# Object Oriented Software Metrics and Quality Assessment: Current State of the Art

Amjan Shaik
M.Tech.(C.S.T.),(Ph.D),
Research Scholar,
Department of CSE,
JNTUH, Hyderabad,
Andhra Pradesh, India.

Dr. C. R. K. Reddy
M.Tech.(C.S.E.),Ph.D.(C.S.E),
HOD & Professor of CSE
CBIT, Hyderabad,
Andhra Pradesh, India

Dr. A. Damodaram
M.Tech.(C.S.E.),Ph.D.(C.S.E),
Professor of CSE &
Director of SCDE,
JNTUH, Hyderabad,
Andhra Pradesh, India.

## ABSTRACT
Necessity for a Productive software has been culminating and Object-Oriented Design technique is providing solution to this as it is the most powerful mechanism for developing proficient software systems. It is helpful not only in declining the cost but also in the development of high quality software systems. Software developers require accurate metrics for developing efficient software system. Object-Oriented Metrics plays a significant role pertaining to this aspect because of their importance in the development of successful software applications. In this paper Assessment of the current state of the art in Metrics and Object-Oriented Software System Quality is done. Further it contains short descriptive taxonomy of the Object-Oriented Design and Metrics.

**Keywords:** System, Metrics, Model, Software, Object-Oriented.

## 1. INTRODUCTION
Object-Oriented Design and Development is a famous way in contemporary software development environment. It enhances software productivity, reusability and flexibility of software systems. Object-Oriented Systems are becoming popular as efficient software systems as they decrease the size of system and number of logical constructs. Object-Oriented software generally have huge number of attributes and these attributes provide more comprehensive descriptions of software's internal nature and structure. These software systems consists of interacting objects which stays in their own local state and perform on their own information. Concepts such as complexity, usability, reusability, testability, understandability etc. are utilized for improving the quality of software system that also have relation with Object-Oriented features and can be utilized for improving the efficiency of Object-Oriented Systems.

Software Metrics have become quite necessary in some areas of software engineering, as they are utilized for measuring software quality and also for estimating the cost and effort of software projects [29].Usually the metrics are utilized to show the software quality in early stage of Software Development Life Cycle (SDLC) for observing the cost impact of modification and also for enhancing the software system, where as almost all the metrics ready for use for Object-Oriented Software Analysis normally will be utilized in later phase of SDLC [10]. As Object-Oriented Metrics need very good understanding of Object-Oriented concepts and no single metric is present which gives all the features of Object-Oriented Software System. Review of the current state of the art in Metrics in Object Oriented Programming is presented here.

Further organization of the paper is as follows. Section II deals with the taxonomy of the Object Oriented Design Methodology and Metrics, section III is about the current state of the art after which conclusion and references follow.

## 2. THE TAXONOMY OF OBJECT ORIENTED DESIGN AND METRICS

### 2.1. Object Oriented Design:
It is required to bring about basic standards and guiding principles which should be followed by the application developer for getting anticipated benefits and profits of Object-Oriented Technology. This technology may be utilized in measurement of the metrics of Object-Oriented Software. There are various design methodologies which suggested the guiding principle for many ways for augmenting Object-Oriented System.

The Booch method [5] explains the analysis and design phases of an Object-Oriented System implementation. This method shows a route from pre-requisites to implementation by utilizing Object-Oriented Analysis and Design and dwells upon the difference between logical view and physical view of a system. Jacobson's Object Oriented Software Engineering (OOSE) method [9] suggested pyramid model for the method of developing Object-Oriented Design, in which tools give aid for the activities in three categories: architecture, method and process. Object Modeling Technique (OMT) as explained by Rumbaugh et al. [14], provides system designers for conceptualizing the overall system architecture. OMT gives rise to 3 distinct models: object model, dynamic model and functional model of the system. Delatte et al. [1] developed Hierarchical Object Oriented Design (HOOD) method. In this method, the Basic Design Step, depends on the recognition of objects by means of Object-Oriented Design techniques. The use of this method is to develop the design as a set of objects that together give functionality to the program. Coad-Yourdon [30, 31] suggested Object-Oriented Analysis and Design method. This method goes in a step by step process for developing Object-Oriented Models. These steps are as follows: finding class & object, identifying structures, defining subjects, defining attributes, and defining services. Reenskaug et al. [39] developed an analysis and design method, developed by Reenskaug, dwells upon the role of objects in the system. This role depends more on the pre-requisites of the system than the properties of the object. So a single object will be able to do different roles at distinct stages of the system. Wirfs- Brock [36] developed the Object-Oriented approach named as Responsibility-Driven Design. According to them, for each class, different responsibilities are defined and in order to

fulfill the responsibilities, they require collaboration with other classes. A set of validation measures for different Object-Oriented Design approaches are developed by the object agency[40]. These measures contain concepts, notations, processes and pragmatics. Various measures for Object-Oriented Designs have been authorized by [6, 33, 41]. For software system, Design-Level Cohesion is suggested [12]. For detailing the quality of software system, more structures connected to the design properties of Object-Oriented System is given by [17, 18, 19, 20].

## 2.2 Metrics:

The Concept of Object Oriented programming that depends on Object-Oriented Metrics joins the design and implementation phases of software system. Different Object-Oriented Metrics are suggested in literature [26].Abreau [3, 4], J. Bansiya et al. [10], Briand et al. [17], Chidamber and Kemerer [37], Lorenz et al.[27], W. Li et al. [42, 43] are the various metrics that are mostly taken for reference in different literatures.

Chidamber and Kemerer (CK) [37] are the researchers who are mostly referred. six metrics were defined by them. They are Weighted Methods per Class (WMC), Response sets for Class (RFC), Lack of Cohesion in Methods (LCOM), Coupling Between Object Classes (CBO), Depth of Inheritance Tree of a class (DIT) and Number of Children of a class (NOC). CK Metrics were defined for evaluating design complexity in relation to their effect on quality factors like usability, maintainability, functionality, reliability etc. Several studies were conducted for authorizing CK Metrics. For example Basili et al. [41] scrutinized the CK Metrics and validated that five metrics of them appear to be useful for surmising class fault proneness. [6, 15] gives theoretical validation of CK Metrics and several experimental studies have been conducted for validating CK Metrics, for e.g. [2, 7, 11, 16, 18, 19, 22, 23, 24, 28, 33, 38, 40, 41, 43]. Table 2 gives the summary of CK Metrics.

CK Metrics aim is to evaluate the design of Object-Oriented System rather than the implementation of the system. This is what that make them more suited to Object-Oriented Paradigm since, in Object-Oriented Design emphasis is more on the design phase of software system.

Lorenz et al. [27] defined metrics for calculating static aspects of software design. These metrics are classified in to various groups depending on class size, class inheritance and class internal. Size-oriented metrics for the Object-Oriented classes lay emphasis on counts of attributes and operations whereas in the case of inheritance-oriented metrics, the emphasis is on the manner in which operations are reused in hierarchy class. Internal class-oriented metrics look at cohesion and code-oriented issues.

