Volume 2 No.4, APRIL 2011 ISSN 2079-8407
Journal of Emerging Trends in Computing and Information Sciences

http://www.cisjournal.org

Investigate the Result of Object Oriented Design Software Metrics on Fault-Proneness in Object Oriented Systems: A Case Study

[1]Amjan. Shaik, [2]N.Satyanarayana, [3]Mohammed Huzaifa, [4]Nazeer Shaik, [5]Mohd Zainuddin Naveed, [6]S.V.Achuta Rao, [7]C.R.K.Reddy

[1]Professor and HOD-CSE, Ellenki College of Engineering and Technology (ECET),Patelguda, Hyderabad, India.
[2]Principal, Nagole Institute of Technology and Science, (NITS),Kuntllor, Hyderabad, India.
[3]Assistant professor –IT, Muffakhamjah College of Engineering and Technology (MJCET),Banjarahills,Hyderabad, India.
[4]Assistant professor –CSE, Moghal College of Engineering and Technology (MCET),Bandlaguda,Hyderabad, India.
[5]Assistant professor –CSE, Muffakhamjah College of Engineering and Technology (MJCET),Banjarahills, Hyderabad, India.
[6]Professor and HOD-CSE&IT, DJR Institute of Engineering and Technology (DJRIET), Vijayawada, India.
[7]Professor andHOD-CSE,Chaitanya Bharathi Institute of Technology(CBIT),Gandipet,Hyderabad,India

ABSTRACT

In the last decade, empirical studies on object-oriented design metrics have shown some of them to be useful for predicting the faults-proneness of classes in object-oriented software systems. It would be valuable to know how object-oriented design metrics and class fault-proneness are related when fault severity is taken into account. In this paper we use logistic regression and principal component methods to empirically investigate the usefulness of object-oriented design metrics, specially a subset of the Chidamber and Kemerer suite in predicting fault-proneness when taking fault severity into account. In the era of Object Oriented software metrics demand for quality software has undergone with rapid growth during the last few years. This is leading to an increase in the development of metrics for measuring the properties of software such as coupling, cohesion and inheritance that can be used in early Quality assessments. Much effort has been developed to the development and empirical validation of software metrics. Quality models that explore the relationship between these properties and quality attributes such as fault proneness, maintainability, effort or productivity are needed to use these metrics effectively. The goal of this work is to empirically explore the relationship between Object Oriented Design Metrics and Fault Proneness of object oriented system classes. The object oriented spatial complexity measures proposed in literature were formulated by keeping C++ language in mind, and there were no spatial complexity measures available for the Java language. Keeping in view the increasing popularity of Java, this paper attempts to define the spatial complexity measures for Java applications. Our results are based on Open Source Java Projects and Post Graduate Engineering student's projects. We are empirically analyzed and tested with our Software Tool.

Keywords: *Object Oriented Metrics, Coupling, Cohesion, Inheritance, Empirical Analysis.*

1. INTRODUCTION

Software Metrics are used to measure software engineering products (design, source code etc.), processes (analyses, design, coding, testing etc.) and professionals(efficiency or productivity of an individual designer). If used properly, software engineering metrics allow us to quantitatively define the degree of success or failure of a product, process, or person. These can also be used to take meaningful and useful managerial and technical decisions related to cost, effort, time, quality etc. Thus , incorporating metrics into software development process is a valuable step towards creating better systems. There are several Metrics proposed for capturing the quality of object oriented design. These metrics provide ways to evaluate the quality of software and their use in earlier phases of software development life cycle. But how do we know which metrics are useful in capturing important quality attributes such as fault proneness, effort or productivity. Empirical studies of real systems can provide the relevant answers. More data based by empirical studies which are capable of being verified by observation or experiment are needed. The validation of software metrics has received much research attention by the software engineers. There are two types of validation that are recognized [9]:internal and external. Internal validation is a theoretical exercise that ensures that the metric is a proper numerical characterization of the property it claims to measure. External validation involves empirically demonstrating that the product metric is associated with some important external metric (such as measures of maintainability or reliability). These are also commonly referred to as theoretical and empirical validation, respectively .The metrics we investigate here consist of CK Metrics suite[2,10,11] and some other metrics. Univariate logistic regression models and principal component method are used as the basis for demonstrating the relationship between object oriented metrics and fault proneness[4].

