A COMPREHENSIVE REVIEW AND ANALYSIS ON OBJECT-ORIENTED SOFTWARE METRICS IN SOFTWARE MEASUREMENT

K.P. Srinivasan
Associate Professor in Computer Science
C.B.M. College, Kovaipudur, Coimbatore – 641 042, India
kpsrinivasanmail@gmail.com

Dr. T. Devi
Head, Department of Computer Applications
School of Computer Science and Engineering
Bharathiar University, Coimbatore – 641 046, India
tdevi5@gmail.com

Abstract— The software development is dynamic and is always undergoing major changes. Today a huge number of tools and methodologies are available for software development and software development refers to all activities that go into producing information system solution. System development activities consist of system analysis, modeling, design, implementation, testing and maintenance and further the state of software metrics in software development during the last decade is encouraging and many researchers are involved in the field of software metrics. The software metrics are being applied and good results are obtained with criticisms. The use of software metrics has proved the process efficiency and product effectiveness. In software engineering, recently, software metrics researchers have introduced new metrics and validated software metrics using empirical and theoretical techniques and software metrics have been used in decisions-making as well as in various process activities and more researchers are involved in empirical studies. The eminent researchers guide the software professionals for evaluating software product effectiveness using software metrics. There are many kinds of software metrics available from traditional metrics to latest computer science field of web science, i.e., web-related metrics in software engineering. In order to propose an object-oriented metrics in software engineering, a thorough understanding of the previous object-oriented metrics is essential in software measurement. A better understanding of existing metrics would lead to clear ideation and developments of concepts to solve the problems of ambiguity in object-oriented metrics. This paper analyzes and reviews the most referred object-oriented metrics in software measurement.

Keywords - Result Based Software Metrics (RBSM); Object-Oriented Metrics; Software Metrics; Software Quality Assessment; Software Quality Attributes; Software Measurement.

I. INTRODUCTION

The software metrics is a consistent topic and research in software engineering [36, 37, 46, 48, 72, 74, 93, 94, 97, 98]. The role of software metrics is to find significant improvement in software products and directs management to take managerial and technical decisions [93, 94]. According to Jones, C. (2014) [46], "In order to solve the problems of software and convert a manual and expensive craft into a modern engineering profession with a high degree of manufacturing automation, the software industry needs much better metrics and measurement disciplines and much more in the way of standard reusable components. Better measures and better metrics are the stepping stones to software engineering excellence" [46]. Today, the software metrics is unfinished, and currently gives the appearance of being more influenced by "metrics validations" and "object-oriented design metrics". The current state of software metrics is still not matured based on standards, new metrics and "it is identified that software metrics research faced more difficulties towards proving usefulness in industry, theoretical validity, empirical validity, defining precise metrics, understanding, methodology of execution, execution time is more to find the metrics values, metrics are executed only by experts, and accuracy on results" [93, 94]. At present, many researchers are involved in research on process and product metrics research [2]. They are also involved in proposing metrics for software process and product measurement [74, 93, 94]. Some researchers are involved in research studies finding usefulness and applications in software environments using software metrics [17, 37, 45, 51, 73, 98]. Few researchers are involved in developing metrics tools for different environments and applied metrics tools for different applications [76]. The main milestones and events of software metrics show that in the past many metrics had been proposed and validated by eminent researchers but most of the metrics lacked in experimental study and few metrics were accepted and used. Although there are many metrics in use and under active investigation, a few metrics are more difficult to apply and execute. As a result, the battle on software metrics is still continuing. The following section discusses the "Comprehensive Metrics Suite" proposed by Srinivasan, K.P., and Devi, T. Section III analyses the most referred object-oriented design metrics proposed by Chidamber, S.R., and Kemerer, C.F. Section IV analyses the most referred object-oriented design metrics called MOOD metrics proposed by F. Brito e Abreu and Section V discusses another important Design metrics proposed by Lorenz, M. and Kidd, J. Section VI discusses the reviews of object-oriented metrics and conclusion includes future directions of the research.

II. THE COMPREHENSIVE SOFTWARE METRICS SUITE FOR OBJECT-ORIENTED DESIGN

Recently, Srinivasan and Devi proposed a set of six result based (RBSM) object-oriented design metrics suite called "Comprehensive Metrics" for measuring object-oriented design attributes [94]. Further, they also introduced a new kind of software metrics for software coding measurement in software engineering called "Program Keyword Metrics (PKM)" [93]. This PKM metrics eliminates the ambiguity criticism of most referred "Halstead Metrics" and "Lines Of Coding (LOC) metrics". And further they eliminated the main criticism of "accuracy on results" in software measurement by "Keyword Metrics (KM)" in Software Engineering [93].

The result based set of object-oriented design metrics [94] and their definitions are shown in Table I and the procedure based metrics system for execution of the object-oriented design metrics is shown in Figure 1.

Table I. A Comprehensive Metrics (RBSM) Suite for Object-Oriented Design

Result Based Software Metric (RBSM)	Definition
MPCF=NPM/ NPM + NNPM (Methods-Per-Class Factor)	MPCF is defined as the ratio of the Number of Public Methods (NPM) to the Number of Public Methods (NPM) and Number of Non Public Methods (NNPM) in the class. Method Per Class Factor excludes inherited methods.
APCF= NPA/NPA+NNPA (Attributes-Per-Class Factor)	APCF is defined as the ratio of the Number of Private (Protected) Attributes (NPA) to the Number of Private Attributes (NPA) and Number of Non Private Attributes (NNPA) in the class. Attribute-Per-Class Factor excludes inherited attributes.
MIF= NIM/NIM+NDM (Method Inheritance Factor)	MIF is defined as the ratio of the Number of Inherited Methods (NIM) to Number of Inherited Methods (NIM) and the Number of Defined Methods (NDM) in the class.
AIF = NIA/NIA+NDA (Attributes Inheritance Factor)	AIF is defined as the ratio of the Number of Inherited Attributes (NIA) to the Number of Inherited Attributes (NIA) and the Number of Defined Attributes (NDA) in the class.
CF=NAC/NPC (Coupling Factor)	NAC is the Number of Actual Couplings with other classes. NPC is the Number of Possible Couplings of this class with other classes of the system. Inheritance is excluded in determining the couplings.
LCF=NDMP/NPMP (Lack-of-Cohesion Factor)	NDMP is the Number of Dissimilar Method Pairs in the class and NPMP is the Number of Possible Method Pairs in the class. If two methods access one or more common attributes, then these two methods are similar. And if two methods have no commonly accessed attribute then these two methods are dissimilar.

This method is a "straightforward method" for execution of object-oriented design metrics called "procedure based metrics system" [92, 93, 94] and it adopts the quality attributes of the object-oriented design which are measured using design properties relationships. This procedure based approach finds the quality effectiveness of the design and yields a value to each of the attributes.

Figure 1 shows the algorithm for measuring the quality of the object-oriented design in software engineering [94]. The set of design quality attributes measured by the proposed RBSM suite of metrics are functionality, understandability, effectiveness, flexibility, extendibility and reusability. This procedure identifies the design properties of an object-oriented design using the metrics encapsulation, inheritance, coupling, cohesion and complexity and form the design metrics-quality attribute relationship as shown in Table II. It gives consolidation of different steps and it shows the relationships of design properties to design attributes, the range values, desired values and the metrics to quality of design attributes [94].

The Table III shows the computation formula and weighted formula for quality attributes of design derived from the quality attributes [94] as given in step 6 of Figure 1. This procedure based system closely examines the quality of the design based on design properties to design attributes, computed formula and modifies the individual classes if necessary and identifies if any class has unacceptable desired values.

Step 1: Select the object-oriented design to measure the quality assessment.
Step 2: Select the quality attributes of the object-oriented design to measure the particular domain.
Step 3: Identify the design properties of an object-oriented design for identifying and related to the quality attributes selected in step 2.
Step 4: Form the related object-oriented design metrics to design properties and desired values for quantify the design properties of step 3 and find design metrics-design attributes relationship.
Step 5: Calculate the metric values of each metrics and tabulate their values for each classes of the entire system for easy operations and find quality assessment of the design.
Step 6: Find the design attribute effectiveness using the obtained metric values from step 5 and further using computation formula. And check the design attributes using the desired values and design properties weights.
Step 7: Closely examine the design attributes from computation formula and metrics values, modify and improve the individual classes of the entire object-oriented system.