MOOD metric set model, proposed by Abreu [3] is one of the basic structural methods of the Object-Oriented Paradigm. They were defined to evaluate the utilization of Object-Oriented Design Methods such as MIF (Method Inheritance Factor), AIF (Attribute Inheritance Factor)) metrics, information hiding (MHF (Method Hiding Factor), AHF (Attribute Hiding Factor)) metrics, and polymorphism (POF (Polymorphism Factor), COF (Coupling Factor)) metrics. Abreu strongly said that metrics definitions and dimensions should be validated as they have a significant role in the process of designing the Object-Oriented Metrics.

Within the framework that, many metrics that are applied to traditional functional development are also applicable to object-oriented development, Rosenberg et al. [21] developed nine metrics for object-oriented system, from which three were traditional metrics viz. Cyclomatic Complexity (CC), Lines of Code (LOC), Comment Percentage (CP) and the other 6 metrics were same as that of the CK Metrics. They validate the six CK metrics were authorized by them at SATC and a link between significant Object Oriented Software quality concepts is given by them.

W. Li et al. [43] suggested a new metric suite comprising a number of Ancestor Classes (NAC), Number of Local Methods (NLM), Class Method Complexity (CMC), Number of Descendent Classes (NDC), Coupling Through Abstract data type (CTA), and Coupling Through Message passing (CTM). These metrics evaluate various internal attributes like Coupling, Complexity and Size.

J. Bansiya et al. [10] defined Quality Model for Object Oriented Design (QMOOD) metrics. The metrics in this model were given given as Average Number of Ancestors (ANA), Cohesion Among Methods of Class (CAM), Class Interface Size (CIS), Data Access Metric (DAM), Direct Class Coupling (DCC), Measure Of Aggregation (MOA), Measure of Functional Abstraction (MFA), Number Of Polymorphic Methods (NOP), Design Size of Class (DSC), Number Of class Hierarchies (NOH), Number of Methods (NOM). In the same way as MOOD Metrics, the QMOOD Metrics are defined to be computable early in the design method. The gist of above scrutinized metrics is given in Table 1.

**Table 1: OBJECT-ORIENTED METRICS FROM VARIOUS SOURCES**

| Source | Metrics |
|---|---|
| Chidamber et al. [37] | WMC, RFC, LCOM, CBO, DIT, NOC |
| Lorenz et al [27] | Class size, Class inheritance, Class internal |
| Abreu [4] | MIF, AIF, MHF, AHF, POF, COF |
| Rosenberg et al. [21] | CC, LOC, CP, WMC, RFC, LCOM, CBO, DIT, NOC |
| Li W. et al.[43] | NAC, NLM, CMC, CMC, NDC, CTA, CTM |
| Bansiya et al. [10] | ANA, CAM, CIS, DAM, DCC, MOA, MFA, NOP, DSC, NOH, NOM |

# 3. CURRENT STATE OF THE ART

Olague, H.M et al[1A] have given an empirical authorization of software metric suites on Agile based software for fault-proneness prediction in Object Oriented Systems. Chidamber and Kemerer (CK) Metrics, Robert C. Martin Metric Suite and McCabe's Metric Suite were the 3 metrics utilized. By Utilizing them, the flaws existing distinct versions of Rhino software have examined for predicting the quality of the software by utilizing the fault proneness concept. Basing on the results and empirical analysis, the authors contended that the distinct metric suites have distinct capacity in prediction of faults. Using this empirical analysis, the authors advised that software professionals for finding out those particular metric suites which can predict faults in the process of developing the quality metric software products utilizing the OO approach.

The 3 distinct quality metric suites and empirically authorized them by applying on software Rhino, a java script engine that was developed in java. The three metrics that are developed

are Chidamber and Kemerer (CK) Metrics, Robert C. Martin's Metric Suite and McCabe's Metric Suite.

The empirical authorization of the chosen metric suits was performed by implementing on Java-Script engine Rhino which was developed in java. The experiments scrutinized the software metrics for the object oriented systems by taking into consideration Rhino software with distinct versions as model for object oriented software. In the beginning experiments were scrutinized and noticed that the software development strategy used in Rhino is highly iterative, with a bottom-up approach.

The experiments examined 3 OO Metrics which suites their capacity for predicting software quality by using the metric measures. The metrics that are Chidamber and Kemerer (CK) metrics, Robert C. Martin Metric Suite and McCabe's Metric Suite. The capacity of these 3 metric suites for predicting the quality for different versions of Rhino is explained here.

The statistical results for some of the metrics are utilized for evaluating the quality of the software. The experiments scrutinized selected metrics such as CK metrics, R. C. Martin Metric Suite and McCabe's Metric Suite by scrutinizing the distribution (mean, median) and variance (standard deviation) of all the measures.

For performing empirical analysis, the experiments contain the number of bugs that are observed and rectified in each class of the system in all of the scrutinized versions. The experiments depend mainly on the primary hypothesis of software quality prediction is that a module which is currently under development is considered to be fault prone in the case of a module with the analogous product or procedure metrics in an earlier project (or release) that was developed in the same environment was fault prone [44]. So the experiments attempted to scrutinize the correlation of the metric values with defects faced from distinct versions of rhino.

**Observation:** Lastly authors came to a conclusion that the proneness in correlation between coherent metric values and defects, and also that the class components in the MOOD metrics suite are not good class fault-proneness predictors. Scrutinizing multivariate binary logistic regression models over six Rhino versions shows that these models may be helpful in evaluating quality in OO Classes developed by utilizing modern highly iterative or agile software development procedures. The empirical analysis is confined to a software developed by using agile model. So there is no hint regarding the fault-proneness and coherent metrics in other very fast and complicated development models like rup, rad and prototype. The experiments require to extend for scrutinizing the relation between coherent metrics and faults in distinct software development models like RUP, RAD and PROTOTYPE.

The other analysis models most frequently quoted in literature are:

Jie Xu et al. [44] have authorized Object-Oriented Design Metrics for faults estimation using empirical analysis. The Chidamber and Kemerer metrics suite were used to appraise the number of faults in the programs. The method includes statistical analysis and neuro-fuzzy techniques. The results showed that we can get dependable fault by using SLOC (Source Lines of Code), WMC (Weighted Methods per Class), CBO (Coupling between Object Classes) and RFC (Response for a Class) metrics. SLOC in particular got the most considerable effect on the number of defects. Yuming Zhou and Hareton Leung [45] have considered fault severity using the logistic regression and machine learning methods in their experimental exploration of the fault-proneness predicting capability of Object-Oriented Design Metrics, in particular, a subset of the Chidamber and kemerer suite. The statistical relation regarding fault severity between most of these design metrics and fault-proneness of classes and the reliance of their prediction competence on severity of faults was made known by the results that acquired on a public domain NASA data set. Further the results showed that the fault-proneness prediction capabilities of these metrics change reasonably with the severity of the defect.

Antoniol G. et al. [46] have experimentally checked size estimation models that are object oriented. The pragmatic examination of Object Oriented Function Points (OOFP) has been extended to a considerable amount with the aid of a bigger data set and by comparing OOFP with other predictors of LOC (Lines of Code) in their work. Linear models where the independent variable is either a conventional OO entity or an OOFP-related measure were built and assayed by using a cross validation approach.

