Evaluation and Application of Package Level Metrics in Assessing Software Quality

Vinay Singh
Usha Martin Academy,
Ranchi,India

Vandana Bhattacherjee
Birla Institute of Technology,
Ranchi, India

ABSTRACT

Today almost all the software industries are overloaded with the maintenance work of already developed software. When any new demand of software arrives, company matches the new problem with the existing product, so that the new product can be easily developed with some new modification in existing products. The reuse of the existing product is only possible when it is measured accurately and efficiently for a longer period.

In this paper, first we will calculate the class level metrics viz. CBO, RFC, WMC etc for the entire package then we will calculate the average value for each class level metric (selected) by dividing the value for each metric from the total number of classes for each package. The new resultant metrics are named as CBO_{avg}, RFC_{avg}, WMC_{avg} and so on. The importance of these metrics is to accurately measure the complexity at package level. We then map these package level class metrics with the quality attributes and finally validate these metrics upon three open source projects i.e. Jedit, FreeCS and Llamma chart. Data extraction has been done through an automated tool JHawk and analysis of data has been done using SPSS 10.0 as statistical tool.

General Terms

Software Engineering

Keywords

Quality model, Object Oriented Attributes, Quality Attributes, Package metrics

1. INTRODUCTION

Software engineering metrics is the unit of measurement, which is used to characterize software engineering products, processes and people. If used properly they can allow us to identify and quantify improvement and make meaningful estimates. The need for software metrics is indeed real. Just as any other engineering discipline, software engineering needs to be made capable to measure the quality of the product that is being produced and delivered to the customer. This is important since the cost of reducing the risk of a flaw within a program before it is coded and compiled is far lower than the cost that might be involved when recalling a delivered product. An important part that needs to be measured is the Object-Oriented design phase. If one can quantify the design and thereby increase the quality of the design, there is a lower probability of the software being flawed.

Software metrics measure different aspects of software complexity and therefore play an important role in analyzing and improving software quality. Measures of software complexity, for example metrics for coupling or cohesion, provide a means of quantifying its internal quality. Internal quality measures are those, which can be performed in terms of the software product itself, and it will be measureable during and after the creation of the software product. However, it has no inherent, practical meaning within itself. To give meaning to these measures it has to be characterized in terms of the external quality. External quality measures are evaluated with respect to how a product relates to its environment. The reliability, maintainability, testability, efficiency and reuse of a product are some examples. These measures are deemed inherently meaningful. Previous studies have indicated that software metrics can be used to obtain useful information on external quality aspects of software [19] [29].

The recent drive towards Object-Oriented technology forces the growth of Object-Oriented software metrics [1] [4] [15] [18] [22] [23] [24]. The metrics suite proposed by Chidamber and Kemerer is one of the best-known Object-Oriented metrics [27] [28]. Various researchers have conducted empirical studies to validate the Object-Oriented metrics for their effects upon program attributes and quality factors such as development or maintenance effort [7] [20]. Alshayeb and Li predict that Object-Oriented metrics are effective (at least in some cases) in predicting design effort [21]. Chae, Kwon and Bae investigated the effects of dependence variables on cohesion metrics for Object-Oriented programs [8]. Several other researchers have validated Object Oriented metrics for effect of class size with the change proneness of classes [3] [17] [29]. Li theoretically validated Chidamber and Kemerer metrics [28] using a metric evaluation framework proposed by Kitchenham et al and discovered some of the deficiencies of metrics in the evaluation process and proposed a new suite of Object-Oriented metrics that overcome these deficiencies [32]. Vinay and Bhattacherjee have studied the effect of coupling metrics in software defect [31].

The object-oriented elements (attribute) play a key role in measuring the quality of the software. Traditional software metrics that evaluate product characteristics such as size, complexity, performance and quality must be changed to rely on some fundamentally different notions such as encapsulation, inheritance, and polymorphism, which are inherent in object orientation. This has led to the definition of many new metrics to measure product using object oriented approach.

The contribution of metrics to the overall objective of software quality is understood and is fully recognized by the software engineering community in general [16][26] and particularly emphasized by the software quality community [32].

