

Comparison of Software Quality Metrics for Object-Oriented System

Amit Sharma¹, Sanjay Kumar Dubey²

¹M.Tech-Scholar, Amity School of Engineering & Technology, Amity University
Noida, Uttar Pradesh, India
amit.krsharma123@gmail.com

²Assistant Professor, Amity School of Engineering & Technology, Amity University
Noida, Uttar Pradesh, India
skdubey1@amity.edu

Abstract

According to the IEEE standard glossary of software engineering, Object-Oriented design is becoming more important in software development environment and software Metrics are essential in software engineering for measuring the software complexity, estimating size, quality and project efforts. There are various approaches through which we can find the software cost estimation and predicates on various kinds of deliverable items. The tools are used for measuring the estimations are lines of codes, function points, and object points. This paper highlight mostly the classification of metrics like software quality metrics and the object-oriented metrics or all the software quality metrics like the HALSTEAD metrics, size metrics, quality metrics and all the object oriented metrics which are proposed from 90's like CK metrics, Moose Metrics, QMOOD Metrics, GQM, MOOSE, EMOOSE and maintain the comparison table through which we can easily analyze the difference between all the object-oriented metrics.

Keywords: *Object-Oriented, class, attributes.*

1. Introduction

Object-Oriented design is more beneficial in software development environment and object-oriented design metrics is an essential feature to measure software quality over the environment [1]. Object-oriented design is those design which contained all the properties and quality of software that is related to any large or small project [2]. It is a degree through which a system object can hold a particular attribute or characteristics. Object-oriented is a classifying approach that is capable to classify the problem in terms of object and it may provide many paybacks on reliability, adaptability, reusability and decomposition of problem into easily understood objects and providing some future modifications [3]. Software metrics makes it possible for software engineer to measure and predict software necessary resource for a project and project work product

relevant to the software development effort. Metrics provide insight necessary to create and design model through the test. It also provide a quantitative way to access the quality of internal attributes of the product, thereby it enables the software engineer to access quality before the product is build [4]. Metrics are the crucial source of information through which a software developer takes a decision for design good software. Some metrics may be transformed to serve their purpose for a new environment.

Software metrics are the tools of measurement. The term metrics is frequently used to mean a set of specific measurement taken for a particular item or process. According to the IEEE standards glossary they defined a metrics as an "a quantitative measure of degree to which a component, system, or a given attributes" [55].

Software metrics are mostly or generally characterize by the software engineering product (example design, source code, and test case), software engineering process (example analysis, design and coding) and software engineering people (example the efficiency of an individual tester or the productivity of an individual designer) [56]. Some most of the software quality metrics features are conclude by many researchers because for a good design metrics the software developer must possess the Object-Oriented design. These quality metrics features contained:-

Compliance: it contains all the ability to cover all the quality factors and all the design characteristics [25].

Orthogonality: It contains all the ability to represent different aspects of the system under measurement [57].

Formality: It contains all ability to get the same value for same system for different peoples. [40].

Minimality: It contains all the ability to used the minimum number of metrics.

Implementability / usability: It contains implementation technology i.e. independent ability to perform the task.

Accuracy: it contains all the quantitative measurements to measure the magnitude of errors [58].

Validity: It refers to the degree through which it accurately reflects or assumes the specific concept that the researchers are used to measure.

Reliability: This is the portability of failure of free software operations for specified periods.

2. Literature survey

Abreu et al. [37] provides a new classification framework for the TAPROOT. This framework was defined with the other two independent vectors these are category and granularity. Six categories of Object-Oriented metrics were defined are design metrics, complexity metrics, size metrics, quality metrics, productivity metrics and reuse metrics and also proposed three

Levels of granularity are software, class and methods but no empirical/theoretical base for the metrics was provided.

M. Alshayeb et al. [13] have given two iterative procedures for the pragmatic study of object oriented metrics. They include the short-cycled agile process and the long cycled framework evolution process. By observing the results, it can be seen that the design efforts and source lines of code added, changed, and deleted were triumphantly predicted by object-oriented metrics in short-cycled agile process where as in the case long-cycled framework process the same features were not successfully predicted by it. This has shown that the design and implementation changes during development iterations can be predicted by object-oriented Metrics, but the same cannot be the case with long-term development of an established system. R.D.Neal et al. [20] also gives the study for the validation of the object-oriented software metrics and found that some of the proposed metrics could not be considered as the valid measure for the dimension then, they could be measured. He defined a model based on measurement theory of the validation through which they can proposed 10 new metrics – Potential Methods Inherited (PMI), Proportion of Methods Inherited by a Subclass (PMIS), Density of Methodological Cohesiveness (DMC), Messages and Arguments (MAA), Density of Abstract Classes (DAC), Proportion of Overriding Methods in a Subclass (POM), Unnecessary Coupling through Global Usage (UCGU), Degree of Coupling Between Class Objects (DCBO), Number

of Private Instance Methods (PrIM), and Strings of Message Links (SML).

R. Harrison et al. [12] suggested a statistical model which is obtained from the logistic regression for identifying threshold values for the Chidamber and Kemerer metrics. The process is authenticated empirically on a large Open-Source System- the Eclipse project. Their conclusion depending on the experimental results is that the Chidamber and Kemerer Metrics have threshold effects at different risk levels. The usefulness of these thresholds on later releases was authenticated with the aid of decision trees. Another conclusion by L.H. Ethzkorn [23] is that the chosen threshold values were more precise than those were chosen depending on either intuitive perspectives or on data distribution parameters. object-oriented design metrics has also been assign the high level design quality attributes for the object-oriented software with the help of hierarchical model. H. Lieu. et al. [33] have given perception that quality of software also plays an important role in terms of safety aspects and financial aspects. They bridged the gap between quality measurement and design of these metrics, with the help of measuring the excellence of object-oriented designs during development and re-development process of the software.

