Estimation of Maintainability in Object Oriented Design Phase: State of the art

Vivek Rai, Akhilesh Mohan Srivastava, Himanshu Pandey, Dr. V. K Singh

ABSTRACT: Object oriented designing is an essential part of software environment. This study focuses on a set of object oriented metrics that can be used to measure the maintainability of an object oriented design. These metrics for object oriented design focus on measurements that are applied to the class and design characteristics. These measurements permit designers to access their software early in process, making changes that will reduce maintainability and improve the continuing capability of the design. In our paper we studied those metrics using empirical analyses for three package designs for the same software. We also found out that value of RFC doesn't need to be low for developing a less fault prone software.

KEYWORDS: maintainability, object-oriented, metrics, designing, classes.

1. INTRODUCTION

A software product requires a number of measures to be taken into account for its designing. The most important measure that must be considered in any software product is its design quality [1]. Among all the quality criteria, software maintainability is broadly accepted as a highly significant quality criterion in the economic success of engineering systems and products. There is a need for software engineers to understand how various components of a design interact in order to maintain and enhance the reliability of software during maintenance. Maintenance of software is one of the most expensive and resource requiring phase of the software development process, maintainability evaluation is an essential component of modern software development life cycle. Evaluation of software maintainability, if done accurately, can be useful in aiding decision making related to the software, efficiency of the maintenance process, comparing productivity and costs among different projects, allocation of resource and staff, and so on. This minimizes the future maintenance effort [4]. .

The study has been conducted in object-oriented paradigm. This is due to the fact that the primary purpose of object-oriented design is to improve software quality criteria such as maintainability, reliability, usability, etc by managing software complexity. The logical complexity of the source code has a strong correlation to the maintainability of the resultant software [8], [9]. Reducing the software development and maintenance costs is the main objective of object-oriented design. In order to facilitate the analysis and evaluation of maintainability of an object-oriented system, Chidamber and Kemerer (CK) metrics [10] have been used. CK metrics are design complexity metrics that aid in identifying certain design and code characteristics in object-oriented software which in turn helps in assessing external software qualities such as software defects, testing, and maintenance effort [11].

According to IEEE Standard Glossary of Software Engineering Terminology Maintainability can be defined as the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment [11]. The object oriented metrics are used to evaluate and predict the quality of software. These metrics are used as an early indicator of externally visible attributes. The externally visible attributes could not be measures until too late in the software development process. Metrics to be a set of standards against which one can measure the effectiveness of object oriented analysis techniques in the design of a system. There is a Mood matrix that stands for maintainability of object oriented design. The object oriented development methods models the system components as the objects and this matrix is used by the object oriented developers to reduce the maintenance effort.

Some of the other popular software quality metrics for measuring the maintainability of the system are:

Testability: Testability is the quality of the software design. Testability is extrinsic property that helps to find out the various kind of bugs presented in the system. It also helps in reducing the bugs for effectiveness of the system. A testable product is used for the complete execution of the test scripts.

Understandability: Understandability of the software system defines as the attributes of the software that uses the user efforts to recognize the various logical concepts.

Modifiability: Modifiability is defined as the changes that occur in the system to increases the performance of the system. Whenever there is the need to change the properties of the system, the developers changes its features according to the demand of the developer.

There are other approaches too to that aim to provide high quality and low maintenance cost. Component based software engineering is the branch of software engineering that emphasizes the separation of concerns. Component based software engineering assembles the software products from pre existing smaller products. These products are known as the components. A component model generally defines a concept of components and rules for their design time composition and is usually accompanied by one or more component technologies, implementing support for composition and interoperation. This approach is primarily used to revolutionize the development and maintenance of software systems.

Factors affecting maintainability:

For evaluating the maintainability of object oriented system five factors are taken complexity, class, coupling, inheritance and number of children. These factors are chosen since they are the design complexity factors and show more impact on the maintainability of object-oriented software system. Brief outlines of all these factors are:

Complexity: By software complexity we mean the difficulty to preserve, modify and comprehend the software.

Class: A class is a basic unit of OOP and it can be said as a set of objects that includes same methods, attributes and relationships.

Coupling: Coupling means the interdependency between different components or functions. Coupling is the measure of interconnections among the modules in a software structure.

