An Overview of Various Object Oriented Metrics

***Brij Mohan Goel, **Prof. Pradeep Kumar Bhatia**
*Research Scholar, Deptt. of Computer Science & Engg., SGVU, INDIA
**Deptt. of CSE., G J University of Science & Technology, Hisar, (Haryana), India
Email: brijmohan.vce@gmail.com, pkbhatia.gju@gmail.com

Abstract: This paper gives the different type of metrics used in object–oriented environments. Our survey includes a set of various object–oriented metrics i.e. those designed specifically for object–oriented programming. We believe that these metrics have significant implications for designing high-quality software products using the object oriented approach. The results can be of great assistance to quality engineers in selecting the proper set of metrics for their software projects.

Keywords: object-oriented metrics, object oriented programming, software designing.

Accepted On: 30.01.2013

1. Introduction

The metrics presented in this paper are object–oriented metrics and metrics proposed for structural programming that could also be applied to object–oriented programming. The aim of this paper is not to mention all the existing metrics or fully present the mentioned metrics but to make the reader aware of their existence and offer references for further reading. The categories chosen to present the metrics are defining a metrics classification and used simply to ease the presentation and sometimes a metric may fall in more than one category.

The aim of this work is to present a broad survey of the existing literature of OO measures that can be applied to measure internal quality attributes of class diagrams, considering the following proposals: Chidamber and Kemerer, Li and Henry, Sharble and Cohen, Kim and Ching, Abreu, Brito e Abreu and Carapuça, Lorenz and Kidd, Briand et al., Harrison et al., Bansiya et al., Genero et al., Tang, Kao and Chen. The objective of the surveyed metrics proposed for OO systems, focusing on product metrics that can be applied to an advanced design or to code.

Several authors have put their suggestions that must be taken into account when defining metrics for software. Metrics must be defined pursuing clear objective and calculation must be easy and it is better if their extraction is automated by a tool.

The objective of this work is to provide researchers with an overview of the current state of metrics for object–oriented programming, focusing on the strengths and weaknesses of each existing proposal. Thus, researchers can have a broad insight into the work already done and that still to be carried out in the field of metrics for object–oriented programming.

2. Object-Oriented Metrics

We will now present those metrics proposals selected for consideration and that may best demonstrate the present-day context of metrics for object-oriented programming.

2.1 CK metrics

In 1991, Chidamber and Kemerer[1] proposed a first version of these metrics and later the definition of some of them were improved and presented in 1994[2]. The six metrics can be summarised as in Table 1.

Table1: CK Metrics

Metric Name	Definition
WMC (Weighted Method Per Class)	Sum of Complexities of local methods of a class.
DIT (Depth of Inheritance)	The depth of a class within the inheritance hierarchy is the maximum number of steps from the class node to the root of the tree and is measured by the number of ancestor classes.
NOC (Number of Children)	A count of the number of direct children of a given class.
CBO (Coupling Between Objects)	This metric is a count of the number of other classes to which the current class is coupled, via non-inheritance-related couples. Two classes are coupled when the methods of one class use methods or attributes of another class.

RFC (Response for a Class)	The RFC is the count of the set of all methods that can be invoked in response to a message to an object of the class or by some method in the class.
LCOM (Lack of Cohesion of Methods)	The cohesion of a class is characterized by how closely the local methods are related to the local attributes.

- **Objective** of CK metrics were defined to measure design complexity in relation to their impact on external quality attributes such as maintainability, reusability, etc.

2.2 Metrics by Li and Henry

Li and Henry in 1993 have proposed several object-oriented software metrics and evaluated the relation between their metrics and the maintenance effort in two commercial systems [3]. They classified three groups of object-oriented metrics in the object-oriented paradigm.

First group contains all the metrics (WMC, DIT, NOC, RFC, LCOM) except CBO proposed in [4].

Second group contains three additional metrics (MPC, DAC, NOM).

Third group contains two size metrics (SIZE1, SIZE2)

The detail of metrics is given in Table 2.