M. Subramanyam et al. [34] proposed some Metrics suites and concluded that for the developers, designs metrics are very important to know the design aspects of the software and to enhance the quality of software. Rachel Harrison et al. [35] discussed about the six properties of metrics for object-oriented design (MOOD) Metrics and measured the object-oriented features like Inheritance, coupling, encapsulation, and polymorphism. In the result they showed that the metrics could be used to provide an overall assessment of the system. A. Goldberg et al. [46] have experimentally checked size estimation models that are object-oriented. The pragmatic examination of object-oriented Function Points has been extended to a considerable amount with the aid of a bigger data set and by comparing Object Oriented Function Points with other predictors of LOC (Lines of Code) in their work. Linear models where the independent variable is either a conventional Object-Oriented entity or an Object-Oriented Function Points-related measure were built and assayed by using a cross validation approach. C. Shyam et al. [14] suggests some software metrics through which we can calculating the quality of modularization of an object oriented software. They aimed that it provide a set of metrics for the large scale object oriented software

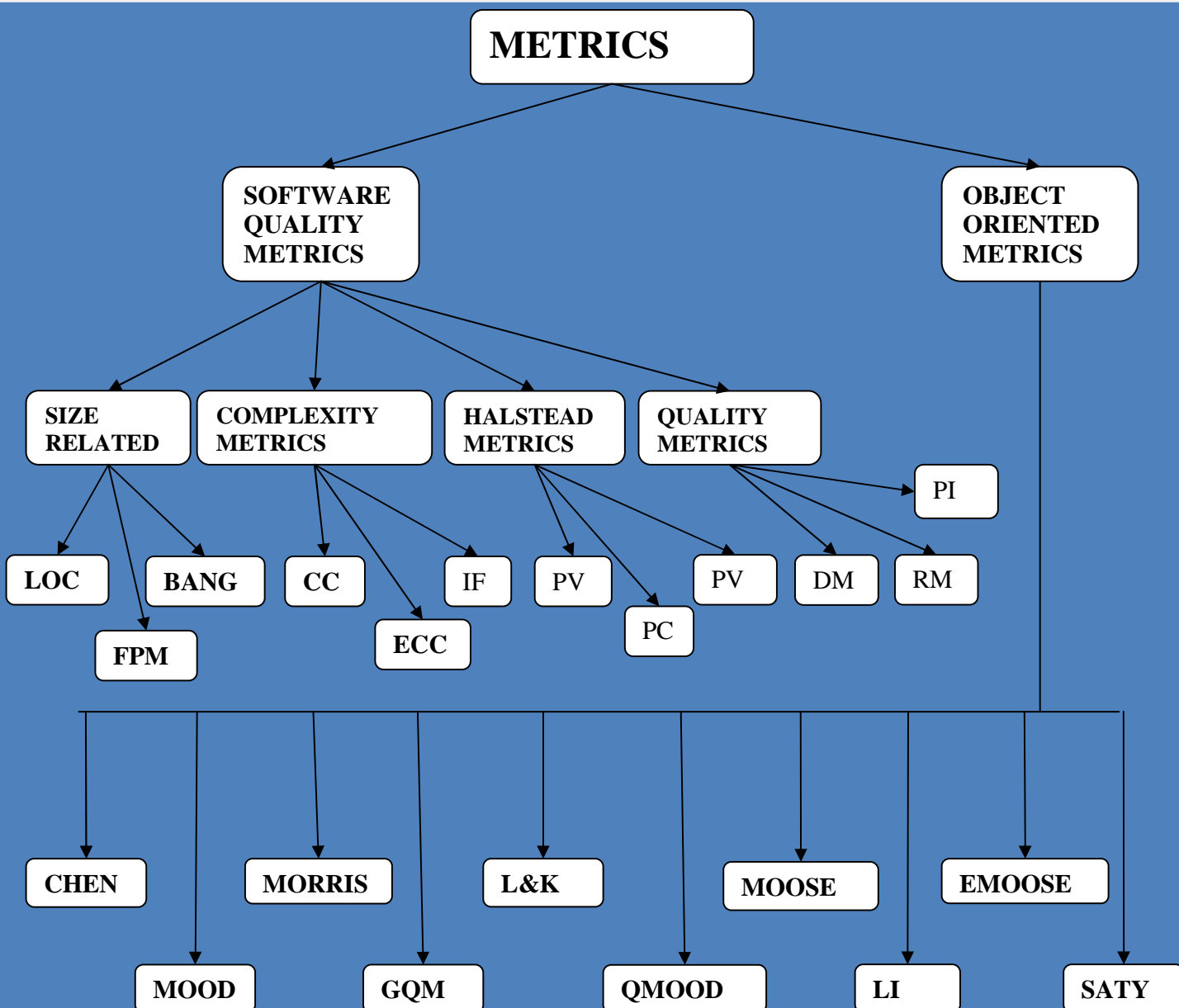
system with having some dependencies and also provide some metrics for characterizing the quality for modularization regarding the APIs of the one side. On the another side, they provide some object-oriented dependencies like inheritance, associates relationship and base class designing. Y. Zhou et al. [54] considered the fault severity using the machine learning methods with their experimental exploration of fault proneness which predict the capability of object-oriented design metrics and all of these of the predictions and the fault severity are also taken from the domain NASA data sets. J. Xu. et al. [53] have proposed an object-oriented metrics which describes the fault estimation using empirical analysis and also used the CK metrics to apprise the number of faults in the particular program. This also includes some neural and fuzzy technique. At last, the result showed that we can get a dependable fault by using CBO,

RFC, WMC, SLOC. Here SLOc is more considerable for the effect on the number of defects. C. Neelamegan et al. [45] surveyed four object-oriented metrics and mostly focused on the measurements that are totally applied on the design and class characteristics. Dr. B.R. Sastry et al. [42] trying to implement the graphics user interaction with the aid of software metrics and also tried to determine the quantity and quality of object oriented software development lifecycle.

