International Journal of Civil Engineering and Technology (IJCIET)
Volume 9, Issue 1, January 2018, pp. 889–899, Article ID: IJCIET_09_01_087
Available online at http://http://www.iaeme.com/ijciet/issues.asp?JType=IJCIET&VType=9&IType=1
ISSN Print: 0976-6308 and ISSN Online: 0976-6316

PREDICTION OF SOFTWARE DEFECTS USING OBJECT-ORIENTED METRICS

Pooja U
Department of Computer Science,
CHRIST (Deemed to be University), Bengaluru, India

Nizar Banu PK
Department of Computer Science,
CHRIST (Deemed to be University), Bengaluru, India

ABSTRACT

In recent years, many of the object-oriented software metrics were proposed for increasing the quality of software design such as prediction of defects and the maintainability of classes and methods. As the word metrics is frequently used for specific measurements taken on a particular process or item and in object-oriented metrics the metrics are the unit of measurements that is used to characterize the data.The fundamental point of this research is to identify the significance difference between software metrics which observes defect prediction and also study about their relation involving in the object oriented metrics that is named as "Chidamber and Kemerer metric suite" which is also known as "CK metrics suite", the number of defects and then finally decide the differences of the metrics in ordering to Eclipse classes as defective and selected with regard to defect prediction.

Key words: Software defects, CK metrics, object-oriented metrics.

Cite this Article: Pooja U and Nizar Banu PK, Prediction of Software Defects using Object-Oriented Metrics. *International Journal of Civil Engineering and Technology*, 9(1), 2018, pp. 889-899.
http://www.iaeme.com/IJCIET/issues.asp?JType=IJCIET&VType=9&IType=1

1. INTRODUCTION

Object oriented metrics helps to create and design model through the testing process. In addition object-oriented metrics also provides a quantitative method to get access with the nature of the item and their interior qualities of the item, as it permits the software designer to give an opportunity to approach the quality of the product when product is made-up. Metrics are the important cause of data that helps the developer to improve the design and helps to develop good software [1].

Software metrics assumes an essential part in developing software products and making them more valuable. Practical support is important to exhibit the estimation of a metric in useful applications. Software metrics are usually distinguished by the software manufacturing

items (like: plan, source code, and testing), software production development (such as investigation, plan and coding) and software designing individuals (like the proficiency of a character analyzer or the efficiency of a character planner) [2].

Object-oriented is a grouping approach that can categorize the issues regarding objects and it can give numerous paybacks on dependability, flexibility, durability and disintegration as difficult problems into effectively reasonable to some objects and giving various upcoming improvements as object-oriented metrics are used to solve and also to predict the quality of the software. Object oriented design metric is a significant division of software development. The most important purpose of object-oriented metrics is to develop the class and effectiveness of software after analyzing the defects. The defects of software can be related to the assistance of object-oriented design metric at the design period of the software. Identification and elimination of defects prior to the client conveyance is basic in the growth of the software. Hence discovering the relation in terms of defects with different estimation measurements during the time spent in software growth becomes necessary [3].

Software defects are some errors or bugs in a program. The defect depends on two factors like size and complexity of the software that can be measured in two ways namely defect density and failure density. Defect density can be premeditated by total number of defects found in each thousand of line of program source code. Failure density can be premeditated by total number of distinguished failures per thousand line of code [4]. Section 2 explains about the background of the object-oriented metrics and parameters of CK metrics, Section 3 explains about methodology Section 4 explains about Experimental results carried out is this work. Section 5 concludes this paper.

2. BACKGROUND

In this division we are discussing about the previous literature surveys related to our research topic. The various parameters of object-oriented metric provides method for calculating various features like area, difficulty level, execution and quality and to convey productive metrics which is effectively used to measure the programs [5]. Bug prediction dataset is collected and analyzed for predicting defects.

The object-oriented software are effectively implemented as user friendly based on the customer requirements but an object-oriented design model is used to increase the size and complexity and also object-oriented design that benefit both the experienced designer and the beginner. In detail there are nine different and determinate physical characteristics in object-oriented design as they are complexity, coupling, sufficiency, completeness, cohesion, size, primitiveness, similarity, volatility. But as in this research the class-oriented metrics is also called as CK metrics suite and in CK metrics the measurable characteristics are size, coupling, cohesion, inheritance.

