International Journal of Computer Applications (0975 – 8887)
Volume 123 – No.1, August 2015

Analysis of Object Oriented Metrics on a Java Application

D.I. George Amalarethinam
Associate Professor
Department of Computer Science
Jamal Mohamed College
Tiruchirappali, India

P.H. Maitheen Shahul Hameed
Associate Professor
Department of Computer Science
Jamal Mohamed College
Tiruchirappali, India

ABSTRACT
Object oriented metrics have become more important in software development environment. They are used to measure software quality and to estimate the cost, to enhance reliability, maintainability and effort of software projects. Object oriented metrics estimate the complexity of OO programs. This paper highlights all the object oriented metrics which are proposed in the last two decades such as CK metrics, Moose Metrics, QMOOD Metrics, GQM, MOOSE, LI Metrics, Chen Metrics, Lorenz Kidd Metrics, Reuse Metrics and EMOOSE. The need for such metrics is particularly acute when an organization is adopting a new technology for which established practices have yet to be improved. This research addresses these desires through the development and implementation of a suite of metrics for OO design. The equations and measurement calculation methods for all mentioned OO metrics are clearly defined. In this research paper a java program is taken as a model with OOP concepts such as inheritance, polymorphism, and abstraction. The above mentioned Object oriented metrics are applied on this java program and the results of each metrics are tabulated clearly. The objective of the research is to select the correct object oriented metrics for their models and application for software developers.

Keywords
Object oriented metrics, Classes, Methods, Inheritance.

1. INTRODUCTION
Object Oriented Design and Development is an interesting area of current research and many authors have done great deal of work in modern years. In fact, Object Oriented Development is not only a different approach to design and implementation, but also an approach to software metrics. To produce high quality Object Oriented applications a strong emphasis on design aspects is highly necessary. Software metrics make it possible for software engineers to measure and predict software systems, essential resources for a project and products relevant for a software evolution. A software quality provides software engineers with a means of quantifying the assessment of a software product. Measurement can be used throughout a software project to assist in estimation, productivity assessment, quality control and project control [1]. Object oriented analysis and design focuses on objects as the primary agents involved in a computation; each class of data and related operations are collected into a single system entity. There are several object oriented programming languages that support object oriented paradigm. Most commonly used are Java, C++, C sharp, and Vb.net [2].

In this paper, a java application with advanced OOP features like inheritance, abstraction, and polymorphism is considered. Object oriented metrics have separate types of metrics for each feature of OOP. Each feature is measured by using correct metrics with their equations and calculated methods.

The metrics presented in this paper are by no means a complete set of object oriented metrics for JAVA. But this analysis can be used as a reference by software developers and managers for building a fault free, consistent and easy to preserve software product in JAVA. There are many distinguished features in JAVA that make it different from other object oriented languages. So future work will be to refine the current metrics and define additional metrics. By using these results the software designers and developers, can easily use the correct metrics for the validation of various types of programming applications.

This research helps the software designers and developers to select the correct metric types for their models and applications. Because in this paper it is clearly explained each and every metrics with their usage, limitations, equations, calculation methods, sources etc. The object oriented metrics have applied and validated for a java application. The results of evaluation methods are clearly described.

This paper is organized as follows. Section 2 presents a very brief summary of the literature review, Section 3 presents various metrics and their characteristics suite for object-oriented programming, Section 4 presents a JAVA program applied metrics for OOP, and Section 5 shows the results and discussion.

2. LITERATURE SURVEY
Gomathi. S, and Edith Linda. P proposed an over view of the OO metrics MOOSE, MOOD, QMOOD, and Chen metrics [3]. This paper highlights, while searching for object oriented metrics and find a particular metrics parameter and many are scattered. This paper mainly aims at collecting all those necessary parameters, organize it and display it in a single paper.

Website Admin proposed comparison and review on object oriented metrics [4]. This paper reviews and analyzes the difference between all the object oriented metrics effectively and maintain the comparison table.

Arti Chhikara and R.S. Chhillar proposed analyzing the complexity of JAVA programs using Object Oriented Software Metrics [5]. In this research, they investigate several object oriented metrics and applied these metrics to several java programs to analyze the complexity of software product.

Amit Sharma and Sanjay Kumar Dubey proposed comparison of software quality metrics for object oriented system [6]. In this research they highlight the classification of metrics like software quality metrics and the object oriented metrics and maintain the comparison table.

Amjan Shaik, C. R. K. Reddy, Bala Manda, Prakashini. C, Deepthi. K proposed an empirical validation of object oriented design metrics in object oriented systems [1]. In this paper we provide empirical evidence underneath the role of object oriented design metrics specifically a subset of the CK metric suite.