Figure 1. Procedure Based Metrics System for Comprehensive Metrics Suite [94]

Table II. Metrics, Ranges, Desired Values, Properties and Attributes for Design

Design Metric	Range	Desired Value	Design Property	Quality Attributes
MPCF	**0 to 1**	**1**	**Complexity**	**Functionality, Reusability**
APCF	**0 to 1**	**1**	**Encapsulation**	**Understanding, Effectiveness, Flexibility**
MIF	**0 to 1**	**0**	**Inheritance, Abstraction**	**Effectiveness, Extendibility**
AIF	**0 to 1**	**0**	**Inheritance, Abstraction**	**Effectiveness, Extendibility**
CF	**0 to 1**	**0**	**Coupling**	**Functionality, Understanding, Flexibility, Reusability**
LCF	**0 to 1**	**0**	**Cohesion**	**Understanding, Reusability**

Table III. Computation Formulae and its Range and Desired Value for Quality Attributes

Quality Attributes	Computation Formula	Range	Desired Value
Functionality	**(MPCF+(1-CF))/2**	**0 to 1**	**1**
Understandability	**(APCF+(1-CF)+LCF)/3**	**0 to 1**	**1**
Effectiveness	**(APCF+(1-MIF)+(1-AIF)/3**	**0 to 1**	**1**
Flexibility	**(APCF+(1-CF))/2**	**0 to 1**	**1**
Extendibility	**((1-MIF)+(1-AIF))/2**	**0 to 1**	**1**
Reusability	**(MPCF+(1-CF)+LCF)/3**	**0 to 1**	**1**

The comprehensive object-oriented design metrics suite has the following advantages and achievements over previous metrics available in literature: ①The complete metrics set having the range and desired values for measuring the design. ②The range value for each metric is between 0 and 1 hence result oriented.③The computation formula values between 0 and 1 hence result oriented. ④ The procedure followed is simple, clear, understandable, unambiguous and consistent. ⑤ The procedural approach (Straightforward Method) is given for execution of software metrics in easy manner.

III. THE CHIDAMBER AND KEMERER METRICS FOR OBJECT-ORIENTED DESIGN METRICS

The metrics suite for object-oriented design proposed by Chidamber, S.R., and Kemerer, C.F. [24, 72, 90] is one of the most widely used and refereed class level object-oriented design metrics [86, 94]. The Chidamber and Kemerer metrics are often popularly referred to in literature as “C-K metrics” [90]. The C-K metrics suite invoked great enthusiasm among researchers and software engineers, and an enormous amount of empirical studies have been conducted to evaluate those metrics. A set of six metrics suite for object-oriented design proposed by Chidamber and Kemerer is shown in Figure 2. This section analyses the six class-level object-oriented design metrics and their validations criteria used by them.

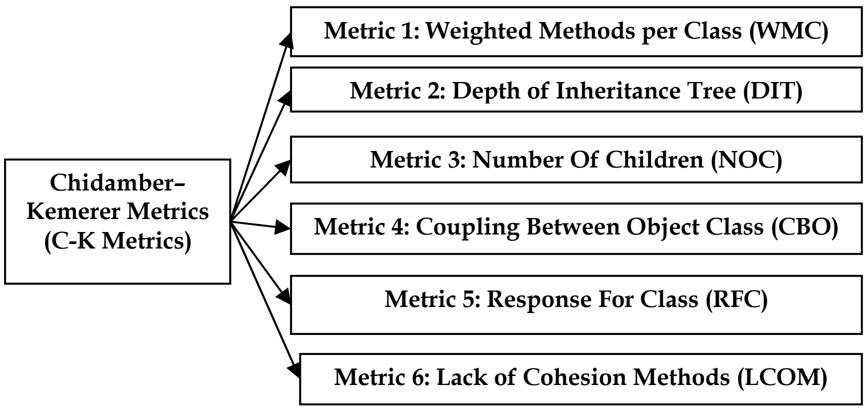
Chidamber–Kemerer Metrics (C-K Metrics)
Metric 1: Weighted Methods per Class (WMC)
Metric 2: Depth of Inheritance Tree (DIT)
Metric 3: Number Of Children (NOC)
Metric 4: Coupling Between Object Class (CBO)
Metric 5: Response For Class (RFC)
Metric 6: Lack of Cohesion Methods (LCOM)

Figure 2. Chidamber - Kemerer (C–K) Metrics Suite

Chidamber and Kemerer adopted Weyuker's properties of measures for validating their metrics [90, 100]. The Weyuker's second property and eighth property are automatically satisfied for all object-oriented design metrics. Weyuker's seventh property "Permutation" is only for traditional metrics and therefore seventh property is not considered for design metrics by Chidamber and Kemerer [24]. The remaining six properties are considered for metrics evaluation criteria for the C-K metrics suite. The properties are changed according to significant object-oriented design metrics. The six class-level metrics are analyzed and empirical studies had been conducted by various researchers. These six metrics are called WMC, DIT, NOC, CBO, RFC, and LCOM and are analysed and empirically evaluated in the different projects.

C-K Metric 1: Weighted Methods per Class (WMC)

Definition: Consider a Class C_1, with methods M_1, $M_2 \ldots M_n$ that are defined in the class. Let $c_1 \ldots\ldots c_n$ be the complexity of methods, then the weighted methods per class (WMC) is formally defined as:

$$WMC = \sum_{i=1}^{i=n} Ci$$

All the methods of WMC complexity are considered to be unity, then WMC = n, the number of methods are weighted methods per class. For the sake of simplicity, C_i is assumed to be unity for all methods of WMC [25]. The C_i is a measure of the complexity of the method, such as the cyclomatic complexity [72]. In order to find WMC metric, there is no need to count indirect methods available through ancestors, friend, or inherited methods, as they are defined outside the class under consideration [26]. According to Briand, L.C., et al. the WMC metric comes under the size metric, which does not satisfy their properties of complexity measures [15]. WMC is not a complexity metric, particularly when C_i is assumed to be unity. The weighted least square model shows that WMC has a positive relationship with number of defects and indicates that increase in the number of methods is going to increase the number of defects [94]. According to validation of WMC, WMC metric satisfies the Weyuker's properties 1,2,3,4 and 5. WMC metrics do not satisfy the property 6 of Weyuker's properties of measures. This metrics give the total methods in the class as the result of WMC metric.

C-K Metric 2: Depth of Inheritance Tree (DIT)

Definition: Depth of inheritance of the class is the DIT metric for the class. In cases involving multiple inheritances, the DIT will be the maximum length from the node to the root of the tree.

The DIT metric is a measure of number of ancestors' classes potentially affecting this class [58]. There are many important research study reports that say that there is almost no DIT inheritance in large projects [11, 25]. This apparent lack of use of inheritance illustrations depicts how care must be taken to actively manage projects in such a way as to achieve benefits aimed directly by object-oriented designers. The DIT metric will reflect the maintainability and complexity of the project as well as cost of the software [25]. In projects, when DIT is large, the probability of fault detection occurrence is more [11]. The DIT metric satisfies the Weyuker's properties [100] 1, 2,3,4,5 and the property 6 is not satisfied by DIT metric.

C-K Metric 3: Number of Children (NOC)

Definition: NOC = Number of immediate subclasses subordinated to a class in the class hierarchy.

The NOC metric is a measure of number of subclasses that are going to inherit the methods of the parent class. According to Singh, P., et al. when the NOC metric value is high, it may require more testing [89]. Surprisingly, large NOC classes are less fault-prone when compared to classes with large DIT. The research report says that there are no children in many large projects [25, 29]. The reusability increases with increase in NOC [13]. The NOC metric satisfies the properties 1,2,3,4, and 5 of Weyuker's properties and the property 6 is not satisfied by NOC.

C-K Metric 4: Coupling Between Object classes (CBO)

Definition: Coupling Between Object (CBO) for a class is a count of the number of other classes to which it is coupled.

The CBO metric relates to an object which is coupled with another object if one of them acts on the other, that is, methods of one use methods or instance variable of another. The couplings are many in number between any two classes which are treated as single coupling according to CBO. The excessive coupling between object classes is preventing its reuse and testing of various parts of a design is complex. The high levels of CBO metric values show managerially poor results and higher rework effort is needed. The significance of CBO is that since classes that have a large number of interconnections with other classes are likely to require greater understanding, it can be hypothesized that they will take more time to develop, test and modify [40, 94] and consequently the productivity levels for such classes may be lower [25].

C-K Metric 5: Response For a Class (RFC)

Definition: RFC = | RS | where RS is the response set for the class.

The response set of a class is a set of methods that can potentially be executed in response to a message received by an object of that class. RFC specifically includes methods called from outside the class and it is a measure of potential communication between the class and other classes. If the RFC value is high in projects then, testing and debugging of class become more complicated. The larger the RFC, the more is the probability of fault occurrence [56]. The RFC metric satisfies Weyuker's properties 1,2,3,4 and 5 and the property 6 is not satisfied.