Inheritance: Inheritance is defined as classes having same methods and operations based on hierarchy. It is a mechanism whereby one object acquires the characteristics from one or more other objects.

Number of Children: Number of Children defines the number of subclasses subordinate to a class in the hierarchy. It indicates the potential influence of a class on design and system.

1.1 LITERATURE SURVEY

MuktamyeeSarker [2] presented a set of object oriented metrics that can be used to measure the quality of an object oriented design. These metrics allows designers to access the software early in development process, making changes that will reduce complexity and improve the capability of the design.Seyyed Mohsen Jamali [3] also stressed on the importance of such metrics specially when an organization is adopting a new technology for which established practices have yet to be developed. Such needs can be addressed by the development and implementation of a suite of metrics for OO design.

Study by Ramanath Subramanyam and M.S. Krishnan [5] enhanced prior empirical literature on OO metrics by giving a new set of results that could validate the association between a subset of CK metrics [10] and defects detected during acceptance testing and those reported by the users. They also pointed out that after controlling for size; some of the metrics in the CK suite of OO design complexity metrics could significantly explain deviations in defects.

A suite of metrics for OO design, called MOOD, can be used to measure the use of OO design mechanisms. Various mechanisms like inheritance, polymorphism, information hiding and coupling, can influence quality characteristics like reliability or maintainability [26].Chidamber & Kemerer's OO metrics are

considered as the best predictors than the best set of "traditional" code metrics [25]. Most authors have used CMK as the basis of their research. Amjan Shaik and C.R.K. Reddy [15] presented a paper that assessed the current state of the art in metrics and object oriented software system quality along with a short descriptive taxonomy of the Object-Oriented Design and Metrics.
M Esperanza Manso, Marcela Genero and Mario Piattinis [19] studied 8 metrics for measuring the structural complexity of class diagrams due to the usage of UML relationships, and 3 metrics to measure their size so that to identify metrics that are really represent as maintainability indicators in a class diagram. They present their work by a study based on Principal Component Analysis. It finally concluded that the metrics related to associations, aggregations, generalizations and dependencies, are the most relevant while those related to size seemed to be redundant.

Melis Dagpinar and Jens H. Jahnke [6] also in their work discussed the impact of various OO metrics on software maintainability. By using an empirical approach they concluded that size and import direct coupling metrics are significant predictors for measuring maintainability of
Classes while inheritance, cohesion, and indirect/export coupling measures are not. Another study also indicated the significance of maintainability at design phase and build up a multivariate linear Maintainability Estimation Model for Object-oriented Design [7] which could estimate the maintainability of class diagrams in respect of their Extendibility and Reusability. Rajendra Kumar and Dr.Namrata Dhanda in their work [12] gave a multivariate regression model called as Maintainability Estimation Model for Object-Oriented design which could be used at Design phase in software development process. Developed model can measure maintainability of object oriented design in respect of Extendibility and Flexibility.
Devpriya Soni, Namita Shrivastava and M. Kumar used the Logical Scoring of Preferences method toevaluate global quality of designs to provide reasonable estimates for factors like functionality, effectiveness, eusability,understandability, and maintainability and also the overall quality of software design [13].

Kiranjit Kaur and Sami Anand proposed a multivariate linear model [14] which could measure the maintainability of a class diagram in the term of reliability, portability that are the sub-characteristics of maintainability. Similar work is done by Geeta Laxmi, Kavita Agrawal and Rizwan begin their paper that stressed on thesignificance of maintainability at design phase [16]. Moreover they build a multivariate linear model called as Maintainability Measurement Model for Object-Oriented Design which could estimate the maintainability of class diagrams in respect of their analyzability, understandability and modifiability.There was another research that also developed a multivariate process model to provide efficient and effective support for object oriented software [23]. This research stressed on the maintainability approach to automatic maintenance of object oriented software which is carried out at the time of software design.

Anil Kumar Malviya and Vibhooti Singh proposed an automation tool that can help the developers to reduce maintenance cost of software project [17].

Luis Reynoso, Marcela Genero and Mario Piattini [18] gave their yet another contribution, they defined a set of metrics for measuring the structural properties of OCL constraints in UML/OCLmodels. Many of these metrics and measurements are defined inters of navigations, a core concept of OCL that defines coupling between the objects. They also stated that relationship exists between object coupling (defined through metrics related to navigations and collection operations) and two maintainability sub-characteristics: understandability and modifiability of OCL expressions [20].