Different things which affect size estimation were recognized by scrutinizing the collected data points and developer practices along with removing function point weighting tables from the OOFP procedure. By observing experimental results, it can be noticed that considerable enhancement in size estimates could be attained by governing these factors, 15% decrease of the normalized mean squared error corresponds to, a 56% reduction.

Mohammad Alshayeb and Wei Li [23] have given 2 iterative procedures for the pragmatic study of object oriented metrics. They include the short-cycled agile process and the long cycled framework evolution process. By observing the results, it can be seen that the design efforts and source lines of code added, changed, and deleted were triumphantly predicted by object oriented metrics in short-cycled agile process where as in the case long-cycled framework process the same features were not successfully predicted by it. This has shown that the design and implementation changes during development iterations can be predicted by Object Oriented Metrics, but the same cannot be the case with long-term development of an established system.

The experimental proof that has been given by Ramanath Subramanyam and M.S. Krishnan [48] is that a subset of the Chidamber and Kemerer suite that are Object Oriented Design complicated metrics performs a significant role in recognizing software faults. Pragmatic results on industry data that belongs to software developed in 2 widespread object oriented development programming languages indicated that the metrics have a considerable nexus with faults even after governing the size of the software. Also, effects of the metrics on faults changed for distinct samples from the 2 programming languages. Significant inferences for designing high-quality Object-Oriented Software were provided by these results.

Hector M. Olague et al. [49] have experimentally checked the software quality predicting capacity of 3 Object-Oriented Metrics suites with respect to their fault-proneness. The 3 Object Oriented Metrics suites examined were Chidamber and Kemerer (CK) metrics, Abreu's Metrics for Object-Oriented Design (MOOD), and Bansiya and Davis' Quality Metrics for Object-Oriented Design (QMOOD). Defect data for six versions of Rhino, an open-source JavaScript application that was written in Java were utilized for knowing the fault-prone classes predicting capability of the 3 metrics suites. The

results authenticated that triumphant statistical models for tracing error-prone classes are produced by the CK and QMOOD suites that have analogous components and good error-prone class predictors are not produced by the class components of the MOOD metrics suite.

Tibor Gyimothy et al. [50] have illustrated a procedure for tracing the fault-proneness of the source code of Mozilla which is an open source Web and e-mail suite, by describing the computing process of Chidamber and Kemerer object-oriented metrics. The use of the metrics for fault proneness prediction was examined by using regression and machine learning procedures for comparing the values that were obtained with the amount of bugs that are present in its bug database known as Bugzilla. The difference in the predicted fault-proneness in the development cycle of the software system was recognized by contrasting the metrics of distinct Mozilla versions.

Mohammad Alshayeb and Wei Li [51] did an experimental study in 2 Object-Oriented (OO) Systems, built using an agile process that resembles Extreme Programming (XP) regarding the class growth and the System Design Instability (SDI) metrics. The day to day collection of evolutionary data of the 2 systems were assayed. They concluded that class growth of the systems got observable trends and project progress with some trends can be shown by the SDI metric. Their other conclusion is that there lies a correlation between SDI metric and XP activities. In early and late development phases, two consistent jumps in the SDI metric values were noticed in both the studied systems. Part of the results concurred with an earlier experimental study in a distinct environment.

Raed Shatnawi et al[2A] suggested a statistical model which is obtained from the logistic regression for identifying threshold values for the Chidamber and Kemerer (CK) metrics. The process is authenticated empirically on a large Open-Source System—the Eclipse project. Their conclusion depending on the experimental results is that the CK Metrics have threshold effects at different risk levels. The usefulness of these thresholds on later releases was authenticated with the aid of decision trees. Another conclusion of the authors is that the chosen threshold values were more precise than those were chosen depending on either intuitive perspectives or on data distribution parameters.

**Observation:** Also, the suggested model can be exploited for knowing the risk level for an arbitrary threshold value. These findings indicate a relationship between risk levels and Object-Oriented Metrics and that these risk levels can be utilized for recognizing threshold effects.

Many techniques were put forth for recognizing threshold effects from controlled case studies and only a few of them are most quite regularly quoted in literature.

Daly et al.[52] analysed the effect of the DIT metric on the effort necessary for fulfilling a maintenance task. They noticed that varying a program using 5 levels of inheritance takes more exertion than altering the same program when reconstructed with zero level of inheritance. Other thing they noticed is that the changes in a program with 3 inheritance levels need less exertion than the program which was rebuilt without having any inheritance, i.e., the optimal value for the DIT metric was 3. In an empirical study, Cartwright [24] replicated the study of Daly et al[52] with distinct settings and tested only the effect of 3 levels of inheritance on the exertion spent to vary a program. The results of Cartwright's study and the results of Daly's work are distinct. Cartwright's study noticed that the changes in a program with 3 levels of inheritance need more exertion than the program when reconstructed without using inheritance. Other studies on inheritance effects, Prechelt et al. [22] and Harrison et al. [32] have proved the results that appeared in Cartwright's study. Benlarbi et al. [56] and El Emam et al. [57] were the only researchers who appraised the threshold values of a number of OO Metrics utilizing a statistical model depending upon the logistic regression model that were proposed by Ulm [58]. But, their study noticed that the threshold values reckoned from the logistic regression were invalid (that is, there was no statistical difference between the two models the no threshold model and the threshold model). In a quantitative study in the epidemiological field, Bender [59] noticed deficiencies in the definition of the threshold model that were suggested in [58] as the model presumed that the defect probability of a class is flat whenever the value of the metric is below the threshold (i.e., whenever a metric value is below the threshold, the probability of finding a fault is a constant) and the fault probability augments in accordance with the logistic function, otherwise. Bender showed that the appraised threshold values (based upon the [58] model) should only be considered adjustable whenever the assumption of the regression model, (i.e., a constant risk below the threshold) seems to be possible [59]. Bender redefined the threshold effects as a risk level that can be accepted. Till now, there is no agreement on the threshold values for software metrics, and conceivably not even for what are the best procedures to utilize in the look over for the threshold effects. In this research analysis of the usefulness of a quantitative methodology that was suggested in [59] for finding the threshold effects, which are redefined as the acceptable risk level, is done.

Santonu Sarkar et al [3A] suggested few metrics for calculating the Quality of Modularization of Large-Scale Object-Oriented Software. They aimed at providing a set of metrics that characterizes large Object-Oriented Software Systems with respect to such dependencies. They suggested few metrics for characterizing the quality of modularization regarding the APIs of the modules on one side. On the flipside, regarding such Object-Oriented inter-module dependencies as produced by inheritance, associational relationships, state access violations, fragile base-class design, etc. The validation process that authors utilized was two-pronged approach and tested it on Open Source applications.