The significance of different process of software metrics are correlated with number of defects and the correlation analysis investigates which metrics were filtered out and also which were selected in the model [6]. The chances of error identification was considered to be the most prominent method of error prone hence, more acceptable to predict the defect of the software [7].

Chidamber and Kemerer presented the OO measures of design and named it as CK metric [8]. An efficient level of knowledge was generated using this metric and are considered to be the popular measurement suite for OO design [9]. CK metrics presents six metrics those are given below in detail

WMC - Weighted Method per class: WMC provides the quantity of the summation of the complication of every method in class. The particular complexity metrics that can be chosen will be normalized as the nominal complexity method. This metric interprets the details such as time consumed and number of methods that are practically measures the work needed for development and maintenance of the class. Quality can be predicted by analyzing the results produced by WMC and are able to predict the quality. High value of WMC yields more errors where the quantities of the bugs are more and fail in the quality [10].

DIT - Depth of inheritance Tree: The maximum venture from the class node to root node of the tree is considered as the class depth and is measured by the predecessor class. As DIT increases when lower-level classes inherit many methods and this leads to possible difficulties when there is an effort to predict the behavior of a class. When the tree with highest depth is considered to have more number of faults and more methods that should be reused [10].

NOC - Number of Children: NOC is the subclasses that are immediately derived from the base class in the class hierarchy are termed as its children. As the number of children grows, the reuse increases but also NOC increases, as this concept is represented as the parent class can be weak when some of the children are not suitable members of the parent class. The width of the hierarchy of a class is measured by NOC. Rise in NOC indicates high reuse of the base class since base class needs high testing and parent class abstraction which is not proper and also increases the amount of testing [10].

CBO - Coupling between Objects: The quantity of different classes that are coupled is the CBO of a class. Two classes are said to be coupled when techniques are expressed in one class that utilize factors characterized by alternate class. Various entries to the single class are considered one access. Just variable reference and method call are counted. The raise of CBO shows the reusability of the class can decrease and additionally an increase in coupling indicates the testing and the modifications can be done to find the fault proneness [10].As in general the CBO values for each class will be low as reasonable.

RFC - Response for a class: The RFC tells the arrangement of all which are conceivably executed in respond of a message that is got by an object class. This incorporates all the methods available inside the class order. Since RFC expands the overall design complexity of the class increments and turns out to be difficult to interpret [10].

LCOM - Lack of Cohesion of methods: LCOM calculates the disparity of the methods in a class by occurrence factors or qualities utilized by techniques. A high union shows great class division. LCOM contains the record of the quantity of techniques whose similarity is not equal to zero. If the LCOM is high then methods might be coupled to one another through traits and after that, class configuration is high in complexity [10].

3. METHODOLOGY

The methodology used in this paper is shown in Figure 1.

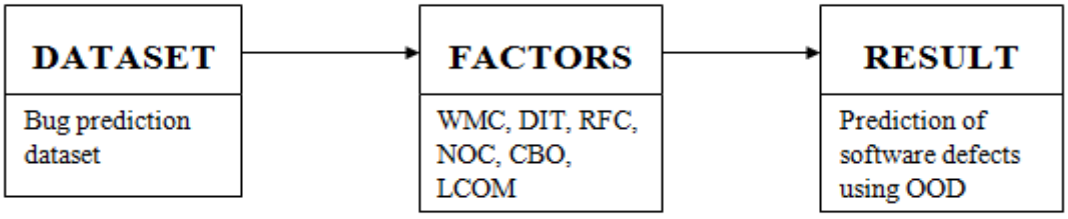

Figure 1 Overview of the research work

Parameters of CK Metrics

The parameters of CK metrics are discussed in this section

3.1. WMC- Weighted Methods per Class

WMC is the addition of complexity for all methods in a class.

WMC = Number of Methods (NOM), where all the methods complexity are viewed as UNITY.