There is one more multivariate linear model called as 'Maintainability Estimation Model for Object-Oriented software in Design phase' or (MEMOOD) [21], which could calculate the maintainability of class diagrams in terms of their understandability and modifiability. D.N.V.Syma Kumar, R.Satya Prasad and R.R.L.Kantam [22] proposed a non-linear maintainability model to find the significant factors for maintainability in the systems which

shows the non-linear behavior in the nature. They used t-test for the reduction of the regression overhead in the process of finding the estimation model of the maintainability with reference to two factors: understandability and modifiability. El-Emam, Khaled and Melo, W in their work [24] developed a set of object-oriented design metrics which can be used to construct such prediction models with high accuracy. They also concluded that an export coupling metric had the strongest association with faultproneness, indicating a structural feature that may be symptomatic of a class with a high probability of later faults.

2. MAINTAINABILITY

Maintainability has previously been described mainly in two ways, either informally or as a function of directly measurable attributes.

Informal Descriptions There are many text descriptions available, which are in essence very similar. We quote the IEEE Standard Glossary of Software Engineering Terminology: M**aintainability:** The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment. [11]

There are other examples of such descriptions [3,4,36].

Such descriptions capture what is intuitively meant with maintainability, but have some problems. They do not in any way guide in how to estimate or measure maintainability. Another problem is that if we follow this approach and try to measure maintainability as "effort", we should bear in mind that the common unit for effort, "manmonth", is in itself very dubious [8]. We can also note that Pfleeger describes maintainability as "the *probability* that […] a maintenance activity can be carried out within a stated time interval […] [it] ranges from 0 to 1" ([8], italics added).

Definitions for Maintainability from Selected Studies

The ease with which a software application or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment

- Software maintenance difficulty is the relative degree to which software maintenance is hard to understand, perform, and manage.
- The article takes *error rate* as independent variable and suggests measures that can be used to highlight features in software design that may require future corrective or preventative maintenance activity.
- *Understandability*: The ease with which a class diagrams can be understood.
- *Analyzability*: The capability of a class diagram to be diagnosed for deficiencies or to identify parts to be modified.
- *Modifiability*: The capability of a class diagram to enable a specified modification to be implemented.
- The ease with which a software system can be modified to correct a fault or conform to changed requirements.
- How difficult it is to make small or incremental changes to an existing software object without introducing errors in logic or design.
- The number of changes made to the code during a maintenance period.
- The probability that the detected fault can be corrected and removed by an arbitrary time *t*.

A. Object Oriented Metrics

The term metrics is frequently used to mean a set of specific measurements taken on a particular process. The object oriented metrics are used to evaluate and predict the quality of software. These metrics are used as an early indicator of externally visible attributes. The externally visible attributes could not be measures until too late in the software development process. Metrics to be a set of standards against which one can measure the effectiveness of object oriented analysis techniques in the design of a system. Object oriented metrics can be applied to analyze source code as an indicator of quality attributes. The source code could be any object oriented language. On the basis of their requirements, object oriented metrics can be classified into two categories. :

1) Project based metrics

2) Design based metrics

A.1. MOOD Matric

Mood matrix stands for maintainability of object oriented design. In the development of the software systems the object oriented developers are promises to reduce the maintenance effort. The object oriented development methods models the system components as the objects. These objects are helpful in allow the designer to separate the interface from the implementation. Earlier the maintainability can be defines in three different ways as,

A.1.1 Testability

Testability is the quality of the software design. It helps in the automated testing. Testability is extrinsic property that helps to find out the various kind of bugs presented in the system. It also helps in reduces the bugs for effectiveness of the system. A testable product is used for the complete execution of the test scripts. When the testability is take place in the system, the customers reports the minimum number of defects. The testable products are easy and the cost to maintain product also less. Testability is an important aspect for the maintainability of software product.

3. OBJECT-ORIENTED DESIGN

An object contains encapsulated data and procedures grouped together to represent an entity. The 'object interface' defines how the object can be interacted with. An object-oriented program is described by the interaction of these objects. Object-oriented design is the discipline of defining the objects and their interactions to solve a problem that was identified and documented during object-oriented analysis.