Table 2: Li and Henry Metrics

Metric Name	Definition
MPC (Message Passing Coupling)	MPC(C) is defined as the number of send statements defined in class C.
DAC (Data Abstracting Coupling)	DAC(C) is defined as the number of ADTs defined in a class C.
NOM (Number of Methods)	NOM(C) is defined as the number of local methods in a class C.
SIZE1 (Size of procedures or functions)	SIZEl(C) is calculated by counting the number of semicolons in a class C.
SIZE2 (Size of properties defined in a class)	SIZE2(C) is calculated by summing the number of attributes and the number of local methods in a class C.

- **Objective** of these metrics were defined to make the relation between different types of metrics.

2.3 Metrics by Li

Li in 1998 developed another Object-Oriented syntactic metrics suite that addressed certain shortcomings in Chidamber and Kemerer's metrics suite[5,6]. The six metrics can be summarised as in Table 3.

Table 3: Li Metrics

Metric Name	Definition
Number of Ancestor Classes (NAC)	This metric measures the total number of ancestor classes from which a class inherits in the class inheritance hierarchy. It addresses a problem with multiple inheritance in the C&K DIT metric.
Number of Descendant Classes (NDC)	This metric measures the number of classes that may potentially be influenced by the class because of inheritance relations. It addresses a problem with the C&K NOC metric, in which the C&K NOC metric counted only the immediate children of a class, and not the grandchildren. A class influences all its subclasses and not just the immediate children.
Number of Local Methods (NLM)	This metric counts the number of local methods defined in a class which are accessible outside the class. Li felt that this metric better defines one of two possible versions of the C&K WMC metric (the other version is better defined by CMC).
Class Methods Complexity (CMC)	This metric is the sum of the internal structural complexity of all local methods, regardless of whether they are visible outside the class or not. Li felt that this metric better defines one of two possible versions of the C&K metric (the other version is better defined by NLM).
Coupling Through Abstract Data Types (CTA)	This metric counts the total number of classes that are used as abstract data types in the data attribute declaration of a class.
Coupling Through Message Passing (CTM)	This metric 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.

Li used different names for his metrics in different papers, but his papers generally employ the same set of metrics [5, 6]. Li still employs a definition of the LCOM metric to measure the cohesion of a class (this LCOM definition is consistent with that of Li and Henry [7], and is different from the later definition of LCOM provided by CK [8]. The CK and Li metrics suites have been examined here since they are widely studied syntactically-based Object-Oriented metrics suites, particularly the CK metrics suites. Other syntactically-based metrics suites have also been defined.

- **Objective** of these metrics measure different internal attributes such as coupling, complexity and size.

2.4 Metrics by Sharble and Cohen

They used nine metrics including six metrics (WMC, DIT,NOC, CBO, RFC, LCOM) for object oriented design proposed in [9], two metrics (WAC, NOT) proposed in the data-driven method, and one metric (VOD) proposed by themselves. In 1993, Sharable represents a set of nine metrics (WMC, DIT, NOC, CBO, RFC, LCOM, WAC, NOT, VOD) defined by Sharble and Cohen. Among the following definitions, WAC and NOT are metrics in the data-driven method and VOD is the one by Sharble and Cohen. The detail of metrics is given in Table 4.

Table 4: Sharble and Cohen Metrics

Metric Name	Definition
WAC (Weighted Attribute per Class)	WAC(C) is defined as the number of attributes, in a class C, weighted by their size.
(NOT) Number of Tramps	NOT(C) is defined as the total number of extraneous parameter in signatures of methods of a class C.
(VOD) Violation of the law of Demeter	VOD(C) is defined as the number of violations of the law of Demeter for a class C.

- **Objective** of the metrics proposed by Sharble and Cohen are used to compare two different methods for the development of object-oriented software that are the responsibility-driven method and the data-driven method [10].

2.5 Metrics by Kim

While existing object-oriented metrics are not as numerous as those in the procedural paradigm. In 1993 Kim has proposed complexity metrics which can be calculated directly from program codes [11]. The detail of metrics is given in Table 5.

Table 5: Kim Metrics

Metric Name	Definition
Static Complexity SC(P)	Static complexity is concerned with a question "How complex is the structure of defined classes?" Then, static complexity is calculated using complexities of method and class.
Dynamic Complexity DC(P)	Dynamic complexity is associated with control flow and system structure. It is measured by calculating the degree of reuse which is invocated by call of method.
Total Complexity TC(P)	TC(P)=SC(P) + DC (P)

- **Objective** of the metrics proposed by Kim are used to evaluate the complexity of C++ programs from two viewpoints: static and dynamic complexities.