These metrics are developed by the authors with the impact of their earlier work [4] that aspired to propose api based metrics for Non Object Oriented Models. Additionally, the metrics suggested in [4,] the inter-module couplings formulated by inheritance, containment, access control, polymorphism, encapsulation, etc., are the new metrics that are discussed. Module quality examination in the model checked in 2 dimensions. One way the module checked as service module and other way as extension module. The metrics suggested are Base-Class Fragility Index(BCFI), Inter-module coupling(IC), and Association-Induced Coupling(AC) to analyse software with regard to the inheritance-induced couplings between the modules. The BCFI, IC, and AC metrics are unaware of the APIs of the modules, nevertheless they define crucial auxiliary software properties without which any calculations that were made by Module Interaction Index (MII) and Non-API Method Closedness Index(NC) would not be that useful. The metric State Access Violation Index(SAV I) suggested that appraises to the range to which software is free of such procedures. The fundamental reasoning for inserting the Plugin Pollution Index( PPI) metric is that Object Oriented Software for large applications depends on third-party plugins

in order to elongate the functionality of the original software; it is common to see the happening of code-bloat in these third-party packages for the plugins. The metric Not-Programming-to-Interfaces Index(NPII) suggested to calculate the range to which the Object-Oriented Software does not follow the recommended practice of programming to interfaces. The other metric Classes related utilized together (CReuM) suggested for proving useful in some applications calculates the range to which classes that are defined together also get utilized together. It is usually true that, when classes that are defined together largely are together utilized, it will be very easy for working out an influence analysis of any alterations to those classes as the influence tends to be relatively localized in the other part of the software. The other metrics suggested are 1) an index for measuring the variability in the number of classes in the modules, MU; 2) an index for calculating the class-size variability by counting the number of methods in the classes, CUM; and 3) an index for calculating the class-size variability by counting the number of lines of code in the classes, CUl are support metrics and size based metrics.

**Observation:** As the metric values are being checked by presuming 2 dimensions for every api, the trustworthiness of the metric values mostly depends on precision of API tracing and concluding classes for service dimension and extension dimension. And the entire discussion is aimed exploring the methodology and formulation for calculating the metric values and their relation in quality assessment. No hint given in the paper regarding the trustworthiness of the modularization of the application as set of application programming interfaces and quality measures determined. Most frequently quoted in literature and analogous to the work illustrated in [3A] are following.

An early work by Coppick and Cheatham [63] attempted to elongate the then popular program-complexity metrics, like the Halstead [64] and the McCabe and Watson complexity measures [65], to OO software. Consequently, other works on OO Software Metrics concentrated mostly on the issue of how to characterize a single class with respect to its own complexity and its connections with other classes. The "one class at a time" concentration can be considered to be applied even when interclass couplings influenced by the procedures of one class calling the methods of other classes are taken into account. Major contributions of this early work are done by Brito e Abreu and Carapuca [66], Chen and Lum [67], Lee et al. [68], Chidamber and Kemerer (CK) [37], Lorenz and Kidd [27], Li and Henry [42], [43], Henderson-Sellers [72], and Briand et al. [73]. These researchers suggested that OO software be characterized by per-class criteria like the average number of attributes, average number of methods, average number of ancestor classes, average number of abstract attributes, Coupling Between Objects (CBO) as calculated by the average number of other-class methods called by every method of a given class and by the average number of other-class attributes utilized by all of the code in a given class, and like wise. Other metrics suggested in the same period—metrics that are straightforwardly redefined on a per-class basis—contain the MOOD metrics of Brito e Abreu et al. [4], [75] and Harrison et al. [33]. Particularly, the following MOOD metrics should be mentioned: the Attribute Hiding Factor (AHF) and the Method Hiding Factor (MHF) metrics for calculating the range of encapsulation, both defined as the ratio of the attributes and methods that are seen in a class as compared with the total number of the same; the Method Inheritance Factor (MIF) metric, that is a ratio of the total number of inherited methods to the total number of the same; and the Coupling Factor (CF) metric that calculates the frequency by which a class references an attribute or a method in another class. Counsell et al. [77] have worked out an arduous mathematical evaluation of 2 already known cohesion metrics, cohesion among methods in a class (CAMC) [78] and normalized Hamming distance (NHD) [79], for understanding their behaviors and analyse their use in calculating class cohesion. The prior work that we have quoted also contains some non-per-class metrics. These comprises the depth of the inheritance tree in a software system, the inheritance fan-out, number of ancestor classes, etc. Another previously suggested nonper-class metric is the system-level Coupling Factor (COF) that was put forwarded by Ghassemi and Mourant [80].

There have been controversies in the literature regarding the advantages of the aforementioned metrics, particularly with respect to the range to which they grasp the subtleties put forwarded by features that are strange to OO software. Going by example, when, in a purely count-based approach to software characterization, the number of attributes and methods defined for a class is known to reveal us something about the complication involved in that class. In OO software, even when a class is explicitly bereft of its own attributes and methods, it may nevertheless maintain a valuable set of the same through inheritance. By using the same token, whenever the code in a class carries out polymorphic method calls, it generally becomes very strenuous to figure out by static analysis as to which piece of code is actually being called for execution. This is what that made some researchers [81], [82], [83], [84], [17], [86] to debate that the quality measures created by the previously mentioned metrics may be open to interpretation.

There is also a body of work in the literature that has concentrated on OO metrics from the view of their capacity to predict software maintainability [87], [83], [88] and design flaws [89], [90], [91]. Much of the work on utilizing metrics for predicting design defects has concentrated on the CK metrics. Other thing that researchers analysed is that whether the fault tolerance of software can be predicted by the same or analogous metrics [41], [18], [50], [48]. Recently, Olague et al. [48] have done an empirical evaluation of the CK and the MOOD metric suites on 6 versions of an open source software for analyzing their capability to predict fault proneness. In a similar fashion, researchers have shown an empirical study of previously known cohesion metrics on a large corpus of software [97] which disclosed a bimodal behavior by almost all of these metrics.

Another work [98] on utilizing metrics to ascertain a necessity for code refactoring (like transferring a method or an attribute from one class to another, deduction of a new class, and so on). It is to be observed that the work of Alshayeb and Li [23] who, by carrying out an empirical study on the Java Development Kit (JDK), illustrated that the same metrics can considerably predict the required refactoring and error correction efforts, more specifically at the end of the Software Development Cycle (and specifically when the design cycle is short). Recently, Carey and Gannod [99] have utilized prevailing OO metrics [37], [72] and machine learning techniques for identifying domain concepts from source code.

Yuming Zhou et al[4A] scrutinized the Confounding Effect of Class Size on the Associations between Object-Oriented Metrics and Change-Proneness. Some experiments on eclipse are performed by them by utilizing 3 size metrics that are cohesion, coupling, and inheritance metrics. The results showed that:

1) The Confounding effect of class size on the associations between OO metrics and change-proneness, generally persists independent of whichever size metric is utilized;

2) The Perplexing effect of class size usually causes an overestimate of the associations between OO metrics and change-proneness; and

3) For many OO metrics, the perplexing effect of class size is the reason for their associations with change-proneness or results in a modification of the direction of the associations. These results profoundly shows that studies authorizing OO metrics on change-proneness should also consider class size as a perplexing variable.