3. REVIEW OF SOFTWARE QUALITY METRICS

Some of the software quality metrics for the development of the software development are-

A. Size related Metrics



- B. Complexity Metrics
- C. Halstead Metrics
- D. Quality Metrics

A. Size related metrics: these are the metrics which can help to quantify the software size. There are three types of software metrics which are using to measure the software size are:

- a) **Line of code (LOC):** It is the oldest metrics which is used to measure the module size but the main issue was developed in line of codes is “what to include in measurement”.
- b) **Function point Metrics:** In which we was measured line of code when the code is available and hence cannot be used in early stage. Albrecht proposed a measure of software size that can be determined early in development life cycle and names as a function point. This was totally depends on the user input, user output, inquiries and intended the values to measure the value to measure program size and thus effort required for the development.
- c) **Bang:** Dc. Marco defined bang as a function metrics. It can be calculated from the certain algorithm and data primitives available from the set of formal specification for the software and give the measures of total functionality and delivered to the user.

B. Complexity Metrics: Mc. Cabe in 1976 describe about the complexity that described the detailed design metrics for finding the complexity of the metrics, these may be described as:

- a) **Cyclomatic Complexity (CC):** T.J.Mc.Cabe in 1976 proposed that complexity measurement may be known as module logical complexity. The basic goal of the metrics is to evaluate the testability and maintainability of the software module, this metrics can also be used as a indicator of reliability in a software system. Mc .Cabe describe as:

$$V(G) = e - n + 2$$

Where,

$V(G)$ =cyclomatic complexity of flow graph G of method in which we interested

e =number of edges in G

n = number of nodes in G

One more way to calculate:

$V(G)=P+1$

Where, P=number of predicate nodes which represent a Boolean statements in code.

- b) **Extended Cyclomatic complexity (ECC):**McCabe measures the program complexity but fails to differentiate in the complexity of cases involving single condition in conditional statement than , Myers suggest the extended cyclomatic complexity that may be defined as:

$$ECC = eV(G) = Pe + 1$$

Where,

Pe =number of predicate nodes in flow graph G weighted by number of compound statements.

- c) **Information flow:** Kafura and Henry proposed the information flow with in a program structure as a metrics for program complexity. In which the metrics may be find by counting the number of local information flows input (fan-in) and flows output (fan-out). The procedure may be defined as:

$$C = [\text{procedure length}] * [(\text{fan-in}) * (\text{fan-out})]^2$$

C. Halstead Metrics: Halstead in 1976 proposed the software science theory, the main aim of this software science theory is to find out the overall software production effort. In which. It contains some vocabulary (n), length(N), volume(V).

- a) **Program vocabulary (n):** in programming languages, programs can be visualized as the set of tokens and these token may be known as operators and operands, halstead define vocabulary (n) as:

$$n = n_1 + n_2$$

where,

n_1 =number of unique operators in the program

n_2 =number of unique operands in the program

- b) **Program length (N):** The program length N may be known as the count of the total number of operators and operands. That may be represented as:

$$N = N_1 + N_2$$

Where,

N_1 =number of operators in the program

N_2 =number of operands in the program

- c) **Program volume (V):** It can be measured as the storage volume required to present in the program. That may be represented as:
 $V=N\text{Log}2n$

E.Quality Metrics: In the quality metrics, there are some of the few quality metrics-

- a) **Defect Metrics:** There is no effective procedure for counting the defects in the program, number of design change, number of intended errors and the error detected by the code inspections and the number of program test may be treated as an alternative measures to the defects.
- b) **Reliability Metrics:** internal product quality is usually measured by the number of bugs in the software or by how long the software can run before the encountering in a crash.
- c) **Maintainability Index:** Dr. paul W. Oman defined a number of functions that predict software maintainability. The maintainability index may be measured as follows-
 $MI=171-5.2*\ln(\text{ave } V)-0.23*\text{ave } V(g)-16.2*\ln(\text{ave } LOC)$

Where,

Ave V= average halstead volume per module

Ave V(g)=average extended cyclomatic complexity per module.

aveLOC=average line of code per module.

4. Review of Object-Oriented Metrics

Some object-oriented metrics for the object-oriented software development. These metrics are-

- A. Chen Metrics
- B. Morris's Metrics
- C. Lorenz and Kidd Metrics
- D. MOOSE Metrics
- E. EMOOSE
- F. MOOD Metrics
- G. Goal Question Metrics
- H. QMOOD Metrics
- I. LI Metrics
- J. SATC for object oriented metrics

A.Chen Metrics

Chen et al. [30] proposed software metrics, through which it can define "What is the behavior of the

metrics in object-oriented design". They may be described all of the behaviors like:

- (i) CCM (Class Coupling Metric),
- (ii) OXM (Operating Complexity Metric),
- (iii) OACM (Operating Argument Complexity Metric),
- (iv) ACM (Attribute Complexity Metric),
- (v) OCM (Operating Coupling Metric),
- (vi) CM (Cohesion Metric),
- (vii) CHM (Class Hierarchy of Method) and
- (viii) RM (Reuse Metric).

Metrics (i) and (iii) are very subjective in nature, Metrics (iv) and metric (vii) mostly involve the count of features; and metric (viii) 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. They proposed all of that each object oriented metrics concept implies a programming behavior.

