Volume 6, Issue 8, August 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Review of Software Quality Metrics for Object-Oriented Design

Sanjeev Kumar Punia
(Ph.D Scholar) Department of Computer Science & Engineering, NIMS University, Jaipur, Rajasthan, India

Dr. Parveen Kumar
(Research Guide) Department of Computer Science & Engineering, NIMS University, Jaipur, Rajasthan, India

Dr. Anuj Gupta
(Research Co-guide) Accurate Institute of Management & Technology, Greater Noida, Uttar Pradesh, India

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 estimate the software cost and predicates on various kinds of deliverable items. These tools are used to measure the estimations of lines of codes, functions points and object points. This paper highlight the classification of different metrics like software quality metrics, object oriented metrics, size metrics, CK metrics, QMOOD metrics, GQM metrics, MOOSE metrics and EMOOSE metrics proposed from early 90's. It also maintains comparison tables that analyze the difference between all object oriented metrics easily.***

Keywords: object oriented system, class attributes

I. INTRODUCTION

Object oriented design is more beneficial in software development environment and it is an essential feature to measure software quality over the environment. Object oriented design contained all the software properties and quality related to small or large project.

The different attributes and characteristics of an object system are held by its degree. Object oriented design is a classification approach that is used to classify the problem in terms of object and it may provide many paybacks on reliability, adaptability, reusability and decomposition to easily understand the objects.

Software metrics makes it possible for software engineer to measure and predict the necessary software resource for a project. Metrics provide insight that is necessary to create and design model through the test. It also provide a quantative way to access the quality of internal attributes of the product hence it enables the software engineer to access quality before building the product. Metrics are the crucial source of information through which a software developer takes a decision to 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.

Generally, software metrics are characterized 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). Most of the software quality metrics must possess the features of object oriented design to design a good metrics. The quality features of these metrics are compliance with the ability to cover all the design characteristics.

Orthogonality: It contains all the ability to represent different aspects of the system under measurement.
Formality: It contains all the ability to get the same value for same system for different peoples.
Minimality: It contains all the ability to use the minimum number of metrics.
Implementability/usuability: It contains all the implementation technology i.e. independent ability to perform the task.
Accuracy: it contains all the quantative measurements to measure the magnitude of errors.
Validity: It refers to the degree through which it accurately reflects or assumes the specific concept used by researchers.
Reliability: This is the portability of free software operations failure for specified periods.

II. LITERATURE SURVEY

Abreu et al. provides a new classification framework for the TAPROOT. This framework was defined by two independent vectors i.e. category and granularity. Six categories of object oriented metrics are defined as design metrics, complexity metrics, size metrics, quality metrics, productivity metrics and reusability metrics. They also proposed three levels of granularity i.e. software level, class level and method level without providing the empirical/theoretical metrics base.

M. Alshayeb et al. 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. In short cycled agile process the

prediction of design efforts and source lines of code may be added, modified and deleted by object oriented metrics where as in long cycled framework process the same features may not be successfully predicted. It shows that design and implementation may change during development iterations and can predicted by object oriented metrics while it is not true in case of long term development of an established system.

R. D. Neal et al. study 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. They defined a model based on validation measurement theory and proposed ten new metrics as potential methods inheritance (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 (PIM) and strings of message links (SML).

R. Harrison et al. suggested a statistical model which is obtained from the logistic regression for identifying threshold values for the Chidamber and kemerer metrics. The process is empirically authenticated on eclipse project. They concluded that the chidamber and kemerer metrics have threshold effects at different risk levels. On later releases, the authentication of these thresholds is shown through the aid of decision trees.

L. H. Ethzkorn shows that the chosen threshold values were more precise than chosen through their intuitive perspectives or 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. Rosenberg Linda perceive that the software quality also play an important role in the safety and financial aspects.

M. Subramanyam et al. proposed some metrics suites and concluded that designs metrics are very important to know the design aspects of the software and to enhance the quality of software for the developers. Harrison et al. discussed six properties 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. find by comparing the object oriented function points with other predictors of lines of code (LOC) that, the object oriented function points can be extended to a considerable amount with the aid of a bigger data set. Linear models are the independent variable to measure the conventional object oriented entity or an object oriented function using a cross validation approach.

C. Shyam et al. suggests some software metrics through which we can calculate the quality of modularization of an object oriented software. On one side they provide a set of metrics for the large scale object oriented software system with some dependencies and some metrics for characterizing the quality for modularization of APIs. On another side, they provide some object oriented dependencies like inheritance, associates relationship and base class designing.

Y. Zhou et al. considered the fault severity using the machine learning methods with their experimental exploration of fault proneness that predict the capability of object oriented design metrics and all of these predictions with fault severity are taken from the data sets of NASA domain. J. Xu. et al. proposed an object oriented metrics that describes the fault estimation using empirical analysis and use CK metrics to apprise the number of faults in the particular program.