As software applications grow in size and complexity, they require some kind of high-level organization. Classes, while a very convenient unit for organizing small applications, are too finely grained to be used as the sole organizational unit for large applications [25]. This is where packages come in; the purpose of a package is to increase the design quality of large applications. By grouping classes into packages, one can reason about the design at a higher level of abstraction. The main goal while defining a new package structure of a software system is how to partition classes into packages. There are different ways to do this, in [25] Robert C. Martin propose a number of package cohesion principle that addresses the goal of package structuring mentioned above. Martin argues that classes that are reused together should be in the same package and classes that are being affected by the same changes should be in the same package. If a class in a package is reusable then all of the classes in that package must be reusable according to Martin.

Another possible approach of package structuring is to group classes into packages in terms of functionality. For instance, the customer determines a system that releases one version with a given functionality A and some time another functionality B. In this case, each function could put in a package of its own to simplify the constructed and developed processes.

One of the most influencing factors of software systems quality, where metrics can play an important role, is the software coding. Being able to predict some software quality characteristics based on package level design is one of our greatest motivation.

In this paper, the QMPOOD (Quality metric of package level in object-oriented design) metrics were validated by three open source java projects [9] [10] [11]. We are measuring the package level design for predicting class failure-proneness. The rest of the paper is organized as follows: section 2 focuses on the existing quality model, section 3 gives our proposed QM$_P$OOD model, section 4 discusses the results, section 5 presents analysis and discussion and finally section 6 concludes the paper.

2. EXISTING QUALITY MODELS

Software quality is still a vague and multifaceted concept, which means different thing to different people. Typically, the way we measure quality depends on the viewpoint we take [2]. This makes the direct assessment of quality very difficult. In order to better quantify quality, researchers have developed indirect models that attempt to measure software product quality by using a set of quality attributes, characteristics, and metrics. An important assumption in defining these quality models is that internal product characteristics (internal quality indicator) influence external product attribute (quality in use), and by evaluating a product's internal characteristics some reasonable conclusion can be drawn about the product's external quality attributes. Software-metrics advocates frequently adopt this product-based approach because it offers an objective and context independent view of quality [2].

One of the earliest software product quality models was suggested by McCall [13] and his colleagues. McCall's quality model defines software-product qualities as a hierarchy of factors, criteria and metrics and was first of the several models of the same form. An international effort has also led to the development of a standard for software product quality measurement, ISO 9126. All of these models vary in their hierarchical definition of quality, but they share the common difficulty. The models are vague in their definition of the lower level details and metric needed to attain a quantitative assessment of product quality. This lack of specifics in these models offers little guidance to software developers who need to build quality products.

Another difficulty with earlier models was the inability to account for dependency among quality attributes. While several high-level attributes [12] were used to refer to product quality, generally, only a subset of these attributes would be relevant for each different viewpoint, since the influence of individual attributes on overall quality might be contradicting. For example, a quality goal for higher flexibility makes it harder to achieve a goal of lower maintainability.

Product based quality model has been developed by Dormey [5] [6]. The author addresses some of the problems of the earlier models such as McCall's and ISO 9126. Dormey's quality framework, like the earlier models, relies on the decomposition of high-level quality attributes into tangible, quality carrying properties of a product's components (requirement, design and implementation). Dormey's generic quality models are having three principal elements: product properties that influence quality, a set of high-level quality attributes, and a means of linking them [6].

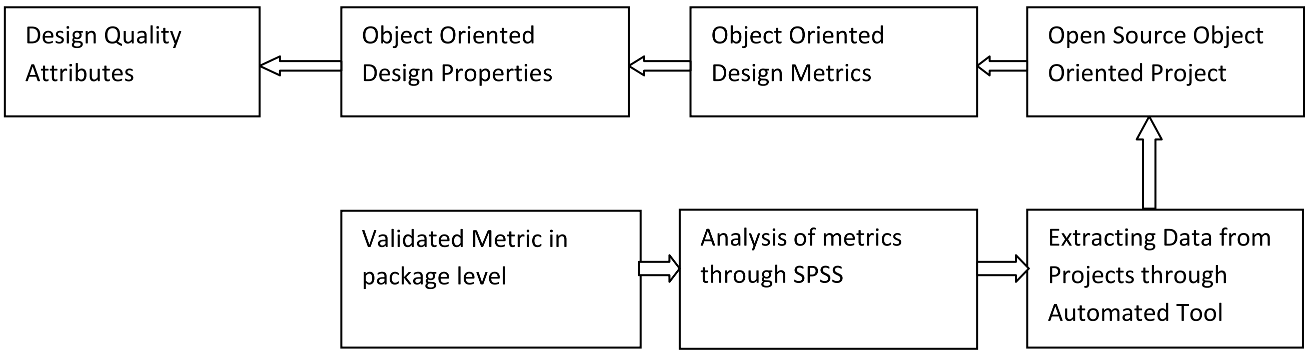
Design Quality Attributes
Object Oriented Design Properties
Object Oriented Design Metrics
Open Source Object Oriented Project
Validated Metric in package level
Analysis of metrics through SPSS
Extracting Data from Projects through Automated Tool