C-K Metric 6: Lack of Cohesion in Methods (LCOM)

Definition: Consider a class C_1 with *n* methods M_1, M_2… M_n. Let $\{ I_j \}$ = set of instance variables used by methods M_i . There are *n* such sets $\{I_1\}$... {In}. Let $P = \{ (I_i, I_j) \mid I_i \cap I_j = 0 \}$ and $Q = \{ (I_i, I_j) \mid I_i \cap I_j \neq 0 \}$. If all n sets $\{I_1\}$… $\{I_n\}$ are 0 then let P = 0.

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

LCOM metric is the count of the number of method pairs whose similarity is 0 minus the method pairs whose similarity is not zero. The LCOM metric is approximately equal to the count of the number of method pairs that do not have common instance variable minus the count of method pairs that have common in the variable.

The cohesiveness is desirable if it is more in the class. LCOM is intimately tied to the instance variable and methods of a class. Cohesion measure exhibits some anomalies with respect to intuitive understanding of the attributes [14, 43]. The cohesion measure measures how well the lines of source code within a module work together to provide a specific piece of functionality. In object-oriented programming, the degree to which methods that implement a single function are described as having high cohesion [19]. High value of LCOM implies that classes should properly be split into two or more sub-classes and cohesiveness of methods within classes is desirable [94]. High LCOM value indicates disparities in functionality provided by the class [39]. High LCOM means higher rework effort will be needed [69]. Metrics that violate class cohesion properties are not well defined and the relatedness of class members is questionable [27, 28, 53]. A class with a large number of common parameter types in its methods is more cohesive than a class with less number of common parameter types in its methods [16, 54]. LCOM metric satisfies the Weyuker's properties of 1, 2, 3 and 5. The properties 4 and 6 are not satisfied by LCOM. The exploratory analysis of C-K metrics were conducted by Chidamber et al. The empirical study was conducted for productivity, rework effort, and design effort on three commercial systems. The empirical study results show that the metrics provided significant explanatory power for variations in the economic variables [25]. Further, Subramanyam, R.S., and Krishnan, M.S. conducted an empirical analysis of C-K metrics for design complexity and implications of software defects. They provided empirical evidence for supporting the role of design complexity metrics of Chidamber and Kemerer metrics for determining software defects. Their empirical study has been conducted for two popular programming languages C++ and Java. They found that the effects of C-K metrics on defects vary across the two programming languages [94]. Shatnawi, R. investigated the acceptable risk levels of object-oriented metrics and concluded that the classes that exceed a threshold value can be selected for more testing to improve their internal quality. He assessed the effectiveness of threshold values for the object-oriented metrics. He identified threshold values for the Chidamber and Kemerer metrics and his empirical results indicate that the C-K metrics have threshold effects at various risk levels [87]. The object-oriented design metrics proposed by Chidamber and Kemerer is approximately used in most empirical studies in software metrics field [81]. Salem, A.M., and Qureshi, A.A. conducted a test on the complexity and cohesion of software. They proved inconsistencies of C-

K metrics and found out that C-K cohesion metric does not distinguish between two classes which have different cohesiveness [77]. C-K metrics is one of the oldest and most reliable metrics among all available software metrics to evaluate object-oriented design. Kumari, R., and Jaspreet. studied Chidamber Kemerer metrics and identified few flaws of C-K metrics. They demonstrated the flaws of C-K metrics and as well as refinement of these metrics [61]. The following section describes another most referred object-oriented design metrics called Metrics for Object-Oriented Design metrics (MOOD).

IV. METRICS FOR OBJECT-ORIENTED DESIGN (MOOD) METRICS SUITE

The Metrics for Object-Oriented Design (MOOD) metrics was proposed by F. Brito e Abreu [1, 91]. These MOOD metrics set has been evaluated by Harrison, R., et al. [42] using the properties of measures proposed by eminent researchers Kitchenham, B., et al [57]. The MOOD metrics set is used to measure encapsulation, inheritance, coupling and polymorphism of the system is shown in Figure 3. These metrics are called MHF, AHF, MIF, AIF, CF, and PF. These six metrics set could be of use to projects as the metrics operate at system level, providing an overall assessment of a system [72, 91]. This section discusses the MOOD metrics [1, 91] and also explains validation criteria of Kitchenham's properties [57] of measure.

MOOD Metric 1: Method Hiding Factor (MHF)

The Method Hiding Factor (MHF) and Attribute Hiding Factor (AHF) metrics were proposed jointly as measures of encapsulation. The encapsulation is to be related to compilation facilities and MHF and AHF metrics could be used for direct and indirect measures of encapsulation. In the MOOD metric set, both MHF and AHF use code visibility to measure information hiding.

The Method Hiding Factor (MHF) metric is a measure of encapsulation and is formally defined as:

$$MHF = \frac{\sum_{i=1}^{TC}\sum_{m=1}^{M_d(C_i)}(1-V(M_{mi}))}{\sum_{i=1}^{TC} M_d(C_i)}$$

Where $M_d(C_i)$ is the number of methods declared in a class C_i and TC is the total number of classes and as a result,

$$V(Mmi) = \frac{\sum_{j=1}^{TC} is\_visible(M_{mi}C_j)}{TC-1}$$

$$is\_visible(M_{mi}C_j) = \{^{1 \; iff j \neq i \wedge C_j \text{may call } M_{mi}}_{0 \qquad \text{otherwise}}$$

Therefore, for all classes, C_1, C_2….C_n, a method counts 0 if it can be used by another class and 1 if it cannot. The total for the system is divided by the total number of methods defined in the system, to give the percentage of hidden methods in the system. If the value of MHF is high, then it means all methods are private which indicates very little functionality. Thus it is not possible to reuse methods with high MHF. The MHF with low value indicates that all the methods are public that implies most of the methods are unprotected [80].

MOOD Metric 2: Attribute Hiding Factor (AHF)

The Attribute Hiding Factor (AHF) metric is to measure the attribute hiding and is formally defined as:

$$AHF = \frac{\sum_{i=1}^{TC}\sum_{m=1}^{A_d(C_i)}((1-V(A_{ai}))}{\sum_{i=1}^{TC} A_d(C_i)}$$

Where $M_d(C_i)$ is the number of attributes declared in a class C_i and TC is the total number of classes and thus,

$$V(A_{ai)} = \frac{\sum_{j=1}^{TC} is\_visible(A_{ai}C_j)}{TC-1}$$

$$is\_visible(A_{ai}C_j) = \{^{1 \; iff j \neq i \wedge C_j \text{may call } A_{ai}}_{0 \qquad \text{otherwise}}$$

Therefore for all classes, C_1, C_2….C_n, an attribute counts 0 if it can be used by another class and 1 if it cannot. The definitions of MHF and AHF cause discontinuities for systems with only one class. The Method Hiding Factor and Attribute Hiding Factor metrics are used for measures of encapsulation. Assuming that these metrics discontinuity is taken into account, MHF and AHF meet three of four criteria for direct measures as proposed by Kitchenham et al. [57]. Kitchenham's properties [57] 1, 2 and 4 are satisfied by MHF and AHF metrics. The property 3 states that 'each unit of an attribute contributing to a valid metric is equivalent'. This says that all the methods and the attributes are equivalent, as far as information hiding is concerned. However, MHF and AHF are intended to measure the relative amount of information hiding and not the quality of information hiding design decisions and therefore property 3 is also satisfied. The criteria for indirect metrics are also satisfied by these two metrics. In the object-oriented system, if the value of AHF is high, it means all attributes are private which indicates very little functionality. Thus it is not possible to reuse attributes with high AHF. The low value of AHF metrics indicates that all attributes are public, that means most of the attributes are unprotected [80].

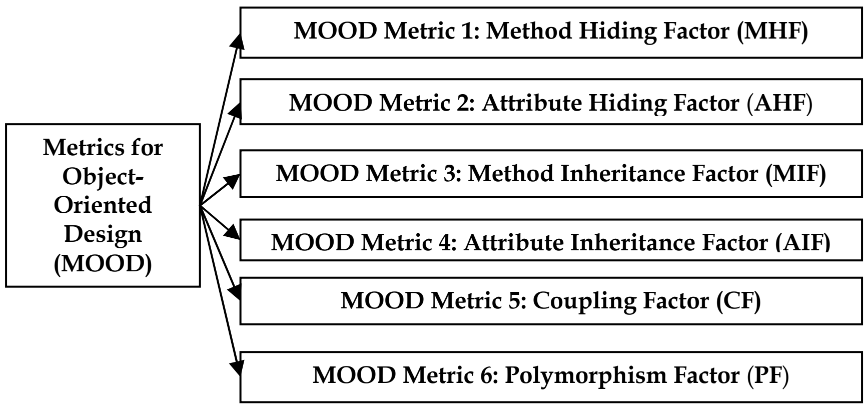
Metrics for Object-Oriented Design (MOOD)
MOOD Metric 1: Method Hiding Factor (MHF)
MOOD Metric 2: Attribute Hiding Factor (AHF)
MOOD Metric 3: Method Inheritance Factor (MIF)
MOOD Metric 4: Attribute Inheritance Factor (AIF)
MOOD Metric 5: Coupling Factor (CF)
MOOD Metric 6: Polymorphism Factor (PF)

Figure 3. Metrics for Object-Oriented Design (MOOD) Metrics

MOOD Metric 3: Method Inheritance Factor (MIF)

The Method Inheritance Factor (MIF) [1, 72] is a measure of inheritance properties and formally defined as:

$$MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)}$$

Where, $M_a(C_i) = M_d(C_i) + M_i(C_i)$, TC is the total number of class, Ma (C_i) is the number of methods that can be involved in association with Ci. $M_d(C_i)$ is the number of methods declared in the class, and $M_i(C_i)$ is the number of methods inherited in the class. In the MIF metric, for each class C_1, C_2…..C_n, , a method counts as 0 if it has not been inherited and 1 if it has been inherited.