Univariate Logistic regression analysis is carried out to test that size, coupling and inheritance increase fault proneness of a class where as cohesion decrease fault proneness of a class and find individual impact of metrics on fault proneness. In addition to coupling and cohesion another important dimension of software quality is its complexity. The software can have better testability, readability and maintainability if it processes low

Volume 2 No.4, APRIL 2011 ISSN 2079-8407
Journal of Emerging Trends in Computing and Information Sciences
©2010-11 CIS Journal. All rights reserved.
http://www.cisjournal.org

coupling, high cohesion and less complexity. Principal component method of factor analysis is used to find whether all these metrics are in dependent or are capturing same underlying property of the object being measured. A high maturity organization is expected to use metrics heavily for process and project management. Though metrics data is collected and even used at the levels. In software organizations metrics are used for project planning, monitoring and controlling a project and overall process management and improvement. Based on empirical experience and analysis by the people ,without software metrics , there is no kind of measurement for a project.

2. RESEARCH BACKGROUND

In this section,we present the theoretical and empirical basis of the object oriented metrics that we attempt to validate.

2.1 Cognitive Theory of Object Oriented Metrics

The understandability of object oriented software has been reported in literature to be dependant on architectural aspects as well along with spatial distances. The architectural aspect of software has been reflected in Cognitive complexity with the help of using weights of various types of Basic Control Structures . A theoretical basis for developing quantitative models relating product metrics and external quality metrics has been provided in [5] and is summarized in Figure[1].

This theory hypothesizes that the structural properties of a software component (such as its coupling) have an impact on its cognitive complexity. The cognitive complexity is an intervening variable between the structural properties of classes and fault-proneness. Cognitive complexity is defined as the mental burden of the individuals who have to deal with the component, for example, the developers, testers, inspectors, and maintainers. High cognitive complexity leads to a component exhibiting undesirable external qualities, such as increased fault-proneness and reduced maintainability. Accordingly, object-oriented product metrics that affect cognitive complexity will be related with fault-proneness.

It would provide us with a clear mechanism that would explain the introduction of faults into object-oriented applications. The current theoretical framework for explaining the effect of the structural properties of object oriented programs on external program attributes can be justified empirically. To be specific, studies that have been performed indicate that the distribution of functionality across classes in object-oriented systems, and the exacerbation of this through inheritance, potentially makes programs more difficult to understand. This suggests that highly cohesive, sparsely coupled, and low inheritance programs are less likely to contain a fault. Therefore, metrics that measure these three dimensions of an object oriented program would be expected to be good predictors of fault-proneness or the number of faults. The empirical question is then whether contemporary object-oriented metrics measure the relevant structural properties well enough to substantiate the above theory.

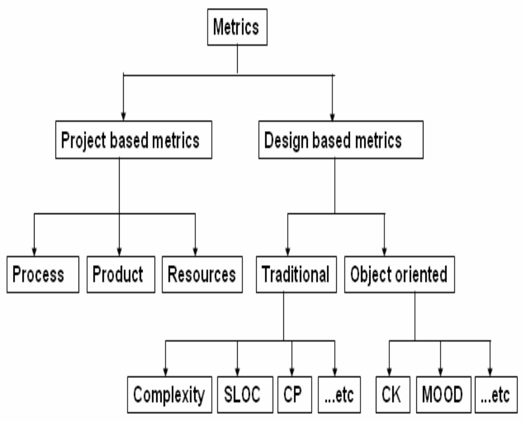


Figure 1: Metrics hierarchy

2.2 Empirical validation of Object Oriented Metrics on Fault Proneness

This paper makes a number of contributions. First, using a publicly available data set, we present new evidence indicating an association between OO design metrics and fault-proneness , thereby providing valuable data in an important area for which limited experimental data is available. Second, we validate the association between OO design metrics and fault-proneness of classes across fault severity. To the best of our knowledge, in spite of its importance, there has been no such previous research. Third , on the methodological front, the fault-proneness of classes is analyzed using not just the familiar method of logistical regression but also applied principal component methods. In this section, we review the empirical studies that investigate the relationship between object-oriented metrics and fault-proneness.