B. Morris Metrics

Morris et al. [27] proposed a metrics suite for the object-oriented metrics systems and they define the system in the form of the tree structure and the following are the Morris's complexity and cohesion metrics. Morris defined the complexity of the object-oriented system in the form of the depth of the tree. Depth of the tree measures the number of the sub nodes of the tree. The more the number of sub nodes of tree the more complex the system. So, complexity of an object is equal to the depth of tree or total number of sub nodes.

C. Lorenz & Kidd Metrics

Lorenz & Kidd [19] proposed a set of metrics that can be grouped in four categories are 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).

D. Metrics for Object-Oriented Software Engineering (MOOSE) : Chidamber and Kemerer (CK) et al. [11] 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-

(i)Weighted Methods per Class (WMC): This measures the sum of complexity of the methods in a class. A predictor of the time and effort required to develop and maintain a class we can use the number of methods and the complexity of each method. A large number of methods in a class may have a potentially larger impact on the children of a class since the methods in the parent will be inherited by the child. Also, the complexity of the class may be calculated by the cyclomatic complexity of the methods. The high value of WMC indicates that the class is more complex as compare to the low values.

(ii)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. The following figure shows that the value of the DIT from a simple hierarchy. DIT represents the complexity and the behavior of a class, and the complexity of design of a class and potential reuse. Thus it can be hard to understand a system with many inheritance layers. On the other hand, a large DIT value indicates that many methods might be reused. A deeper class hierarchy indicates that the more methods was used or inherited through which this making more complex to predict the behavior of the class and the deeper tree indicates that there is high complexity in the design because all of the facts contained more methods and class are involved. A deep hierarchy of the class may indicates a possibility of the reusing an inherited methods.

(iii)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[14,15]. These points are come out as NOC is used to measure that “How many subclasses are going to inherit the methods of the parent class”. The greater the number of children, the greater the potential for reuse, since inheritance is a form of reuse. The greater the number of children, the greater the likelihood of improper abstraction of the parent

class. The number of children also gave an idea of the potential influence for the class which may be design.

(iv)Coupling between Objects (CBO): CBO is used to count the number of the class to which the specific class is coupled. The rich coupling decrease the modularity of the class making it less attractive for reusing the class and more high coupled class is more sensitive to change in other part of the design through which the maintenance is so much difficult in the coupling of classes. The coupling Between Object Classes (CBO) metric is defined as “CBO for a class is a count of the number of non-inheritance related couples with classes”. It claimed that the unit of “class” used in this metric is difficult to justify, and suggested different forms of class coupling: inheritance, abstract data type and message passing which are available in object-oriented programming.

(v)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. Larger value also complicated the testing and debugging of the object through which, it requires the tester to have more knowledge of the functionality. The larger RFC value takes more complex is class is a worst case scenario-value for RFC also helps the estimating the time needed for time needed for testing the class.

(vi)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. The disjoint methods have no common instance variables in the methods, while the similar methods have at least one common instance variable. It is used to measuring the pairs of methods within a class using the same instance variable. Since cohesiveness within a class increases encapsulation it is desirable and due to lack of cohesion may imply that the class is split in to more than two or more sub classes. Low cohesion in methods increase the complexity, when it increases the error proneness during the development is so increasing.

E. Extended Metrics For Object-Oriented Software Engineering EMOOSE :

W.Li et al. [9] proposed this metrics of the MOOSE model. They may be described as-

(i)Message Pass Coupling (MPC): It means that the number of message that can be sent by the class operations.

(ii)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.

(iii)Number of Methods (NOM): It is used to count the number of operations that are local to the class i.e. only those class operation which can give the number of methods to measure it.

(iv)Size1:- It is used to find the number of line of code.

(v)Size2:-It is used to count the number of local attributes & the number of operation defined in the class.

F. Metrics For Object-Oriented Design (MOOD):
 F.B. Abreu et al. [37] defined MOOD (Metrics for Object-Oriented Design) metrics. MOOD refers a structural model of the object oriented paradigm like encapsulation as (MHF, AHF), inheritance (MIF, AIF), polymorphism (POF), and message passing (COF). 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 [38]. In MOOD metrics model, there are two main features are methods and attributes. Attributes are used to represent the status of object in the system and methods are used to maintained or modifying several kinds of status of the objects [5].

Metrics are defined as:

(i)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 under consideration. The invisibility of a method is the percentage of the total classes from which this method is not visible.

(ii)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.

(iii)Method Inheritance Factor (MIF): MIF is defined as the ratio of the sum of the inherited methods in all classes of the system under consideration to the total number of available methods (locally defined plus inherited) for all classes.

(iv)Attribute Inheritance Factor (AIF): AIF is defined as the ratio of the sum of inherited attributes in all classes of the system under consideration to the

total number of available attributes (locally defined plus inherited) for all classes.

(v)Polymorphism Factor (PF): PF is defined as the ratio of the actual number of possible different polymorphic situation .

MIF & AIF are used to measure the inheritance of the class & also provide the similarity into the classes. CF is used to measure the coupling between the classes. the coupling are of two types static & dynamic coupling, due to which is increase the complexity of the class & reduce the encapsulation & potential reuse that provide better maintainability. Software developers for the object-oriented system always avoid the high coupling factor. Polymorphism potential of the class are used to measure the polymorphism in the particular class & also arise from inheritance

G. Goal Question Metrics (GQM):V. L. Basili [18] developed GQM approach. This approach was originally defined for evaluating defects for a set of projects in the NASA Goddard Space Flight Center environment. 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:

- (i) List major goals of the development or maintenance project.
- (ii) Derive from each goal the questions that must be answered to determine if the goals are being met.
- (ii) Decide what must be measured in order to be able to answer the questions adequately.