WMC focuses on the following:

- The quantity of techniques and the complication of methods included are to display time and effort is important to create and sustain the class.
- The improved quantity of methods in a class. The superior of the class in the possible effect on children, since children will acquire every methods characterized by the class.
- Classes with substantial quantities of methods are more application particular, restricting the opportunity of reuse [11].

For instance, In Figure 2, WMC for Account is 3 (consider method of each class complexity to be unity)

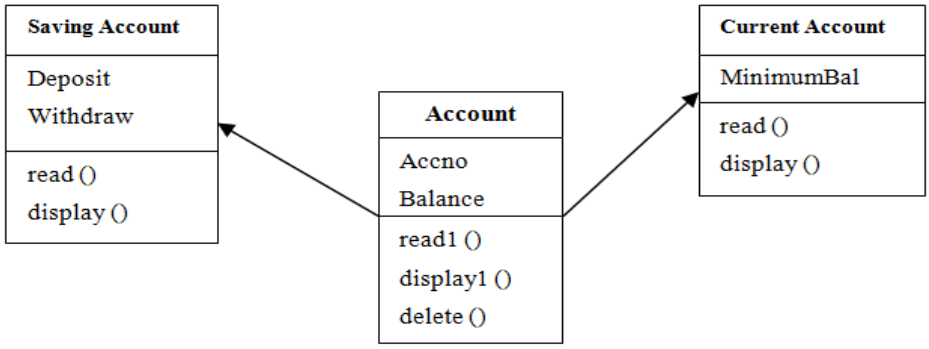

Figure 2 Class diagram of Account Information system.

3.2. RFC- Response for a Class

RFC is calculated by adding the number of methods in the response set. The response set of a class is "a set of methods that can probably be executed in response to a message received by an object of that class". The complexity of a class is calculated in terms of number of methods.

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

RFC focuses on the following:

- If a huge number of methods can be invoked in response to a message, the error detection and testing of the class makes more difficult. Because it needs a higher standard of interpretation on the testing part.
- A most horrible scenario rate for probable responses will support in suitable designation of testing time.

For instance, In Figure 2, class Account has two capacities read1 and display1 which call methods

Saving Account::read (), Current Account::read (), Saving Account::display (), Current Account::display ().

RS = {Account :: read1, Account :: display1,Account :: delete} ∪{Saving Account :: read, Saving Account :: display}∪{Current Account :: read, Current Account :: display}

RFC = 7

3.3. DIT- Depth of Inheritance Tree

DIT = depth of the class in the inheritance tree.

The depth of a node of a tree refers to the length of the maximal way from the node to the tree [12].

DIT focuses on the following:

- The deeper a class is its hierarchy, the larger the method of quantity is expected to inherit, making it more difficult to know its performance.
- Deeper trees represent better complexity design, because of more classes and methods are concerned.
- The deeper a specific class is in the chain of importance, the more prominent the likely reuse of acquired techniques.

For instance, In Figure 3, DIT for Vehicle class is 2 as it has 2 predecessor classes Bus/Car and Transportation. DIT for Bus and Car is 1 as it has one predecessor class Transportation.

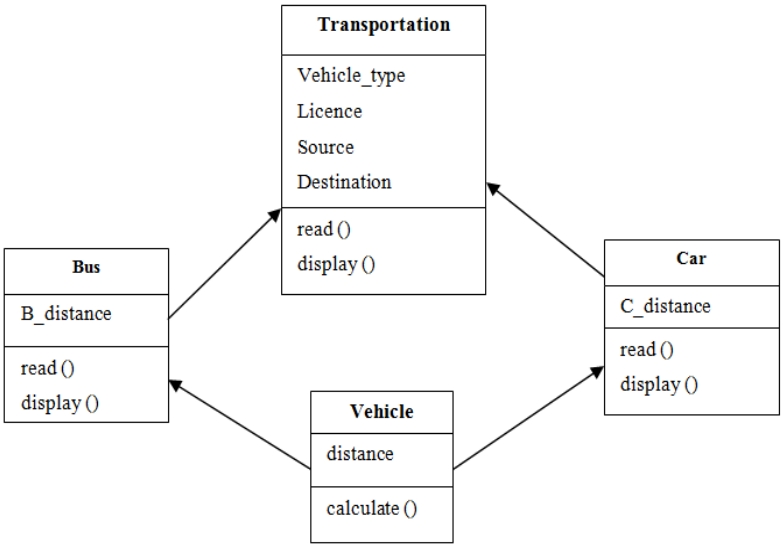

Figure 3 Class diagram of Transportation.