Seyyed Mohsen Jamali proposed object oriented metrics [7]. This research addresses the needs through the development and implementation of suite of metrics for object oriented design.

K.K.Aggarwal, Yogesh Singh, Arvinder Kaur, Ruchika Malhotra proposed empirical study of object oriented metrics [8]. This paper investigates 22 metrics proposed by various researchers.

3. VARIOUS METRICS

Object oriented metrics are based on the data and procedure model of structured analysis, Object Oriented metrics are also based on the objects and their characteristics. Various types of metrics are Chidamber & Kemerer's Metrics, LI Metrics, Lorenz and Kidd Metrics, Chen Metrics, MOOSE Metrics, EMOOSE Metrics, MOOD Metrics, QMOOD Metrics, Reuse Metrics, Goal Question Metrics (GQM).

3.1 Chidamber & Kemerer's Metrics (CK Metrics)

Chidamber and Kemerer's metrics suite for OO Design is the deepest research in OO metrics examination. They have characterized six metrics for the OO design. In this paper, a complete description of their metrics is given [7].

3.1.1 Weighted Methods per Class (WMC)

This measures the sum of complexity of the methods in a class. To predict the time and effort required to develop and maintain a class it can use the number of methods and the complexity of each method. It is used to count the methods implemented within a class [7]. A class C1, with methods M1... Mn , that are defined in the class. When C1... Cn are the complexity of the methods, then

$$WMC = \sum_{i=1}^{n} Ci \qquad \text{eq. (1)}$$

If all method complexities are to be unity, then WMC = n, the number of methods [7].

3.1.2 Depth of Inheritance Tree (DIT)

Depth of inheritance of the class is the DIT metric for the class. In case, involving multiple inheritances, the DIT would be the maximum length from the node to the root of the tree. It also helps to find out the inheritance depth of the tree from current node to the ancestor node [7]

3.1.3 Number of children (NOC)

Number of immediate sub-classes subordinated to a class in the class hierarchy is defined as NOC. This is used to measure the subclass subordinate to a class in the hierarchy [7].

3.1.4 Coupling between object classes CBO)

It provides the number of other modules that are coupled to the current module either as a client or a supplier. A class is coupled to another if it uses the member functions and/or instance variables of the other class [9].

3.1.5 Response for a Class (RFC)

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. RFC counts the occurrences of calls to other classes from a particular class [10].

RFC = | RS | where, RS is the response set for the class. The response set for the class can be expressed as:

$$RS = \{ M \} \cup \text{all } i \{ R_i \} \qquad \text{eq. (2)}$$

where $\{ R_i \}$ = set of methods called by method i and{ M } = set of all methods in the class [7].

3.1.6 Lack of Cohesion in Methods (LCOM)

LCOM measures the extent to which methods reference the classes instance data. Cohesion is a degree of methods through which all the methods of the class are inter-related with one another and provide a well bounded behavior [10]. Consider a Class1 with n methods Me_1, Me_2..., Me_n. Let $\{I_j\}$ = set of instance variables used by methods Me_i. There are n such sets $\{I_1\}$,... $\{I_n\}$.

Let $P = \{ (Ii, Ij) \mid Ii \cap Ij = \phi\}$ and $Q = \{ (Ii, ,Ij) \mid Ii \cap Ij \neq \phi\}$. If all n sets $\{I_1\}$,... $\{I_n\}$ are ϕ then let $P = \phi$.

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

calculated value [7].

3.2 LI Metrics

LI discovered some metrics as the discovered problems with Chidamber and Kemerer’s metrics during the course of defining the unit definition model for the metrics. An alternative suite of six object-oriented metrics was proposed by LI [11].

Number of Ancestor Classes (NAC), Number of Local Methods (NLM), Class Method Complexity (CMC), Number of Descendent Classes (NDC), Coupling Through Abstract Data Type (CTA), and Coupling Through Message Passing (CTM). were proposed in order to overcome some limitations found in Chidamber and Kemerer’s metrics [11].

3.2.1 Number of ancestor classes (NAC)

The Number of Ancestor classes (NAC) metric proposed as an alternative to the DIT metric measures the total number of ancestor classes from which a class inherits in the class inheritance hierarchy.

3.2.2 Number of local methods (NLM)

The Number of Local Methods metric is defined as the number of the local methods defined in a class which are accessible outside the class. It measures the attributes of a class that WMC metric intends to capture.

3.2.3 Class method complexity (CMC)

This metric CMC is defined as the summation of the internal structural complexity of all local methods.

3.2.4 Number of descendent classes (NDC)

It metric as an alternative to NOC is defined as the total number of descendent classes (subclass) of a class.

