) of a class can be defined as:

$$LCOM = |P| - |Q|, \text{ if } |P| > |Q|$$
$$LCOM = 0 \text{ otherwise}$$

The high value of LCOM indicates that the methods in the class are not really related to each other and vice versa. According to above definition of LCOM the high value of LCOM implies low similarity and low cohesion, but a value of LCOM = 0 doesn't implies the reverse (Mayer et. al. 1999).

Consider the example in figure 4 (a) the value of LCOM is 8 (as | P | =9 and | Q | = 1). Whereas in figure 4 (b) the value of LCOM is also 8 (as | P | =18 and | Q | = 10), but figure 4 (a) example is more cohesive than figure 4 (b) example. So the above said definition of CK metric for LCOM is not able to distinguish the more cohesive class from the less ones. This is simple violation of the basic axiom of measurement theory, which tells that a measure should be able to distinguish two dissimilar entities. So this deficiency offends the purpose of metric.

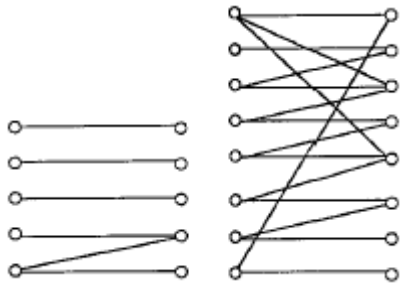

Figure 4: Two examples of (a) Less cohesive class and (b) Densely cohesive Class

In another test of validity of the LCOM metric consider the example of figure 5:

I1 = {a, b, c, d, e}
I2 = {a, b, e}
I3 = {x, y, z, d, e}
I4 = {x, y, z, d}

Figure 5: Example for the calculation of LCOM

Consider a class supporting the first three sets then |P| = 2, |Q| = 1⇒ LCOM = 1 implies less cohesion but when considering a class that supports all four sets then |P| = 3, |Q| = 3 ⇒ LCOM = 0 implies high cohesion. But this is just the reverse that we are expecting when we analyze the above sets as I1 and I2 are a pair of cohesive methods as are I3 and I4 and the good design recommends the formation of two classes, not one (Henderson et. al. 1996).

Table 1: Evaluation of the Chidamber and Kemerer metrics against Weyukar's nine axioms (Chidamber et. al. 1991)

CK Metrics	Weyukar's nine axioms								
	1	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***
WMC	Y	Y	Y	Y	Y	Y	N	Y	N
DIT	Y	Y	Y	Y	N	Y	N	Y	N
NOC	Y	Y	Y	Y	Y	Y	N	Y	N
RFC	Y	Y	Y	Y	Y	N	N	Y	N
LCOM	Y	Y	Y	Y	Y	Y	N	Y	N
CBO	Y	Y	Y	Y	Y	Y	N	Y	N

2.3 A New Measure for LCOM

For solving the above problems this paper recommends the two alternatives, one version is the amendments in the basic CK metric of LCOM, denoted by $LCOM^{new1}$ and other version is

the new formula for the LCOM which considers the number of data members of the classes, number of classes and the number of data members, denoted by $LCOM^{new2}$.

$$LCOM^{new1} = \frac{1}{m}\sum_{m}^{i=1} \frac{1}{\eta(M_i)} - \frac{1}{a} \qquad \text{.....(1)}$$

where "η(Mi)" is the number of data variables that i^{th} method is using, "m" are the total number of methods and "a" is total number of data variables used by the methods.

According to Second version is the modified form of existing CK metric for LCOM as that metric is not considering the relative importance of the elements of set P and set Q used in calculating the LCOM of a class. First component of this version is representing the normalized weightage of the "$\{ (I_i, I_j) \mid I_i \cap I_j \}$" constituents of the methods considered. The second component represents the normalized weightage of the "$\{ (I_i, I_j) \mid I_i \cap I_j \neq \varnothing \}$" constituents of the methods in the class. As the $| I_i \cap I_j |$ is the number of data variables shared by the two methods ,on the other hand, $| I_i \cup I_j |$ is the union of variables of both the methods. The basis of $LCOM^{new2}$ is same as that of the LCOM metric.

$$LCOM^{new2} = \frac{1}{ma}\left\{ \sum_{\substack{\text{for each element} \\ (I_i, I_j) \\ \text{of P set}}} \frac{|I_i \cup I_j|}{a} - \sum_{\substack{\text{for each element} \\ (I_i, I_j) \\ \text{of Q set}}} \frac{|I_i \cap I_j|}{|I_i \cup I_j|} \right\}$$

.....(2)

To verifying the correctness of the $LCOM^{new1}$and $LCOM^{new2}$ lets consider the six graded examples of figure 6. Examples (A) and (B) of Figure 6, should clearly be divided into two as they have very less cohesiveness. At the other extreme, example (F) is maximally cohesive in which all methods access all attributes. The intermediate parts (C–E) show various degrees of cohesion. The values of the two proposed versions of LCOM are given in Table 2.