3.4. NOC- Number of Children

NOC is a quantity of instantaneous subclasses which will be lesser to a class in hierarchy.

NOC focuses on the following:

- When an NOC begins in a class it increments the reuse but the abstraction might be weakened.
- Depth is normally improved compared to breadth in class progression, because it advances reprocess of methods by the Inheritance.
- NOC gives a thought of prospective authority for a class that may have on the design.
- Class which has huge number of children needs more testing.

For instance, In Figure 3, NOC value for class Transportation is 2.

3.5. CBO- Coupling between Objects

CBO is a count of the amount of preceding classes to which it is coupled for a class.

CBO focuses on the following:

- CBO is the quantity of combination among two different class
    - As association expands, recycle of that class reduces.
    - More number of classes used by a certain classes characterize coupling of different objects and classes thus they are disagreeable.
    - More number of inputs of classes signifies better design objects and elevated point of reusable.
    - Non potential to sustaining more number of inputs of classes and less number of classes used by a certain classes is over the complete framework.

For instance, In Figure 4, Account class contains validations of samples of the classes Saving Account and Current Account. The Account class designates its Saving Account and Current Account problems to instances of the Saving Account and Current Account classes. The importance of metric CBO for class Account is 2 and for class Saving Account and Current Account is zero.

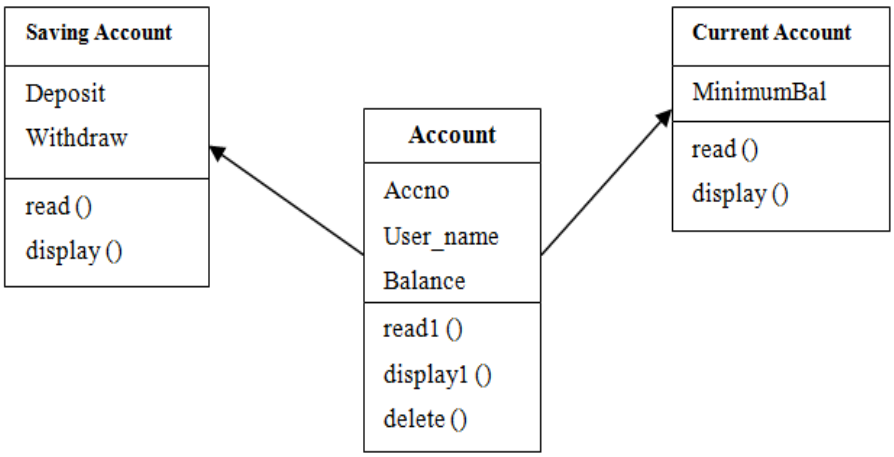

Figure 4 Representation of User details

3.6. LCOM- Lack of Cohesion in Methods

The difference between methods in a class through instanced factors is estimated.

LCOM focuses on the following:

- Combination of methods inside a class has become popular thus it advances encapsulation.
- Shortage of cohesion involves classes must possibly divide to more than two sub-classes.
- Any calculation of divergence of methods leads to recognize faults based on the classes in the design.
- Less cohesion expands difficulty level, by this means growing the probability of bugs during the improvement process.

For instance, In Figure 5, there are three methods read1 (), display1 (), delete () in class Account, where I1, I2, I3 and I4 are instance variable used by these methods.

I1= {Accno, User_name, Balance, Type, Date}, I2= {Accno}, I3= {Balance}, I4= {User_name} where $I_1 \cap I_2$, $I_2 \cap I_3$ and $I_3 \cap I_4$ are non - null but $I_2 \cap I_3$, $I_2 \cap I_4$ and $I_3 \cap I_4$ are null sets. LCOM will be is 0 in this case [|P|=|Q|=3].

