**Information Science and Technology**
ISSN: 0976-917X & ISSN: 0976-9188, Volume 3, Issue 2, 2014, pp.-066-070.
Available online at http://www.bioinfopublication.org/jouarchive.php?opt=&jouid=BIJ0000002

# OBJECT ORIENTED DESIGN METRICS, FRAMEWORKS AND QUALITY MODELS

**DESAI C.G.***
Department of MCA, MIT, Aurangabad- 431 009, MS, India.
*Corresponding Author: Email- chitragdesai@gmail.com

Received: November 27, 2013; Accepted: July 17, 2014

**Abstract-** Software quality has multidimensional facet and when it comes to assessment of quality of software it is essential to understand what quality attributes contribute. These attributes should fit in specific framework of metrics so that the expected quality parameter assessment is attained. Object oriented design has good set of metrics which can be used for object oriented design assessment.

This paper highlights the existing quality models; framework and metrics on object oriented design and provides an insight motivating a thought process for new paradigm for object oriented quality model.

**Keywords-** Object oriented design, quality models, design metrics,

**Citation:** Keshamoni K. and Harikrishna M. (2014) Improved Visual Cryptography Scheme for Data Security, Information Science and Technology, ISSN: 0976-917X & ISSN: 0976-9188, Volume 3, Issue 2, pp.-066-070.



## Introduction

The object oriented approach to software development promises better management of system complexity and a likely improvement in project outcomes such as quality and the project cycle time [1]. This requires creating a good design. The term 'good' refers to a design which is clear, easy to implement and easy to maintain. Design complexity has been conjectured to play a strong role in the quality of the resulting software system in object oriented development environment [1]. This necessitates the early assessment and evaluation of the object oriented design.

Quality measures of object oriented code or design artifacts usually involve analyzing the structure of these artifacts with respect to the interdependencies of classes and components as well as their internal elements. The underlying assumption is that such measures can be used as objective measure to predict various external quality aspects of the code or design artifacts, e.g., maintainability and reliability. Quality is viewed from one's perspective and hence the set of metrics that evolves comes from context independent view of quality.

One of the earliest software quality models was suggested by McCall [7] and his colleagues. McCall's quality model defines software product qualities as a hierarchy of factors, criteria and metrics and was the first of the several models of the same form. The quality model defined in ISO/EIC 9126-1 "Software engineering product quality" standard classifies quality attributes as external, visible on system and internal, properties of subsystem and components. All these models vary in their hierarchical definition of quality, but they share a common difficulty. The models are vague in their definition of lower levels of details and metrics need to attain a quantitative assessment of product quality [2]. Another difficulty with the earlier models was the inability to account for dependency among quality attributes.

Most aspects of software development process and respective products are too complex to be adequately captured by one single metric. This necessitates the framework requirement for object oriented design metrics. The framework is a methodology for the development of quality models in a bottom-up fashion, providing an approach that will ensure that the lower level details are well specified and computable.

As per the observation and also reported by [2] there are no known comprehensive and complete models or frameworks that evaluate the overall quality of design developed using an object oriented approach based on its internal design properties. This paper highlights the existing quality models; framework and metrics on object oriented design and provides an insight motivating a thought process for new paradigm for object oriented quality model.

## Quality Models

Dromey [8,9] has addressed some of the problems of earlier models such as McCall's and ISO 9126. The Quality Model for Object Oriented Design QMOOD [2] extended the Dromey's generic quality model methodology. This model has the lower-level design metrics well defined in terms of design characteristics, and quality is assessed as an aggregation of the model's individual high-level quality attributes. The high level quality attributes are assessed using a

set of empirically identified and weighted object oriented design properties, which are derived from object oriented metrics which measure the lowest level structural, functional, and relational details of a design.

Although this model gives the indication that model of this type can be effectively used in monitoring the quality of software product it has not considered quality attributes such as reliability, maintainability and testability. There is evidence that design metrics is related to a variety of quality characteristics of software product such as reliability, testability and maintainability [10]. A set object oriented design metrics for these attributes are not considered by QMOOD. In QMOOD across each design property a single metric is used which is not self sufficient in itself to justify the design properties.