MOOD Metric 4: Attribute Inheritance Factor (AIF)

The Attribute Inheritance Factor (AIF) is a measure of inheritance properties metric and it is formally defined as:

$$AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)}$$

Where $A_a(C_i) = A_d(C_i) + A_i(C_i)$ and TC is total number of classes, $A_a(C_i)$ is the number of attributes that can be involved in association with Ci, $A_d(C_i)$ is the number of attributes declared in the class and $A_i(C_i)$ is the number of attributes inherited in the class. In AIF metric, for each class C_1, C_2…C_n,, an attribute counts as 0 if it has not been inherited and 1 if it has been inherited. The total AIF for the system is divided by the total number of attributes, including any which have been inherited. The MIF and AIF metrics measure directly the number of inherited methods and attributes respectively as a proportion of the total number of methods or attributes. The MIF and AIF satisfy the properties 1, 2, 3 and 4 of direct measures of validation properties. Thus, MIF and AIF metrics are valid for direct measure of inheritance factor [91].

MOOD Metric 5: Coupling Factor (CF)

The Coupling Factor (CF) metric is a measure of coupling and proposed as a measure of coupling between classes excluding coupling due to inheritance [1, 42]. The coupling factor metric has been defined as:

$$CF = \frac{\sum_{i=1}^{TC}\left[\sum_{j=1}^{TC} is\_client(C_{i,}C_j)\right]}{TC^2 - TC}$$

Where, $$is\_visible(C_c C_s) = \begin{cases} 1 & iff C_c \Rightarrow C_s \wedge C_c \neq C_s \\ 0 & \text{otherwise} \end{cases}$$

And the C_c => C_s represents the relationship between a client class C_c and a supplier class C_s. The coupling factor is calculated by considering all possible pairwise sets of classes, and whether the classes in the pair are related, either by message passing or by semantic association links. These relationships are considered to be equivalent as far as coupling is concerned. The coupling factor metric satisfies properties of direct measures. Therefore CF is a direct measure of the size of a relationship between two classes and for all pairwise relationships between classes in a system. A high level interclass coupling will have a high CF value. The coupling between modules is more difficult to understand, change, and correct these modules and thus making the software system more complex. A good software system should exhibit low coupling between its units [55]. There are different programs that could have the same CF value then in which case they could exhibit the same amount of coupling. Thus coupling factor satisfies the valid direct measure of inter-class coupling also.

MOOD Metric 6: Polymorphism Factor (PF)

The Polymorphism Factor (PF) metric is proposed for a measure of polymorphism in object-oriented environment. The PF metric is formally defined as:

$$PF = \frac{\sum_{i=1}^{TC} M_o(C_i)}{\sum_{i=1}^{TC}\left[M_n(C_i) \times DC(C_i)\right]}$$

M_d (C_i) = M_n (C_i) + M_o (C_i) and TC is the total number of classes and M_n (C_i) is the number of new methods and M_o (C_i) is the number of overriding methods and DC (C_i) is the descendants count. In the MOOD metrics, polymorphism factor is the number of methods that redefine inherited methods, divided by the maximum number of possible distinct polymorphic situations. The denominator includes, as a multiplier, the number of descendant classes of a base class. Therefore, the value of PF for a system without any inheritance will always be undefined. Thus, the metric exhibits an unexpected discontinuity, giving an undefined result where a result of 0 would have been expected. Thus, PF is not a valid metric. If the PF metric discontinuity is removed, then it would be theoretically a valid metric. These six object-oriented design metrics refer to a basic structural mechanism of the encapsulation - MHF and AHF metrics, inheritance - MIF and AIF metrics, message-passing- CF metric and polymorphism - PF metric, for system level object-oriented design measurement. Olague, H.M., et al. (2007), conducted an empirical study of MOOD metrics suite. They conducted an empirical research and they also empirically validated MOOD metrics. They have implemented metrics in commercial tools. They compared individual MOOD metrics with the Chidamber and Kemerer metrics also [70]. Sarkar, S., et al. conducted research study on MOOD metric set and they identified attribute hiding factor and method hiding factor metrics are measuring the extent of encapsulation and both are defined as the ratio of the attributes and methods that are visible in a class [78, 79]. Jassim, F., and Altaani, F. conducted the study on MOOD metrics using a statistical approach and they used linear regression model to find the relationship between factors of MOOD metrics and their influences on object-oriented software measurements [44].

V. THE LORENZ AND KIDD OBJECT-ORIENTED METRICS

The Lorenz and Kidd proposed a set of metrics to measure object-oriented systems [64, 65, 72]. This section discusses the Lorenz and Kidd metrics and their significance on object-orientation. Lorenz and Kidd proposed class based metrics into four categories as size, inheritance, internals and externals (Figure 4). The size metrics are proposed for object-oriented class and which count the attributes and operations for individual class and average values for system. The inheritance metrics are proposed for operations, internals metrics are proposed for cohesion and external metrics are proposed for coupling [82]. Lorenz and Kidd metrics are called AC, MC, NOO, NOA, SI, AOS and ANP.

L-K Metric 1: Attribute Count (AC)

The Attribute Count (AC) metric is the total number of attributes in a class. Both inherited attributes and the attributes defined in the class are counted for AC. A large number of attributes indicates that the class has too many properties in it.

L-K Metric 2: Method Count (MC)

The Method Count (MC) metric is the total number of methods in a class. Both inherited methods and the methods defined in the class are counted for MC.

L-K Metric 3: Number of Operation Overridden by a Subclass (NOO)

The Number of Operation Overridden by a subclass (NOO) metric is the total number of operations overridden by a subclass. There are occasions, a subclass replaces an operation inherited from its superclass with a specialized version of its own use and it is called as “overriding”. The large value of NOO indicates that there is a problem in design and this should result in unique new method names. The number of operations overridden by a subclass metric value is high then testing is more in object-oriented software and that can be difficult to modify it. NOO metric of Lorenz and Kidd [64, 65, 72] is similar to polymorphism metric of the MOOD set [1].

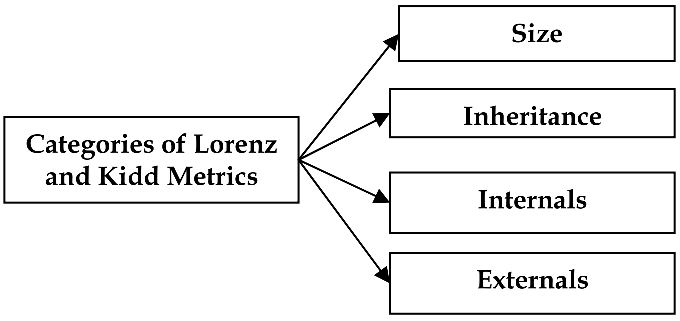

Figure 4. Categories of L-K Metrics

L-K Metric 4: Number of Operations Added by a Subclass (NOA)

The Number of Operations Added by a subclass (NOA) metric is the total number of specialized methods and attributes added by a subclass.

The large values of NOA generally indicate that the subclass drifts away from the abstraction implied by the super class. In general, as the depth of the class hierarchy increases, the value for NOA at lower levels in the hierarchy should go down.

L-K Metric 5: Specialization Index (SI)

The Specialization Index (SI) metric is proposed for indication of degree of specialization of each of subclasses in system. The Specialization Index (SI) metric can be defined as:

$$SI = \frac{[NOO * Level]}{MC}$$

Where NOO is the number of operations overridden by a subclass and Level is the level in the class hierarchy at which the class resides and MC is the total number of methods for the class. This is same as DIT metric of Chidamber and Kemerer metrics [24]. Specialization can be achieved by adding or deleting operations or by overriding. A high value of SI indicates that the class hierarchy has classes that do not conform to the superclass abstraction.

L-K Metric 6: Average Operation Size (AOS)

The Average Operation Size (AOS) metric is the average of the number of messages sent by each method of the class. This metric is quite similar to RFC metric of Chidamber and Kemerer metric [24]. As the number of messages sent by a single operation increases, it is likely that responsibilities have not been well allocated within a class.