Figure1. QMpOOD Model

Bansiya [14] and his colleagues propose an excellent model in the context of earlier models. Their model defines the new design size metrics, relationships, and evaluates the weights. All the models defined so far measured the software at class level either in design size or code level which is not very significant in predicting the quality of the software. It is to overcome the drawback that we propose our QMpOOD (Quality metric of package level in object-oriented design) model. This model is presented in the next section.

3. MODEL DEVELOPMENT

Recently, a long needed framework for building product based quality model has been developed by Dormey [5], [6]. This quality framework, like earlier models relies on the decomposition of high-level quality attributes into tangible, quality-carrying properties of a product's components (requirements, design, and implementation). The model used in the development of the Quality metric of package level in object-oriented design (QMpOOD) extends Bansiya's quality model for object oriented design and involves the steps shown in Fig 1.

3.1 Identifying Quality Properties

Quality attribute – "Reusability", "flexibility", "understandability", "functionality", "extendibility" and "effectiveness" were selected as the initial set of quality attribute in our model.

Reusability is the process of creating a new product from an existing one without significant effort. The new product can contain the feature of the old one but should also have new features in it. Software is said to be reusable if it is good to understand and less complexity.

Software is said to be flexible if any changes made is adaptable, the ability of a design to be adapted to provide functionally related capabilities.

Understandability is directly related to the complexity of the design structure. Good design of a package and classes tends to be more understandable.

Functionality defines the responsibilities assigned to the classes and packages. Packages having too much functionality are difficult to understand and maintain.

Extendibility allows for the incorporation of new requirements in the design, more flexibilities are tends to be more extendible.

Effectiveness is the ability to achieve the desired functionality and behavior using object oriented concepts and technique.

3.2 Identifying Object Oriented Attribute

Object oriented concept plays an important role in well-designed program. The designed properties of abstraction, encapsulation, polymorphism, coupling, cohesion are frequently used as being representative of design quality characteristics in both structural as well as object-oriented development. Packaging of classes and method is an important role of the programmer's in object-oriented design. In this context, the need for optimized packaging is getting more and more importance.

The definition of key object-oriented terms for metrics are given as follows

- Cohesion – The degree to which the methods with in a class are related to one another.
- Coupling – Object X is coupled to Object Y if and only if X sends a message to Y.
- Encapsulation – class X is said to be encapsulated if the methods and attribute of the class is tied together.

- Abstraction – the percentage of private properties of the class, higher the percentage of private properties higher the abstraction is.

- Polymorphism – The ability of an object to became many different form the lower the value of WMC more the class is polymorphic.

3.3 Suite of Package level metrics for object-oriented design

We now present the definition of the package level metrics used for validation

- **Number of classes in a package**

  Average number of classes ($NOCL_{avg}$) in a package is defined as

$$\frac{\sum_{i=1}^{n} P_i}{n}$$

Where p_i is the number of classes in package i

$NOCL_{avg}$ is the average number of classes in the package, large number of classes in the package indicates that the package is heavily loaded hence may be difficult to maintain, where as the package, which are, having very few classes implies that the package is doing nothing and need to be restructured.

- **Number of statements in a package**

  Average number of statements (NOS_{Avg}) in a package is defined as

$$\frac{\sum_{i=1}^{n} NOS_i}{n}$$

Where, NOS_I is the number of statements in a package i, N is the number of classes in a package.

NOS_{Avg} can be calculated by counting the Number of statements in a package. This is the total number of statements in the package and the number of statements for each class defined in the package. Large the number of statements in a package indicates that the package is more stable.

- **Weighted method per class in a package**

  Average number of WMC of classes (WMC_{Avg}) in a package i is defined as

$$\frac{\sum_{i=1}^{n} M_i}{n}$$

Where M_i is the count of the number of methods in classes in package i.

WMC_{Avg} can be calculated as counting the number of methods in a specific package dividing by the number of classes in package. WMC can be used as a predictor of how much time and effort is required to develop and maintain the class. A large value of WMC will have a great impact on the children of the class. Package with large **WMC_{Avg}** value limit the possibility of reuse. High number of average method in a package can be an indicator that the package is doing much, hence testability and maintainability is difficult.