Inspired from QMOOD [2] a reliability focused quality model RFQ MOOD was proposed. The initial set of design quality attributes in QMOOD is functionality, effectiveness, understandability, extendibility, reusability and flexibility. However, the quality attributes concentrated by RFQMOOD are reliability, reusability, testability and maintainability.

As discussed earlier a quality model takes the form of the viewpoint we take of quality which is still a vague and multifaceted concept, the earlier quality models provides a framework from which to proceed. The next session provides overview on the existing object oriented design metrics and framework.

## Overview of the Existing Objects Oriented Metrics and Frameworks

Software measures are a tool to measure the quality of software. The area of software measurement is also known as software metrics. A metric here is not considered in the sense of a metric space, it is considered as: measurement is a mapping of empirical objects to numerical objects by homomorphism. A homomorphism is a mapping, which preserves all relations and structures. Put in words: Software quality should be linearly related to software measure.

Measuring the relatedness in software started changing with the changing approach towards software development i.e., from traditional to object- oriented. History of software measurement by Horst Zuse gives complete overview of the milestones in the development of software measures. According to him more than 5000 paper about software measurements were published till 1996 and more till date.

As this paper is related in identifying the candidate metrics for object oriented design it restrict the survey to the existing object oriented metrics and frameworks. In the next section a insight on some of the most widely cited object oriented metrics is brought forth.

## Existing Object Oriented Metrics

### Moreau and Dominick

They proposed following metrics [11]:

a. Message Vocabulary Size (MVS)
b. Inheritance Complexity (IC)
c. Message domain Size (MDS)

The three defined metrics need classification such as what exactly is meant by “sending message” and how the metrics are to be computed. However we can draw some parallels between these metrics and the three object-oriented software quality abstractions of coupling, inheritance complexity and cohesion.

### Chidamber and Kemerer

The Chidamber and Kemerer metric suite [12] is the most cited set of metrics and also most criticized. There are six metrics in the suite, all of them being design metrics:

a. Weighted method per class (WMC)
b. Depth of inheritance tree (DIT)
c. Number of children (NOC)
d. Coupling between object classes (CBO)
e. Response for class (RFC)
f. Lack of cohesion in methods (LCOM)

Churcher and Shepperd [14] point out that definition of some of the basic direct counts are imprecise, which could have impact on the defined metrics. The main concern lies with the number of methods in a class count, used directly in computation of WMC and indirectly in LCOM. Due to the various possibilities in counting the methods, the results could vary dramatically, leading to confusion. Hitz and Montazeri [15] argue that CBO is not sensitive enough measure of coupling, since it considers all couples to be of equal strength. Henderson-Seller [16] shows LCOM measure is not sensitive enough for cases of high cohesion.

Li and Henry [17] conducted their own empirical experiments, and showed that by using a combination of five of six CK metrics (Omitting CBO), along with some newly defined metrics, it is possible to predict maintenance effort required for a software system. Basili, et al [18] shows that five of the six CK metrics were useful in predicting class fault-proneness during the high and low level phases of life cycle.

### Li and Henry

They [17] present ten metrics in their system; they include five of the six metrics defined by CK, namely DIT, NOC, RFC, LCOM and WMC. In addition they define five more metrics of their own. These are:

a. Message Passing Coupling (MPC)
b. Data Abstraction Coupling (DAC)
c. Number of Methods (NOM)
d. Number of Semicolons (SIZE1)
e. Number of Properties (SIZE2)

Experimenting with these metrics the authors concluded that there is a strong relationship between metrics and maintenance effort in object oriented systems. Also maintenance effort can be predicted from combinations of metrics collected from source code.

However, in SIZE1, the authors use number of semicolons in a class, which is language-dependent and also not derivable until the source code is available. The DIT metric is used as a measure of complexity, where the larger the value of DIT, the more complex the system is supposed to be. But trying to minimize DIT leads to the guideline “do not use inheritance at all”, while inheritance is one of the major advantages of the object oriented paradigm.