2.6 Metrics by Kim and Ching

In 1994, Kim examine in detail the metrics from the viewpoint of three key aspects in object-oriented paradigm: (1)Syntax complexity, (2)Inheritance complexity and (3)Interaction complexity. For each key aspect, we expand it to five attributes, which can be calculated quantitatively [12, 13, 14, 15]. The detail of metrics is given in Table 6.

Table 6: Kim and Ching Metrics

Metric Name	Definition	Attribute	Definition
Syntax Complexity SX(P)	Syntax complexity is a coding structure of program. That is, it measures the volume of program and the effort of coding.	1. A_{IMC}=Degree of internal method complexity	The complexity of method is a fundamental metric in an object-oriented program and tells how much efforts and times are needed in composing and managing the program.
		2. A_{NOM}= number of methods in a class	The complexity of a class is influenced by the number of methods in the class and the complexities of methods. The larger the number of methods is, the more complex the class is.
		3. A_{NOCL}==Number of classes in a program	The complexity of program is related to the number of classes, and the number of classes is related to the degree of reuse of class. The larger the number of classes is, the greater the efforts of programmer and the less the degree of reuse of class is.
		4. A_{LCOM} = Degree of	The cohesion of a class is characterized by how closely

		lackness of cohesion in methods	local methods are related to local instance variables in the class. Class size tends to decrease with increasing cohesion of the method in the class.
		5. A_{UOC} = Degree of usability of classes in a program.	Sometimes, classes are only defined and not used. In this case, we consider that the usability of the class decreases and the complexity of use of the class increases.
Inheritance Complexity IH(P)	Inheritance complexity measures the degree of reuse by inheritance.	1. A_{DIT} = Depth of inheritance tree	The deeper a class is in the inheritance hierarchy, the greater the number of methods it will inherit is, making it more complex. But, as the reusability of a class increases, the value of complexity decreases gradually because inheritance permits code reusability. As the result, reusing of existing code saves time and cost and increases a program's reusability.
		2. A_{NOC}=Number of children[2]	A_{NOC} relates to the notion of the scope of properties. It is a measure to evaluate how many sub-classes are going to inherit the methods of parents.
		3. A_{NOA}= Number of inheriting ancestor directly	The number of inheriting ancestor directly is related to reusability of inheritance. Since a class is inherited by superclass, the number of methods in the class decreases and the degree of reusability of superclass increase.
		4. A_{DOR} = Degree of reuse by inheritance	A_{DOR} means how the complexity of a class is reduced by increasing the degree of reuse by inheritance.
		5. A_{NOD} = Number of disjoint inheritance trees	The number of disjoint inheritance trees gives much influence on designing efforts.
Interaction Complexity IT(P)	Interaction complexity measures the degree of coupling.	1. A_{CBI} = Number of couplings by inheritance	Inheritance promotes software reuse in the object-oriented paradigm. However, it also creates the possibility of violating encapsulation and information hiding[5].
		2. A_{RFC} = Number of responses for a class[2]	If a large number of methods can be invoked in response to a message, then testing and debugging of the object becomes more complicated.
		3. A_{CBO} = Number of coupling between objects.	When a function calls another function, the degree of changes of called function is depended to the number of passed parameters.
		4. A_{VOD} = Number of Violations of the law of Demeter[7]	The law of Demeter attempts to minimize the coupling between classes. If a class follows this law, then its methods can invoke only the methods within a limited set of classes.
		5. A_{MPC} = Number of coupling through message passing	Message-passing coupling is used to measure the complexity of message passing among classes.
Complexity of Program P: COMP(P)	SX(P) + IH(P) + IT(P)		

- **Objective.** On the basis of conventional object-oriented metrics and our analytic studies on object-oriented programs, we propose a new framework which evaluates the scope or the capability of complexity metrics for object-oriented programs.

2.7 Abreu Metrics

The set of six metrics developed by Abreu in 1995 [16] were intended to be design metrics. The emphasis behind the development of the metrics is on the features of inheritance, encapsulation and coupling. The six metrics can be summarised as in Table 7.