Metrics Studied

The metrics of coupling, cohesion, inheritance and size are the independent variables used in this study[10,11]. Our focus is on OO metrics that are used as independent variables in a prediction model that is usable at early stages of software development. The metrics selected in this paper are summarized in Table 1.

Table 1: Object-Oriented Metrics

Metric	1) **Definition**
Coupling between Objects (CBO)	CBO for a class is count of the number of other classes to which it is coupled.
Coupling between Objects (CBO1)	Same as CBO, except that inheritance based coupling is not counted.
Lack of Cohesion (LCOM1)	It counts number of null pairs of methods that do not have common attributes.

Lack of Cohesion (LCOM2)	It measures the dissimilarity of methods in a class by looking at the instance variable or attributes used by methods.
Number of Children (NOC)	The NOC is the number of immediate subclasses of a class in a hierarchy.
Depth of Inheritance (DIT)	The depth of a class within the inheritance hierarchy is the maximum number of steps from the class node to the root of the tree and is measured by the number of ancestor classes.
Weighted Methods per Class (WMC)	The WMC is a count of sum of complexities of all methods in a class.
Response for a Class (RFC)	The response set of a class (RFC) is defined as set of methods that can be potentially executed in response to a message received by an object of that class.
IFCAIC ACAIC OCAIC FCAEC DCAEC OCAEC IFCMIC ACMIC DCMIC FCMEC DCMEC OCMEC IFMMIC AMMIC OMMIC FMMEC DMMEC OMMEC	These coupling metrics count number of interactions between classes. The metrics distinguish the relationship between the classes (friendship, inheritance, none), different types of interactions, and the locus of impact of the interaction. The acronyms for the metrics indicates what interactions are counted: • The first or first two characters indicate the type of coupling relationship between classes (A: Ancestor, D:Descendents, F: Friend classes, IF: Inverse Friends (classes that declare a given class a as their friend), O:Others, i.e., none of the above relationships). • The next two characters indicate the type of interaction: CA: There is a Class-Attribute interaction if class x has an attribute of type class y. CM: There is a Class-Method interaction if class x consist of a method that has parameter of type class y. MM: There is a Method-Method interaction if class x calls method of another class y, or class x has a method of class y as a parameter. • The last two characters indicate the locus of impact: IC: Import coupling, counts the number of other classes called by class x. EC: Export coupling, count number of other classes using class y.
Lines Of Code (LOC)	It is the count of lines in the text of the source code excluding comment lines

Measuring quality

Measurement enables to improve the software process, assist in the planning, tracking the control of a design. A good software engineer uses measurements to asses the quality of the analysis and design model, the source code, the test cases, etc. What does quality mean?

"Quality refers to the inherent or distinctive characteristics or property of object, process or other thing. Such characteristics or properties may set things apart from other things, or may denote some degree of achievement or excellence".

Many quality measures can be collected from literature, the main goal of metrics is to measure errors and defects. The following quality factor should have every metrics [10,11]

• **Efficiency** - Are the constructs efficiently designed?
The amount of computing resource and code required by a program to perform
its function.
• **Complexity** - Could the constructs be used more effectively to decrease the
architectural complexity?
• Understandability - Does the design increase the psychological complexity?
• **Reusability** - Does the design quality support possible reuse?
Extent to which a program or part of a program can be reused in other application , related to the packaging and scope of the functions that the program performs.
• **Testability/Maintainability** - Does the structure support ease of testing and changes?
Effort required locating and fixing an error in a program, as well as effort required to test a program to ensure that it performs its intended function.