L-K Metric 7: Average Number of Parameters per method (ANP)

The Average Number of Parameters per method (ANP) metric is the average of the number of parameters passed to each of the methods of the class. The larger the number of operation parameters, the more complex the collaboration between objects. In general, average number of parameters per operation should be kept as low as possible.

VI. REVIEWS ON OBJECT-ORIENTED METRICS AND SOFTWARE MEASUREMENT IN SOFTWARE ENGINEERING

The improvement of the management process depends upon ability to identify, measure, and control essential parameters of the development process. This is achieved through effective software metrics and the measurement of the essential parameters of software development. The demand for efficient software is increasing day by day and object-oriented design technique is able to fulfill this demand because it is the most powerful mechanism to develop efficient software systems. This section reviews the major object-oriented metrics and their significances. Most of the work in software industry is related to maintenance. The maintainability of a software system is desirable and it is an important characteristic [34, 35]. Kanmani, S., et al. proposed external system characteristics assessment using object-oriented inheritance metrics. They proposed the methodology to measure the C++ source code through inheritance matrix representation [49].

Bansiya, J., and Davis, C.G. proposed a set of metrics for object-oriented design called “Quality Model for Object-Oriented Design (QMOOD)" [10, 94]. This is a hierarchical model for the assessment of high level design quality attributes of object-oriented design which is called as QMOOD. This model evaluates the structural and behavioral design properties of classes, objects and their relationships using a suite of object-oriented design metrics. This hierarchical model relates design properties of encapsulation, modularity, coupling, and cohesion to high-level quality attributes such as flexibility, reusability and complexity. The relationship from design properties to quality attributes are weighted in accordance with their influence and importance. A key attribute of the model is that it can be easily modified to include different relationships and weights. The model is empirically validated on large commercial object-oriented systems.

Aggarwal, K.K., et al. proposed a model for integrated complexity measurement for measuring the software complexity based on lines of code, average variable statement, cyclomatic complexity and degree of control nesting [3]. Chae, H.S., et al. presented an approach for improving the cohesion by considering the characteristics of the dependent instance variables in an object-oriented program. They investigated the effects of dependent instance variables on cohesion metrics for object-oriented programs and they proposed an approach to identifying the dependency relations among instance variables [18]. Aggarwal, K.K., et al. proposed two design metrics for object-oriented software and these metrics are analytically evaluated against Weyuker's properties of measures [4].

Aggarwal, K.K., et al. conducted an investigation on 22 metrics proposed by various researchers and applied these metrics on projects for empirical study [5]. Sarkar, S., et al. proposed a set of metrics that measure the quality of modularization of a non-object-oriented software system. They proposed design principles to capture the notion of modularity and they defined metrics centered on principles. Their metrics characterize the software from a variety of perspectives as structural, architectural, and notions such as the similarity of purpose and commonality of goals. Their metrics are based on information-theoretic principles and tested their metrics on popular open-source systems [79]. Aggarwal, K.K., et al. conducted effect of design metrics on fault proneness in object-oriented systems. They empirically investigated the relationship between object-oriented design metrics and fault-proneness of object-oriented systems [6].

Sarkar, S., et al. proposed 13 metrics for measuring the modularization of large-scale object-oriented software. Their 13 metrics characterise the quality of modularisation with respect to such object-oriented inter-module dependencies [78]. Alghamdi, J.S. presented a scheme for measuring coupling between program components. His scheme makes the measurement of coupling easier by breaking it down into two major steps and provides a systematic procedure for each step [8]. Kaur, K., and Singh, H. validated component based software development on reuse of software components. They have validated object-oriented metrics to measure structural properties of commercial software components [52]. Bawane, N., and Srikrishna, C.V. proposed a metric for software and the process of selecting the metrics that support the goal of measuring design and code quality [12]. Kaur, K., and Singh, H. conducted a study on system behavior for object-oriented systems using metrics. They conducted empirical studies using two object-oriented languages [53]. Object-oriented metrics can play an important role in object-oriented software development. The object-oriented metrics are important in the development of successful software applications [31]. Ma, Y.T., et al. (2010), proposed a hierarchical set of metrics for coupling and cohesion. They conducted empirical study on 12 open-source object-oriented software systems for validating their set. Their experimental results show the correlations between cross-level metrics and they provided more effective information about fault-prone classes in practice [66]. Kumar, S.A., et al. proposed the significance of software metrics to quantify design and code quality and discussed on the needs of development and implementation of metrics [62].

Okike, E. presented a pedagogic evaluation about the Chidamber Kemerer LCOM metric using field data from three industrial systems. They suggested that the LCOM metric measures class cohesiveness and appropriateness in the determination of properly and improperly designed classes [68]. Babu, S., and Parvathi, R.M.S. proposed an approach to the computation of dynamic coupling measures in distributed object-oriented systems. The motivation of measures is to complement existing measures that are based on static analysis by actually measuring coupling at runtime in the hope of obtaining better decision and prediction models [9]. Ahmed, M., and Shoaib, M. proposed design metrics to measure real time environment and the aim of the set of new metrics is to measure the design before handing over to the implementation team [7]. The measurement can distinguish the characteristics of entity from another by analysis and drawing the conclusion that software metrics are used to measure the attributes of an entity. It is accepted that quality of software product is strongly dependent on the quality of its design [83, 88] Yadav, A., and Khan, R.A. proposed coupling metrics for complexity normalization. They proposed a method to improve reliability of object-oriented design by normalizing complexity which is closely correlated with coupling and coupling complexity normalization (CCN) metric is used to minimize complexity of object-oriented design [99].

Chhikara, A., and Chhillar, R.S. proposed an aspect-orientated object-oriented metrics. Aspect-Oriented Paradigm is the emerging paradigms that promise to enhance software design and promotes reuse. Their research studies the object-oriented metrics and how the introduction of aspects affects these metrics [22]. Chhikara, A., et al. conducted the impact of different types of inheritance on the object-oriented software. Their research paper focused on effects of inheritance on object-oriented environment [23]. Gandhi, P., and Bhatia, P.K. proposed two metrics called Message Received Coupling (MRC) and Degree of Coupling (DC) metrics for the automatic detection of design problems along with an algorithm to apply these metrics to redesign an object-oriented source code. They designed a Method Calling Graph for calculating the value of proposed metrics [38]. Sharma, R., and Chhillar, R.S. discussed the merits and demerits of various metrics. They proposed a new system for measuring the goodness of implementation phase. The concept of object-oriented metrics has also been explored [84]. Sharma, A.H., et al. presented a review of the quality metrics suites of CK, MOOD, and LK metrics. They select some metrics and discard other metrics based on the definition and capability of the metrics [86].

Reda, S., et al. presented a methodology for software design quality assessment. Their methodology helps the designer to measure and assess the changes in design due to design enhancements. They illustrated the methodology using practical software design examples and analyzed its utility in industrial projects [75]. Kumar, R., and Gupta, D., proposed heuristics for object-oriented metrics. They proposed heuristics for CK, MOOD, and LK object-oriented metrics [60]. Krishnaiah, R.V., and Prasad, B.S. (2012), studied a suite of metrics for object-oriented design. The metric values have been calculated using a semi-automated tool. They analyzed the resulting values of CK and MOOD metrics and provided significant insight about the object oriented characteristics of the projects [59].

Dubey, S.K., and Rana, A. proposed a fuzzy model to quantify maintainability of object-oriented software system using Chidamber and Kemerer object-oriented metrics. The model takes object-oriented projects and evaluates its maintainability and fuzzy model is validated by using analytical hierarchy processing technique [32]. Dubey, S.K., et al. (2012), reviewed object-oriented metrics and they analyzed the difference between the object-oriented metrics and they studied object-oriented metrics which assures to reduce cost and the maintenance effort by serving as early predictors to estimate software faults [33]. Dash, Y, et al. (2012), studied artificial neural network and they explored the application of evaluate maintainability of the object-oriented software and they studied maintenance effort [30].

Chawla, M.K., and Chhabra, I. (2012), has conducted mapping of program characteristics into five structural complexity metrics and behavior of an information system. They applied and obtained results from three java based sorting programs [21]. Jyothi, V.E., et al. (2012), have studied agile software development refactoring to improve software quality and improve software internal structure without changing its behavior. They proposed an object-oriented software metric tool called “Metric Analyser” and the tool was tested on different codebases [47]. Gupta, A., et al. discussed the most commonly used metrics suite of CK, MOOD and LI on the basis of characteristic they measure. Further, they identified strengths and weaknesses of these metrics and concluded that none of the metrics suite is foolproof. Moreover, there is no single metric that can measure all the aspects of an object-oriented System [41]. Sharma, A.K., et al. reviewed the metrics of CK, MOOD, and LK metrics. They analyzed the metrics and recommended that are useful in evaluation of software quality [85].