What follows is a description of the class-based subset of object-oriented design, which does not include object prototype-based approaches where objects are not typically obtained by instancing classes but by cloning other (prototype) objects. Object-oriented design is a method of design encompassing the process of object-oriented decomposition and a notation for depicting logical and physical as well as state and dynamic models of the system under design.

Input (sources) for object-oriented design

The input for object-oriented design is provided by the output of object-oriented analysis. Realize that an output artifact does not need to be completely developed to serve as input of object-oriented design; analysis and design may occur in parallel, and in practice the results of one activity can feed the other in a short feedback cycle through an iterative process. Both analysis and design can be performed incrementally, and the artifacts can be continuously grown instead of completely developed in one shot.

Some typical input artifacts for object-oriented design are:

- Conceptual model: The result of object-oriented analysis, it captures concepts in the problem domain. The conceptual model is explicitly chosen to be independent of implementation details, such as concurrency or data storage.
- Use case: A description of sequences of events that, taken together, lead to a system doing something useful. Each use case provides one or more scenarios that convey how the system should interact with the users called actors to achieve a specific business goal or function. Use case actors may be end users or other systems. In many circumstances use cases are further elaborated into use case diagrams. Use case diagrams are used to identify the actor (users or other systems) and the processes they perform.
- System sequence diagram: A system sequence diagram (SSD) is a picture that shows, for a particular scenario of a use case, the events that external actors generate their order, and possible inter-system events.
- User interface documentations (if applicable): Document that shows and describes the look and feel of the end product's user interface. It is not mandatory to have this, but it helps to visualize the end-product and therefore helps the designer.
- Relational data model (if applicable): A data model is an abstract model that describes

how data is represented and used. If an object database is not used, the relational data model should usually be created before the design, since the strategy chosen for object-relational mapping is an output of the OO design process. However, it is possible to develop the relational data model and the object-oriented design artifacts in parallel and the growth of an artifact can stimulate the refinement of other artifacts.

Object-oriented concepts

The five basic concepts of object-oriented design are the implementation level features that are built into the programming language. These features are often referred to by these common names:

- **Object/Class:** A tight coupling or association of data structures with the methods or functions that act on the data. This is called a *class*, or *object* (an object is created based on a class). Each object serves a separate function. It is defined by its properties, what it is and what it can do. An object can be part of a class, which is a set of objects that are similar.

- **Information hiding:** The ability to protect some components of the object from external entities. This is realized by language keywords to enable a variable to be declared as *private* or *protected* to the owning *class*.
- **Inheritance:** The ability for a *class* to extend or override functionality of another *class*. The so-called *subclass* has a whole section that is derived (inherited) from the super class and then it has its own set of functions and data.
- **Interface (object-oriented programming):** The ability to defer the implementation of a *method*. The ability to define the *functions* or *methods* signatures without implementing them.
- **Polymorphism (specifically, Subtyping):** The ability to replace an *object* with its sub objects. The ability of an *object-variable* to contain, not only that *object*, but also all of its sub objects.

Designing concepts

- Defining objects, creating class diagram from conceptual diagram: Usually map entity to class.

- Identifying attributes.

- Use design patterns (if applicable): A design pattern is not a finished design, it is a description of a solution to a common problem, in a context.[1] The main advantage of using a design pattern is that it can be reused in multiple applications. It can also be thought of as a template for how to solve a problem that can be used in many different situations and/or applications. Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved.

- Define application framework (if applicable): Application framework is a term usually used to refer to a set of libraries or classes that are used to implement the standard structure of an application for a specific operating system. By bundling a large amount of reusable code into a framework, much time is saved for the developer, since he/she is saved the task of rewriting large amounts of standard code for each new application that is developed.

- Identify persistent objects/data (if applicable): Identify objects that have to last longer than a single runtime of the application. If a relational database is used, design the object relation mapping.

- Identify and define remote objects (if applicable).

Output (deliverables) of object-oriented design

- **Sequence diagram**: Extend the system sequence diagram to add specific objects that handle the system events.

  A sequence diagram shows, as parallel vertical lines, different processes or objects that live simultaneously, and, as horizontal

arrows, the messages exchanged between them, in the order in which they occur.

- **Class diagram**: A class diagram is a type of static structure UML diagram that describes the structure of a system by showing the system's classes, their attributes, and the relationships between the classes. The messages and classes identified through the development of the sequence diagrams can serve as input to the automatic generation of the global class diagram of the system.