- **Lack of cohesion of methods** (**$LCOM_{avg}$**)

Average number of Lack of cohesion of methods of classes ($LCOM_{avg}$) in a package I is defined

$$\frac{\sum_{i=1}^{n}\left(\left(\frac{1}{INST}\right)*(Numref\ -Wmc\ )\right)/(1-Wmc\ ))}{n}$$

Where, **Numref** is the sum of the number of attribute references in each of the methods in the class

Inst is the number of instance variables defined in the class.

Wmc is the number of methods in the class.

The $LCOM_{avg}$ value was calculated for each class by dividing the number of classes in a package. This version of LCOM has value in the range 0 to 2.Lower values are better any value over 1 should be viewed as indicator of poor code. Effective object-oriented designs maximize cohesion in order to promote encapsulation. A large number of LCOM implies that the class is attempting to model more than a single concept and thus may need to be decomposed into several classes.

- **Coupling Between Object**(CBO_{avg})

Average number of Coupling between object (CBO_{avg}) in a package is defined as

  Let A_i be the set of classes which class I references and B_i be the set of classes which references class i.

$$\frac{\sum_{i=1}^{n}(A_i \cap B_i\ )}{n}$$

The more independent a class is, the easier it is to reuse it in another application. The large number of couplings the higher the sensitivity of changes would have to other parts of the design, and therefore maintenance is more difficult. High CBO values for a class suggest that it will be difficult to reuse as it indicates that the package within the classes is too dependent on the other class.

- Response for class (RFC_{avg})

  Average number of Response for class in a package is defined as

$$\frac{\sum_{i=1}^{n} \text{NOMT} + \text{EXT}}{n}$$

A high value for RFC indicates a class that is more complex and therefore more difficult to test and maintain.

- Number of packages imported by class (PackAvg)

Average Number of packages imported by class (Pack_{avg}) is defined as

$$\frac{\sum_{i=1}^{n} Packimp}{n}$$

Where, Packimp is the Number of packages imported by this class. Classes that import a large number of packages become more difficult to maintain due to this interdependencies

3.4 Mapping of Object-Oriented design principle with metrics

Table 1 Mapping of object oriented design principle with Metrics

Quality Attribute	**Object Oriented Design Principle**							
Functionality	**Cohesion**	**Coupling**	**Polymorphism**	**Complexity**	**Design Size**	**Abstraction**	**Encapsulation**	**Inheritance**
Reusability	LCOM_{avg}	RFC_{avg},EXT_{avg}, PACK_{avg},CBO_{avg}			WMC_{avg} NOCL_{avg} NOS_{avg}			
Flexibility		RFC_{avg},EXT_{avg}, PACK_{avg},CBO_{avg}	RFC_{avg} WMC_{avg}				WMC_{avg}, LCOM_{avg}	
Understandability	LCOM_{avg}		RFC_{avg} WMC_{avg}	RFC_{avg} LCOM_{avg}	WMC_{avg} NOCL_{avg} NOS_{avg}	WMC_{avg}, LCOM_{avg}	WMC_{avg}, LCOM_{avg} WMC_{avg}	
Functionality	LCOM_{avg}		RFC_{avg} WMC_{avg}		WMC_{avg} NOCL_{avg} NOS_{avg}			
Extendibility		RFC_{avg},EXT_{avg}, PACK_{avg},CBO_{avg}	RFC_{avg} WMC_{avg}			WMC_{avg}, LCOM_{avg}		NOCL_{avg}
Effectiveness			RFC_{avg} WMC_{avg}			WMC_{avg}, LCOM_{avg}	WMC_{avg}, LCOM_{avg}	NOCL_{avg}

Where, NOMT is the number of method declared in the class and EXT is the number of methods external to the class.

Response for class was calculated by totaling the number of methods declared in the class and the number of methods external to the class called from code within the class. i.e.NOMT +EXT.