Patidar, K., et al. (2013), presented a measurement of the coupling and cohesion between objects that measures the association between numbers of classes, check the direct dependencies, indirect dependencies, I/O dependencies, number of out and in metrics in object-oriented programming [71]. Michura, J., et al. proposed a set of metrics to quantify and measure the attributes. They proposed complexity metrics which are used to determine the difficulty in implementing changes through the measurement of method complexity, method diversity, and complexity density [67]. Lamrani, M., et al. (2013), presented an approach to express software design metrics based on a formal definition of the UML Meta model. They applied their approach to the well known suite of metrics called the CK metrics and MOOD metrics [63]. Chawla, S. (2013), has reviewed the set of MOOD and QMOOD metrics sets and they discussed the usefulness of each metrics [20]. Kaur, A., and Kaur, P.J. (2013), studied class cohesion metrics measured during the design phase to predict software quality. They used cohesion to evaluate class based on the information that is available during design phases [50].

VII. CONCULSION

The design metrics play an important role in helping designer and developers to understand design aspects of software and it improves the software quality and productivity. In general, object-oriented metrics serve many purposes for software engineers and software metrics are used by the project manager, developer, and tester in assuring the quality of the software products. In today’s software development environment, object-oriented design and development is important and there is strong relationship between the object-oriented metrics and the testability efforts in object-oriented system. This paper has analyzed the most referred object-oriented design metrics proposed by Chidamber and Kemerer, MOOD metrics set, and Lorenz and Kidd metrics. This paper also discussed the recently proposed “Comprehensive Metrics” suite for object-oriented design quality assessment and the review of object-oriented metrics proposed by various researchers and their significances are also outlined. The use of existing metrics and development of new metrics will be important factors in future software engineering process and product development [37, 46, 93, 94]. In future, research work will be based on using software metrics in software development for the improvement of the time schedule, cost estimates and quality and can be improved through software metrics.

REFERENCES

[1] Abreu, F.B., and Melo, W., "Evaluating the Impact of Object-Oriented Design on Software Quality," *Proceedings of the 3rd International Software Metrics Symposium*, IEEE, Berlin, Germany, March, 1996.

[2] Alemneh, E., "Current States of Aspect Oriented Programming Metrics," International Journal of Science and Research, Volume 3 Issue 1, January 2014, pp. 142-146.

[3] Aggarwal, K.K., Singh, Y., and Chhabra, J.K., "A Multiple Parameter Software Complexity Measure," *The Journal of Computer Society of India*, Vol. 33, No 1, March 2003, pp. 22-30.

[4] Aggarwal, K.K., Singh, Y., Kaur, A., and Malhotra, R., "Software Design Metrics for Object-Oriented Software," *Journal of Object Technology*, Vol. 6, No. 1, January-February 2006, pp. 121-138.

[5] Aggarwal, K.K., Singh, Y., Kaur, A., and Malhotra, R., "Empirical Study of Object-Oriented Metrics," *Journal of Object Technology*, Vol. 5. No. 8, November-December 2006, pp. 149-173.

[6] Aggarwal, K.K., Singh, Y., Kaur, A., and Malhotra, R., "Investigating effect of Design Metrics on Fault Proneness in Object-Oriented Systems", *Journal of Object Technology*, Vol. 6, No. 10, November-December 2007, pp. 127-141.

[7] Ahmed, M., and Shoaib, M., "Novel Design Metrics to Measure Real Time Environment Application Design," *Journal of American Science*, 7(7), 2011, pp. 222-226.

[8] Alghamdi, T.S., "Measuring Software Coupling," *The Arabian Journal for Science and Engineering*, Vol. 33, No. 1B, April 2008 , pp. 119-129.

[9] Babu, S., and Parvathi, R.M.S., "Design Dynamic Coupling Measurement of Distributed Object Oriented Software Using Trace Events," *Journal of Computer Science*, 7 (5), 2011, pp. 770-778.

[10] Bansiya, J., and Davis, C.G., "Hierarchical Model for Object-Oriented Design Quality Assessment," *IEEE Transactions on Software Engineering*, Vol. 28, No. 1, January 2002, pp. 4-17.

[11] Basili, V.R., Briand, L.C., and Melo, W.L., "A Validation of Object-Oriented Design Metrics as Quality Indicators," *IEEE Transactions on Software Engineering*, Vol. 22, No. 10, October 1996, pp. 751-761.

[12] Bawane, N., and Srikrishna, C.V., "A Survey of Quality Assessment through Object-Oriented Metrics," *CSI Communications*, Vol. 33, No 7, October 2009, pp. 21-25.

[13] Bhatia, P.K., and Mann, R., "An Approach to Measure Software Reusability of OO Design," *Proceedings of 2nd National Conference on Challenges and Opportunities in Information Technology*, RIMT-IET, Mandi Gobindgarh, March 2008, pp. 26-30.

[14] Bieman, J.M., and Ott, L, M., "Measuring Functional Cohesion," *IEEE Transactions on Software Engineering*, Vol.20, No.8, August 1994, pp. 644-657.

[15] Briand, L.C., Morasca, S., Basili, V.R., "Property-Based Software Engineering Measurement," *IEEE Transactions on Software Engineering*, Vol. 22, No.1, January 1996, pp. 68-85.

[16] Chae, H.S., and Bae, D.H., "Improving Cohesion Metrics for Classes by Considering Dependent Instance Variables," *IEEE Transactions on Software Engineering*, Vol. 30, No. 11, November 1998, pp. 826-832.

[17] Chauhan, R., Singh, R., Saraswat, A., Joya, A.H., Gunjan, V.K., "Estimation of Software Quality using Object Oriented Design Metrics," International Journal of Innovative Research in Computer and Communication Engineering, Vol. 2, Issue 1, January 2014, pp. 2581-2586.

[18] Chae, H.S., Kwon, Y.R., and Bae, D.H., "Improving Cohesion Metrics for Classes by Considering Dependent Instance Variables," *IEEE Transactions on Software Engineering*, Vol. 30, No. 11, October 2004, pp. 826-832.

[19] Chandrika, S.M., Babu, E.S., and Srikanth, N., "Conceptual Cohesion of Classes in Object Oriented Systems," *International Journal of Computer Science and Telecommunications*, Volume 2, Issue 4, July 2011.

[20] Chawla, S., "Review of MOOD and QMOOD Metric Sets," *International Journal of Advanced Research in Computer Science and Software Engineering*, Vol. 3, Issue 3, March 2013, pp. 448-451.

[21] Chawla, M.K., and Chhabra, I., "A Multiple Parameter Software Complexity Measure," *International Journal of Engineering Research and Technology*, Vol. 1 Issue 5, July 2012, pp. 1-5.

[22] Chhikara, A., and Chhillar, R.S., "Impact of Aspect Orientation on Object Oriented Software Metrics," *International Journal on Computer Science and Engineering*, Vol. 2 No. 3, June-July 2011, pp. 466-469.

[23] Chhikara, A., Chhillar, R.S., and Khatri, S., "Evaluating the Impact of Different Types of Inheritance on the Object Oriented Software Metrics," International Journal of Enterprise Computing and Business Systems, Vol.1 Issue 2 July 2011.

[24] Chidamber, S.R., and Kemerer, C.F., "A Metrics Suite for Object-Oriented Design," *IEEE Transactions on Software Engineering*, Vol. 20, No. 6, June 1994, pp. 476-493.

[25] Chidamber, S.R., Darcy, D.P., and Kemerer, C.F., "Managerial use of Metrics for Object-Oriented Software: An Exploratory Analysis," *IEEE Transactions on Software Engineering*, Vol.24, No. 8. August 1998, pp. 629 – 639.

[26] Churcher, N.I., and Sheppard, M.J., "Comments on a Metrics Suite for Object – Oriented Design," *IEEE Transactions on Software Engineering*, Vol.21, No.3, March 1995, pp. 263-265.

[27] Dallal, J.A., "Mathematical Validation of Object-Oriented Class Cohesion Metrics," *International Journal of Computers*, Issue 2, Vol. 4, 2010, pp. 45-52.

[28] Dallal, J.A., "Measuring the Discriminative Power of Object-Oriented Class Cohesion Metrics," *IEEE Transactions on Software Engineering*, Vol. 37, No. 6, November-December 2011, pp. 788-804.

[29] Dange, A.S., Joshi, S. D., " Fault Prediction in Object Oriented System Using the Coupling and Cohesion of Classes," *International Journal of Computer Science and Management Studies*, Vol. 11, Issue 02, August 2011, pp. 48-51.