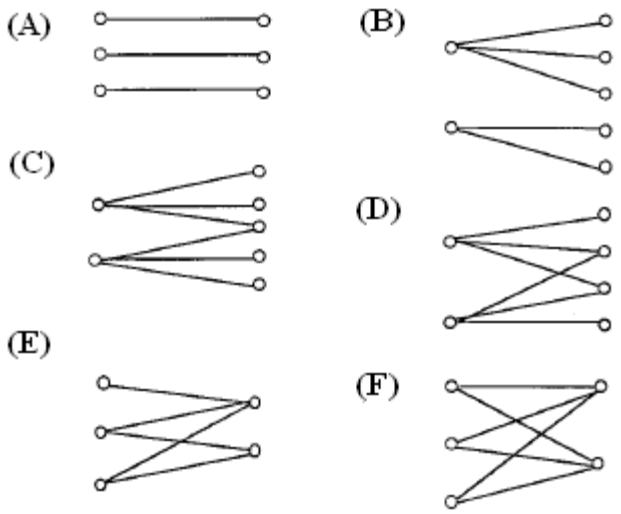

Figure 6: Six graded examples (the values of the three versions of LCOM are given in Table 2). Parts (A) and (B) should clearly be divided into two. At the other extreme, part (F) is maximally cohesive in which all methods access all attributes. The

Table 2: Values of two proposed versions of LCOM from Equations 1 & Equation 2 for the examples of Figure 6

Version	(A)	(B)	(C)	(D)	(E)	(F)
$LCOM^{new1}$	0.67	0.21	0.133	0.083	0.167	0.00
$LCOM^{new2}$	0.22	0.1	-0.013	-0.0625	-0.33	-0.5

By analyzing the table 2, is clear that the $LCOM^{new1}$ gives the least value zero for the maximum cohesive example (F) and for other examples the values of $LCOM^{new1}$ is in accordance with the cohesiveness level. The values calculated using $LCOM^{new2}$ are also verified as the value decreases from (A) to (F) as the cohesiveness increases from (A) to (F).

When $LCOM^{new1}$ and $LCOM^{new2}$ are applied to the figure 4 examples then $LCOM^{new1}$ =0.7, $LCOM^{new2}$= 0.124 for figure 4(a) example and $LCOM^{new1}$= 0.417, $LCOM^{new2}$= 0.078 for the example of figure 4(b).

When we consider 3 sets of example of figure 5 the $LCOM^{new1}$ and $LCOM^{new2}$ values are 0.164 and 0.047 respectively, but when we consider 4 sets then the $LCOM^{new1}$ and $LCOM^{new2}$ values are 0.16 and 0.059 respectively.

3. Conclusion

As the two proposed versions are able to alleviate the problems that are there with CK metric of LCOM. CK metric of LCOM measure is not very discriminating (i.e. it again fails Weyuker's basic first axiom (Weyuker 1988)) for low cohesive structures as reflected in the example of figure 5, where it has produced extreme contrast values. But second proposed version is able to solve the problem by indicating high value of $LCOM^{new2}$, when we consider the four sets as compared to the value when three sets are considered. But the second version is able to solve the problem completely by producing the values 0.059, when we consider the four sets and 0.047, when we consider three sets of example of figure 5, showing the superiority over the first version. As clear from the values of $LCOM^{new1}$, the first proposed version is able to solve the problem of figure 5 partially but not fully.

As the value of $LCOM^{new2}$ for problem of figure 4 (a) which is less cohesive is 0.124 as compared to 0.078 value for more cohesive problem of figure 4 (b), that shows that the second version is able to discriminate the less cohesive candidate from the more cohesive ones that, the CK metric of LCOM is not able to do. The first version is also able to discriminate the less cohesive candidate from the more cohesive ones by producing higher value of $LCOM^{new1}$ for less cohesive problem.

Hence the second version is able to solve both the problems that the original CK metric of LCOM was not able to solve so it can become the true candidate of the rectified version of CK metric as far as the cohesiveness of a class is concerned.

4. References

Basili, V., Briand, L., and Melo, W. "A Validation of Object Oriented Design Metrics as Quality Indicators," *IEEE Trans. Software Engineering*., vol. 22, 1996, pp. 751-761.

Chidamber, S.R., and Kemerer, C.F. "A Metric Suite for Object Oriented Design," *IEEE Trans. Software Engineering*, Vol. 20, 1994, pp. 476-493.

Chidamber, S.R., and Kemerer, C. F. "Towards a Metrics Suite for Object Oriented Design," *Proc. Conf. Object Oriented Programming Systems, Languages, and Applications (OOPSLA'91),* vol. 26, no. 11, 1991, pp. 197-211.

Henderson-Seller, B., and Constantine, L. L. "Coupling and Cohesion (towards a valid metrics suite for object oriented analysis and Design", *Object Oriented Systems,* 3, 1996, pp. 143-158.

Henderson-Sellers, B. "Some metrics for object oriented software engineering," In J. Plotter, M. Tokoro, and B. Meyer, editors, *TOOLS 6: Technology of Object-Oriented Languages and Systems*, Englewood Ciffs, N. J., 1991, Prentice Hall TOOLS Conference Series, pages 131-139.

Krishnan, M. S., Kriebel, C. H., Kekre, S., and Mukhopadhyay, T. "An Empirical Analysis of Productivity and Quality in Software Products," *Management Science*, vol. 46, 2000, pp. 745-759.

Li, W., Henry, S., Kafura, D., and Schulman, R. "Measuring Object- Oriented Design," *Journal of Object Oriented Programming,* July-August 1995, pages 48-55.

Lorenz, M., and Kidd, J. *Object – Oriented Software Metrics*, Prentice Hall, Englewood Cliffs, N. J.,1994

Mayer, T., and Hall, T. "Critical Analysis of Current OO Design Metrics", *Software Quality Journal*, 8, 1999, pp. 97-110,

Pressman, R. " A Practitioner's Approach to Software Engineering," Mc-grawhill Publications, 2001, pp. 658-662.

Sheldon, F. T., Jerath, K., and Chung, H. "Metrics for Maintainability of Class Inheritance Hierarchies", *Journal of Software Maintenance*, issue 3, May 2002, pp. 147-160.

Weyuker, E. "Evaluating software complexity measures," *IEEE Trans. Software Eng*., 14, 1988, pp. 1357–1365.