Depending on Eclipse source build and class size metrics SLOC, NMIMP, and NumPara the experiments conducted for scrutinizing the potentially perplexing effect of class size on 55 OO metrics encompassing cohesion, coupling, and inheritance metrics and came to a conclusion that the bewildering effect of class size on the associations between OO metrics and change-proneness persists for many size factors.

When metrics are utilized for capturing some external quality attributes, like change-proneness, the effect of confounding from the class size must be taken off. Or else, we may get misleading results.

**Observation:** The work reported by Yuming Zhou et al[40] is impressive but confined for checking the association with traditional metrics and not taking into consideration the limits of the metrics brought into light in recent literature[50]. The association between class size and influence of metrics is to be corroborated empirically.

Jehad Al Dallal et al[50] enumerated and exchanged views about cases lacking discrimination anomaly (LDA) problem exists, as described using 16 cohesion metrics. Additionally, the authors empirically studied the regular happening of the LDA problem by applying considered metrics to classes in five open source Java systems. Then suggested a metric and a simulation-based methodology for calculating the discriminative power of cohesion metrics. Final conclusion is that the suggested discrimination metric calculates the probability that cohesion metric will give different cohesion values for classes with the same number of attributes and methods. However, distinct connectivity pattern of cohesive interactions (CPCIs) and also contended that a highly differentiated cohesion metric is more desirable as it has a lower chance of incorrectly taking into consideration classes to be cohesively equal when they have distinct CPCIs.

The suggested metric DPC defined as "A class model is defined by its number of methods and attributes/parameter-types, irrelevant of the CPCI". When cohesion metric and a class model are taken into consideration, the discriminative power of class cohesion metric (DPC) is the probability that it will get distinct cohesion values for classes of the same model , however with distinct CPCIs.

The DPC measurement procedure was developed based on the reasoning that "Models with larger numbers of methods and attributes have much larger numbers of possible distinct CPCIs". Hence, whenever the discrimination metric takes all models together for consideration, its value will be domineered by the larger models as they have much larger number of distinct CPCIs. For solving this problem, the DPC of a metric is calculated for each model individually. Comparison of discriminative power of different metrics can be done by considering the DPC values for distinct models.

**Observation:** When cohesion metric and a class model are taken into consideration, the DPC measuring methodology gets the accurate DPC value as it takes into account all obtainable CPCIs. But, few bounds has been noticed in DPC metric that are given below.

The DPC metric is utilized for comparing the discriminative power of distinct cohesion metrics. No threshold is required for evaluating the fitness of cohesion metric by using its discriminative power. The DPC metric is model-dependent. Given cohesion metric, the models mostly vary in terms of DPC values. In some of the cases, a cohesion metric has higher DPC values than another metric for some models .However, it has lower DPC values than the same metric for other models. The DPC calculating procedure is intensive in computation. The possible number of CPCIs augments exponentially as the size of the model increases. Assessment of few other analogous suggestions that are regularly quoted in literature follows:

Some authors have stated problems in discrimination with some class cohesion metrics, but this is done without further study. For example, Briand et al. [74] condemn LCOM3 as a component that is connected can have distinct degrees of connectivity. Those authors also said that LCOM2 has little power to discriminate. Similar criticism is shown towards LCOM2 by Bonja and Kidanmariam [76]. Counsell et al. [77] show that CAMC does not differentiate between classes that have the same number of methods, the same number of distinct parameter types, and the same total number of cohesive interactions but that show distinct connectivity patterns. They also show that NHD is unable to differentiate between classes having the same number of methods and different parameter types, where each parameter type is utilized in the same number of methods, independent of the connectivity pattern. Bonja and Kidanmariam [76] give examples to illustrate that CC has more differentiating power than LCOM2 or CAMC. Fernandez and Pena [53] give examples illustrating that SCOM has more differentiating power than LCOM2, LCOM3, and LCOM5.

Al Dallal [102] suggests an HLD cohesion metric and gives examples for comparing the suggested metric to CAMC and NHD in terms of differentiating power. In this particular paper, we suggest a formal definition for discriminative power and explain a process for calculating. Additionally, we have given examples for showing which of the 16 metrics taken into consideration have LDA problems.

The authenticity of a metric has to be studied and scrutinized both empirically and theoretically [54]. Empirical validation checks if the calculated and estimated values are consistent with each other. Theoretical validation checks if the metric shows the necessary properties of the calculated attribute. Several researchers have addressed how to empirically authenticate class cohesion metrics, encompassing [73], [70], [71], [69], [23], [50], [92], [55], [96] and [85]. Several properties are put forwarded for authenticating software metrics theoretically.

The first 4 properties were put forwarded by Briand et al. [74], the following six properties were illustrated by Chidamber and Kemerer [37], the next seven properties were explained by Ferna´ndez and Pen˜a [53], and the last property was demonstrated by Fenton and Pfleeger [29].We can observe that some of the properties are specific for class

cohesion metrics and others are usually connected with almost all the software metrics. Authors [74], [73], [47], [62], [72], [76], [53], [94], [95], [96], and [85] utilized some of these properties—and especially the first four—to authenticate many class cohesion metrics. The sensitivity property is very near to the introduced discriminative power property, but it is more refined and is conformed for class cohesion metrics.

# 4. CONCLUSION

This paper appraised contemporary art in metrics and quality assessment of Object Orient Software Systems. Assessment illustrates that there is huge progress in utilization of metrics for OO Software Quality Assessment. Along with that it is also clear that the new dimensions in usage of metrics and invention of new metrics gives a greater scope for Research in Object Oriented Software Assessment Directions and Strategies. It can also be noticed that large scope in metrics that gives guidance for indicating the progress an OO Software System has developed and the quality of design. By seeing the growing fame of Object-Oriented Software, possibility of developing models is very high, which would predict the usability and maintainability of Object-Oriented Software System in an efficient manner. So we are sanguine regarding future work in this particular direction.

# 5. REFERENCES

[1] B. Delatte, M. Heitz, and J. F. Muller, HOOD Reference Manual 3.1, Masson, Paris, 1993.

[2] B. Unger and L. Prechelt, The impact of inheritance depth on maintenance tasks – Detailed description and evaluation of two experimental replications, Technical Report, Karlsruhe University: Karlsruhe, Germany, 1998.

[3] F. B. Abreu and R. Carapua, "Candidate Metric for OOS within taxonomy framework, Journal of System & Softwrae, Vol. 26, No. 1, July 1994.

[4] F. B. Abreu, "The MOOD Metrics Set", In Proc. ECOOP'95, Workshop on Metrics, 1995.

[5] G. Booch, Object-oriented analysis and design, Benjamin-Cummings, U.S.A, pp.107-215, 1994.

[6] G. Poels and G. Dedene, DISTANCE: A Framework for Software Measure Construction, Research Report DTEW9937, Dept. Applied Economics, Katholieke Universiteit Leuven, Belgium, 1999, pp 46.