Some design principles and strategies

Dependency injection: The basic idea is that if an object depends upon having an instance of some other object then the needed object is "injected" into the dependent object; for example, being passed a database connection as an argument to the constructor instead of creating one internally.

- **Acyclic dependencies principle:** The dependency graph of packages or components (the granularity depends on the scope of work for one developer) should have no cycles. This is also referred to as having a directed acyclic graph. For example, package C depends on package B, which depends on package A. If package A also depended on package C, then you would have a cycle.
- **Composite reuse principle:** Favour polymorphic composition of objects over inheritance.

4. ESTIMATION PROCESS

Estimation theory is a branch of statistics and signalprocessing that deals with estimating thevalues of parameters based on measured/empirical data that has a random component. Theparameters describe an underlying physical setting in such a way that their value affects thedistribution of the measured data. An estimator attempts to approximate the unknown parametersusing the measurements.

For example, it is desired to estimate the proportion of a population of voters who willvote for a particular candidate. That proportion is the parameter sought; the estimate is based on a smallrandom sample of voters.Or, for example, in radar the goal is to estimate the range of objects (airplanes, boats etc.) byanalyzing the two-way transit timing of received echoes of transmitted pulses. Since the reflectedpulses are unavoidably embedded in electrical noise, their measured values are randomly distributed, so that the transit time must be estimated.

In estimation theory, two approaches are generally considered.

- The probabilistic approach (described in this article) assumes that the measured data is random with **probability distribution** dependent on the parameters of interest

- The setmembership approach assumes that the measured data vector belongs to a set which depends on the parameter vector.

For example, in electrical communication theory, the measurements which contain informationregarding the parameters of interest are often associated with a noisy signal. Without randomness, or noise, the problem would be deterministic and estimation would not be needed.

Metrics Used For Evaluating Maintainability

These metrics are aimed at assessing the design of object-oriented system rather than implementation. This makes them more suited to object-oriented paradigm as object-oriented design puts great emphasis on the design phase of software system. The CK metric suite consists of six design complexity metrics- WMC, DIT, NOC, CBO, RFC and LCOM. Except for LCOM, all these metrics can be used as maintainability predictors as LCOM is uncorrelated with the maintainability of the software. Thus the CK metrics (except LCOM) are briefly described as follows [10]:

WMC (Weighted Methods per Class)

It is a weighted sum of all the methods defined in a class. It measures the complexity of a class. It also predicts how much time and effort is required to develop and maintain the class. High WMC indicates greater complexity and hence low maintainability.

DIT (Depth of Inheritance Tree)

It is the length of the longest path from a given class to the root class in the inheritance hierarchy and is measured by the number of ancestor classes. So this metric calculates how far down a class is declared in the inheritance hierarchy. High DIT indicates greater design complexity and more fault-proneness.

NOC (Number of Children)

It is equal to the number of immediate child classes derived from a base class. High NOC means greater level of reuse, more effort required for testing, more complexity and fault-proneness.

CBO (Coupling Between Objects)

For a class, CBO is measured by counting the number of other classes to which it is coupled. Coupling is a measure of interdependence of two objects. Two classes are coupled if methods of one use methods and/or instance variables of the other. High CBO indicates complex design, decreases modularity, and complicates testing of the class.

RFC (Response for a Class)

It is the count of all the methods which can potentially be executed (directly or indirectly) in response to a message to an object of that class or by some method in the class. (This includes all methods accessible within class hierarchy). High RFC means more effort required for testing, greater design complexity and fault-proneness.
The values of all the above metrics are inversely proportional to the maintainability of a system [9].

5. PROPOSED WORK

In our paper we will use the CK metrics [10] to study the effect of various factors related to class and find out which of them have more relevance in measuring the maintainability of software as early as in its design process. We used sample data of three packages each for a model design of an online shopping website. All the three models were used to construct a software, based on the analysis of those designs and the difficulties faced by the developers in later stages of maintainability we will conclude the impact of the metrics on maintainability. Analysis is done by a empirical evaluation of the class diagram of each package with respect to stated metrics.