3.2.5 Coupling through abstract data type (CTA)

It is defined as the total number of classes that are used as abstract data types in the data-attribute declaration of a class.

3.2.6 Coupling through message passing(CTM)
This is 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.

3.3 Lorenz & Kidd Metrics
Lorenz & Kidd proposed a set of metrics that can be grouped in four categories of size, inheritance, internal and external. Size oriented metrics for object oriented class may be focused on count of the metrics, operations and attributes of an individual class and average value of object-oriented software as a whole. Inheritance based metrics is totally concentrated in which operations that are reused through the class hierarchy. Metrics for the class intervals are totally oriented towards the cohesion, while the external metrics were used to examine and reuse. It divide the class based metrics into the broad categories like size, internal, external inheritance and the main metrics which are focused on the size and complexity are class size (CS), Number of operations overridden by a subclass (NOO), Number of operations added by a subclass (NOA), Specialization index (SI), Average operation size (OS), Operation complexity (OC), Average number of parameters per operation (NP) [6].

3.3.1 Class Size metric (CS)
The overall size of a class can be found by using the following measurements:

1. Total number of methods that are encapsulated within the class
2. Total number of attributes that are encapsulated within the class

When the value of CS is increased, it becomes harder to understand, reuse, test, and maintain [12].

3.3.2 Number of Operations (methods) Overridden by a subclass (NOO)
There are instances when a subclass changes a method, inherited from its super class with a specialized version, for its own use. This type of replacement is called overriding [12].

3.3.3 Number of Operations (methods) Added by a subclass (NOA).
Subclasses are specialized by adding methods and attributes. When the value of NOA increases, the subclass discards away from the abstraction implied by the super class [12].

3.3.4 Specialization Index (SI)
SI provides a rough indication of the degree of specialization for each of the subclasses in an OO software system. Specialization can be achieved by either adding or overriding methods. Specialization can be calculated as follows (Pressman, 2000; Alhadithi and Taka, 2002) [12].

$$SI = (NOO * level) / M\ total \qquad \text{eq. (3)}$$

where NOO: number of operations overridden by a subclass.

Level: level in the class hierarchy at which the class resides

M total: total number of methods of a class [12].

3.4 Chen Metrics
3.4.1 *Chen* et al. proposed software metrics, through which it can define “What is the behavior of the metrics in object-oriented design” [4].

3.4.2 RM (Reuse Metric)
RM is a Boolean (0 or 1) indicator metric. Therefore, all of the terminologies in object oriented language, consider as the basic components of the paradigm are objects, classes, attributes, inheritance, method, and message passing [4].

3.5 MOOSE Metrics
Metrics for Object-Oriented Software Engineering (MOOSE): Chidamber and Kemerer (CK) et al. proposed some metrics that have generated a significant amount of interest and are currently the most well-known object-oriented suite of measurements for Object-Oriented software. The CK metrics suite consists of six metrics that assess different characteristics of the object-oriented design are given below. [4]

3.5.1 Weighted Methods per Class(WMC).
This measures the sum of complexity of the methods in a class. The complexity of the class may be calculated by the cyclomatic complexity of the methods [4].

3.5.2 Depth of Inheritance Tree (DIT).
DIT metric is used to find the length of the maximum path from the root node to the end node of the tree. DIT represents the complexity and the behavior of a class, and the complexity of design of a class and potential reuse [4].

3.5.3 Number of children (NOC).
According to Chidamber and Kemerer, the Number of Children (NOC) metric may be defined for the immediate sub class coordinated by the class in the form of class hierarchy [4].

3.5.4 Coupling Between Objects (CBO).
CBO is used to count the number of the class to which the specific class is coupled [4].

3.5.5 Response for class (RFC).
The response set of a class (RFC) is defined as set of methods that can be executed in response and messages received a message by the object of that class [4].

3.5.6 Lack of Cohesion in Methods (LCOM).
This metric is used to count the number of disjoints methods pairs minus the number of similar method pairs used. It is used to measuring the pairs of methods within a class using the same instance variable [4].

3.6 EMOOSE Metrics
Extended Metrics for Object-Oriented Software Engineering (Emoose). W.Li et al. proposed this metrics of the Moose model. They may be described as-

3.6.1 Message Pass Coupling (MPC)
It means that the number of message that can be sent by the class operations [9]. So if two different methods in class A1 access the same method in class B1, then MPC = 2. [8]

3.6.2 Data Abstraction Coupling (DAC)
It is used to count the number of classes which an aggregated to current class and also defined the data abstraction coupling [4].