Table 7: Abreu Metrics

Metric Name	Definition
Polymorphism Factor (PF)	This metric is based on the number of overriding methods in a class as a ratio of the total possible number of overridden methods. Polymorphism arises from inheritance, and Abreu claims that in some cases, overriding methods reduce complexity, so increasing understandability and maintainability.

Coupling Factor (CF)	This metric counts the number of inter-class communications. There is a similarity here with the NCR metric of L&K. Abreu views coupling as increasing complexity, reducing both encapsulation and potential reuse and limiting understandability and maintainability.
Method Hiding Factor (MHF)	This metric is the ratio of hidden (private or protected) methods to total methods. As such, MHF is proposed as a measure of encapsulation.
Attribute Hiding Factor (AHF)	This metric is the ratio of hidden (private or protected) attributes to total attributes. AHF is also proposed as a measure of encapsulation.
Method Inheritance Factor (MIF)	This metric is a count of the number of inherited methods as a ratio of total methods. There is a similarity here with the NCR metric of L&K. Abreu proposes MIF as a measure of inheritance, and consequently as a means of expressing the level of reuse in a system. It could also claim to be an aid to assessment of testing needed.
Attribute Inheritance Factor (AIF)	This metric counts the number of inherited attributes as a ratio of total attributes. Just as for the MIF, Abreu proposes AIF as a means of expressing the level of reuse in a system. It is claimed, however, that too much reuse causes a deterioration in understandability and testability.

- **Objective** of these metrics measure design in the development of inheritance, encapsulation and coupling.

2.8 MOOD metrics

The original proposal of MOOD metrics by Britoin 1994 was improved by Brito in 1996a, and recently extended to MOOD2 metrics by Brito in 1998, which consider metrics defined at different levels of granularity, not only at class diagram level. Brito e Abreu in 2001 also presented a formal definition of MOOD2 metrics using the Object Constraint Language (OCL) by Warme in 1999. Given that MOOD metrics were more explored as empirically as theoretically, we will only refer to them in the rest of this section (we consider the improved version defined by Brito e Abreu and Melo in 1996a [17,18,19,20,21]. Table 8 shows six of the MOOD metrics applied at class diagram level.

Table 8: Mood Metrics

Metric Name	Definition
MHF	See from Table 7
AHF	
MIF	
AIF	
PF	
CF	

- **Objective** They were defined to measure the use of OO design mechanisms such as inheritance (MIF and AIF) metrics, information hiding (MHF and AHF metrics), and polymorphism (PF metric) and the consequent relation with software quality and development productivity.

2.9 Lorenz and Kidd Metrics

We now describe ten metrics proposed by Lorenz and Kidd in 1994. We note that many other metrics were suggested by L&K in [22]. However, the ten metrics described give a fair cross-section of the broad areas covered. Unlike the C&K metrics, most of the L&K metrics are direct metrics, and include more directly countable measures, e.g., the Number of Methods (NM) metric, and the Number of Variables (NV) metric. Although relatively simple to collect, doubt can be cast on the usefulness of such metrics because they give only a limited insight into the architecture of the system under investigation. For each of the metrics considered, L&K offered some justification for the existence of that metric and we include that justification in the following analysis. The ten metrics can be summarised as in Table 9.

Table 9: Lorenz and Kidd Metrics

Metric Name	Definition
Number of Public Methods (PM)	This simply counts the number of public methods in a class. According to L&K, this metric is useful as an aid to estimating the amount of work to develop a class or subsystem.
Number of Methods (NM)	The total number of methods in a class counts all public, private and protected methods defined. L&K suggest this metric as a useful indication of the classes which may be trying to do too much work themselves; i.e., they provide too much functionality.
Number of Public	Number of Public Variables per class (NPV). This metric counts the number of