Goal (Conceptual level): A goal is defined for an object, for a variety of reasons, with respect to various models of quality, from various points of view, relative to a particular environment. Objects of measurement are products, processes and resources.

Question (Operational level): A set of questions is used to characterize the way the assessment/achievement of a specific goal is going to be performed based on some characterizing model.

Metric (Quantitative level): A set of data is associated with every question in order to answer it in a quantitative way. This data can be objectives and subjective, if they depend only on the objects that can be measured and not on the viewport from which they may be taken. For example, number of versions of a document, staff hours spent on a task, size of a program.

The GQM approach define some goals, refine those goals into a set of questions, and the questions are further refined into metrics. Consider the following figure, for a particular question; G1 and G2 are two goals, Q2 in common for both of these goals. Metric M2 is required by all three questions. The main idea of GQM is that each metric identified is placed within a context, so metric M1 is collected in order to answer question Q1 to help achieve the goal G1.

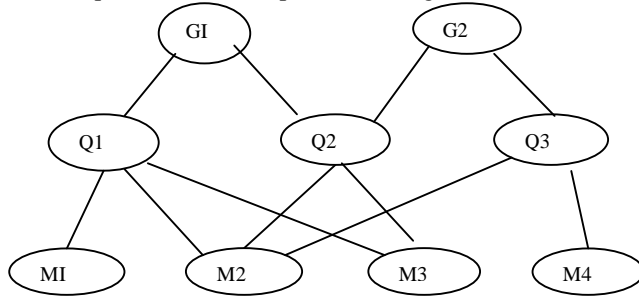


Fig-2
Goal Question Metrics Hierarchy

H. Quality Model for Object-Oriented Design (QMOOD):

The QMOOD [25] is a comprehensive quality model that establishes a clearly defined and empirically validated model to assess object-oriented design quality attributes such as understandability and reusability, and relates it through mathematical formulas, with structural object-oriented design properties such as encapsulation and coupling. The QMOOD model consists of six equations that establish relationship between six object-oriented design quality attributes (reusability, flexibility, understandability, functionality, extendibility, and effectiveness) and eleven design properties.

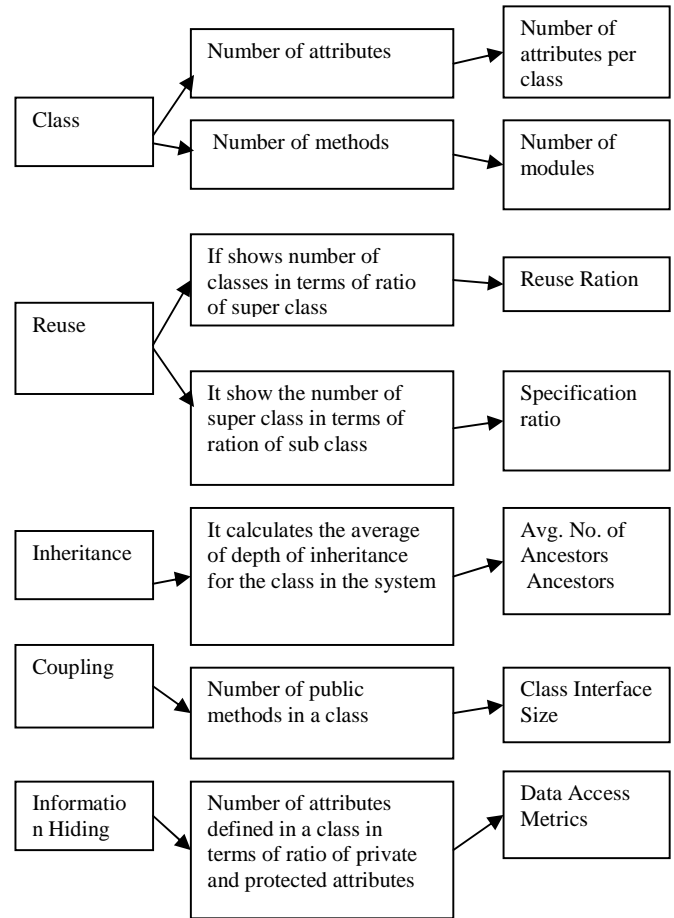


Fig-3
QMOOD Metrics [25]

The whole description for QMOOD can be get from the Bansiya's thesis through which, The QMOOD metrics can further classified into two measures are:

System Measures: System measures describe such metrics are DSC (Design Size in Metrics), NOH (Number of Hierarchies), NIC (Number of Independent classes), NSI (Number of Single Inheritance), NMI (Number of multiple Inheritance), NNC (Number of Internal Classes), NAC (Number of Abstract Classes), NLC (Number of Leaf Classes), ADI (Average Depth of Inheritance), AWI (Average Width of Classes), ANA (Average Number of Ancestors).

Class Measures: Class measure metrics are those metrics which can define some metrics are MFM (Measure of Functional Modularity), MFA (Measure of Functional Abstraction), MAA (Measure of Attribute Abstraction), MAT (Measure of Abstraction), MOA (Measure of Aggregation), MOS

(Measure of Association), MRM (Modeled Relationship Measure), DAM (Data Access Metrics), OAM (Operation Access Metrics), MAM (Member Access Metrics), DOI (Depth of Inheritance), NOC (Number of Children), NOA (Number of Ancestor), NOM (Number of Methods), CIS (Class Interface Size), NOI (Number of Inline Method), NOP (Number of Polymorphic Method), NOO (Number of Overloaded Operators), NPT (Number of Unique Parameter Types), NPM (Number of Parameter per Method), NOA (Number of Attributes), NAD (Number of Abstract Data Types), NRA (Number of Reference Attributes), NPA (Number of Public Attributes), CSB (Class Size in Bytes), CSM (Class Size in Metrics), CAM (Cohesion Among Methods of class), DCC (Direct Class Coupling), MCC (Maximum Class Coupling), DAC (Direct Attribute based Coupling), MAC (Maximum Attribute based Coupling), DPC (Directed Parameter based Coupling), MPC (Maximum Parameter based Coupling), VOM (Virtual ability Of Method), CEC (Class Entropy Complexity), CCN (Class Complexity based on Data), CCP (Class Complexity based on method Parameter), CCM (Class Complexity based on Members).