[7] G. Poelsand and G. Dedene, "Evaluating the Effect of Inheritance on the Modifiability of Object-Oriented Business Domain Models", 5th European Conference on Software Maintenance and Reengineering (CSMR 2001), Lisbon, Portugal, 2001, pp. 20-29.

[8] H. Sneed, Encapsulating Legacy Software for Reuse in Client/Server Sstem, In proceedings of WCRE-96, IEEE press, 1996, Monterey.

[9] I. Jacobson, Object-Oriented Software Engineering, Addison-Wesley, 1992 .

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

[11] J. Daly, A. Brooks, J. Miller, M. Roper and M. Wood, "An Empirical Study Evaluating Depth of Inheritance on Maintainability of Object- Oriented Software", Empirical Software Engineering, Vol. 1, No. 2, 1996, pp. 109-132.

[12] J. M. Bieman, and B. K. Kang, "Measuring Design-Level Cohesion", IEEE Transactions on Software Engineering, Vol. 24, No. 2, pp. 111- 124, 1998.

[13] J. Pinson Lewis and Richard S. Wiener, An Introduction to Objectoriented Programming and Smalltalk, Addison-Wesley pp 49-60, 1988.

[14] J. Rumbaugh, M. Blaha, W. Lorensen, F. Eddy, and W. Premerlani, Object-Oriented Modeling and Design, Prentice-Hall, 1991

[15] L. C. Briand, S. Morasca and V. Basili, "Property-Based Software Engineering Measurement", IEEE Transactions on Software Engineering, Vol. 22, No. 6, pp. 68-86, 1996.

[16] L. C. Briand, J. W. Daly, V. Porter, and J. Wust, A Comprehensive Empirical Validation of Product Measures for Object-Oriented Systems. Technical Report, ISERN-98-07, 1998.

[17] L. C. Briand, J. W. Daly and J. Wust, "A Unified Framework for Coupling Measurement in Object-Oriented Systems", IEEE Transactions on Software Engineering, Vol. 25, No. 1, pp. 91–121, 1999.

[18] L. C. Briand, J. W. Daly, V. Porter, and J. Wust, "Exploring the Relationships Between Design Measures and Software Quality in Object Oriented Systems", Journal of Systems and Software, Vol. 51, No. 3, pp. 245-273, 2000.

[19] L. C. Briand and J. Wust, "The Impact of Design Properties on Development Cost in Object-Oriented Systems", Proc. 7th Int'l Software Metrics Symposium (METRICS 01), IEEE CS Press, 2001.

[20] L. C. Briand, W. L. Melo and J. Wust, "Assessing the Applicability of Fault Proneness Models Across Object-Oriented Software Projects", IEEE transactions on Software Engineering, Vol. 28, No. 7, 2002.

[21] L. H. Rosenberg and L. Hyatt, "Software Quality Metrics for Object- Oriented Environments", Crosstalk Jounal, 1997.

[22] L. Prechelt, B. Unger, M. Philippsen and W. Tichy, "A controlled experiment on inheritance depth as a cost factor for code maintenance", The Journal of Systems and Software, Vol. 65, 2003, pp. 115-126.

[23] M. Alshayeb, and M. Li, "An Empirical Validation of Object-Oriented Metrics in Two Different Iterative Software Processes", IEEE Transactions on Software Engineering archive, Vol. 29, 2003, pp.1043 – 1049.

[24] M. Cartwright, An Empirical view of inheritance, Information and Software Technology, Vol. 40, No. 4, 1998, pp. 795-799.

[25] M. El Wakil, A. El Bastawissi, M. Boshra and A. Fahmy, Object- Oriented Design Quality Models – A Survey and Comparison. 2nd International Conference on Informatics and Systems, 2004.

[26] M. G. Bocco, M. Piattini and C. Calero, "A Survey of Metrics for UML Class Diagrams", Journal of Object Technology, Vol. 4, 2005, pp. 59- 92.

[27] M. Lorenz and J. Kidd, Object-Oriented Software Metrics, Prentice Hall, 1994.

[28] M. Tang, M. Kao and M. Chen, An Empirical Study on Object-Oriented Metrics, 6th IEEE International Symposium on Software Metrics, 1998.

[29] N. E. Fenton and S. L. Peeger, Software Metrics: A Rigorous and Practical Approach, PWS Publishing Company, Boston, Massachusetts, USA, 1997.

[30] P. Coad and E. Yourdon, Object-Oriented Analysis, Yourdon Press, Prentice Hall, New Jersey, 1990.

[31] P. Coad and E. Yourdon, Object-Oriented Design, Yourdon Press, Prentice Hall, New Jersey, 1991.

[32] R. Harrison, S. Counsell and R. Nithi, "Experimental Assessment of the Effect of Inheritance on the Maintainability of Object-Oriented Systems", The Journal of Systems and Software, Vol. 52, 2000, pp. 173-179.

[33] R. Harrison, S. Counsell and V. Reuben, "An Evaluation of the MOOD Set of Object-Oriented Software Metrics", IEEE Transactions on Software Engineering, Vol. 24, No. 6, pp. 491-496, 1998.

[34] R. Subramanya and M. S. Krishnan, "Empirical of CK Metrics for Object-Oriented Design Complexity: Implication for Software Defects", IEEE Transaction on Software Engineering, Vol. 29, 2003, pp. 297-310.

[35] R. W. Selby and V. R. Vasili, "Analyzing Error-Prone Systems Structure", IEEE Transactions on Software Engineering, Vol. 17, 1991, pp. 141-152.

[36] R. Wirfs-brock, B. Wilkerson, and L. Weiner, Designing Object- Oriented Software, Prentice-Hall, 1990.

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

[38] S. R. Chidamber, D. P. Darcy, and C. F. Kemerer, "Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis", IEEE Transactions on Software Engineering, Vol. 24, No. 8, pp. 629-637, 1998.

[39] T. Reenskaug, E. Andersen A. Berre, A. Hurlen, A. Landmark, O. Lehne, E. Nordhagen, E. Ness-Ulseth, G. Oftedal, A. Skaar, and P. Stenslet , "OORASS: seamless support for the creation and maintenance of object oriented systems", Journal of Object Oriented Programming, Vol. 5, No. 6, 1992, pp. 7-41.

[40] The Object Agency, A comparison of Object–Oriented Development Methodologies, 1996. http://www.toa.com.

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

[42] W. Li, and S. Henry, "Object-Oriented Metrics that Predict Maintainability". Journal ofSystems and Software, Vol. 23, No. 2, pp. 111-122, 1993. [43] W. Li, "Another Metric Suite for Object Oriented Programming", The Journal of Systems and Software, Vol. 44, No. 2, pp. 155-162, 1998.

[44] Jie Xu, Danny Ho, Luiz Fernando Capretz, "An Empirical Validation of Object-Oriented Design Metrics for Fault Prediction," Journal of Computer Science, Vol: 4, No: 7, pp. 571-577, 2008.

[45] Yuming Zhou, Hareton Leung, "Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults, " IEEE transaction on software engineering, Vol. 32, No. 10, pp. 771-789, 2006.

[46] Antoniol G, Fiutem R, Lokan C, "Object-Oriented Function Points: An Empirical Validation," In Kluwer Academic Publishers, pp: 225-254, 2003.