We are going to use WMC (Weighted Methods per Class), DIT (Depth of Inheritance Tree), NOC (Number of Children), CBO (Coupling between Objects) and RFC (Response for a Class) metrics for evaluation of the package designs. The details of the various packages are given in the following tables:

CLASSES	WMC	DIT	NOC	CBO	RFC
1	0	0	0	2	0
2	0	0	0	0	0
3	0	0	3	2	0
4	0	1	2	4	0
5	0	0.633	0	2	0
6	0	2	0	3	0
7	0	2.33	0	3	0
8	0	2	0	3	0
9	0	2.65	0	1	0
10	0	0	0	0	0

Table 1: detailed information of package 1

CLASSES	WMC	DIT	NOC	CBO	RFC
1	4	1	0	1	7
2	3	0	4	3	10
3	1	1	2	3	6
4	1	0	0	2	5
5	1	2	0	1	5
6	1	0	0	1	1
7	1	2	0	1	5

Table 2: detailed information of package 2

CLASSES	WMC	DIT	NOC	CBO	RFC
1	0	0	0	2	1
2	0	0	0	0	0
3	0	1	0	2	0
4	1	0	0	2	1
5	0	0	0	0	0
6	0	0	2	4	1
7	0	0.33	0	2	0
8	0	0.33	0	1	0
9	0	0	0	2	0
10	0	1	0	1	0

Table 3: detailed information of package 3

As we can see through the tables package 1 is pretty simple package that have no defined functions and therefore has zero response for functions in its entire structural framework. Package 2 is more defined with functions and inheritance between classes. It has also got high values for RFC for only 7 classes. High values of RFC are generally considered to attain more complexity for the system development [10]. Now, package 3 with 10 classes also has only one method in its structure. This package consists of a total of 2 RFC but well defined variables.

Now to conduct an empirical study on the various metrics for the three packages we will calculate their mean and median values for each of the metrics. First, Weighted method per class (WMC):

PACKAGES	MEAN	MEDIAN
Package1	0	0
Package 2	1.714	1
Package 3	0.01	0

Table 4: mean and median values for WMC

PACKAGES	MEAN	MEDIAN
Package1	1.061	0.816
Package 2	0.857	1
Package 3	0.266	0

Table 4: mean and median values for DIT

PACKAGES	MEAN	MEDIAN
Package1	0.05	0
Package 2	0.857	0
Package 3	0.02	0

Table 4: mean and median values for NOC

PACKAGES	MEAN	MEDIAN
Package1	2	2
Package 2	1.714	1
Package 3	1.60	2

Table 4: mean and median values for CBO

PACKAGES	MEAN	MEDIAN
Package1	0	0
Package 2	5.571	5
Package 3	0.03	0

Table 4: mean and median values for RFC

These designs were further used to create an online shopping website. .Now, we will analyze the sample data empirically using a graphical representation.

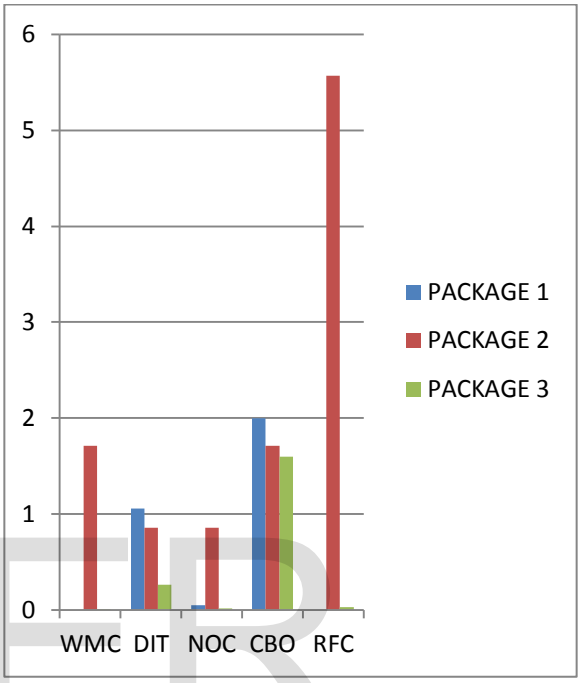

Figure 1: graphical representation of mean values for the metrics of the three packages

We chose bar chart graphical representation to analyze our results because it is more clear and easier to understand, even for novice students. Package 2 was the most useful design for constructing the software and required little maintenance as compared to the other software developed using package 1 and 3 designs. We analyzed that even though high value of RFC is considered to bring complexity in the design [10] but having zero or almost no value isn't helpful either. CBO should be kept as minimum as possible so as to maintain the reusability of the class. Our prime focus was to analyze the impact of RFC on the overall quality of the software design and prove that even though it increases the overall complexity of the design and efforts required by the testers in testing each and every path possible it yet results in better less fault prone software.