I. LI W. METRICS

Li et al. [16] proposed six metrics are 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).

(i)Number of Ancestor Classes (NAC): The Number of Ancestor classes (NAC) metric proposed as an alternative to the DIT metric measures the total number of ancestor classes from which a class inherits in the class inheritance hierarchy. The theoretical basis and viewpoints both are same as the DIT metric. In this the unit for the NAC metric is “class”, justified that because the attribute that the NAC metric captures is the number of other classes’ environments from which the class inherits.

(ii)Number of Local Methods (NLM): The Number of Local Methods metric (NLM) is defined as the number of the local methods defined in a class which are accessible outside the class. It measures the attributes of a class that WMC metric intends to capture. The theoretical basis and viewpoints are different from the WMC metric. The theoretical basis describes the attribute of a class that the NLM metric captures. This attribute is for the usage of the class in an object-oriented design because it indicates the size

of a class’s local interface through which other classes can use the class. They stated three viewpoints for NLM metric as following:

- 1) The NLM metric is directly linked to a programmer’s effort when a class is reused in an Object-Oriented design. More the local methods in a class, the more effort is required to comprehend the class behavior.
- 2) The larger the local interface of a class, the more effort is needed to design, implement, test, and maintain the class.
- 3) The larger the local interface of a class, the more influence the class has on its descendent classes.

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

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

(v)Coupling through Abstract Data Type (CTA): The Coupling through Abstract Data Type (CTA) is defined as the total number of classes that are used as abstract data types in the data-attribute declaration of a class. Two classes are coupled when one class uses the other class as an abstract data type [16]. The theoretical view was that the CTA metric relates to the notion of class coupling through the use of abstract data types. This metric gives the scope of how many other classes’ services a class needs in order to provide its own service to others.

(vi)Coupling through Message Passing (CTM): The Coupling through Message Passing (CTM) defined as the number of different messages sent out from a class to other classes excluding the messages sent to the objects created as local objects in the local methods of the class. Two classes can be coupled because one class sends a message to an object of another class, without involving the two classes

through inheritance or abstract data type [Li., 98]. Theoretical view given was that the CTM metric relates to the notion of message passing in object-oriented programming. The metric gives an indication of how many methods of other classes are needed to fulfill the class' own functionality.

J.SATC's Metrics

Rosenberg Linda [48] proposed to select object oriented metrics that supports the goal of measuring the code, quality, result and they proposed many object-oriented metrics due to lack of theoretical basis and that can be validated. These metrics may be used to evaluate the object oriented concepts like methods, coupling and inheritance and mostly focus on both of the internal and external efficiency measures of the psychological complexity factors that affect the ability of the programmer. It proposed three traditional metrics and six new metrics for the object-oriented system metrics-

Traditional Metrics

(i)Cyclomatic Complexity (CC): Cyclomatic Complexity is used to measure the complexity of an algorithm in a method of class. Cyclomatic Complexity of methods can be combined with other methods to measure the complexity of the class. Generally, this is only used for the evaluation of quality attribute complexity.

(ii)Line of Code: It is a method used to evaluate the ease of understandability of the code by the developer and the maintainer. It can easily be counted by the counting the number of lines for the code and so on. Generally, used to measure the reusability and maintainability.

New Object Oriented Metrics

The six new object oriented metrics are may be discussed as:

(i)Weight Method per Class (WMC): It is used to count the methods implemented within a class. The number of methods and complexities involved as predictors, how many time and effort is required to develop and maintain the class.

(ii)Response for a Class (RFC): It is used to the combination of the complexity of a class through the number of methods and the communication of methods with other classes. This is used to evaluate the understandability and testability.

(iii)Lack of Cohesion of Method (LCOM): 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. It also measures the degree of similarity of methods by data inputs variables and attributes. Generally, it is used to evaluate the efficiency and reusability.

(iv)Depth of Inheritance Tree (DIT): Inheritance is a relationship between the class that enables the programmer to use previously defined object including the operators and variables. It also helps to find out the inheritance depth of the tree from current node to the ancestor node. It is used to evaluate the reusability, efficiency, understandability and testability.

(v)Number of Children (NOC): This is used to measure the subclass subordinate to a class in the hierarchy. Greater the number of children means greater reusability and inheritance i.e. in the form of reuse. Generally, it is used to measure efficiency, testability and reusability.

SATC focused on some selected criteria for the object oriented metrics as:

- (i) Efficiency of constructor design to decrease architecture complexity.
- (ii) Specification of design and enhancement in testing structure
- (iii) Increase capacity of psychological complexity.

Source Construct	Metrics	Object-Oriented Structure
Traditional metrics	Cyclomatic complexity (CC)	Methods
	Line of Codes	Methods
	Comment percentage (COM)	Methods
New object oriented metrics	Weight method per class (WMC)	Methods / Class
	Response for a class (RFC)	Class / Message
	Lack of cohesion of methods (LCOM)	Class / Cohesion
	Coupling between Object (CBO)	Coupling
	Depth of Inheritance Tree (DIT)	Inheritance
	Number of children (NOC)	Inheritance

5. Conclusion and future Works

This manuscript contributes to an increased understanding of the state of the software metrics. This can also provide some software quality metrics and the object-oriented metrics, which can define that “how to measure the functionality of the software and How we can improve their characteristics. A mechanism is provided for comparing all the object oriented software metrics which define all the methods, attributes are used in software engineering environment. The increase in software development means the measurement was also so high. The increasing significance being placed on software measurement which has to lead to an increase in the amount of research on developing the new software measures. In this paper, we have presented all of the software metrics for object oriented development. They provided a basis for measuring all of the characteristics like size, complexity, performance and quality. In reliance of some notions the quality may be increased by adding some features like abstraction, polymorphism and inheritance which are inherent in object orientation. This paper provides some help for researchers and practitioners for better understanding and selection of software metrics for their purposes.