3.6.3 Number of Methods (NOM)
It is used to count the number of operations that are local to the class [4].

3.7 MOOD Metrics
Metrics For Object-Oriented Design (MOOD): Each of the metrics was expressed to measure where the numerator defines

the actual use of any one of the feature for a particular design. In MOOD metrics model, there are two main features, viz, methods and attributes. The attributes are used to show the status of objects in the system and methods are used to maintained or modifying several kinds of status of the objects. Metrics are defined as given below [4].

3.7.1 Method Hiding Factor (MHF)

MHF is defined as the ratio of the sum of the invisibilities of all methods defined in all classes to the total number of methods defined in the system. The invisibility of a method is the percentage of the total classes from which this method is not visible [4].

MHF, a measure of encapsulation is defined as:

$$\text{MHF} = \frac{\sum_{i=1}^{TC}\sum_{m=}^{Md(Ci)}(\mathbf{1}-V(Mmi))}{\sum_{i=1}^{TC} Md(Ci)} \qquad \text{eq. (4)}$$

where Md(Ci) is the number of methods declared in a class,

$$\text{V(Mmi)} = \frac{\sum_{j=1}^{TC} is\_visible(Mmi,Cj)}{TC-1} \qquad \text{eq. (5)}$$

where TC is the total number of classes, and

$$\text{is visible(M mi ,C j)} = \begin{cases} 1 & iff\ j \neq \text{i} \wedge Cj\ may\ call\ Mmi \\ & 0\ otherwise \end{cases}$$

eq. (6)

Thus, for all classes, C1, C2…Cn, a method counts as 0 if another class, may use it and 1 if it cannot be used by another class. The total for the system is divided by the total number of methods defined in the system [8].

3.7.2 Attribute Hiding Factor (AHF)

AHF is defined as the ratio of the sum of the invisibilities of all attributes defined in all classes to the total number of attributes defined in the system under consideration. It is defined formally as [4]

$$\text{AHF} = \frac{\sum_{i=1}^{TC}\sum_{a=1}^{Ad(Ci)}(\mathbf{1}-V(Aai))}{\sum_{i=1}^{TC} Ad(Ci)} \qquad \text{eq. (7)}$$

where Ad(Ci) is the number of methods declared in a class, and

$$\text{V(Aai)} = \frac{\sum_{j=1}^{TC} is\_visible(Aai,Cj)}{TC-1} \qquad \text{eq. (8)}$$

where TC is the total number of classes, and

$$\text{is visible(Aai, Cj)=} \begin{cases} 1\ iff\ j \neq i \wedge\ Cj\ may\ call\ Aai \\ \quad 0\ otherwise \end{cases}$$

[5] eq. (9)

3.7.3 Method Inheritance Factor (MIF)

The Method Inheritance Factor (MIF) is defined as the ratio of the Number of Inherited Methods (NIM) to the Number of Defined Methods (NDM) and inherited methods in the class[13].

$$\text{MIF} = \frac{NIM}{NDM+NIM} \quad \text{or} \quad \text{MIF} = \frac{\sum_{i=1}^{TC} Mi(Ci)}{\sum_{i=1}^{TC} Ma(Ci)} \qquad \text{eq. (10)}$$

where, Ma(Ci) = Mi(Ci) + Md(Ci)

TC = total number of classes

Md(Ci) is the number of methods described in a class

Mi(Ci is the number of methods inherited in a class [8].

3.7.4 Attribute Inheritance Factor (AIF)

The Attribute Inheritance Factor (AIF) is defined as the ratio of the Number of Inherited Attributes (NIA) to the Number of Defined Attributes (NDA) and inherited attributes in the class. AIF is equal to that of MOOD metrics [13].

$$\text{AIF} = \frac{NIA}{NDA+NIA} \qquad \text{eq. (11)}$$

It is also defined as follows

$$\text{AIF} = \frac{\sum_{i=1}^{TC} Ad(Ci)}{\sum_{i=1}^{TC} Aa(Ci)} \qquad \text{eq. (12)}$$

where, Aa(Ci)= Ai(Ci)+ Ad (Ci)

TC= total number of classes

Ad(Ci) which is number of attribute declared in a class

Ai(Ci) which is number of attribute inherited in a class

AIF = 0 % for class lacking inheritance [8].

3.7.5 Polymorphism Factor (PF)

PF defines the ratio of the actual number of possible different polymorphic situation for class Ci to the maximum number of possible distinct polymorphic situations for class Ci.

It is defined as below

$$\text{PF} = \frac{\sum_{i=1}^{TC} Mo(Cj)}{\sum_{i=1}^{TC}[Mn(Ci)XDC(Ci)} \qquad \text{eq. (13)}$$

Mn (Ci) = Number of New Methods

Mo(Ci) = Number of Overriding Methods

DC(Ci) = Descendants Count [8].