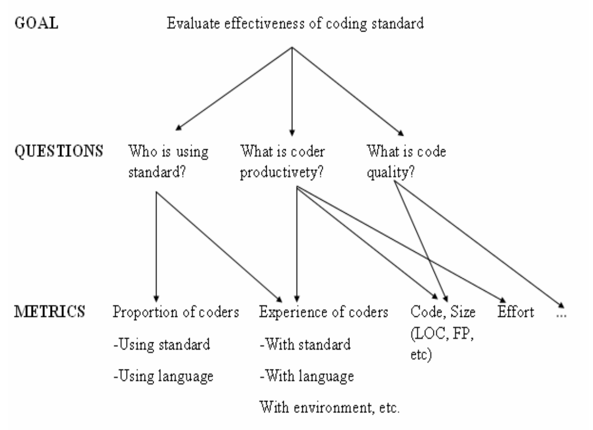


Figure 2: Example of deriving metrics from goal and questions

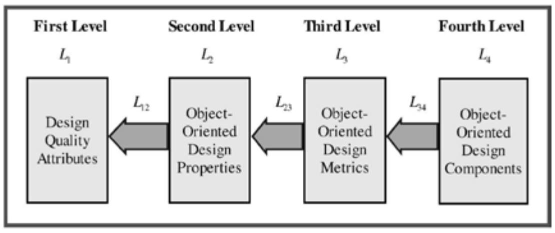
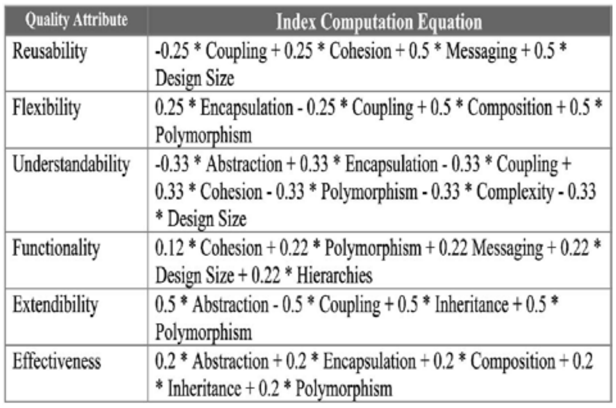

First Level
L_1
Design Quality Attributes
Second Level
L_2
Object-Oriented Design Properties
Third Level
L_3
Object-Oriented Design Metrics
Fourth Level
L_4
Object-Oriented Design Components
L_{12}
L_{23}
L_{34}

Figure 3: Levels and links



Quality Attribute	Index Computation Equation
Reusability	-0.25 * Coupling + 0.25 * Cohesion + 0.5 * Messaging + 0.5 * Design Size
Flexibility	0.25 * Encapsulation - 0.25 * Coupling + 0.5 * Composition + 0.5 * Polymorphism
Understandability	-0.33 * Abstraction + 0.33 * Encapsulation - 0.33 * Coupling + 0.33 * Cohesion - 0.33 * Polymorphism - 0.33 * Complexity - 0.33 * Design Size
Functionality	0.12 * Cohesion + 0.22 * Polymorphism + 0.22 Messaging + 0.22 * Design Size + 0.22 * Hierarchies
Extendibility	0.5 * Abstraction - 0.5 * Coupling + 0.5 * Inheritance + 0.5 * Polymorphism
Effectiveness	0.2 * Abstraction + 0.2 * Encapsulation + 0.2 * Composition + 0.2 * Inheritance + 0.2 * Polymorphism

Table 2: Computation formulas for quality attributes

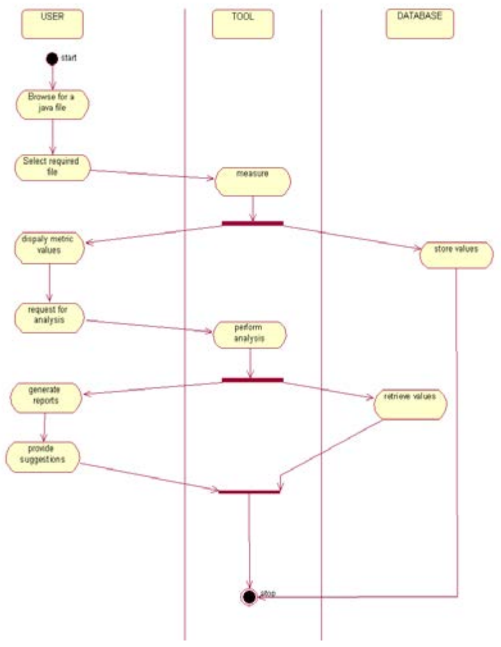


Figure 4: Activity Diagram for OODMAJ Tool

Empirical Data Collection

Case Study1:

We consider five Java projects [10] mentioned below that are licensed as GNU open source from various domains :

Project 1:BLACKDUCKKODERS (http://www.koders.com/):10versions chosen
Project 2: STRAR UML-One of the UML tool to design UML diagrams (http://www.osalt.com/staruml): 5.0 versions chosen.
Project 3: OpenOffice Draw 3.0 (http://www.openoffice.org/product/draw.html): 3.0 versions chosen
Project 4: InfraRecorder 0.50 (http://infrarecorder.org/):0.5 versions chosen
Project 5: Gimpshop 2.2.11 (http://plasticbugs.com/?page_id=294):2.2.11 versions Chosen

Case Study 2:

To analyze the metrics chosen for this work, their values are computed for ten different systems. These systems are developed by M.Tech Students. The following relevant data was collected:
1. The design and source code of the java programs and
2. The faulty data found by the testing team.

The 10 systems under study consists of 200 classes out of which 130 are system classes and 70 are standard library classes available in java language. These classes contain functions to manipulate files, strings, lists, hash tables, frames, windows, menus, threads, socket connection etc. All metric values are computed on system classes whereas coupling and inheritance metrics are also calculated between 'system classes' and 'standard library classes'.

Observations

It was observed during testing on both the Case Studies the classes coupled with standard library classes were less fault prone than those coupled with system classes.

3. RESEARCH METHODOLOGY

In this section, we review the research methodology that investigate the relationship between object-oriented metrics and fault-proneness. The product metrics cover the following dimensions: coupling, cohesion, inheritance, and complexity. Coupling metrics characterize the static usage dependencies among the classes in an object-oriented system [6]. Cohesion metrics characterize the extent to which the methods and attributes of a class belong together[7]. Inheritance metrics characterize the structure of the inheritance hierarchy.

Logistic Regression Model:

Logistic Regression (LR) model is the most widely used technique in literature to predict dependent variable from set of independent variables. In our work independent variable are OO metrics and dependent variable is fault proneness. LR is of two types: (i) Univariate LR (ii) Multivariate LR Univariate LR is a statistical method that formulates a mathematical model depicting relationship among each independent variable and dependent variable to determine if the measure is statistically related, in the expected direction, to fault proneness. Multivariate LR is used to construct a prediction model for the fault-proneness of classes. In this method combination of metrics are used to determine the effect on dependent variable.

In our research we used univariate logistic regression model. The general form of an LR model is:

$$\pi = \frac{1}{1+e^{-\left(\beta_0+\sum_{i=1}^{k}\beta_i x_i\right)}}, \qquad (1)$$

where π is the probability of a class having a fault, and the xs are the independent variables. In a univariate analysis only one xi, x1, is included in the model, and this is the product metric that is being validated:

$$\pi = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1)}}. \qquad (2)$$

When controlling for size, a second xi, x2, is included that measures size:

$$\pi = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1+\beta_2 x_2)}}. \qquad (3)$$

The magnitude of an association can be expressed in terms of the change in odds ratio as the x1 variable changes by one standard deviation. The odds ratio is a measure of association. The odds of an event, is the ratio of the number of ways the event can occur to the number of ways the event cannot occur. The odds ratio is the ratio of faulty classes and non-faulty classes. If a metric is not related to fault-proneness, then the odds ratio is equal to one. If there is a positive association, then the odds ratio will be greater than one, and if there is a negative association, then the odds ratio will be a fraction. Let D denote the presence of a fault (D . 1) or absence (D . 0), and let x be our coupling metric. Then,

$$\Pr(D=1|x) = \frac{1}{1+e^{-(\beta_0+\beta_1 x)}} \qquad (4)$$

is the probability of a fault given the value of x. The probability of there not being a fault given the value of x is:

$$q_x = 1 - p_x =$$
$$\Pr(D=0|x) = 1 - \frac{1}{1+e^{-(\beta_0+\beta_1 x)}} = \frac{e^{-(\beta_0+\beta_1 x)}}{1+e^{-(\beta_0+\beta_1 x)}}. \qquad (5)$$