3.5 Mapping of Object-Oriented design principle with Quality Attributes

Table2. Mapping of Metrics Quality Attribute

Object Oriented Design Principle	Metrics
Coupling	RFC_{avg},EXT_{avg},$Pack_{avg}$,CBO_{avg}
Cohesion	$LCOM_{avg}$
Design Size	WMC_{avg}, $NOCL_{avg}$, NOS_{avg}
Polymorphism	RFC_{avg}, WMC_{avg}
Abstraction	WMC_{avg}, $LCOM_{avg}$
Inheritance	DIT_{avg},NOC_{avg}
Complexity	WMC_{avg},NCP_{avg},RFC_{avg},$LCOM_{avg}$
Encapsulation	WMC_{avg}, $LCOM_{avg}$

Table 2 summarizes the design metrics and influence of each of the design properties on the quality attribute. The reviewed information indicates that the cohesion property has a significant influence on reusability, understandability and functionality. Design size metrics is viewed to promote reuability, understandability and functionality. Low coupling and high cohesion is considered good for reusability. Low upling with High Polymorphism and high encapsulation increases flexibility. Understandability is directly infuenced by complexity, abstraction, design size, polymorphism and cohesion. High abstraction, large design size and high complexity has adversely influenced understandability. The quality attribute effectiveness is directly influence by higher abstraction, encapsulation, polymorphism and inheritance.

4. RESULTS

To validate the package level metrics we have taken three large opensource projects[9][10][11] namely Llamachart, JEDIT and FreeCS . below is the statistics and the interpretation of individual metrics taken with the open sourse projects. After that we have found the comparison of design size metrics (WMCavg , NOSavg and NOCLavg) with other metrics (CBOavg, RFCavg, LCOMavg and PCkavg) using JFree chart. The analysis has been done using SPSS 10.0.

Interpretation of NOS_{avg} Metrics

Nos_{avg} metric is used to count the average number of executable statement in a package. Very less number of statement indicates that the package is doing nothing. After analyzing all the three projects we observed that JEdit has some of the classes in a package having very few executable statements (minimum of 6 statements). This indicates that the functionality defined in the package is not uniformely distributed. Very large number of statement is also not good as it indicates that the package is too overloaded (in case of FreeCS maximum of 5079 statements) hence difficult to understand and maintain.

Table 3 SUMMARY SATISTICS FOR NOS_{avg}

Site	Number Of Packages	Min	Max	Mean	Std.Dev
Llama Chart	4	7	88.11	50.47	41.27
JEdit	37	6.33	240	56.33	47.30
FreeCS	17	47	5079	1173.35	1389.88

Interpretation of WMC_{avg} Metrics

WMC_{avg} is the design size metrics. High WMC value increases reusability, and will decrease the understandability of the project. Some of the classes in JEdIT projects has average wmc value 1 which indicates the poor reusability. There may be chance of improvement by merging these classes with some other classes with in the same package without affecting the LCOM values, that is without affecting the encapsulation of the classses. FreeCS is having High Lcom value which is the indication of poor understandability.

Table 4 SUMMARY SATISTICS FOR WMC_{avg}

Site	Number Of Packages	Min	Max	Mean	Std.Dev
Llama Chart	4	2	8.57	6.51	3.09
JEdit	37	1	26.14	5.98	4.57
FreeCS	17	2	69	12.34	15.37

Interpretation of $LCOM_{avg}$ Metrics

$LCOM_{avg}$ metric influences the quality factor such as encapsulation and abstraction , high lcom value is a serious issue as it indicates the poor encapsulation. Among all the three projects FreeCS has the highest Average Lcom value at package level indicating poor encapsulation. It may introduce possibility of error in case of reuse of packages in development

Table 5 SUMMARY SATISTICS FOR LCOM$_{avg}$

SITE	Number Of Packages	Min	Max	Mean	Std.Dev
Llama Chart	4	2.92	31.11	19.18	13.22
JEdit	37	2	88	19.97	15.1
FreeCS	17	28	1607	350.05	447.97

Interpretation of CBO$_{avg}$ Metrics

CBO$_{avg}$ is to count the number of classes that is coupled with other classes in a package. There can be high coupling between classes in a package but the package may have low coupling between other packages in a system. Low coupling is desirable as it increases the maintainability, reusability and understandability.

Amongst the three projects Llama chart is the best in quality(in case of maintainability, reusability and understandability) having low coupling, minimum of 2.44 and maximum of 8.5. Next is the Jedit with minimum of 0 and maximum of 12.59. In FreeCS project the packages are more dependent as it is having very high coupling with a minimum of 8 and maximum of 416 indicating that the system is difficult to maintain and reuse.