### Martin’s Package Metrics

Martin [19] identifies criteria for the proper distribution of classes into packages. These criteria are essentially based on he notion of dependency. The goal is to reduce dependency, especially dependencies on concrete class.

Unfortunately, martin does not define what a dependency exactly is? He only says that dependencies are caused by class relationships like inheritance, aggregation and uses. As an educated guess the depends-on-relation, which includes the examples given by Martin, is used for formal definitions.

Martin does not consider nested packages, even though dependencies of classes in packages nested inside a package to classes within that package can be considered to have a special status, as they are more "local" than dependencies from classes in outside packages. The metrics proposed by Martin is as follows:

a. Relational cohesion (H)
b. Afferent coupling (Ca)
c. Efferent coupling (Ce)
d. Abstractness (A)
e. Instability (I)
f. Distance from main sequence (D)

Martin's metrics focus on high level, architectural design issues, so they can be formalized easily. There are some vague points in the original definitions, but these could be overcome in the formalization by educated guesses.

**Brito e Abreu**

e Abreu, [20,21] derived a set of six metrics known as the MOOD (Metrics for Object Oriented Design) metrics. It includes:

a. Method Hiding Factor (MIF)
b. Attribute Hiding Factor (AHF)
c. Method Inheritance Factor (MIF)
d. Attribute Inheritance Factor (AIF)
e. Polymorphism factor (POF)
f. Coupling Factor (COF)

These metrics refer object-oriented paradigm in following ways:

1. Encapsulation (MHF and AHF)
2. Inheritance (MIF and AIF)
3. Polymorphism (POF)
4. Message Passing (COF)

And are expressed as quotients

Metric = X/ Total

The numerator represents the actual use of those mechanisms for a design. The denominator acting as a normalizer represents the hypothetical maximum achievable use for the same mechanism on the same design. The value for each metric will therefore be in the range 0-1 i.e., between 0-100%.

Definition for MIF and AIF are inconsistent with the 0-1 scale as shown in [22]. Also the AIF is meaningless in the sense that the concept of inheritance concerns the behavior defined in a method, an attribute does not have behavior, and thus cannot be overridden or inherited.

The MOOD metrics have been subjected to much empirical evaluation, with claims made regarding the usefulness of the metrics to assess external attributes such as quality. The theoretical evaluation of MOOD metrics by [23] show that any empirical validation is premature due to the majority of the MOOD metrics being fundamentally flawed. The metrics either fails to meet the MOOD team's own criteria or is founded on an imprecise, and in certain cases inaccurate, view of Object oriented paradigm.

**Lorenz and Kidd**

Lorenz and Kidd [24] defined many object oriented design metrics, but did not validate nor thoroughly test them [13]. The metrics are listed in [Table-1] along with the level at which they are taken.

**Table 1-** Lorenz and Kidd Metrics

| Property | Associated Metric |
|---|---|
| Method Size | Number of messages send, number of statements, lines of code, average method size. |
| Method internals | Method complexity, strings of message send. |
| Class Size | Number of public instance methods per class, number of instance method per class, average number of instance method per class, number of instance variables per class, number of class variables per class. |
| Method Inheritance | Number of methods over ridden by a sub class (NOV), number of methods inherited by a subclass, number of methods added in sub class, specialization index. |
| Class intervals | Class cohesion, global usage, average number of parameters per method, use of friend function, percentage of function oriented code, average number of comment lines per method, average number of commented methods, number of problem reports per class or contracts |
| Class externals | Class coupling, number of times a class is reused, number of classes per method over thrown away. |

**Existing Framework for Object Oriented Metrics**

Most aspects of software development process and respective products are too complex to be adequately captured by one single metric. However, the choice of a set of metrics exposes the well-known pitfalls of measuring:

- Too much, there by getting overwhelmed by a big amount of unmanageable numeric data;
- Too little, thereby not gaining sufficient insight to be able take corrective actions,
- The wrong attributes, thereby deriving delusive conclusions.