The odds of a class having a fault given the value of x is

$$\frac{p_x}{q_x} = \Psi(x) = e^{\beta_0+\beta_1 x}. \qquad (6)$$

The odds of a class having a fault if the product metric value x is increased by one standard deviation is:

$$\Psi(x+\sigma) = e^{\beta_0+\beta_1(x+\sigma)}. \qquad (7)$$

The change in odds by increasing the value of x by one standard deviation is:

$$\Delta\Psi = \frac{\Psi(x+\sigma)}{\Psi(x)} = e^{\beta_1\sigma}. \qquad (8)$$

In this subsection we find the relationship of independent variables (OO metrics) with dependent variable (fault proneness). Univariate LR analysis is done on 85 system classes. The table 2 provides the coefficient (B), standard error (SE), statistical significance (sig), R2 statistic and odds ratio (exp(B)), for each measure. Metrics with no variance or lower variance are excluded from the table. The metrics with a significant relationship to fault proneness, that is, below or at the significance (named as Sig. in Table 2) threshold of 0.05 are shown in bold (see Table 2). The metrics that are not shown in bold do not have a significant relationship with fault proneness.

Table 3: Statistical results for fault proneness.

2) Metric	3) B	S.E.	Sig.	R2	Exp(B)
CBO	0.8436	0.2802	0.0026	0.1206	2.3246
CBO1	0.6180	0.2491	0.0131	0.077	1.8553
LCOM1	0.0612	0.0244	0.0121	0.2155	0.0631
LCOM2	0.0800	0.0347	0.0212	0.1982	1.0832
DIT	0.7518	0.4279	0.0789	0.0344	0.4715
NOC	0.3147	0.2666	0.2379	0.0172	1.3698
LOC	0.0100	0.0033	0.0025	0.273	1.0101
RFC	0.1817	0.041	0.0000	0.536	1.1993

Volume 2 No.4, APRIL 2011 ISSN 2079-8407
Journal of Emerging Trends in Computing and Information Sciences
©2010-11 CIS Journal. All rights reserved.
http://www.cisjournal.org

		0			
WMC	0.2466	0.0646	0.0001	0.375	1.2796
OCAEC	0.0731	0.2552	0.7746	0.0000	1.0758
OCAIC	0.9381	0.3594	0.0090	0.077	2.5552

Principal Component Method:

Principal Component Method(PCM) is a mathematical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of uncorrelated variables called **principal components. PCM** is used to maximize the sum of squared loadings of each factor extracted in turn. The PC Method aims at constructing new variable (Pi), called Principal Component (PC) out of a given set of variables *Xj*' *s*(*j* = 1,2,...., *k*).The variables with high loadings help identify the dimension P.C. is capturing, but this usually requires some degree of interpretation. In order to identify these variables, and interpret the PC.s, we consider the rotated components. As the dimensions are independent, orthogonal rotation is used. There are various strategies to perform such rotation. We used the varimax rotation, which is the most frequently used strategy in literature. Eigenvalue (or latent root) is associated with each PC. It refers to the sum of squared values of loadings relating to dimension, and then the sum is referred to as eigenvalue. Eigenvalue indicates the relative importance of each dimension for the particular set of variables being analyzed. In our study, the PC.s with eigenvalue greater than 1 is taken for interpretation.

The coupling of system classes to system classes is counted separately from coupling of system classes to standard library classes. SL is suffixed with the metric name when coupling to standard library classes is counted. For instance CBO metric in such case is named as CBO_SL. The PC extraction method and varimax rotation method is applied on all metrics. The rotated component matrix is given in Table 3. The values above 0.7 (shown in bold in Table 3) are the metrics that are used to interpret the PC.s. For each PC., we also provide its eigenvalue, variance percent and cumulative percent. The interpretations of PCs are given as follows:

• **P1:** CBO_SL, OCAIC_SL, OCMIC_SL, CBO1_SL and OMMIC_SL measure coupling from standard library classes.

• **P2:** LCOM1, LCOM2, WMC and OCMIC. This dimension includes coupling, cohesion and size metrics. This indicates that import coupling and cohesion metrics have correlation with size.