Variables per class (NPV)	public variables in a class. L&K consider the number of variables in a class to be one measure of its size. The fact that one class has more public variables than another might imply that the class has more relationships with other objects and, as such, is more likely to be a *key class,* i.e., a central point of co-ordination of objects within the system.
Number of Variables per class(NV)	This metric counts the total number of variables in a class. The total number of variables metric includes public, private and protected variables. According to L&K, the ratio of private and protected variables to total number of variables indicates the effort required by that class in providing information to other classes. Private and protected variables are therefore viewed merely as data to service the methods in the class.
Number of Methods Inherited by a subclass (NMI)	This metric measures the number of methods inherited by a subclass. No mention is made as to whether that inheritance is public or private. In a language such as C++, we have to consider the possibility that the inheritance may be private, Then, any classes using methods from a subclass would not necessarily have access to all of the inherited methods.
Number of Methods Overridden by a subclass (NMO)	A subclass is allowed to re-define or override a method in its superclass(es) with the same name as a method in one of its superclasses. According to L&K, a large number of overridden methods indicates a design problem, indicating that those methods were overridden as a design afterthought. They suggest that a subclass should really be a specialisation of its superclasses, resulting in new unique names for methods.
Number of Methods Added by a subclass (NMA)	According to L&K, the normal expectation for a subclass is that it will further specialise (or add) methods to the superclass object. A method is defined as an added method in a subclass if there is no method of the same name in any of its superclasses.
Average Method Size	The average method size is calculated as the number of non-comment, nonblank source lines (NCSL) in the class, divided by the number of its methods. AMS is clearly a size metric, and would be useful for spotting outliers, i.e., abnormally large methods.
Number of times a class is Reused (NCR)	The definition of NCR given by L&K is somewhat ambiguous. We assume the metric is intended to count the number of times a class is referenced (i.e., reused) by other classes. In this sense, we could view reuse in a similar way to coupling. We could then consider NCR as a measure of the extent of inter-class communication, and in this respect, a high value for NCR as undesirable.
Number of Friends of a class (NF)	This metric measure, for each class, the number of friends of that class. Friends allow encapsulation to be violated, and as such should be used with care. A high number of friends within a class could indicate a potential design flaw, an oversight in design, which has filtered through to the coding stage; we note in passing that friends are a concept specific to the C++ language. NF is a measure of class coupling, since friends may rely on a particular class (or classes) to operate properly.

- **Objetive.** Lorenz and Kidd's metrics were defined to measure the static characteristics of software design, such as the usage of inheritance, the amount of responsibilities in a class, etc.

2.10 Briand et al.'s Metrics

In 1997, Brian metrics are defined at the class level, and are counts of interactions between classes [23]. The detail of metrics is given in Table 10.

Table 10: Briand et al.'s Coupling Metrics

Metric Name	Definition
ACAIC **OCAIC** **DCAEC** **OCAEC** **ACMIC** **OCMIC** **DCMEC** **OCMEC**	These measures distinguish the relationship between classes different type of interactions, and the locus of impact of the interaction. The acronyms for the measures indicate what interactions are counted: ▪ The first letter indicates the relationship (A: coupling to ancestor classes, D: Descendants, O: Others, i.e. none of the other relationships). ▪ The next two letters indicate the type of interaction: • CA: there is a Class-attribute interaction between classes c and d, if c has an attribute of type d. • CM: There is a Class-Method interaction between classes c and d, if class c has a method with a parameter type class d. ▪ The last two letters indicate the locus of impact: • IC: Import coupling, the measure counts for a class c all

	interactions where c is using another class. • EC: Export coupling: count interactions where class d is the used class.

- **Objective.** The aim of these metrics is the measurement of the coupling between classes.

2.11 Harrison et al. 's Metrics

The Harrison in 1998 has proposed the metric **Number of Associations (NAS)**, which is defined as the number of associations of each class, counted by the number of association lines emanating from a class in a class diagram [24].

- **Objective.** The NAS metric measures the inter-class coupling.

2.12 Bansiya et al.'s Metrics

The metrics defined by Bansiya in 2002 and Davis which can be applied at class level.

Table 11: Bansiya and Davis's Metrics [Bansi02]

Metric Name	Definition
DAM	The Data Access metric is the ratio of the number of private (protected) attributes to the total number of attributes declared in the class.
DCC	The Direct Class Coupling metric is a count of the different number of classes that a class is directly related to. The metric includes classes that are directly related by attribute declarations and message passing (parameters) in methods.
CAMC	The Cohesion Among Methods of Class metric computes the relatedness among methods of a class based upon the parameter list of methods [Bansiya99]. The metric is computed using the summation of the intersection of parameters of a method with the maximum independent set of all parameter types in the class.
MOA	The Measure of Aggregation metric is a count of the number of data declarations whose types are user defined classes.
MFA	The Measure of Functional Abstraction metric is the ratio of the number of methods inherited by a class to the total number of methods accessible by member methods of the class.