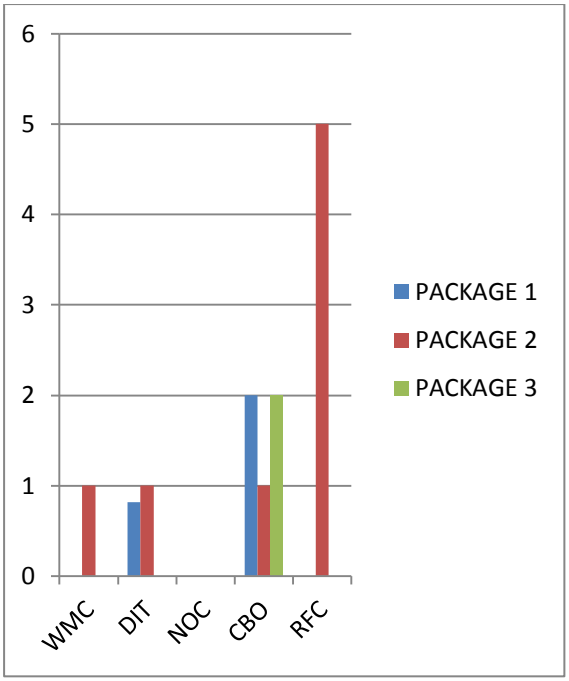

Figure 1: graphical representation of median values for the metrics of the three packages

Median denotes the middle value of the sample data, in the figure 2 we have shown the graphical representation of the median values for all of the metrics of three packages. In the figure as we can see that RFC for package 3 is quite high and NOC is situated at 0.

6. CONCLUSION AND FUTURE WORK

Maintainability is considered as an important quality factor for developing the efficient software system.Object-oriented approach enhances the maintainability of software system. In literature there are no well-defined criteria to evaluate maintainability.Measuring maintainability of a software system in the design phase may help a software designer improve the maintainability of software system before delivery to the customer and hence reduces a lot of efforts, time and cost.

In this paper we aimed to present the state of art of object oriented designs and analyze its various metrics that effect maintainability. We found out that despite of its traditional view on RFC to cause increase in complexity and efforts during testability it lead to development of a less fault prone software. So instead of trying to keep its value as low as possible the designers must try to keep the class diagram fully elaborate with detailed description of each method so as to have better understanding for the codes developers.

In future we aim to study the MOOD metrics and finally present our own multivariate model that can estimate maintainability of the software design.

7. REFERENCES

[1] Ragab, S. R., Ammar, H. H., 2010 "Object-Oriented design metrics and tools: a survey", Proc. Of Informatics and Systems (INFOS), pp. 1 – 7

[2] MuktamyeeSarker, 2005 "An overview of Object Oriented Design Metrics", Master Thesis, Umeå University, Sweden.

[3] Seyyed Mohsen Jamali, January 2006 "Object Oriented Metrics(A Survey Approach)", Tehran Iran.

[4] Lucia, A. De, Pompella, E., Stefanucci, S., 2005 "Assessing effort estimation models for corrective maintenance through empirical studies," Information and Software Technology, vol. 47, no. 1, pp. 3-15.

[5] RamanathSubramanyam and M.S. Krishnan, May 2003 "Empirical analysis of CK metrics for object torienteddesign complexity: implications forsoftware defects". ARTICLE in IEEE TRANSACTIONS ON SOFTWARE ENGINEERING.

[6] MelisDagpinar and Jens H. Jahnke, 2003, "Predicting Maintainability with Object-OrientedMetrics - An Empirical Comparison".Proceedings of the 10th Working Conference on Reverse Engineering (WCRE'03) IEEE.

[7] NupurSoni, Dr. MazharKhaliq, 2015, "Maintainability Estimation of Object Oriented Software: Design Phase Perspective". International Journal of Advanced Research in Computer and Communication EngineeringVol. 4, Issue 3, March 2015.

[8] Pfleeger S. L., Software Engineering, Theory and Practice, Prentice-Hall, Inc., 1998.