• **P3:** OMMIC, RFC are coupling metrics. These metrics count import coupling from system classes through method invocations.

• **P4:** AMMIC_SL, OCAIC are import coupling metrics.

• **P5:** CBO, CBO1 are coupling metrics that count both import and export coupling.

• **P6:** NOC is an inheritance metric that counts number of children of a class.

Table 4: Results for Principal Component Method

PC	P1	P2	P3	P4	P5	P6
Cumulative%	32.608	44.97	56.010	63.676	70.424	75.603
Variance %	32.608	12.3	11.03	7.665	6.748	5.1788
Eigenvalue	6.84	2.59	2.31	1.60	1.41	1.08
CBO	0.12	0.00	0.18	0.14	0.91	-0.05
CBO_SL	0.80	0.15	0.07	0.37	0.12	0.16
CBO1	0.03	-0.03	0.18	-0.11	0.94	-0.01
LCOM1	0.28	0.87	0.26	0.06	-0.07	0.01
LCOM2	0.28	0.88	0.21	0.01	-0.08	0.00
DIT	-0.25	-0.14	0.36	-0.29	-0.28	-0.25
NOC	0.10	-0.07	0.17	-0.04	-0.08	0.80
LOC	0.27	0.41	0.68	0.02	0.05	0.17
RFC	0.20	0.34	0.76	0.15	0.07	0.17
WMC	0.35	0.74	0.49	0.16	0.01	0.17
OCAEC	0.48	-0.00	-0.04	0.41	-0.09	-0.41
OCAIC	0.10	0.19	0.08	0.74	0.04	0.39

4. CONCLUSION

In real-life systems, faults can differ significantly in their impact on the operation of a software system. It would be valuable to use OO design metrics to help to identify the fault-proneness of classes when the severity of fault is taken into account. Previous studies have raised the need to validate OO design metrics across different fault severities. However , little is currently known on this subject. Our study attempts to fill this gap by empirically validating OO design metrics for different fault severities. In this study we first find the interrelationships among selected metrics and then found the individual and combined effect of selected metrics on fault proneness. The results of univariate LR analysis show that most of the import coupling and cohesion metrics are found related to fault proneness. On the other hand inheritance metrics were not found related to fault proneness . We are also applied Principal component method to these metrics to get the Fault proneness. The number of dimensions captured in PC analysis is much lower than the number of metrics. This simply supports the fact that many of the metrics proposed are based on comparable ideas and therefore provide somewhat redundant information. It was observed during testing the classes coupled with standard library classes were less fault prone than those coupled with system classes.

ACKNOLEDGEMENTS

The authors would like to thank Ellenki College of Engineering & Technology‘s Research & Development Center for their cooperation and support in data collection.

REFERENCES

[1] K.El Emam, W. Melo, J. Machado, “The Prediction of Faulty Classes Using Object-Oriented Design Metrics”, *Journal of Systems and Software*, vol. 56, 63-75, 2001.

[2] S.Chidamber and C.Kemerer, “A metrics Suite for Object-Oriented Design *”,IEEE Trans. Software Engineering*, vol. SE-20, no.6, 476-493, 1994.

[3] M-H. Tang, M-H. Kao, and M-H. Chen, ªAn Empirical Study on Object Oriented Metrics,º Proc. Sixth Int'l Software Metrics Symp., pp. 242-249, 1999.

[4] L. Briand, J. Wuest, J. Daly, and V. Porter, ªExploring the Relationships Between Design Measures and Software Quality in Object-Oriented Systems,º J. Systems and Software, vol. 51, pp. 245- 273, 2000.

[5] L. Briand, J. Wuest, S. Ikonomovski, and H. Lounis, ªA Comprehensive Investigation of Quality Factors in Object-Oriented Designs: An Industrial Case Study,º Technical Report ISERN-98-29, Int'l Software Eng. Research Network, 1998.

[6] L. Briand, J. Daly, and J. Wuest, ªA Unified Framework for Coupling Measurement in Object-Oriented Systems,º IEEE Trans. Software Eng., vol. 25, no. 1 pp. 91-121, Jan. 1999.