Bansiya in 2002 and Davis has used the metrics shown in Table 11 above and others taken from the literature (see Table 12) to build a model for evaluating the overall quality of an OO design based on its internal design properties.

Table 12: Others OO Dsign Metrics used in [Bansi02]

Metric Name	Definition
DSC	This metric counts the total number of classes in the design.
NOH	The metric counts the total number of class hierarchies in the design.
ANA	The Average Number of Ancestors metric is computed by determining the number of classes along all paths from the "root" class(es) to all classes in an inheritance structure.
NOP	This metric counts the total number of polymorphic methods.

This hierarchical model called QMOOD has the lower-level design metrics well defined in terms of design characteristics, and quality is assessed as an aggregation of the model's individual high-level quality attributes. The high-level attributes are assessed using a set of empirically identified and weighted OO design properties, which are derived from the metrics shown in table 11 and 12, which measure the lowest-level structural, functional and relational details of a design (see Table 13).

Table 13: Metrics for Design Properties

Design property	Derived Design Metric
Design size	DSC
Hierarchies	NOH
Abstraction	ANA
Encapsulation	DAM
Coupling	DCC
Cohesion	CAMC
Composition	MOA
Inheritance	MFA
Polymorphism	NOP
Messaging	CIS
Complexity	NOM

Lastly, the effectiveness of the initial model in predicting design quality attributes has been validated against numerous real-world projects. The quality predicted by the model shows good correlation with evaluator assessment of projects designs and predicts implementation qualities well [25, 26].

- **Objective.** These metrics were defined for assessing design properties such as encapsulation (DAM), coupling (DCC), cohesion (CACM), composition (MOA) and inheritance (MFA).

2.13 Genero et al. 's metrics

In 2000, 02 Genero et al.'s metrics were grouped into: Class-scope metrics (applied to single classes) and Class-diagram scope metrics (applied at diagram level) (see Table 14 and 15 respectively)[27].

Table 14: Class Diagram-Scope Metrics

Metric Name	Definition
NAssoc	The Number of Association metric is defined as the total number of associations within a class diagram. This is a generalization of the NAS (Number of Associations) metric [Harri00] to the class diagram level.
NAgg	The Number of Aggregation metric is defined as the total number of aggregation relationships within a class diagram (each whole part pair in an aggregation relationship).
NDep	The Number of Dependencies metric is defined as the total number of dependency relationships within a class diagram.
NGen	The Number of Generalization metric is defined as the total number of generalization relationships within a class diagram (each parent-child pair in a generalization relationship).
NGenH	The Number of Generalization metric is defined as the total number of generalization relationships within a class diagram (each parent-child pair in a generalization relationship).
NAggH	The Number of Aggregation Hierarchies metric is defined as the total number of aggregation hierarchies within a class diagram.
MaxDIT	The Maximum DIT metric is defined as the maximum between the DIT value obtained for each class of the class diagram. The DIT value for a class within a generalization hierarchy is the length of the longest path from the class to the root of the hierarchy [Chida94].
MaxHAgg	The Maximum HAgg metric is defined as the maximum between the HAgg value obtained for each class of the class diagram. The HAgg value for a class within an aggregation hierarchy is the length of the longest path from the class to the leaves.

Table 15: Class-Scope Metrics

Metric Name	Definition
NAssocC	The Number of Association per Class metric is defined as the total number of associations a class has with other classes or with itself.
HAgg	The height of a class within an aggregation hierarchy is defined as the length of the longest path from the class to the leaves.
NODP	The Number of Direct Parts metric is defined as the total number of "direct part" classes which compose a composite class.
NP	The Number of Parts metric is defined as the number of "part" classes (direct and indirect) of a "whole" class.
NW	The Number of Wholes metric is defined as the number of "whole" classes (direct or indirect) of a "part" class.
MAgg	The Multiple Aggregation metric is defined as the number of direct "whole" classes that a class is part-of, within in an aggregation hierarchy.
NDepIn	The Number of Dependencies In metric is defined as the number of classes that depend on a given class.
NDepOut	The Number of Dependencies Out metric is defined as the number of classes on which a given class depends.