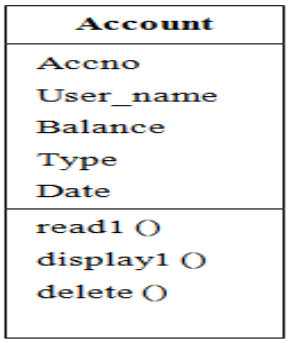

Figure 5 Class diagram of Account class in Bank Information.

4. EXPERIMENTAL RESULT

4.1. Dataset

Table 1 shows the bug prediction dataset that is taken from website. It contains source code metrics gathered from a few frameworks to help experimental investigations on source code progression. The dataset includes the information on the growth of the java-based systems and the bug prediction dataset is from Eclipse classes [13].

Table 1 Dataset of the object oriented metrics

EntityName	LOC	WMC	DIT	NOC	LCOM	CBO	RFC
org::eclipse::jdt::core::dom::ArrayType	150	23	3	0	9	10	38
org::eclipse::jdt::internal::compiler::util::WeakHashSet	135	32	1	0	7	6	20
org::eclipse::jdt::internal::core::builder::QualifiedNameSet	44	14	1	0	1	3	7
org::eclipse::jdt::internal::core::JavaModelManager::PerProjectInfo	45	8	1	0	0	8	8
org::eclipse::jdt::internal::core::PackageFragmentInfo	36	6	3	1	0	5	7
org::eclipse::jdt::internal::core::search::matching::VariablePattern	14	2	4	2	0	1	4
org::eclipse::jdt::internal::core::util::HandleFactory	283	59	1	0	1	44	69
org::eclipse::jdt::internal::eval::VariablesInfo	38	7	1	0	0	3	4
org::eclipse::jdt::internal::compiler::env::NameEnvironmentAnswer	60	10	1	0	18	4	7
org::eclipse::jdt::internal::core::ParameterizedBinaryType	15	3	8	0	0	3	7
org::eclipse::jdt::internal::codeassist::InternalCompletionProposal	187	48	1	1	0	10	28
org::eclipse::jdt::internal::compiler::ast::MagicLiteral	12	3	7	3	1	0	3
org::eclipse::jdt::core::dom::PrimitiveType	111	17	3	0	5	13	35
org::eclipse::jdt::internal::core::search::IndexSelector	124	26	1	0	1	20	42
org::eclipse::jdt::internal::core::ImportDeclaration	76	15	4	0	18	8	26

Bug prediction dataset from Eclipse classes are shown in Table 1 and parameters that predict the defects are given below.

- LOC-Lines of code
- WMC-Weighted method per class
- NOC-Number of classes
- RFC-Response of a class
- LCOM-Lack of cohesion of methods
- CBO-Coupling between object
- DIT-Depth of Inheritance

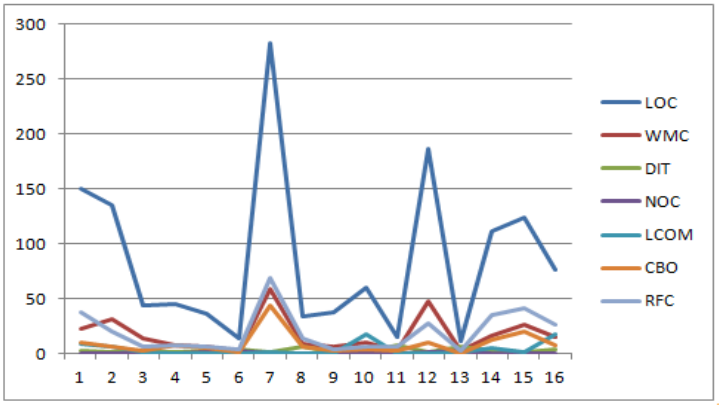

Figure 6 Defects of the object-oriented metrics

Figure 6 represents the defects of the sample dataset of object-oriented metrics.

4.2. Implementation

The actual suite of CK metrics has six metrics as mentioned above are WMC, DIT, LCOM, CBO, RFC and NOC .The lack of relevance of different metrics in the CK suite to our representation and the possible complexity in processing these procedures lead us to eliminate them.