This also includes some neural and fuzzy technique and the result showed that we can get a dependable fault by using CBO, RFC, WMC and SLOC. Dr. B. R. Sastry et al. implement the graphics user interaction by using software metrics and determine the quantity and quality of object oriented software development life cycle.

III. REVIEW OF SOFTWARE QUALITY METRICS

There are different software quality metrics for the software development as

A. Size metrics
B. Complexity metrics
C. Halstead metrics
D. Quality metrics

A. Size metrics:

The size metrics help to quantify the software size. There are three types of software metrics used to measure the software size as

a) Line of code metrics: It is the oldest metrics used to measure the module size but the main issue is to decide the different line of codes to use in the count.

b) Function point metrics: In this the line of code is measured at the availability of the code. Henry et al. proposed function point metrics to measure the software size early in development life cycle. This depends totally on the user input, output inquiries used to measure the program size.

c) Bang: Chidamber et al. defined function metrics as a bang and calculated by certain algorithm and data primitives set of formal software specification measures total functionality.

B. Complexity Metrics:

Complexity metrics described the detailed metrics design to find the complexity. These metrics are described as:

a) Cyclomatic complexity (CC): Brito et al. proposed complexity measurement metrics whose basic goal 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. The cyclomatic complexity can be calculated as:

V (G) = e-n+2
Where
V(G) = Cyclomatic complexity of flow graph G
e = Number of edges in G
n = Number of nodes in G
Another way to calculate the cyclomatic complexity as:

s

V(G) = P+1
Where,
P = Number of predicate nodes that used to represent a boolean statements in the code.

b) Extended cyclomatic complexity (ECC): Brito et al. measures the program complexity but fails to differentiate complexity in single conditional statement. H Fijiwara suggest that the extended cyclomatic complexity can be defined as:
ECC = eV(G) = Pe+1
Where,
Pe = Number of predicate weighted nodes by number of compound statements in flow graph G.

c) Information flow: Kafura and Henry proposed that a program structure metrics can be used to calculate program complexity. The metrics is calculated by the number of local information flows input (fan-in) and flows output (fan-out) counting. The complexity is counted as:
= [procedure length]*[(fan-in)*(fan-out)]2

C. Halstead metrics:

Halstead proposed the software science theory to find out the overall software production effort. Halsted metrics should contain the program vocabulary (n), length (N) and volume (V) defined as:

a) Program vocabulary (n): The programs are visualized as the set of tokens known as language operators and operands. Halstead define vocabulary (n) as:
n = n1+n2
Where,
n1 = number of unique operators in the first program
n2 = number of unique operands in the second program

b) Program length (N): The program length N is the count of the total number of operators and operands. That may be represented as:
N = N1+N2
Where,
N1 = number of operators in the first program
N2 = number of operands in the second program

c) Program volume (V): It is the measurement of storage volume that is required to present the program. That may be represented as:
V = NLog2n

D. Quality metrics:

The different types of quality metrics are describes as:

a) Defect metrics: Defect metrics do not have an effective procedure to count the defects in the program as number of design change, number of intended errors, 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: The quality of internal product is usually measured by the number of bugs in the software and by the duration of software metrics crash.

c) Maintainability index: Dr. Paul W. Oman defined a number of functions to predict software maintainability. As the measurement of maintainability index can be done as:
MI = 171-5.2*In(ave V)-0.23*ave V(g)-16.2*In(ave LOC)
Where,
Ave V = Average halstead volume per module
Ave V(g) = Average extended cyclomatic complexity per module.
Ave LOC = Average line of code per module.

IV. REVIEW OF OBJECT-ORIENTED METRICS

There are some object-oriented metrics for the object-oriented software development as follows:

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. 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:

a) CCM (Class Coupling Metric),
b) OXM (Operating Complexity Metric),
c) OACM (Operating Argument Complexity Metric),
d) ACM (Attribute Complexity Metric),
e) OCM (Operating Coupling Metric),
f) CM (Cohesion Metric),
g) CHM (Class Hierarchy of Method) and
h) RM (Reuse Metric).

Metrics (a) and (c) are very subjective in nature, Metrics (d) and metric (g) mostly involve the count of features; and metric (h) 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. 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 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 (NP).

D. 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-

a) 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.

b) 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.

c) 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. 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 as inheritance is a form of reuse.

The number of children will be greater if the likelihood of improper abstraction of the parent class is greater. The number of children also gave an idea of the potential influence for the class which may be design.

d) 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.

e) 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 in estimating the time needed for time needed for testing the class.

f) 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. proposed this metrics of the MOOSE model. They may be described as-

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

b) 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.

c) 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.

d) Size1: It is used to find the number of line of code.

e) 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. 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. 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.

Metrics are defined as:

a) 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.

b) 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.

c) 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.

d) 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.

e) 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 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:

a) List major goals of the development or maintenance project.
b) Derive from each goal the questions that must be answered to determine if the goals are being met.
c) 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.

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.