Table 6 SUMMARY SATISTICS FOR CBO$_{avg}$

Site	Number Of Packages	Min	Max	Mean	Std.Dev
Llama Chart	4	0.1	0.46	0.34	0.16
JEdit	37	0	1.49	0.24	0.27
FreeCS	17	0.02	15.68	3.92	4.39

Interpretation of RFC$_{avg}$ Metrics

RFC$_{avg}$ is to count the average number of sum of local methods and methods called by local methods in a package. The data from all the projects except FreeCS suggest that most of the classes in a package invoke a small number of methods indicating high encapsulation hence increasing understandability and maintainability.

Table 7 SUMMARY SATISTICS FOR RFC$_{avg}$

SITE	Number Of Packages	Min	Max	Mean	Std.Dev
Llama Chart	4	2.44	8.50	5.08	2.51
JEdit	37	0	12.59	4.48	3.08
FreeCS	17	8	416	99.25	121.62

Interpretation of NOCL$_{avg}$ Metrics

The NOCL$_{avg}$ metric is to count the number of classes in a package,. Very few number of classes in a package suggest that the package need to merge with other packages without affecting the CBO, RFC and LCOM. In JEDIT and FreeCS the classes is not uniformely distributed amon the package. Some package has large number of classes as seen in JEDIT and FreeCS(maximum of 231 and 56 number of classes) .Packages having very large number of classes(say more than 50) are difficult to understand and maintain.

Table 8 SUMMARY SATISTICS FOR NOCL$_{avg}$

SITE	Number Of Packages	Min	Max	Mean	Std.Dev
Llama Chart	4	2	13	7.75	4.57
JEdit	37	1	231	33.13	46.27
FreeCS	17	1	56	11.7	12.97

5. ANALYSIS AND DISCUSSION

In this section we present the inter relationship of various metrics for the FreeCS project

.

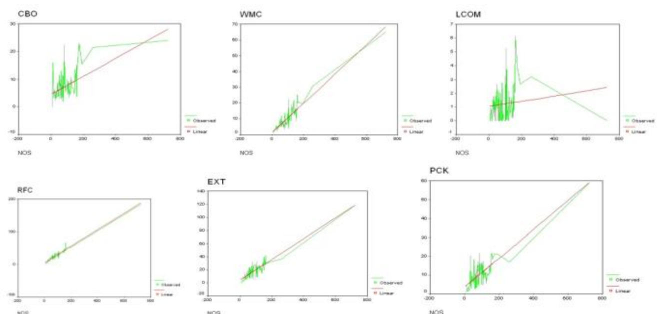

Figure 2 Relationship of design size metrics (NOS$_{avg}$) with coupling and cohesion metrics

Figure 2 shows the distribution point of NOS_{avg} metric with other metric ie; PCK_{avg}, WMC_{avg}, $LCOM_{avg}$, CBO_{avg}, RFC_{avg} and EXT_{avg}. The relationship between NOS_{avg} metric with the PCK_{avg} shows very high fluctuation of PCK_{avg} when Number of executable statements is between 0-200. The PCK_{avg} becomes more stable when NOS_{avg} has increased to over 200. The similar nature is also observed with the WMC_{avg} which is unstable between 0-200 NOS but becomes more stable when NOS_{avg} increases. The $LCOM_{avg}$ metric goes down to negative as the NOS_{avg} increased and there is high flutuation between 0-200 NOS. The CBO_{avg} metric becomes stable when NOS increases above 300. This means that the class has more number of executable statements which are capable of operating seperately without referencing other classes. There is a linear relationship as shown with RFC_{avg}. The relationship with EXT_{avg} is almost same as that of RFC_{avg} As the number of NOS increases the relationship becomes linear.