- **Objective.** They were defined to measure class diagram complexity, due to the use of different kinds of relationships, such as associations, generalizations, aggregations and dependencies, in relation with their impact on external quality attributes such as class diagram maintainability.

2.14 Tang, Kao and Chen Metrics

In 1999 Tang present a set of new metrics shown in Table 16 which are derived from our observations in studying CK metrics [28].

Table 16: Tang, Kao and Chen Metrics

Metric Name	Definition
1. Inheritance Coupling (IC)	The IC provides the number of parent classes to which a given class is coupled. A class is coupled to its parent class if one of its inherited methods is functionally dependent on the new or redefined methods in the class. In general, a class is coupled to its parent class if one of the following conditions holds: 1. One of its inherited methods uses a variable (or data member) that is defined in a new/redefined method. 2. One of its inherited methods calls a redefined method and uses the return value of the redefined method. 3. One of its inherited methods is called by a redefined method and uses a parameter that is defined in the redefined method. 4. One of its inherited methods uses a variable X, and the value of X depends on the value of a variable Y which is defined in a new/redefined method. The motivation behind the IC metric

	is that when a data member, which is used by an inherited method, is modified by a new or redefined method, it is likely to introduce new faults into the inherited method.
2. Coupling Between Methods (CBM)	The CBM provides the total number of new/redefined methods to which all the inherited methods are coupled. An inherited method is coupled to a new/redefined method if it is functionally dependent on a new/redefined method in the class. Therefore, the number of new/redefined methods to which an inherited method is coupled can be measured. The CBM measures the total number of function dependency relationships between the inherited methods and new/redefined methods. As a matter of fact, this metric is a variant of the IC metric. The motivation behind this metric is that the IC only measures the number of parent classes to which a given class is coupled, without the CBM, additional function dependency complexity at the methods level is not considered.
3. Number of Object/Memory Allocation (NOMA)	It measures the total number of statements that allocate new objects or memories in a class. The indirect allocations, ie. the allocations caused by calling other methods, are not considered. The motivation behind thismetric is that classes with large numbers of object/memory allocation statements tend to introduce additional complexity for object/memory management. Therefore, the higher the NOMA, the higher the probability of detecting object management faults.
4. Average Method Complexity (AMC)	The AMC provides the average method size for each class. Pure virtual methods and inherited methods are not counted. The assumption behind this metric is that a large method, which contains more code, tends to introduce more faults than a small method.

- **Objective.** To achieve more reliable testing and yet minimize redundant testing efforts. we present a set of new metrics which we consider useful as indicators of OO fault-prone classes. Therefore, these new metrics can be utilized to decide which classes need to be tested using OO testing techniques.

3. Conclusions and Future Work

This paper introduces the basic metric suite for object-oriented design. Metric data provides quick feedback for software designers and managers. Analyzing and collecting the data can predict design quality. If appropriately used, it can lead to a significant reduction in costs of the overall implementation and improvements in quality of the final product. The improved quality, in turn reduces future maintenance efforts. Using early quality indicators based on objective empirical evidence is therefore a realistic objective.
In Future, we will design a set of new metrics that can find the impact of reusability of a class, complexity and quality of system using the CK metrics and other metrics available in the literature.

References

[1] Chidamber S. and Kemerer C.: “Towards a Metrics Suite for Object Oriented Design”, Conference on Object-Oriented Programming: Systems, Languages and Applications (OOSPLA 91), Published in SIGPLAN Notices, vol. 26, no. 11, pp. 197-211, 1991.

[2] Chidamber S. and Kemerer C.: “A Metrics Suite for Object Oriented Design”, IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476-493, 1994.

[3] W. Li and S. Henry: “Object-oriented metrics that predict maintainability,” The Journal of Systems and Software, 23, pp.111-122(1993).

[4] Chidamber S. R. and Kemerer C. F.:“Towards a metrics suite for object-oriented design,” Proc. of OOPSLA’91, pp.197-211(1991).

[5] Li, W. Another Metric Suite for Object-Oriented Programming. Journal of Systems and Software, 44, 1998, pp.155- 162.