[47] Y. Zhou, L. Wen, J. Wang, Y. Chen, H. Lu, and B. Xu, "DRC: A Dependence Relationships Based Cohesion Measure for Classes," Proc. 10th Asia-Pacific Software Eng. Conf., pp. 1-9, 2003.

[48] Ramanath Subramanyam, M.S. Krishnan, "Empirical Analysis of CK Metrics for Object- Oriented Design Complexity: Implications for Software Defects," IEEE transaction on software engineering, Vol. 29, No. 4, pp. 297-310, 2003.

[49] Hector M. Olague, Letha H. Etzkorn, Sampson Gholston, and Stephen Quattlebaum "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 transaction on software engineering, Vol: 33, No: 6, pp. 402-419, 2007.

[50] Tibor Gyimothy, Rudolf Ferenc, Istvan Siket, "Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction", IEEE Transactions on Software Engineering, Vol. 31, No. 10, October 2005.

[51] Mohammad Alshayeb, Wei Li, "An empirical study of system design instability metric and design evolution in an agile software process", Journal of Systems and Software, Vol: 74, No: 3, pp: 269 - 274, 2005.

[52] J. Daly, A. Brooks, J. Miller, M. Roper, and M. Wood, "Evaluating Inheritance Depth on the Maintainability of Object-Oriented Software," Empirical Software Eng. vol. 1, no. 2, pp. 109-132, 1996.

[53] L. Ferna´ndez and R. Pen˜ a, "A Sensitive Metric of Class Cohesion," Int'l J. Information Theories and Applications, vol. 13, no. 1, pp. 82-91, 2006.

[54] B. Kitchenham, S.L. Pfleeger, and N. Fenton, "Towards a Framework for Software Measurement Validation," IEEE Trans. Software Eng., vol. 21, no. 12, pp. 929-944, Dec. 1995.

[55] A. Marcus, D. Poshyvanyk, and R. Ferenc, "Using the Conceptual Cohesion of Classes for Fault Prediction in Object-Oriented Systems," IEEE Trans. Software Eng., vol. 34, no. 2, pp. 287-300, Mar./Apr. 2008.

[56] S. Benlarbi, K. El Emam, N. Goel, and S. Rai, "Thresholds for Object-Oriented Measures," Proc. 11th Int'l Symp. Software Reliability Eng., pp. 24-38, 2000.

[57] K. El Emam, S. Benlarbi, N. Goel, W. Melo, H. Lounis, and S.N. Rai, "The Optimal Class Size for Object-Oriented Software," IEEE Trans. Software Eng., vol. 28, no. 5, pp. 494-509, May 2002.

[58] K. Ulm, "A Statistical Method for Assessing a Threshold in Epidemiological Studies," Statistics in Medicine, vol. 10, no. 3, pp. 341-349, 1991.

[59] R. Bender, "Quantitative Risk Assessment in Epidemiological Studies Investigating Threshold Effects," Biometrical J., vol. 41, no. 3, pp. 305-319, 1999.

[60] R. Strnisa, P. Sewell, and M. Parkinson, "The Java Module System: Core Design and Semantic Definition," Proc. ACM SIGPLAN Conf. Object-Oriented Programming Systems, Languages and Applications, vol. 42, no. 10, pp. 499-514, 2007.

[61] S. Sarkar, G.M. Rama, and A.C. Kak, "API-Based and Information- Theoretic Metrics for Measuring the Quality of Software Modularization," IEEE Trans. Software Eng., vol. 33, no. 1, pp. 14-32, Jan. 2007.

[62] Z. Chen, B. Xu, and Y. Zhou, "Measuring Class Cohesion Based on Dependence Analysis," J. Science and Technology, vol. 19, no. 6, pp. 859-866, 2004.

[63] C.J. Coppick and T.J. Cheatham, "Software Metrics for Object- Oriented Systems," Proc. ACM Ann. Computer Science Conf., pp. 317-322, 1992.

[64] M.H. Halstead, Elements of Software Science. Elsevier, 1977.

[65] T.J. McCabe and A.H. Watson, "Software Complexity," Crosstalk, J. Defense Software Eng., vol. 7, no. 12, pp. 5-9, Dec. 1994.

[66] F. Brito e Abreu and R. Carapuca, "Candidate Metrics for Object- Oriented Software within a Taxonomy Framework," J. Systems and Software, vol. 26, pp. 87-96, 1994.

[67] J.-Y. Chen and J.-F. Lum, "A New Metric for Object-Oriented Design," Information of Software Technology, vol. 35, pp. 232-240, 1993.

[68] Y.-S. Lee, B.-S. Liang, and F.-J. Wang, "Some Complexity Metrics for Object-Oriented Programs Based on Information Flow," Proc. Sixth IEEE Int'l Conf. Computer Systems and Software Eng., pp. 302- 310, 1993.

[69] L.C. Briand and J. Wust, "Empirical Studies of Quality Models in Object-Oriented Systems," Advances in Computers, pp. 97-166, Academic Press, 2002.

[70] L.C. Briand, J. Wust, J. Daly, and V. Porter, "Exploring the Relationship between Design Measures and Software Quality in Object-Oriented Systems," J. System and Software, vol. 51, no. 3, pp. 245-273, 2000.

[71] L.C. Briand, J. Wu¨ st, and H. Lounis, "Replicated Case Studies for Investigating Quality Factors in Object-Oriented Designs," Empirical Software Eng., vol. 6, no. 1, pp. 11-58, 2001.

[72] B. Henderson-Sellers, Object-Oriented Metrics: Measures of Complexity. Prentice Hall, 1996.

[73] L.C. Briand, S. Morasca, and V.R. Basili, "Defining and Validating Measures for Object-Based High-Level Design," IEEE Trans. Software Eng., vol. 25, no. 5, pp. 722-743, Sept./Oct. 1999.

[74] L.C. Briand, J. Daly, and J. Wuest, "A Unified Framework for Cohesion Measurement in Object-Oriented Systems," Empirical Software Eng.—An Int'l J., vol. 3, no. 1, pp. 65-113, 1998.

[75] F.B. e Abreu, M. Goulao, and R. Estevers, "Towards the Design Quality Evaluation of OO Software Systems," Proc. Fifth Int'l Conf. Software Quality, 1995.

[76] C. Bonja and E. Kidanmariam, "Metrics for Class Cohesion and Similarity between Methods," Proc. 44th Ann. ACM Southeast Regional Conf., pp. 91-95, 2006.

[77] S. Counsell, S. Swift, and J. Crampton, "The Interpretation and Utility of Three Cohesion Metrics for Object-Oriented Design," ACM Trans. Software Eng. and Methodology, vol. 15, no. 2, pp. 123- 149, 2006.

[78] J. Bansiya, L. Etzkorn, C. Davis, and W. Li, "A Class Cohesion Metric for Object-Oriented Designs," J. Object Oriented Program, vol. 11, no. 8, pp. 47-52, 1999.