[7] L. Briand, J. Daly, and J. Wuest, ªA Unified Framework for Cohesion Measurement in Object-Oriented Systems,º Empirical Software Eng., vol. 3, pp. 65-117, 1998.

[8] L. Dales and H. Ury, ‘An Improper Use of Statistical Significance Testing in Studying Co-variables,’ International Journal of Epidemiology, vol. 7, no. 4, pp. 373-375, 1978.

[9] N. Fenton, ªSoftware Metrics: Theory, Tools and Validation,’ Journal of Software Engineering’., pp. 65-78, Jan. 1990.

[10] Amjan Shaik, C. R. K. Reddy, Bala Manda, Prakashini. C, Deepthi. K” An Empirical Validation of Object Oriented Design Metrics in Object Oriented Systems” International Journal of Emerging Trends in Engineering and Applied Sciences (IJETEAS) 1 (2): 216-224 (ISSN: 2141-7016).

[11] Amjan Shaik, C. R. K. Reddy, Bala Manda, Prakashini. C, Deepthi,” Metrics for Object Oriented Design Software Systems: A Survey ”International Journal of Emerging Trends in Engineering and Applied Sciences (IJETEAS) 1 (2): 190-198 (ISSN: 2141-7016).

ABOUT THE AUTHORS

.

Amjan Shaik is working as a Professor and Head, Department of Computer Science and Engineering at Ellenki College of Engineering and Technology (ECET), Hyderabad, India. He has received M.Tech. (Computer Science and Technology) from Andhra University. Presently, he is a Research Scholar of JNTUH, Hyderabad, India. 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 Engineering, Software Metrics, Software Testing , Software Quality and Object Oriented Design.

Prof. Dr. N. Satyanarayana is Working as a Principal and Professor of CSE at Nagole Institute of Technology and Science (NITS), Hyderabad, India. He has received M.Tech.(Computer Science and Engineering) from Indian School of Mines (ISM),

Dhanbad and Ph.D in Computer Science and Engineering from Acharya Nagarjuna University(ANU), Guntur. He has widely published and presented Research and Technical Papers in International Journals, National Journals, International Conferences and National Conferences. He is a distinguished academician and administrator . At Present 8 Research Scholars are doing Ph.D under his esteemed guidance in various Universities. His main research Interests are Software Architectures, Software Metrics. Networking, Communication Systems, Security and Data Mining .

Prof. Dr. C.R.K. Reddy is working as a Professor and Head, 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 Conferences, and National , International Journals. 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.

Mohammed Huzaifa is working as an Assistant Professor , Department of Information Technology at Muffakhamjah College of Engineering and Technology(MJCET), Banjarahills, Hyderabad, India. He has received M.Tech (Software Engineering) from JNTUH Hyderabad. He has presented number of technical papers in International and National Conferences. His research interests are Data Mining, Information Security and Software Engineering.

Mohd Zainuddin Naveed is working as an Assistant Professor , Department of Computer Science and Engineering at Muffakhamjah College of Engineering and Technology (MJCET), Banjarahills, Hyderabad, India. He has received M.Tech (CSE) from Shadan College of Engineering and Technology, Affiliated to JNTUH Hyderabad . He has presented number of technical papers in International and National Conferences. His research interests are Data Mining, Information Security and Software Engineering.

Nazeer. Shaik is working as an Assistant Professor , Department of Computer Science and Engineering at Moghal College of Engineering and Technology (MCET), Hyderabad, India. He has received M.Tech (CSE) from Bharth University, Chennai. He has presented number of technical papers in National Conference. His research interests are Software Engineering, Software Project Management, Computer Networks and Mobile Computing.

S.V. Achuta Rao is working as a Professor and Head, Department of CSE and IT at DJR Institute of Engineering and Technology (DJRIET), Vijayawada, India. He has received M.Tech. (Computer Science and Engineering) from JNTU, Kakinada, India. Presently, he is a Research Scholar of Rayalaseema University (RU), Kurnool, India. He has been published and presented good number of Research and technical papers in International and National Conferences. His main research interests are Data Mining, Networking, Image Processing, Software Engineering and Software Metrics.