3.7.6 Coupling Factor (CF)

NAC is the Number of Actual Couplings with other classes and NPC is the Number of Possible Couplings of this class with other classes of the system. Clearly, the numbers of possible [13].

$$\text{CF} = \frac{NAC}{NPC} \quad \textbf{or}$$

It is formally defined as:

$$\text{CF} = \frac{\sum_{i=1}^{TC}\sum_{j=1}^{TC}[is\_client(Ci,Cj)]}{TC^2-TC} \qquad \text{eq. (14)}$$

$$\text{is\_client(Cc,Cs)=} \left\{ \begin{matrix} 1\ iff\ Cc => Cs \wedge\ Cc\ \neq Cs \\ 0\ otherwise \end{matrix} \right\} \quad \text{eq (15)}$$

Couplings due to the use of the inheritance are not included in CF, because a class is heavily coupled to its ancestors via inheritance. If no classes are coupled, CF = 0 %. If all classes are coupled with all other classes, CF is equal to 100 % [8].

3.8 QMOOD Metrics

Quality Model for Object-Oriented Design (QMOOD): This is a comprehensive quality model that establishes a clearly defined and empirically validated model to assess object-oriented design quality attributes such as understandability, reusability, and relates it through mathematical expressions, with structural object-oriented design properties such as encapsulation and coupling [4].

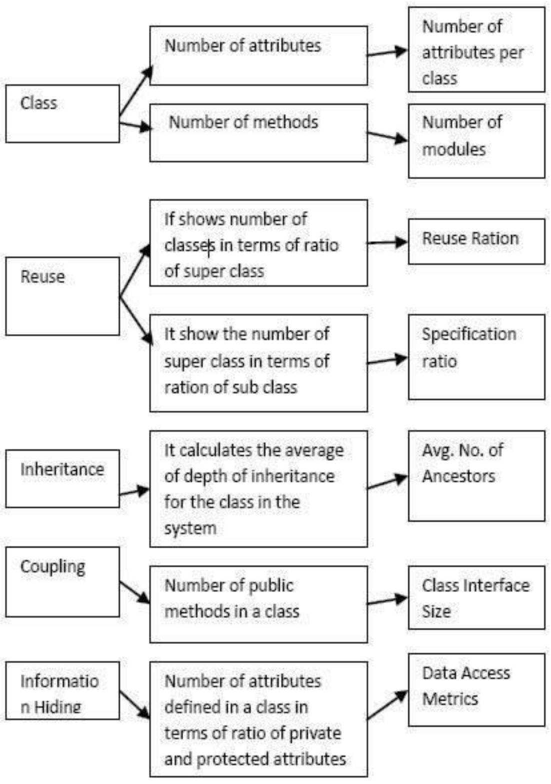
Class
Number of attributes
Number of attributes per class
Number of methods
Number of modules
Reuse
If shows number of classes in terms of ratio of super class
Reuse Ration
It show the number of super class in terms of ration of sub class
Specification ratio
Inheritance
It calculates the average of depth of inheritance for the class in the system
Avg. No. of Ancestors
Coupling
Number of public methods in a class
Class Interface Size
Information Hiding
Number of attributes defined in a class in terms of ratio of private and protected attributes
Data Access Metrics

Figure 1- QMOOD Metrics [7]

The QMOOD metrics given in figure 1 can further be classified into two measures namely,

1. System Measures: - System measures explain such metrics are DSC (Design Size in Metrics), NOH (Number of Hierarchies), NAC (Number of Abstract Classes), NLC (Number of Leaf Classes), ADI (Average Depth of Inheritance) [4].
2. Class Measures:- Class measure metrics are those metrics which can define some metrics are NOP (Number of Polymorphic Method), DCC (Direct Class Coupling), MCC (Maximum Class Coupling) [4].

The QMOOD metrics measures are shown in the Table 1.

The set of metrics in QMOOD are as follows:

3.8.1 Design Size in classes (DSC).
DSC metric is a count of the total number of classes in the design.

3.8.2 Number of Hierarchies (NOH).
NOH metric is a count of the number of class hierarchies in the design [4].

3.8.3 Direct Class Coupling (DCC).
DCC is a count of different number of classes that a class is directly related to.

3.8.4 Number of Abstract Classes (NAC)
It is a count of total number of abstract classes.

3.8.5 Number of Leaf Classes (NLC)
It is a count of total number of leaf classes.

3.8.6 Average of Depth of Inheritance (ADI)
It is an average value of the depth of inheritance.

3.8.7 Number of Polymorphic methods (NOP)
NOP metric is a count of the methods that can exhibit polymorphic behavior [4].