[30] Dash, Y, Dubey, S.A., and Rana, A., "Maintainability Prediction of Object Oriented Software System by Using Artificial Neural Network Approach," *International Journal of Soft Computing and Engineering*, Vol. 2, Issue. 2, May 2012, pp. 420-423.

[31] Dubey, S.K., and Rana, A., "A Comprehensive Assessment of Object-Oriented Software Systems Using Metrics Approach," *International Journal on Computer Science and Engineering*, Vol. 2, No. 8, 2010, pp. 2726-2730.

[32] Dubey, S.K., and Rana, A., "A Fuzzy Approach for Evaluation of Maintainability of Object Oriented Software System," *International Journal of Computer Applications*, Vol. 49, No 21, July 2012, pp. 1-6.

[33] Dubey, S.K., Sharma, A., and Rana, A., "Comparison Study and Review on Object- Oriented Metrics," *Global Journal of Computer Science and Technology*, Vol. 12, Issue 7, April 2012, pp. 38-47.

[34] Emam, K.E., and Melo, W., "The Prediction of Faulty Classes Using Object-Oriented Design Metrics," *Institute for Information Technology*, National Research Council, 1999, Canada.

[35] Emam, K.E., Melo, W., and Machado, J.C., "The Prediction of Faulty Classes Using Object-Oriented Design Metrics," *The Journal of Systems and Software*, 56(2001), 2001, pp. 63-75.

[36] Fenton, N.E., and Pfleeger, S.L., *Software Metrics: A Rigorous and Practical Approach*, Thomson Asia, Singapore, 2004.

[37] Fernando, W.K.S.D., Wijayarathne, D.G.S.M., Fernando, J.S.D., Mendis, M.P.L., and Guruge, I., "The Importance of Software Metrics: Perspective of a Software Development Projects in Sri Lanka," Proceedings of the SAITM Research Symposium on Engineering Advancements, Sri Lanka, April 2014, pp. 91-95.

[38] Gandhi, P., and Bhatia, P.K., "Optimization of Object-Oriented Design using Coupling Metrics," *International Journal of Computer Applications*, Vol. 27, No 10, August 2011, pp. 41-44.

[39] Gelinas, J.F., Badri, M., and Badri, L., "A Cohesion Measure for Aspects," *Journal of Object Technology*, Vol. 5, No. 7, September-October 2006, pp. 97 - 114.

[40] Goyal, G., and Patel, S., "Analysis of Object Oriented Class Inheritance and Interfaces Using Coupling Measures," *International Journal of Engineering and Social Science*, Vol. 2, Issue5, 2012, pp. 93-103.

[41] Gupta, A., Batra, G., and Vijaylaxmi., "Analyzing Theoretical Basis and Inconsistencies of Object Oriented Metrics," *International Journal on Computer Science and Engineering*, Vol. 4, No 5, May 2012, pp. 803- 808.

[42] Harrison, R., Counsel, S.J. and Nithi, R.V., "An Evaluation of the MOOD Set of Object-Oriented Software Metrics," *IEEE Transactions on Software Engineering*, Vol.24, No.6, June 1998, pp.491-496.

[43] Hitz, M., and Montazeri, B., "Chidamber and Kemmerer's Metrics Suite: A Measurement Theory Perspective," *IEEE Transactions on Software Engineering*, Vol.22, No4, April 1996, pp.267-271.

[44] Jassim, F., and Altaani, F., "Statistical Approach for Predicting Factors of MOOD Method for Object Oriented," *International Journal of Computer Science Issues*, Vol. 10, Issue 1, No. 1, January 2013, pp. 589- 593.

[45] Jayalakshmi, N and Satheesh, N., "Software Quality Assessment in Object Based Architecture," International Journal of Computer Science and Mobile Computing, Vol. 3, Issue. 3, March 2014, pg.941 – 946.

[46] Jones, C., "Evaluating Software Metrics and Software Measurement Practices," Version 4, Namcook Analytics, March 2014. (http://Namcookanalytics.com).

[47] Jyothi, V.E., Srikanth, K., and Rao, K.N., 2012, "Effective Implementation of Agile Practices – Object Oriented Metrics Tool to Improve Software Quality," *International Journal of Software Engg. and Applications*, Vol. 3, No 4, pp. 13-24.

[48] Kan, S.H., *Metrics and Models in Software Quality Engineering*, Pearson Education, India, 2006.

[49] Kanmani, S., Sankaranarayanan, V., and Thambidurai, P.,"External System Characteristics Assessment Using Object-Oriented Inheritance Metrics," *The Journal of Computer Society of India*, Vol. 32, No 2, June 2002, pp. 5-12.

[50] Kaur, A., and Kaur, P.J.,"Class Cohesion Metrics in Object Oriented Systems," *International Journal of Software and Web Sci.*, 3(2), Dec 2013, pp. 78-82.

[51] Kapila, H., and Singh, S., "Bayesian Inference to Predict Smelly classes Probability in Open Source Software," International Journal of Current Engineering and Technology, Vol.4, No.3, June 2014, pp. 1724-1728.

[52] Kaur, K., and Singh, H., "Metrics to Evaluate Object-Oriented Software Components" *CSI Communications*, Vol. 31, No 11, February 2008, pp. 23-26.

[53] Kaur, K., and Singh, H., "Exploring Design Level Class Cohesion Metrics," *Journal of Software Engineering and Applications*, Vol. 3, 2010, pp. 384-390.

[54] Kaur, K., and Singh, H., "An Investigation of Design Level Class Cohesion Metrics," *International Arab Journal of Information Technology*, Vol. 9, No. 1, 2012, pp. 66-73.

[55] Kaur, M., Batra, P., and Khare, A., "Static Analysis and Run-Time Coupling Metrics," International Journal of Information Technology and Knowledge Management, Vol. 3, No. 2, July-December 2010, pp. 707-710.

[56] Kaur, P.J., Verma, A., and Thapar, S., "Software Quality Metrics for Object-Oriented Environments," *Proceedings of National Conference on Challenges and Opportunities in Information Technology*, RIMT-IET, Mandi Gobindgarh, 2007, pp. 13-16.

[57] Kitchenham, B., Pfleeger, S.L., and Fenton, N., "Towards a Framework for Software Measurement Validation," *IEEE Transactions on Software Engineering*, Vol. 21, No.12, December 1995, pp. 929-943.

[58] Koh, T.W., Selmat, M.H., Ghani, A.A.A., and Abdullah, R., "Review of Complexity Metrics for Object Oriented Software Products," *International Journal of Computer Science and Network Security*, Vol.8, No.11, November 2008, pp. 314-320.

[59] Krishnaiah, R.V., and Prasad, B.S., "Analysis of Object Oriented Metrics," *International Journal of Computational Engineering Research*, Vol. 2, Issue 5, September 2012, pp. 1474 – 1479.

[60] Kumar, R., and Gupta, D., "Heuristics Based on Object Oriented (OO) Metrics," *International Journal of Emerging Technology and Advanced Engineering*, Vol. 2, Issue 5, May 2012, pp. 393-395.

[61] Kumari, R and Jaspreet., "Implementation of Hybrid Metrics to Evaluate Software Performance Encapsulating CK Metrics," *International Journal of Engineering Research and Technology*, Vol. 2, Issue 3, March 2013, pp. 1-4.

[62] Kumar, S.A., Kumar, T.A, and Swarnalatha, P., "Significance of Software Metrics to Quantify Design and Code Quality," *International Journal of Computer Applications*, Vol. 11, No.9, December 2010, pp. 36-42.

[63] Lamrani, M., Amrani, Y.E., and Ettouhami, A., "A Formal Definition of Metrics for Object Oriented Design: MOOD Metrics," *Journal of Theoretical and Applied Information Technology*, Vol. 49, No 1, March 2013, pp. 1-10.

[64] Lorenz, M., 1993, *Object-Oriented Software Development*, A practical guide, PTR prentice Hall, Englewood Cliffs, New Jersey, 1993.

[65] Lorenz, M., and Kidd, J., *Object-Oriented Software Metrics*, Prentice Hall, Englewood Cliffs, New Jersey.

[66] Ma, Y., He, K., Li, B., Liu, J., and Zhou, X., "A Hybrid Set of Complexity Metrics for Large-Scale Object-Oriented Software Systems," *Journal of Computer Science and Technology*, 25(6): November 2010, pp. 1184–1201.

[67] Michura, J., Capretz, M.A.M., and Wang, S., "Extension of Object-Oriented Metrics Suite for Software Maintenance," *ISRN Software Engineering*, Vol. 2013, pp. 1-14.

[68] Okike, E., "A Proposal for Normalized Lack of Cohesion in Method (LCOM) Metric Using Field Experiment," *International Journal of Computer Science Issues*, Vol. 7, Issue 4, No 5, July 2010, pp. 19-27.