References

- [1] C. Neelamegam, M. Punithavali, “A survey on object oriented quality metrics”, Global journal of computer science and technologies, pp 183-186, 2011.
- [2] A. Deepak, K. Pooja, T. Alpika, S. Sharma, “Software quality estimation through object oriented design metrics”, IJCSMS International journal of computer science and network security, april 2011, pp 100-104.
- [3] B. Henderson, seller, “object oriented metrics: measure of complexity”, Prentice Hall, 1996.
- [4] A. Shaik, C.P.K. Reddy, B. Manda, prakashine, K. Deepti, “Metrics for object oriented design software system:A Survey”, Journal of emerging trend in engineer and applied science (JETEAS), pp 190-198, 2010
- [5] N. Fenton et al, “Software metrics: a rigorous and practical approach”, International Thomson computer press 1996.
- [6] L.C.Briand, J.Wuest, J.Daly and V. Porter, “Exploring the Relationships Between Design Measures and Software Quality In Object Oriented Systems”, Journal of Systems and Software, 51, 2000.
- [7] L.C. Briand, W.L. Melo and J.Wust, “Assessing the Applicability of Fault Proneness Models Across Object Oriented Software Projects”, IEEE transactions on Software Engineering. Vol. 28, No. 7, 2002.
- [8] P.Coad and E.Yourdon, “Object Oriented Analysis”, Yourdon Press, 1990.
- [9] W. Li, Sallie, Henry “Metrics for Object-Oriented system”, Transactions on Software Engineering, 1995.
- [10] L.H. Rosenberg and L.Hyatt, “Applying and interpreting object oriented metrics”, Proceedings of software technology conference, utah, April 1998.
- [11] C. Shyam, Kemerer, F. Chris, "A Metrics Suite for Object- Oriented Design" M.I.T. Sloan School of Management, pp. 53-315, 1993
- [12] R. Harrison, Samaraweera, L.G. Dobie and Lewis, P.H: Comparing Programming Paradigms: An Evaluation of Functional and Object-Oriented Programs, Software Eng. J., vol. 11, pp. 247-254, July 1996.
- [13] M. Alshayeb and Li.W., "An empirical validation of object-oriented metrics in two different iteration software processes", IEEE transaction on Software Engineering, Vol-29, no-.11, Nov 2003.
- [14] C. Shyam and C. F. Kemerer, “Towards a Metrics Suite for Object Oriented Design”, Proceeding on Object Oriented Programming Systems, Languages and Applications Conference (OOPSLA'91), ACM, Vol. 26, Issue 11, Nov 1991, pp. 197-211.
- [15] C. Shyam and C. F. Kemerer, “A Metrics Suite for Object Oriented Design”, IEEE Transactions on Software Engineering, Vol. 20, No. 6, June 1994, pp. 476-493.
- [16] Li W., “Another Metric Suite for Object-oriented Programming”, The Journal of System and Software, Vol. 44, Issue 2, December 1998, pp. 155-162.
- [17] C. Jones, “Estimating Software Costs: Bringing Realism to Estimating”, 2nd Edition, Mc Graw Hill, New York, 2007.
- [18] V.L.Basili, L. Briand and W. L. Melo, “A validation of object-oriented Metrics as Quality Indicators”, IEEE Transaction Software Engineering. Vol. 22, No. 10, 1996, pp. 751-761.
- [19] M. Lorenz, J. Kidd, “Object Oriented Software Metrics”, Prentice Hall, NJ, (1994).
- [20] R. D. Neal, “The Measurement Theory Validation of Proposed Object-Oriented Software Metrics”, Dissertation, Virginia Commonwealth University, (1996).
- [21] J. V. Gurf, J. Bosch, “Design, Implementation and Evolution of Object-Oriented Frameworks: Concepts and Guidelines”, Software- Practice and Experience, 31, (3), (2001), 277–300.