[9] Dubey, S. K., Rana, A., 2011 "Assessment of Maintainability Metrics for Object-Oriented Software System", ACM SIGSOFT SEN, vol. 36, No. 5.

[10] Chidamber, S. R., Kemerer, C. F., 1994 "A Metrics Suite for Object Oriented Design," IEEE

Transactions on Software Engineering, vol. 20, no. 6, pp. 476-493.

[11] IEEE, IEEE Standard Glossary of SoftwareEngineering Terminology, report IEEE Std 610.12-1990, IEEE, 1990.

[12] Rajendra Kumar, Dr.NamrataDhanda, 2015, "Maintainability Measurement Model for Object-Oriented Design". International Journal of Advanced Research in Computer and Communication Engineering, Vol. 4, Issue 5, May 2015.

[13] DevpriyaSoni, Dr. NamitaShrivastava and Dr. M. Kumar, 2010, "A Methodology for Empirical Quality Assessment ofObject-Oriented Design". (IJCSIS) International Journal of Computer Science and Information Security, Vol. 7, No.2, 2010.

[14] KiranjitKaur. ,SamiAnand, 2013, "A Maintainability Estimation Model and Metrics for Object-Oriented Design (MOOD)". International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 2, No 5, May 2013,ISSN: 2278 – 1323 .

[15] AmjanShaik, Dr. A. Damodaram, Dr. C. R. K. Reddy , 2012, "Object Oriented Software Metrics and Quality Assessment: Current State of the Art ".International Journal of Computer Applications (0975 – 8887) Volume 37– No.11, January 2012.

[16] GeetaLaxmi,Mrs. KavitaAgrawal , Dr. Rizwan Beg, 2014 ," Maintainability Measurement Model of Object Oriented Design ".International Journal of Advanced Research in Computer Science and Software Engineering ,Volume 4, Issue 11, November 2014 ISSN: 2277 .

[17] Anil Kumar Malviya, Vibhooti Singh , 2015 "Some Observations on Maintainability Estimation Model for Object Oriented Software in requirement, Design, Coding and Testing Phases". International Journal of Advanced Research in Computer Science and Software Engineering, Volume 5, Issue 3, March 2015 ISSN: 2277 .

[18] M EsperanzaManso, Marcela Genero, Mario Piattini, "No-Redundant Metrics for UML Class DiagramStructural Complexity".

[19] Luis Reynoso, Marcela Genero,Mario Piattini,2005, "Assessing the Impact of Coupling on the Understandability and Modifiabilityof OCL Expressions within UML/OCL Combined Models". IEEE- 2005, Spain.

[20] S. W. A. Rizvi and R. A. Khan ,2010, "Maintainability Estimation Model for Object-Oriented Software in Design Phase(MEMOOD) ".JOURNAL OF COMPUTING, VOLUME 2, ISSUE 4, APRIL 2010, ISSN 2151-9617.

[21] Mr.D.N.V.Syma Kumar,Dr.R.Satya Prasad,Dr.R.R.L.Kantam, 2015 ,"Maintainability of Object-Oriented SoftwareMetrics Using Non-Linear Model". International Journal of Advanced Research inComputer Science Engineering and Information Technology, Volume: 5 Issue: 3 20-Mar-2015,ISSN_NO: 2321-3337.

[22] Anshul Mishra and Ajay Kumar Yadav, 2014, "Proposed Maintainability Model for Software Development: Design Issues".American International Journal of Research in Science, Technology, Engineering & Mathematics, AIJRSTEM 14-377, 2014.

[23] El-Emam, Khaled; Melo, W., 1999, "The Prediction of Faulty Classes Using Object-Oriented DesignMetrics ". NRC Publications Archive (NPArC) ,Publisher's version: J. Syst. Software, 1999.

[24] Victor R. Basili, Lionel Briand and Walcélio L. Melo, 1995, "A VALIDATION OF OBJECT-ORIENTED DESIGNMETRICS AS QUALITY INDICATORS". Technical Report, Univ. of Maryland, USA. April 1995.

[25] Fernando Brito e Abreu, WalcélioMelo, 1996, "Evaluating the Impact of Object-Oriented Design on Software Quality".IEEE, Berlin, Germany, March 1996.Originally published in Proceedings of the 3rd InternationalSoftware Metrics Symposium (METRICS'96).