[69] Okike, E., "A Pedagogical Evaluation and Discussion about the Lack of Cohesion in Method (LCOM) Metric Using Field Experiment," *International Journal of Computer Science Issues*, Vol. 7, No 3, March 2010, pp. 36-43.

[70] Olague, H.M., Etzkorn, L.H., Gholston, S., and Quattlebaum, S., "Empirical Validation of Three Software Metrics Suites to Predict Fault-Proneness of Object-Oriented Classes Developed Using Highly Iterative or Agile Software Development Processes," IEEE Transactions on Software Engineering, Vol. 33, No. 6, June 2007, pp. 402-419.

[71] Patidar, K., Gupta, R., and Chandel, G.S., "Coupling and Cohesion Measures in OO Programming," *International Journal of Advanced Research in Computer Science and Software Engineering*, Vol. 3, Issue 3, March 2013, pp. 517-521.

[72] Pressman, R.S., *Software Engineering a Practitioner's Approach*, 5th Edition, McGraw Hill, India, 2001.

[73] Pasupathy, S and Bhavani, R, "Analyzing the Efficiency of Program Through Various OOAD Metrics," Journal of Theoretical and Applied Information Technology, Vol. 61 No.2, March 2014. pp. 346-351.

[74] Rajnish, K., "Another New Complexity Metric for Object-Oriented Design Measurement," International Journal of Hybrid Information Technology, Vol.7, No.2, 2014, pp.203-216.

[75] Reda, S., Ammar, H., and Hegazy, O., "A Methodology for Software Design Quality Assessment of Design Enhancements," *International Journal of Computer Science and Network*, Vol. 1, Issue 6, December 2012, pp. 101-109.
[76] Singh, H, Kumar, A, "A Novel Approach to Enhance the Maintainability of Object Oriented Software Engineering During Component Based Software Engineering," International Journal of Computer Sci. and Mobile Computing, Vol. 3, Issue 3, Mar 2014, pp.778 – 786.
[77] Salem, A.M., and Qureshi, A.A., "Analysis of Inconsistencies in Object Oriented Metrics," *Journal of Software Engg. and Applications*, 4, 2001, pp. 123-128.
[78] Sarkar, S., Kak, A.C., and Rama, G.M., "Metrics for Measuring the Quality of Modularization of Large-Scale Object-Oriented Software," *IEEE Transactions on Software Engineering*, Vol. 34, No. 5, September-October, 2008, pp. 700-720.
[79] Sarkar, S., Rama, G.M., and Kak, A.C., "API-Based and Information-Theoretic Metrics for Measuring the Quality of Software Modularization," *IEEE Transactions on Software Engineering*, Vol. 33, No. 1, January 2007, pp. 14-32.
[80] Sastry, J.S.V.R.S., Ramesh, K.V., and Padmaja, M., "Measuring Object-Oriented Systems Based on the Experimental Analysis of the Complexity Metrics," *International Journal of Engg. Sci. and Technology*, Vol. 3, No. 5, 2011, pp. 3726-3731.
[81] Saxena, P., and Saini, M., "Empirical Studies to Predict Fault Proneness: A Review," *International Journal of Computer Applications*, Vol. 22, No.8, May 2011, pp. 41-45.
[82] Shaik, A., Reddy, C.R.K., and Damodaran, A., "Object Oriented Software Metrics and Quality Assessment: Current State of the Art," *International Journal of Computer Applications*, Vol. 37, No, 11, 2012, pp. 06-15.
[83] Sharma, M., Singh, G., Arora, A., and Kaur, P., "A Comparative Study of Static Object Oriented Metrics," *International Journal of Advancements in Technology*, Vol. 3 No.1, January 2012, pp.25-34.
[84] Sharma, R., and Chhillar, R.S., "Novel Approach to Software Metrics," *International Journal of Soft Computing and Engg.*, Vol. 2, Issue 3, 2012, pp. 232-236.
[85] Sharma, A.K., Kalia, A., and Singh, H., "Empirical Analysis of Object Oriented Quality Suites," *International Journal of Engineering and Advanced Technology*, Vol.1, Issue-4, 2012, pp. 163-167.
[86] Sharma, A.K., Kalia, A., and Singh, H., "Metrics Identification for Measuring Object Oriented Software Quality," International Journal of Soft Computing and Engineering, Vol. 2, Issue 5, November 2012, pp. 255- 258.
[87] Shatnawi, R., "A Quantitative Investigation of the Acceptable Risk Levels of Object-Oriented Metrics in Open-Source Systems," *IEEE Transactions on Software Engineering*, Vol. 36, No. 2, March-April 2010, pp. 216-225.
[88] Shoaib, S.M., Shah, A., and Majeed, F., "Software Design Quality Metrics for Web Based Applications," *Pakistan Journal of Sci.*, Vol. 63, No. 1, 2011, pp. 19-25.
[89] Singh, P., Chaudhary, K.D., and Verma, S., "An Investigation of the Relationships between Software Metrics and Defects," *International Journal of Computer Applications*, Vol. 28, No.8, August 2011, pp. 42-45.
[90] Srinivasan, K.P., Devi, T., and Thiagarasu, V., "An Analysis of Chidamber - Kemerer Metrics for Object-Orientation Design," *Proceedings of National Conference on Emerging Trends in Computer Science*, Avinasilingam University for Women, Coimbatore, March 2009.
[91] Srinivasan, K.P., and Devi, T., "Design and Development of a Procedure to Test the Effectiveness of the Object-Oriented Design," International Journal of Engineering Research and Industrial Applications, Vol. 2, No.VI, 2009, pp. 15-25.
[92] Srinivasan, K.P., and Devi, T., "Design and Development of a Procedure for new Object-Oriented Design Metrics," *International Journal of Computer Applications*, Vol.24, No.8, June 2011, pp. 30-35.
[93] Srinivasan, K.P., and Devi, T.,"A Novel Software Metrics and Software Coding Measurement in Software Engineering," *International Journal of Advanced Research in Computer Science and Software Engineering*, Vol. 4, Issue 1, January 2014, pp. 303-308.
[94] Srinivasan, K.P., and Devi, T., "A Complete and Comprehensive Metrics Suite for Object-Oriented Design Quality Assessment," *International Journal of Software Engineering and Its Applications*, Vol. 8, No. 2, February 2014, pp.173-188. (This paper is recognized as a "Quality Paper" by SERSC, Republic of Korea and Published with Free of Cost).
[95] Subramanyam, R., and Krishnan, M.S., "Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects," *IEEE Transactions on Software Engineering*, Vol. 29, No. 4, April 2003, pp. 297-310.
[96] Thirugnanam, M., and Swathi J.N., "Quality Metrics Tool for Object Oriented Programming," International Journal of Computer Theory and Engineering, Vol. 2, No. 5, October 2010, pp. 712-717.
[97] Torkamani, M.A., "Metric Suite to Evaluate Reusability of Software Product Line," International Journal of Electrical and Computer Engineering, Vol. 4, No. 2, April 2014, pp. 285-294.
[98] Umamaheswari. E., Ghosh, D.K., "Software Quality: Dual Experts Opinion and Conditional Based Aggregation Method," International Journal of Engineering and Technology, Vol. 6 No. 2, Apr-May 2014, pp, 1167-1175.
[99] Yadav, A., and Khan, R.A., "Coupling Complexity Normalization Metric-An Object Oriented Perspective," *International Journal of Information Technology and Knowledge Management*, Vol. 4, No.2, July-Dec. 2011, pp. 501-509.
[100] Weyuker, E.J., "Evaluating Software Complexity Measure," *IEEE Transactions on Software Engineering*, Vol. 14, No. 9, September 1988, pp. 1357-1365.

AUTHORS PROFILE

K.P. Srinivasan received his Master of Computer Applications Degree from Bharathiar University, Coimbatore, India in 1993 and M.Phil Degree in Computer Science from the Bharathiar University, Coimbatore, India in 2001. Presently, he is working as an Associate Professor in Computer Science in C.B.M. College, Kovaipudur, Coimbatore under Bharathiar University, Coimbatore, India since 1997. He is doing his research work in Software Engineering. He has published five conference papers and six journal papers. He has received the best paper award from a conference and "quality paper" recognition from a reputed journal. His current research interests are in the areas of Software Engineering and Object-Oriented Systems.

Dr T. Devi received Master of Computer Applications Degree from P.S.G. College of Technology, Coimbatore, India in 1987 and the Ph.D. Degree from the University of Warwick, United Kingdom in 1998. Presently, she is working as an Associate Professor and Head of the Department of Computer Applications, School of Computer Science Engineering, Bharathiar University, Coimbatore, India. Prior to joining Bharathiar University, she was an Associate Professor in Indian Institute of Foreign Trade, New Delhi, India. She has contributed more than 140 papers in various Journals and Conferences. Her current research interests are in the areas of Software Engineering and Concurrent Engineering.