Table 1: QMOOD Parameters

Acronym	**Description**
DSC	Design Size in Metrics
NOH	Number of Hierarchies
DCC	Direct Class Coupling
NAC	Number of Abstract Classes
NLC	Number of Leaf Classes
ADI	Average Depth of Inheritance
NOP	Number of Polymorphic Method

3.9 Reuse Metrics
An object-oriented development environment supports design and code reuse, the most straightforward type of reuse being the use of a library class (of code), which perfectly suits the requirements. Yap and Henderson-sellers analysed two measures, designed to evaluate the level of reuse possible within classes [8].

3.9.1 Reuse Ratio (U)
The reuse ratio, U, is given by

$$U = \frac{\textbf{Number of superclasses}}{\textbf{Total Number of classes}} \qquad \text{eq. (16)}$$

3.9.2 Specialization Ratio (S)
Specialization ratio, S, is given as

$$S = \frac{\textbf{Number of subclasses}}{\textbf{Number of superclasses}} \qquad \text{eq. (17)}$$

3.10 Goal Question Metrics (GQM)
Goal Question Metrics (GQM): V. L. Basili developed GQM approach. He has also provided the set of sequence which are helpful for the designers. The goal of GQM is to express the meaning of the templates which covers purpose, perspective and environment; a set of guidelines also proposed for driving question and metrics. It provides a framework involving three steps: [4]

- Goal (Conceptual level): List major goals of the development or maintenance project.
- Question (Operational level): Derive from each goal the questions that must be answered to determine if the goals are being met.
- Metric (Quantitative level): Decide what must be measured in order to be able to answer the questions adequately [4].

Consider the following figure 2, for a particular question; OOPS and FAULT-FREE are two goals, Q5 (Quality) in common for both of these goals. The main idea of GQM is that each metric identified is placed within a context, so all metrics are collected in order to answer all questions Q1… Q5, which help to achieve the goals OOPS and FAULT-FREE.

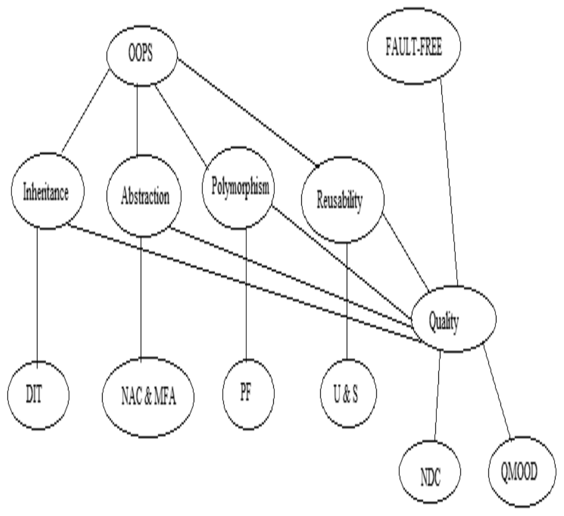

Figure 2 - Model for Goal Question Metrics Hierarchy

4. ALGORITHM - EB Bill

To better define and understand how these metrics are calculated, an algorithm for EB bill calculation is used as an example. Figure 3 shows the class diagram.

Step 1: Start

Step 2: Declare an abstraction class named as EB.

Step 3: Declare the variables cust_id and
Cust_name for the class EB.

Step 4: Define the function EB with the reference parameters as c_id and cname.

Step 5: Declare the abstraction functions Ucalc() and Acalc().

Step 6: Declare another class named as Cust_details which extends the EB class.

Step 7: Define a function named as Cust_details with the reference parameters c_id and cname.

Step 8: Print the values of c_id and cname.

Step 9: Declare another class named as Units which extends the EB class.

Step 10: Declare the variable cunit and punit for the class Units.

Step 11: Get the values for cunit and punit.

Step 12: Print the values of cunit and punit.

Step 13: Declare another class named as Calculations which extends the EB class.

Step 14: Declare the variables Tunits and amt for the class Calculations.

Step 15: Calculate the Tunit (Total units) value.

Step 16: Calculate the amt (amount) value using elseif ladder function.

Step17: Print the Tunit and amt values.

Step 18: Stop.