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:

a) 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).

b) 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. 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).

a) 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.

b) 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:

i) 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.
ii) The larger the local interface of a class, the more effort is needed to design, implement, test, and maintain the class.
iii) The larger the local interface of a class, the more influence the class has on its descendent classes.

c) 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.

d) 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.

e) 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. 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.

f) 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 typ. 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 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.

a) 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.

b) 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. It is generally, used to measure the reusability and maintainability.

New Object Oriented Metrics

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

a) 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.

b) 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.

c) 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, ii is used to evaluate the efficiency and reusability.

d) 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, understandability and testability.

e) 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:

a) Efficiency of constructor design to decrease architecture complexity.
b) Specification of design and enhancement in testing structure
c) Increase capacity of psychological complexity.

V. 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 is software development means the measurement was also so high. The increasing significance being placed software measurement which has to lead and increase 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 rely of some notions the quality may be increased by added 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, 2016
[2] A. Deepak, K. Pooja, S. Sharma "Software quality estimation through object oriented design metrics" IJCSNS International journal of computer science and network security, 2014
[3] B. Henderson "object oriented metrices: measure of complexity", Prentice Hall, 2012
[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, 2015
[5] N. Fenton "Software metrices: a rigorous and practical approach" International Thomson computer press 2009
[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, pp. 51, 2010
[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. 2014
[8] P.Coad and E.Yourdon "Object Oriented Analysis" Yourdon Press, 2012
[9] W. Li, Sallie, Henry "Metrics for Object-Orient ed system" Transactions on Software Engineering, 2005
[10] L.H. Rosenberg and L.Hyatt "Applying and interpreting object oriented metrics" Proceedings of software technology conference, utah, 2015
[11] C. Shyam, Kemerer, F. Chris "A Metrics Suite for Object- Oriented Design" M.I.T. Sloan School of Management, pp. 53-315, 2013
[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, 2006
[13] M. Alshayeb "An empirical validation of object-oriented metrics in two different iteration software processes" IEEE transactionod Software Engineering, Vol-29, no-.11, Nov 2013
[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 ACM, Vol. 26, Issue 11, Nov 2011
[15] C. Shyam and C. F. Kemerer "A Metrics Suite for Object Oriented Design" IEEE Transactions on Software Engineering, Vol. 20, No. 6, June 2014
[16] Li W. "Another Metric Suite for Object-oriented Programming" The Journal of System and Software, Vol. 44, Issue 2, December 2008
[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 "Avalidation of object-oriented Metrics as Quality Indicators" IEEE Transaction Software Engineering. Vol. 22, No. 10, 2006
[19] M. Lorenz, J. Kidd "Object Oriented Software Metrics" Prentice Hall, NJ, 2015
[20] R. D. Neal "The Measurement Theory Validation of Proposed Object-Oriented Software Metrics" Dissertation, Virginia Commonwealth University, 2008
[21] J. V. Gurp, J. Bosch "Design, Implementation and Evolution of Object-Oriented Frameworks: Concepts and Guidelines" Software Practice and Experience, 2013
[22] "Study of Certain Object-Oriented Software Metrics" Journal of Systems and Software, 2012
[23] K. Morris "Metrics for Object-oriented Software Development Environments" Masters Thesis, MIT, 2014
[24] Booch "G: Object-Oriented Analysis and Design with Applications" 2nd ed., Benjamin Cummings, 2008
[25] R. S. Pressman: Software Engineering, Fifth edition, ISBN 007709, 2014
[26] Chen J.Y Lum: "A New Metrics for Object-Oriented Design" Information of Software Technology 35, 2011
[27] Abreu, B. Fernando: "The MOOD Metrics Set" Proc. ECOOP'95 Workshop on Metrics, 2009
[28] Alexander et al "Mathematical Assessment of Object-Oriented Design Quality" IEEE Transactions on Software

Engineering, 2013
[29] H.Lilu, K.Zhou and S.Yang: "Quality metrics of OOD for Software development and Redevelopment" First Asia-Pacific Conference on Quality Software, August 2012
[30] M. Subramanyam and R. Krishnan: "Emphirical Analysis of CK metrics for OOD complexity: Implication for software defect" IEEE transaction on software engineering, 2014
[31] R.Harrison, S.J.Counsell and R.V.Nithi: "An evaluation of the MOOD set of OOSM", IEEE Transaction on Software Engineering" vol. no.6, pp.491-496, June 2009
[32] 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, 2014
[33] L. H. Etzkorn, W. E. Hughes, C. G. Davis "Automated Reusability Quality Analysis of OO Legacy Software" Information and Software Technology, 2014
[34] 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" 2012
[35] J. Bansiya, C. G. Davis "A Hierarchical Model for Object-Oriented Design Quality Assessment" IEEE Transactions on Software Engineering, 2014
[36] J.Eder, G.Kappel and M.Schreft "Coupling and Cohesion in Object Oriented Systems" Technical Report University of Klagenfurt, 2012
[37] B. F. Abreu: "Design metrics for OO software system" Quantitative Methods Workshop, 2015