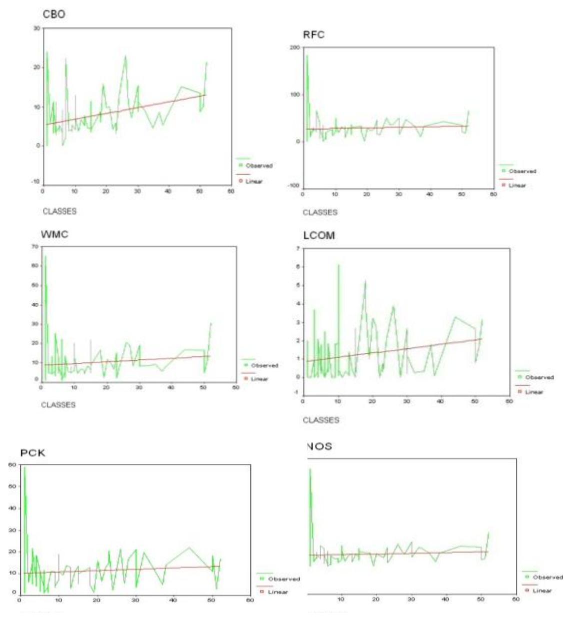

Figure 3 Relationship of $NOCL_{avg}$ with Coupling and Cohesion metrics

Figure 3 shows the comparison of different design size metric $NOCL_{avg}$ (Average Number of classes) with other metrics ie; PCK_{avg}, WMC_{avg}, $LCOM_{avg}$, CBO_{avg}, RFC_{avg} . Relationship with WMC_{avg} shows that there is a high fluctuation of WMC in class size 0-10 but as the class size increases, say up to 50 classes the relationship becomes stable. Relationship with $LCOM_{avg}$ is almost similar to the WMC_{avg} in that as the class size increases the $LCOM_{avg}$ becomes more stable, but when the size is more than 50 it becomes little unstable. Relationship with CBO_{avg} is that the lower the number of classes more the classes are dependent so as the number of classe increases the relationship becomes more stable. The same relationship is shown with the RFC_{avg}, the less the number of classes the relationship becomes more unstable.

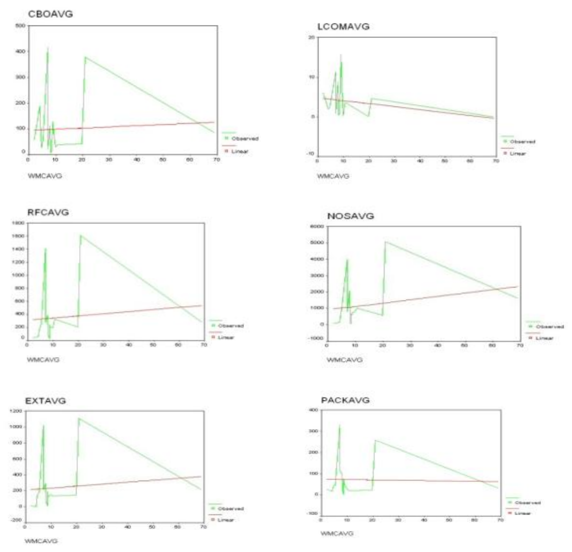

Figure 4 Relationship of WMC_{avg} with Coupling and Cohesion metrics

Figure 4 shows the relationship of design size metric WMC_{avg} with the other metrics ie; PCK_{avg}, $LCOM_{avg}$, CBO_{avg}, RFC_{avg} and EXT_{avg} . Relationship with $LCOM_{avg}$ shows that there is very high fluctuation of $LCOM_{avg}$ when WMC size is 0-10 but as the size of WMC increases, say up to 60 classes the relationship becomes more stable and reaching towards 0. Relationship with CBOavg seems to be unpredicatble. However as WMC reaches above 20 the relationship becomes inversely linear. The same relationship is shown with the RFC_{avg}, EXT_{avg}, and NOS_{avg}.

6. CONCLUSION

In this paper we have calculated the average package level metrics then a mapping of these metrics with object- oriented design properties and quality attributes has been done. Finally these metrics have been validated upon three open source projects.

After interpretation of all metrics it was observed that metrics for FreeCs had the maximum coefficient of variation (NOS_{avg} - 118%, WMC_{avg} – 124% , CBO_{avg} – 122%, $LCOM_{avg}$ – 112%, RFC_{avg} – 128%) reflects inconsistency among the samples within the group and hence we chose this project for analysis and discussion in section 5. It was further observed that most of the metrics value stablized when the number of classes in a package exceeds 200.

We conclude that the prediction or outcome of FreeCs software is far away from standards. Therefore this software should be re-evaluated and corrected for better results. Hence we can conclude that well formulated metrics can be used as prediction tools for software quality.

7. REFERENCES

[1] B. Henderson-Sellers and J.M. Edwards, "Books Two of Object oriented Knowledge : The Working Object", Prentice-Hall Sydney, 1994.

[2] B. Kitchenham and S.L. Pfleeger, "Software Quality: The Elusive Target, "IEEE Software vol. 13, no. 1, pp. 12-21, s1996.

[3] E. Arishlom, L.C. Briand., Foyen, "A Dynamic coupling measures for Object Oriented Software", IEEE Trans. On Software Engineering, 30, 8(2004) 491-506.