To achieve the above purpose the data chosen from an open source code in object oriented program. We have taken some sample classes of Eclipse compiler and other tools of java and then variance is analyzed for that data and then based on the result we have predicted the significant difference of defects with the help of parameters of CK metrics in object oriented metrics.

4.2.1. Test Procedure

The test procedure measures the means between groups and within groups and also controls any of individual means are statistically changed from each other.

H_0: $\mu 1=\mu 2=\ldots..=\mu k$

H_1: $\mu 1\neq\mu 2\neq\ldots..\neq\mu k$

Where μ is group mean, k is number of groups

In this case,

H_0: There is no significant difference of defects in the metrics

H_1: There is a significant difference of defects in the metrics.

When H_0 is rejected, we accept alternative hypothesis that is H_1.

ANOVA: Single factor

The single factor analysis of variance that means ANOVA is used to calculate any statistical modifications among the means of two or more independent samples.

ANOVA uses F-test to test the equality of means.

ANOVA Single factor

Table 2 Variance of analysis: single factor is calculated.

Groups	Count	Sum	Average	Variance
LOC	16	1364	85.25	5666.466667
WMC	16	282	17.625	273.7166667
DIT	16	47	2.9375	6.0625
NOC	16	7	0.4375	0.795833333
LCOM	16	61	3.8125	38.29583333
CBO	16	144	9	111.8666667
RFC	16	319	19.9375	346.0625

In Table 2 summary of variance of analysis, count, sum and average is calculated.

Source of Variation	DF	SS	MS	F
Regression	1	$SSR = \sum_{i=1}^{n} (\hat{y}_i - \bar{y})^2$	$MSR = \frac{SSR}{1}$	$F^* = \frac{MSR}{MSE}$
Residual error	n-2	$SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$	$MSE = \frac{SSE}{n-2}$	
Total	n-1	$SSTO = \sum_{i=1}^{n} (y_i - \bar{y})^2$		

Figure 7 Calculating the methods for defect prediction.

Figure 7 is used for calculating the methods of ANOVA by using those formulas [14].

In Table 3 single factor Variance of analysis is calculated by this result we can prove that our assumption of test procedure is right or wrong.

Where,

SS – Sum of square

df – Degree of freedom

BGS-Between Groups (row – 1)

WGS-Within Groups (N – row)

Total (N-1)

MSS – Mean sum of square

MSB-Between Groups (SSB/df)

MSW-Within Groups (SSW/df)

F cal – Calculated value (MSB/MSW)

F tab – Calculated by table (when $\alpha = 0.05$)

ANOVA

Table 3 Methods for defect prediction

Source of Variation	*SS*	*Df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	85118.714	6	14186.45238	15.412239	1.32E-12	2.1861
Within Groups	96649	105	920.4666667			
Total	181767.71	111				

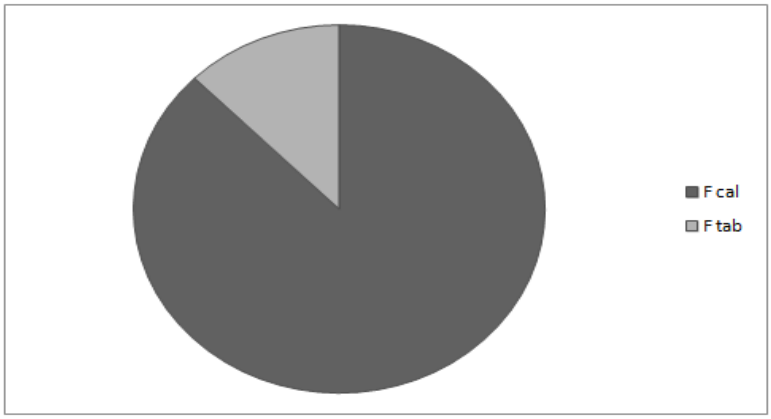

Figure 7 Comparison of F cal and F tab

4.2.2. Inference

As Inference produces the comparing of several groups and as the inference is calculated by taking two kind of variance: between group and within group. If between group variation > within group variation, we reject H_0 and accept H_1 .In this case, if F cal > F tab, we reject H_0 by Figure 7 and 8 shows that 15.4122 > 2.186. Therefore, there is a significant difference of defects in the metrics.