- [22] H. Fujiwara, S. Kusumoto, K. Inoue, A. Suzuki, T. Ootsubo, K. Yuura, "Case Studies To Evaluate a Domain Specific Application Framework Based on Complexity and Functionality Metrics", *Information and Software Technology*, 45, (1), (Jan 2003), 43–49.
- [23] L. H. Etzkorn, W. E. Hughes, C. G. Davis, "Automated Reusability Quality Analysis of OO Legacy Software", *Information and Software Technology*, 43, (5), (2001), 295–308.
- [24] G. Manduchi, C. Taliercio, "Measuring Software Evolution at a Nuclear Fusion Experiment Site: A Test Case for Applicability of OO and Reuse Metrics in Software Characterization", 44, (10), (2002), 593–600.
- [25] J. Bansiya, C. G. Davis, "A Hierarchical Model for Object-Oriented Design Quality Assessment", *IEEE Transactions on Software Engineering*, 28, (1), (2002), 4–17.
- [26] G. Subramanian, W. Corbin, "An Empirical Study of Certain Object-Oriented Software Metrics", *Journal of Systems and Software*, 59, (1), (2001), 57–63.
- [27] K. Morris, "Metrics for Object-oriented Software Development Environments," Masters Thesis, MIT, 1989..
- [28] Booch, G: *Object-Oriented Analysis and Design with Applications*, 2nd ed., Benjamin Cummings, 1994.
- [29] R. S. Pressman: —*Software Engineering*, Fifth edition, ISBN 0077096770.
- [30] Chen J.Y Lum: "A New Metrics for Object-Oriented Design." *Information of Software Technology* 35, 4(April 1993):232-240.
- [31] Abreu, B. Fernando: "The MOOD Metrics Set," *Proc. ECOOP'95 Workshop on Metrics*, 1995.
- [32] Alexander et al 2003, "Mathematical Assessment of Object-Oriented Design Quality.", *IEEE Transactions on Software Engineering*, VOL. 29, NO. 11, November 2003.
- [33] H.Lilu, K.Zhou and S.Yang: "Quality metrics of OOD for Software development and Re-development", *First Asia-Pacific Conference on Quality Software*, August 2002.
- [34] M.Subramanyam and R.Krishnan: "Emphirical Analysis of CK metrics for OOD complexity: Implication for software defect", *IEEE transaction on software engineering*, 2003.
- [35] R.Harrison, S.J.Counsell and R.V.Nithi: "An evaluation of the MOOD set of OOSM", *IEEE Transaction on Software Engineering*, vol.24 no.6, pp.491-496, June 1998. JürgenWüst, "SD METRICS TOOL", in *der Lache* 17, 67308
- [36] J.Eder, G.Kappel and M.Schreft, "Coupling and Cohesion in ObjectOriented Systems", *Technical Report University of Klagenfurt*, 1994.
- [37] B. F. Abreu: "Design metrics for OO software system", *ECOOP'95, Quantitative Methods Workshop*, 1995.
- [38] Abreu, B. Fernando, Rita, Miguel, G.: "The Design of Eiffel Program: Quantitative Evaluation Using the MOOD metrics", *Proceeding of TOOLS'96 USA*, Santa Barbara, California, July 1996.
- [39] A. Kaur, S. Singh and K. S. Kahlon, *Evaluation and Metrication of Object Oriented System*", *Proceedings of the International Multi Conference of Engineers and Computer Scientists 2009, I IMECS 2009*, March 18 - 20, 2009, Hong Kong.
- [40] F.Brito, E. Abreu and W. Melo, 1996, "Evaluating the Impact of Object-Oriented Design on Software Quality", *3rd Int'l S/W Metrics Symposium*, March 1996, Berlin, Germany
- [41] S.R.Chidamber and C.F.Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering* , pages 476 – 493, June 1994.
- [42] Dr B.R. Sastry, M.V. Vijaya Saradhi, "Impact of software metrics on Object Oriented Software Development life cycle", *International Journal of Engineering Science and Technology*, Vol 2 (2), pg 67-76, 2010.
- [43] R. Subramanyam and M.S. Krishnan, "Empirical Analysis of CK metricsfor Object Oriented Design Complexity: Implications of Software defects" *IEEE transactions on Software Engineering*, vol 29, no- 4, 2003.
- [44] Dr K. M. Breesam, "Metrics for Object Oriented design focusing on class Inheritance metrics", *2nd International conference on dependability of computer system IEEE*, 2007.
- [45] C. Neelamegan, Dr. M. Punithavalli, "A Survey-Object Oriented quality metrics", *Global journal of Computer Sc. And Technology*, Vol 9, no 4, 2009.
- [46] A. Goldberg, Robson, D., "Smalltalk-80: the language and its implementation", Reading, MA: Addison Wesley, 1983.
- [47] M.A. Eliss & B. Stroustrup, "The annotated C++ Reference manual", Reading MA: Addison-Wesley, 1990.
- [48] Rosenberg Linda, "Software Quality Metrics for Object Oriented System Environments", *A report of SATC's research on OO metrics*.
- [49] F. Brito, E. Abreu (INESC/ISEG), Miguel Goulao, Rita Esteves (INESC/IST), "Towards the Design Quality Evaluation of Object Oriented

- Software System”, Revised version: originally published in Proceedings of 5th International Conference of Software Quality, Austin, Texas, 23 to 26 October, 1993.
- [50] A. L. Baroni, “Formal Definition of Object Oriented Design Metrics”, MS Thesis, Vrije Universities Brussels-belgium, 2002.
- [51] Abreu, Brito F. and Carapuca, Rogerio, “Candidate Metrics for Object –Oriented Software within a Taxonomy Framework”, Proceedings of AQUIS’93 (Achieving Quality In Software), Venice, Italy, October 1993: selected for reprint in the Journals of Systems and Software, Vol. 23 (I), pp. 87-96, July 1994.
- [52] S. Chidamber and C. Kemerer, “A metrics Suite for Object Oriented Design”, IEEE Transaction on Software Engineering, June 1994, pp 476-492.
- [53] J. Xu, H. Danny, L. Fernando Capretz, An Empirical Validation of Object-Oriented Design Metrics for Fault Prediction,. Journal of Computer Science, Vol: 4, No: 7, pp. 571-577, 2008.
- [54] Y. Zhou, H. Leung, “Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults”, IEEE transaction on software engineering, vol. 32, no.10, pp-771-789,2006.
- [55] R.S.Pressman, “Software Engineering-A practitioners Approach” Fourth Edition, Mc. Graw Hill International Edition 1997
- [56] E.V.Berrad, “Metrics for Object-Oriented Software engineering”, The Object Agency.
- [57] V.L &C.C., Manager, SATC “Principal Components of orthogonal Object-oriented metrics (323-08-14)”, white paper analysing the result of NASA object-oriented data, May 29, 2003.
- [58] Fairy Richard, “Software engineering Concepts”, Tata Mc Graw Hill, 2003.