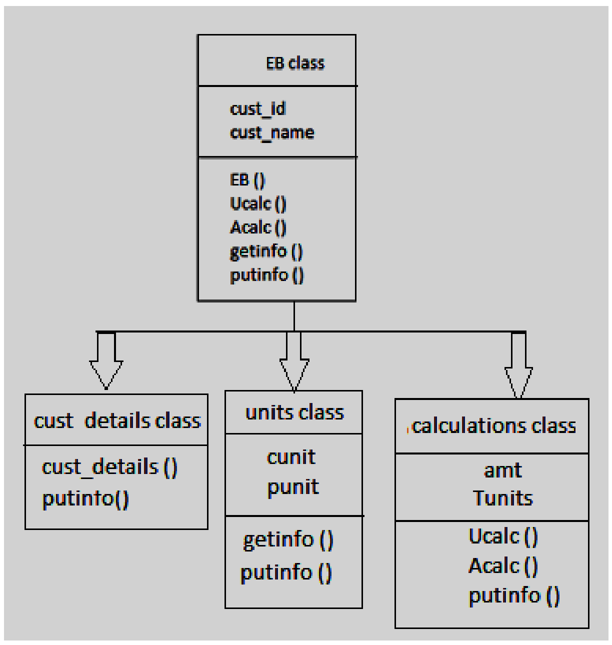

Figure 3- Class Diagram

5. RESULTS AND DISCUSSION

These metrics were calculated and tested on a Java program for EB bill application and following results are obtained. Table 2 shows the 32 object oriented metrics and its measured values.

Table 2: Object Oriented Metrics and its measured values

Sources	Metrics	EB class	Cust_ details class	Units class	Calculations class
CK Metrics	WMC	5	3	2	3
	RFC	7	3	2	9
	DIT	0	1	1	1
	NOC	3	0	0	0
	CBO	0	1	1	1
	LCOM	2			
LI Metrics	NAC	0	1	1	1
	NLM	0	1	2	1
	NDC	3	0	0	0
	CTA	1	0	0	0
	CTM	2	0	1	0
Lorenz &Kidd Metrics	CS	6	2	0	2
	NOO	0	2	2	3
EMOOSE	MPC	0	3	2	5
	DAC	0	2	0	2
	NOM	5	2	2	3
	NUS	5	3	2	3
MOOD	MHF	1			
	AHF	1			
	MIF	0.54			
	AIF	0.25			
	PF	0.46			
	CF	1			
QMOOD	DSC	4			
	NOH	3			

	DCC	**2**
	NAC	**1**
	NLC	**3**
	ADI	**0.7**
	NOP	**2**
Reuse Metrics	**Reuse ratio (U)**	**0.25**
	Speci alizati -on ratio(S)	**3**
	Inheritance Dependenci-es (ID)	**3**

The following points are observed from the Table 2 regarding the complexity of the java program for EB bill application.

The value of RFC is 9. The maximum value of RFC is high for the program as it also counts the method invocations.

The values for NOM and WMC are similar as method complexities are generally considered to be unity.

CBO value is normally less in sample data, hence classes are easy to understand, reuse and maintain.

DAC has less values representing that the developers has less tendency to use data abstractions.

MPC is a dynamic measure. It provides more information than rest of the class coupling measures.

NOM is a subset of RFC and is simple to measure.

LCOM value is 2 because the number of pairs of methods having access to common attributes is more than the number of pairs of methods having no common attributes. It mentions that the classes are cohesive.

The DIT and NOC values are not same in the program; this shows that less inheritance is used in most of the classes to optimum level.

The value of AIF is 0.25, suggesting low use of attribute inheritance.

The MIF value is 0.54. It is observed that there are very less methods in super classes; they contain only abstract methods that are overridden in subclasses.

The PF value is 0.46. So it is moderate in our program.

The MHF and AHF values are same. So, all methods or attributes are private in this program.

The value of CF is 1. It is observed that all classes in this program are coupled.

It is observed that 4 classes used in the DSC, 3 class hierarchies in the design, 2 classes are directly coupled in the design, 1 abstract class in the design, 3 leaf classes in design, 2 polymorphic methods used in the design.

The reuse ratio is 0.25. It is also observed from figure 3 that one super class and 4 classes. The specialization Index is 3. It is also observed that 3 subclasses and 1 super class.

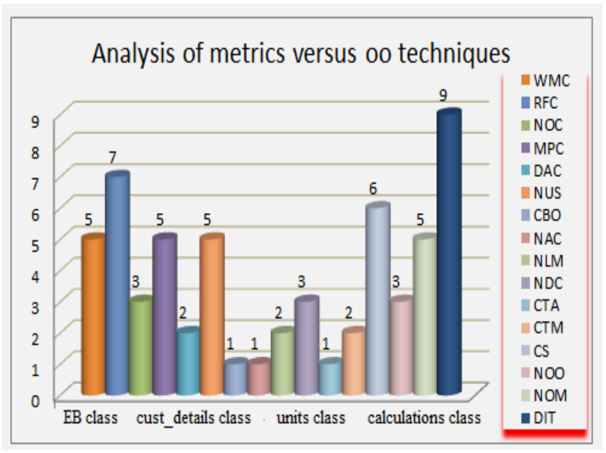

Figure 4: Analyzed Object Oriented Metrics values for all classes

The object oriented metrics for all classes are analyzed and represented in figure 4. It is concluded that the maximum value of RFC is 9 and the minimum value of DIT, CBO and NAC is 1.