[79] S. Counsell, E. Mendes, S. Swift, and A. Tucker, "Evaluation of an Object-Oriented Cohesion Metric through Hamming Distances," Technical Report BBKCS-02-10, Birkbeck College, Univ. of London, 2002.

[80] M.D. Ghassemi and R.R. Mourant, "Evaluation of Coupling in the Context of Java Interfaces," Proc. ACM SIGPLAN Conf. Object- Oriented Programming Systems, Languages and Applications, pp. 47- 48, 2000.

[81] M. Hitz and B. Montazeri, "Chidamber and Kemerers Metrics Suite: A Measurement Theory Perspective," IEEE Trans. Software Eng., vol. 22, pp. 267-271, 1996.

[82] N. Churcher and M. Shepperd, "Comments on "A Metrics Suite for Object-Oriented Design"," IEEE Trans. Software Eng., vol. 21, no. 3, pp. 263-265, Mar. 1995.

[83] R.K. Bandi, V.K. Vaishnavi, and D.E. Turk, "Predicting Maintenance Performance Using Object-Oriented Design Complexity Metrics," IEEE Trans. Software Eng., vol. 29, no. 1, pp. 77-86, Jan. 2003.

[84] L. Etzkorn, C. Davis, and W. Li, "A Practical Look at the Lack of Cohesion in Methods Metrics," J. Object Oriented Programming, vol. 11, no. 5, pp. 27-34, 1998.

[85] J. Al Dallal and L. Briand, "A Precise Method-Method Interaction- Based Cohesion Metric for Object-Oriented Classes," ACM Trans. Software Eng. and Methodology, vol. 20, no. 6, Nov. 2011.

[86] H. Kabaili, R.K. Keller, and F. Lustman, "Cohesion as Changeability Indicator in Object-Oriented Systems," Proc. Fifth European Conf. Software Maintenance and Reengineering, pp. 39-46, 2001.

[87] P. Oman and J. Hagemeister, "Constructing and Testing of Polynomials Predicting Software Maintainability," J. Systems and Software, vol. 24, no. 3, pp. 251-266, Mar. 1994.

[88] M. Dagpinar and J.H. Jahnke, "Predicting Maintainability with Object-Oriented Metrics—An Empirical Comparison," Proc. 10th Working Conf. Reverse Eng., p. 155, 2003.

[89] R. Marinescu, "Detecting Design Flaws via Metrics in Object Oriented Systems," Proc. 39th Int'l Conf. and Exhibition on Technology of Object-Oriented Languages and Systems, pp. 173-182, 2001.

[90] B. Lague, D. Proulx, E.M. Merlo, J. Mayrand, and J. Hudepohl, "Assessing the Benefits of Incorporating Function Clone Detection in a Development Process," Proc. Int'l Conf. Software Maintenance, 1997.

[91] K. Kontogiannis, "Evaluating Experiments on the Detection of Programming Patterns Using Software Metrics," Proc. Working Conf. Reverse Eng., pp. 44-54, 1997.

[92] K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra, "Investigating Effect of Design Metrics on Fault Proneness in Object- Oriented Systems," J. Object Technology, vol. 6, no. 10, pp. 127- 123, 2007.

[93] J. Al Dallal, "A Design-Based Cohesion Metric for Object-Oriented Classes," Proc. Int'l Conf. Computer and Information Science and Eng., Nov. 2007.

[94]] J. Al Dallal, "Software Similarity-Based Functional Cohesion Metric," IET Software, vol. 3, no. 1, pp. 46-57, 2009.

[95] J. Al Dallal, "Mathematical Validation of Object-Oriented Class Cohesion Metrics," Int'l J. Computer Science, vol. 4, no. 2, pp. 45-52, 2010.

[96] J. Al Dallal and L. Briand, "An Object-Oriented High-Level Design-Based Class Cohesion Metric," Information and Software Technology, vol. 52, no. 12, pp. 1216-1221, 2010.

[97] R. Barker and E. Tempero, "A Large-Scale Empirical Comparison of Object-Oriented Cohesion Metrics," Proc. 14th Asia-Pacific Software Eng. Conf., pp. 414-421, 2007.

[98] F. Simon, F. Steinbruckner, and C. Lewerentz, "Metrics Based Refactoring," Proc. Fifth European Conf. Software Maintenance and Reengineering, pp. 30-38, 2001.

[99] M.M. Carey and G.C. Gannod, "Recovering Concepts from Source Code with Automated Concept Identification," Proc. 15th IEEE Int'l Conf. Program Comprehension 2007.

## 6. AUTHORS PROFILE

**Amjan.Shaik** is a Research Scholar, Department of Computer Science and Engineering, JNTUH, Hyderabad, India. He has received M.Tech.(Computer Science and Technology) from Andhra University. He has been published and presented more than 30 Research and Technical papers in International Journals , International Conferences and National Conferences. His main research interests are Software Metrics, Software Engineering, Software Testing, Software Quality and Object Oriented Design.

**Prof. Dr. C.R.K. Reddy** is working as a Professor and HOD, Department of Computer Science and Engineering at Chaitanya Bharathi Institute of Technology (CBIT),Hyderabad, India. He has received M.Tech.( Computer Science and Engineering ) from JNTUH, Hyderabad and Ph.D in Computer Science and Engineering from Hyderabad Central University (HCU). He has been published and presented wide range of Research and Technical Papers in National ,International Journals and National ,International Conferences. At present 8 Research Scholars are doing Ph.D under his esteemed guidance. His main research Interests are Program Testing, Software Engineering , Software Metrics , Software Architectures, Neural Networks and Artificial Intelligence.

**Prof. Dr. Avula Damodaram** joined as faculty of Computer Science & Engineering at JNTU, Hyderabad in the year 1989. In his over 2 decades of dedicated service. Dr. Damodaram performed distinguished services to the University as a Professor, Head of the Department, Vice Principal, Director of UGC-Academic Staff College and now Director, School of Continuing & Distance Education. Dr. Damodaram has successfully guided 6 Ph.D. and 2 MS Scholars apart from myriad M.Tech projects. He is currently guiding 9 scholars for Ph.D and 1 scholar for MS. Dr. Damodaram is on the editorial board of 2 International Journals and a number of Course materials. He successfully executed an AICTE research project at a cost of 7 Lakhs. Dr. Damodaram has been a UGC nominee for a number of expert and advisory committees of various Indian Universities. He has been associated with conduct of many entrance tests in the state such as ECET and ICET. Dr .Damodaram has been a Life Member, Vice-President, Director and President of a number of core committees spread all over the country. Dr. Damodaram has served the interests of the College and University teachers at the University, State and National levels. He has organized as many as 30 Workshops, Short Term Courses and other Refresher and Orientation programmes. Dr .Damodaram has published more than 50 well researched papers in national and International journals. He has also presented 45 papers at different National and International conferences. Dr. Damodaram visited the Universities of Austria and the United Kingdom for presenting papers at International conferences. On the basis of his scholarly achievements and other multifarious services, Dr .Damodaram was honoured with the award of DISTINGUISHED ACADAMICIAN by Pentagram Research Centre, India, in January 2010.