[4] F. Brotoeabreu, "The MOOD Metrics Set", in Proc. ECOOP 95 Workshop Metrics, 1995.

[5] G.R. Dormey "A model for Software Product Quality, " IEEE Trans. Software Eng., vol 21, no. 2, pp. 146-162, Feb. 1995.

[6] G.R. Dormey , "Cornering the Chimer, "IEEE Software, vol. 13, no. 1, pp. 33-43, 1996.

[7] H.Kabaili, R.K. Keller and F.Lustman, "Cohesion as changeability indicator in object-Oriented System", in Proc. Fifth European Conf. Software Maintenance and Reengineering, 2001.

[8] H. S. Chae, Y.R. Kwon and D.H. Bae, "Improving Cohesion Metrics for Classes by considering Dependent Instanse Variables" , IEEE Trans on Software Engineering, 20, 6 (1994), 476-493.

[9] Source code of freeCS is taken from the URL http://sourceforge.net/projects/FreeCS

[10] Source code of Llama chart is taken from URL http://sourceforge.net/projects/Llamachart

[11] Source code of Jedit is taken from URL http://sourceforge.net/projects/jedit

[12] I. Sommerville. "Software Engineering" (4th edition), Addison-Wesley, 1992.

[13] J.A. McCall, P.K. Richard, and G.F. Walters, "Factors in software Quality", Vol 1,2 and 3, AD/A-049-014/015/055 Nat'1 Tech. Information Service, Springfield, Va., 1977.

[14] J.Bansiya, C.G.Davis , "A Hierarchical Model for Object-Oriented Design Quality assessment," IEEE Trans. On Software Engineering , vol 28. no 1, 2002

[15] J. M. Bieman and B.K. Kang "Cohesion and Reuse in an Object-Oriented System", in Proc. Symp. Software Reliability, 1995, pp.259-262

[16] J. Vincent, A. Walters, and J.Sinclair, Software Quality Assurance, Vol. 1. Prentice Hall, 1988.

[17] K. EL. Emam, S. Benlarbi, N. Goel, and S.N.Rai, "The Confounding effect of the Class size on the Validity of Object-Oriented Metrics", IEEE Trans. On Software Engineering, 27, 7,(2001).

[18] L.C.Briand, J.W. Daly and J.K. Wust, "A unified Framework for Cohesion Measurement in Object-Oriented Systems" Empirical Software eng, 1, 1 (1998), 65-117.

[19] L.C.Briand, "Empirical Investigation of Quality Factors in Object-Oriented Software", Empirical studies of software Engineering, ottawa, Canada, 1999.

[20] L.C. Briand and J.K.Wust , "Modeling Development effort in Object-Oriented system using Design properties", IEEE Trans on Software Enginerring , 27, 11(2001), 963-986

[21] M. Alshayeb and Li. W, "An empirical Validation of Object oriented Metrics in two different Iterative Software 1043-1049

[22] M. Hitz and B.Montazeri, Correspondance, Chidamber and Kemerer's Metric Suite. " A Measurement Theory Prespective", "IEEE Trans. On Software Engineering, 22, 4 (1996), 267-271

[23] M. Lorenz, and J.Kidd, "Object oriented software Metrics", A practical Guide, 1994

[24] N.I. Churcher and M.J. Sheppered, Comments on "A Metric suite for Object-Oriented Design", IEEE Trans. On Software engineering, 21 (1995), 263-265

[25] Robert C. Martin , agile Software Development, Pearson Education Inc, 2003.

[26] R. Pressman. Software Engineering a Practitioner's Approach" 3rd edition), McGraw-Hill, 1992

[27] S.R. Chidamber and C.F. Kemerer, "Towards a Metric Suite for Object-Oriented Design", in Proc Sixth OOPSLA Conf., (1991), 197-211.

[28] S.R. Chidamber and C.F. Kemerer, "A Metric Suite of Object-Oriented design", IEEE Trans on Software Engineering, 20, 6(1994), 476-493

[29] V.R Basil, LC.Briand,WL.Melo, "A Validation of Object oriented Design as Quality Indicators", IEEE Trans on Software Engineering, Vol. 22, No.10, 1996, pp. 751-761

[31] Vinay,Bhattacherjee and Sandeep, "An Analysis Of Dependency of Coupling On Software Defects", ACM SIGSOFT Software Engineering Notes", January 2012 Volume 37 Number 1.

[32] W. Humphery. "Managing the Software Process" , SEI Series in Software Engineering, Addison-Wisley, 1989.