[6] Li, W., Etzkom, L., Davis, C., and Talburt, J. An Empirical Study of Design Evolution in a Software System. Information and Software Technology, accepted, to appear 2000.

[7] Li, W., and Henry, S. Object-oriented Metrics that Predict Maintainability. The Journal of Systems and Software, 23,2, 111-122(1993).

[8] Chidamber, S.R., and Kemerer, C.F. A Metrics Suite for Object-Oriented Design.

IEEE Trans. on Software Engineering, 20,6,476-493,1994.

[9] Chidamber S. R. and Kemerer C. F.:"Towards a metrics suite for object-oriented design," Proc. of OOPSLA'91, pp.197-211(1991).

[10] Sharble R. C. and Cohen S. S.:"The object oriented brewery: A comparison of two object oriented development methods," ACM SIGSOFT Software Engineering Notes, 18, 2, pp.60-73(1993).

[11] Kim E. M: An experimental evaluation of OOP complexity metrics: SOMEFOOT, Master thesis, Chonbuk National University(1993).

[12] Kim E. M, Chang O. B, Kusumoto S, Kikuno T: "Analysis of metrics for object-oriented program complexity" Computer Software and Applications Conference, IEEE Computer, pp. 201-207 (1994).

[13] Chidamber S. R. and Kemerer C. F.:"Towards a metrics suite for object-oriented design," Proc. of OOPSLA'91, pp.197-211(1991).

[14] W. Li and S. Henry: "Object-oriented metrics that predict maintainability," The Journal of Systems and Software, 23, pp.111-122(1993).

[15] Sharble R. C. and Cohen S. S.:"The object oriented brewery: A comparison of two object oriented development methods," ACM SIGSOFT Software Engineering Notes, 18, 2, pp.60-73(1993).

[16] F. Brito e Abreu, M. Goulao, and R. Esteves. Toward the design quality evaluation of 00 software systems. In 5th Int Conf on Software Quality, 1995.

[17] Brito e Abreu F. and Carapuça R.: "Object-Oriented Software Engineering: Measuring and controlling the development process", 4th International Conference on Software Quality, Mc Lean, VA, USA, 1994.

[18] Brito e Abreu F. and Melo W.: "Evaluating the Impact of Object-Oriented Design on Software Quality", 3rd International Metric Symposium", pp. 90-99, 1996a.

[19] Brito e Abreu F., Ochoa L. and Goulao M.: "The MOOD metrics set", INESC/ISEG Internal Report, 1998.

[20] Brito e Abreu F., Zuse H., Sahraoui H. and Melo W.: "Quantitative Approaches in Object-Oriented Software Engineering", ECOOP´99 Workshop Reader, LNCS vol. 1743, Springer-Verlag, pp. 326-337, 1998.

[21] Brito e Abreu F.: "Using OCL to formalize object oriented metrics definitions", Technical Report ES007/2001, FCT/UNL and INESC, 2001.

[22] Lorenz M. and Kidd J.: Object-Oriented Software Metrics: A Practical Guide, Prentice Hall, Englewood Cliffs, New Jersey, 1994.

[23] Briand L., Devanbu W. and Melo W.: "An investigation into coupling measures for C++", 19th International Conference on Software Engineering (ICSE 97), Boston, USA, pp. 412-421, 1997.

[24] Harrison R., Counsell S. and Nithi R.: "Coupling Metrics for Object-Oriented Design", 5th International Software Metrics Symposium Metrics, pp. 150-156, 1998.

[25] Bansiya J., Etzkorn L., Davis C. and Li W.: "A Class Cohesion Metric For Object-Oriented Designs", The Journal of Object-Oriented Programming, vol. 11, no. 8, pp. 47-52, 1999.

[26] Bansiya J. and Davis C.: "A Hierarchical Model for Object-Oriented Design Quality Assessment", IEEE Transactions on Software Engineering, vol. 28, no. 1, pp. 4-17, 2002.

[27] Genero M.: "Defining and Validating Metrics for Conceptual Models", Ph.D. thesis, University of Castilla-La Mancha, 2002.

[28] Tang M.H., Kao M.H., Chen M.H.: "An Empirical study on object-oriented Metrics" Software Metrics Symposium, 1999. IEEE Computer, Proceedings. PP. 242-249(1999).

View publication stats