6. CONCLUSION AND FUTURE WORK

This paper clearly explains all the object oriented metrics which are proposed in the last two decades like CK metrics, MOOD Metrics, GQM, QMOOD Metrics, MOOSE, LI Metrics, Chen Metrics, Lorenz and Kidd Metrics, Reuse Metrics, EMOOSE. The need for such metrics is particularly acute when an organization is adopting a new technology for which established practices have yet to be developed. The calculations for each object oriented metrics are clearly defined. In this research paper a JAVA Application is taken as a model. The possible metrics measurements are calculated for the JAVA application. Finally the measured Object oriented metrics are tabulated and their results are given.

This research paper helps researchers and software developers to select the correct object oriented metrics for their models and applications. Here, the advanced concepts of OOPs like inheritance, polymorphism, abstraction are used in our example JAVA Application. The analysis of object oriented metrics for all classes can be used as a reference by software developers and managers for building a fault free, reliable and simple to maintain software product in JAVA In future we can add more advanced features like Exception – Handling and Reliability of OOPs concepts in our applications and measure their metrics values and performances. The man power, time, and cost will be reduced by using these metrics for developing a Java software system.

7. REFERENCES

[1] Amjan Shaik, C. R. K. Reddy, Bala Manda, Prakashini. C, Deepthi. K " An Empirical Validation of Object Oriented Design Metrics in Object Oriented Systems", Journal of Emerging Trends in Engineering and Applied Sciences (JETEAS) 1 (2): 216-224c Scholarlink Research Institute Journals, 2010.

[2] Arti Chhikara, R.S.Chhillar, Sujata Khatri ,"Applying Object Oriented Metrics to C# Programs", Int. J. Comp. Tech. Appl., Vol 2 (3),666-674, 2008.

[3] Gomathi. S, Edith Linda. P, "An Overview of Object Oriented Metrics A complete Survey", International Jounal

of Computer Science & Engineering Technology, Vol. 4 No.09 Sep 2013.

[4] Website Admin, “Comparison study and Review On Object Oriented Metrics”, Wednesday, 27 November 2013.

[5] Arti Chhikara and R.S.Chhillar, “ Analyzing the Complexity of Java Programs using Object Oriented Software Metrics”, International Journal of Computer Science Issues, Vol 9, Issue 1, No 3, January 2012.

[6] Amit Sharma, Sanjay Kumar Dubey, “Comparison of software Quality Metrics for Object-Oriented System”, International Journal of Computer Science & Management Studies, Special Issue of Vol. 12, June 2012.

[7] Seyyed Mohsen Jamali, “object oriented metrics”, Software Assurance Technology Center (SATC), 2006.

[8] K.K.Aggarwal, Yogesh Singh, Arvinder Kaur, Ruchika Malhotra “Empirical Study of Object Oriented Metrics”,Journal Of Object Technology,Vol. 5, No. 8, November-December 2006.

[9] Mei-Huei Tang Ming-Hung Kao Mei-Hwa Chen, “An Empirical Study on Object-Oriented Metrics”, SUNY at Albany, Albany, NY 12222, November 1999

[10] Daniel Rodriguez Rachel Harrison, “An Overview of Object Oriented Design Metrics”, RUCS/2001/TR/A March 2001.

[11] Dr. Rakesh Kumar, Gurvinder Kaur, “Comparing Complexity in Accordance with ObjectOriented Metrics”, International Journal of Computer Applications (0975 – 8887) Volume 15– No.8, February 2011.

[12] Jubair J. Al-Ja'afer and Khair Eddin M. Sabri,” Chidamber-Kemerer (CK) and Lorenz-Kidd(LK) Metrics to Access Java Programs”, IFAC 2004.

[13] K.P. Srinivasan, Dr T. Devi, “Design and Development of a Procedure for new Object-Oriented Design Metrics”, International Journal of Computer Applications (0975 – 8887) Volume 24– No.8, June 2011.

[14] Meenakshi Kandpal and Anmol Kandpal, “ Critical Analysis of Traditional Size Estimation Metrics for Object Oriented Programming”, International Journal of Computer Applications (0975 – 8887) Volume 58– No.13, November 2012.