5. CONCLUSIONS

The purpose of this research is to predict defect using CK object-oriented metrics. A suite of metrics has been produced to present the information that will help in the preservation of object-oriented systems. The use of the planned methodology is to predict the significant difference of defects in the metrics. In this research we have taken a sample dataset that is from the bug prediction dataset of source code metrics as the data is based on Eclipse classes and for this data is single factor variance of analysis is calculated and based on the result we can overcome the outcome as in CK metrics we can reject the significant difference of defects in object oriented metrics.

The proposed methodology is only based on the parameters of CK metrics, this can be further implemented to all the object oriented metrics, and this can be implemented for all the classes. This could be the possible future work.

REFERENCES

[1] Aarti Sharma and Manoj Wadhwa, “Empirical Study of Object Oriented Software Metrics for evaluating software defects: CKMOOD suits”, International Journal of Computer Science and Engineering, Vol-3(5) May 2015.

[2] Tibor Gyimothy, Rudolf Ferenc and Istvan Siket, “Empirical Validation of Object-oriented metrics on open source software for fault prediction”, IEEE transaction on software engineering, Vol.31 October 2005.

[3] Michelle Cartwright and Martin Shepperd, “An Empirical investigation of an Object-oriented software system”, IEEE transaction on software engineering, Vol.28 August 2000.

[4] Chen-Huei Chou, “Metrics in Evaluating Software Defects”, International Journal of Computer Application, Vol.63 February 2013.

[5] Gomathi.S, "An overview of Object-Oriented Metrics: A complete survey", International Journal of Computer Science and Engineering Technology (IJCSET), Vol.4 September 2013.

[6] Marian Jureczko, "Significance of different software metrics in defect prediction", Software Engineering: An international journal (SEIJ), Vol.1 September 2011.

[7] Victor R. Basili, Fellow, IEEE, Lionel C. Briand and Walcelio L. Melo, Member, IEEE Computer Society, "A validation of Object-oriented Design Metrics as Quality Indicators", IEEE transactions on Software Engineering, Vol.22 October 1996.

[8] Shyam R. Chidamber and Chris F. Kemerer, "Towards a metric suite for object oriented design", ACM digital library on Object-oriented programming systems, languages and applications, Vol.26 November 1991.

[9] C. Neelamegam and Dr. M. Punithavalli, "A Survey - Object Oriented Quality Metrics", Global Journal of Computer Science and Technology, Pg-No: 183-186, 2011.

[10] Thapaliyal and Garima Verma, "Software defects and Object oriented metrics – An empirical Analysis", International Journal of Computer Applications, Vol.9 November 2010.

[11] P. Ashok Reddy, Dr. K. Rajasekhara Rao, Dr. M. Babu Reddy, "Study of the Efficacy of Various Kinds of Object Oriented Metrics to Ensure Better Understand ability and Maintainability of Object Oriented Design", International Journal of Advanced Research in Computer Science and Software Engineering, Vol.5 April 2015.

[12] Shyam R. Chidamber and Chris F. Kemerer, "A Metrics Suite for Object Oriented Design", IEEE transactions on Software Engineering, Vol.20 June 1994.

[13] http://aserg.labsoft.dcc.ufmg.br/comets/dataset/jdt/metrics/

[14] https://onlinecourses.science.psu.edu/stat501/node/266

[15] M. Karthikeyan and Dr. S. Veni, Software Defect Pre diction Using Improved Support Vector Machine Classifier. International Journal of Mechanical Engineering and Technology, 7(5), 2016, pp. 417–421.

[16] Dr. R. Dillibabu, K. Karnavel And L. Sudha, Predicting Software Defects With Association Mining To Discover Defect Pattern By Using ABDP, International Journal of Information Technology & M anagement Information System (IJITMIS), 4(3), 2013, pp. 136–146.

[17] Dadi Mohankrishna Rayudu, Naresh. E And Dr. Vijaya Kumar B. P. The Impact Of Test-driven Development on Software Defects and Cost: A Comparative Case Study, International Journal of Computer Engineering and Technology (IJCET), 5(2), 2014, pp. 98–107.