To avoid these traps a framework for the implementation of metrics initiatives has to be adopted. Several researchers have proposed different frameworks for object oriented metrics along different dimensions in an attempt to organize the metric collection [Table-2]. Shows the framework by Henderson-Seller [25].

**Table 2-** Henderson-Seller Framework

| Perspective | Measures | Metrics |
|---|---|---|
| Inside a class | Size and Complexity | WMC, NOM, NO Attribute Count |
| External at the class level | Concerns interface of classes. | Metrics here can be viewed as measuring the services offered by a class. |
| System level | Measures from the above two perspectives | |
| System level relationships | Coupling | |
| Inheritance coupling | Inheritance hierarchy and coupling | |

Sheetz, et al [4] defines four levels along which metrics can be classified. All the metrics measures the complexity of the software. [Table-3] gives the details of Sheetz framework.

Yet another approach to classifying metrics comes from Bellin [4] as shown in [Table-4].

**Table 3-** Sheetz Framework

| Level | Metrics |
|---|---|
| Variable level | Variable fan-in, Variable fan-out. |
| Method level | Method input parameters, method parameters returned, object variables accessed, method |

**Table 4-** Bellin Framework

| Group | Objective | Metrics |
|---|---|---|
| A | Capturing statistical aspects of OO design | Number of classes, Number of methods, number of messages, number of receiving classes, Number of sender classes, Number of levels in hierarchy. |
| B | Dealing with code reuse | Number of classes reused, percent of reused classes modified. |
| C | Deals with the quality of on abstraction of OO system. | Coupling, cohesion. |

e Abreu [20] has pointed out several frameworks, summarized in [Table-5] to [Table-7], below based on different perspectives such as target, structure and obtainment criterion. He also pointed out that the above taxonomies, although relevant, do not cover the semantics of metrics usage. Neither have they covered the level of abstraction within the paradigm concept. To overcome these problems a new framework was proposed i.e., TAPROOT (Taxonomy Precise for Object Oriented Metrics) framework [5].

**Table 5-** Target Taxonomy for Metrics

| Type | Description | Examples |
|---|---|---|
| Product Metrics | Quantification of attributes of the software development deliverables | Length in words of the user manual, lines of source code, number of relations in database. |
| Process Metric | Quantification of attributes of the software development process | Design duration, coding effort, maintenance cost. Average effort for the application of 1 test |
| Hybrid Metrics | Mixture of product and process metrics | Cost per function point, time to deliver n LOC, average monthly failure rate per I/O interface. |

The metrics are classified along two “independent Vectors”, category and granularity. The authors [5] reveal that the categories were derived after a sample of 128 references was reviewed in order to find a common denominator in the extensive metric literature. The categories are design, size, complexity, reuse, productivity and quality. The second dimension granularity further refines the categories by considering metrics in each category at the method, class and system level.

**Table 6-** Structure taxonomy for Metrics

| Type | Description | Examples |
|---|---|---|
| Elementary Metrics | Quantification of a single attribute of the software development process or deliverables. | Requirement specification dimension in words, LOC, time to complete the design phase, |
| Composite Metrics | Mathematical combinations of several elementary metrics. | Man Month per KLOC, average time for correcting one error, testing efficiency. |

**Table 7-** Obtainment Criterion Taxonomy for Metrics

| Type | Description | Examples |
|---|---|---|
| Objective Metrics | Precisely defined and equally obtainable on a repeatable fashion, irrespective of the collector or time. | Number of uncommented lines of source of code, average number of yearly produced versions, the number of input screens. |
| Subjective Metrics | Depends upon the collector’s judgment; may lead to incoherent and non repeatable measures | Programmer’s experience, average learning time, ease of utilization of a certain application |

**Table 8-** TAPROOT Classification Framework

| | Method | Class | System |
|---|---|---|---|
| Design | MD | CD | SD |
| Size | MS | CS | SS |
| Complexity | MC | CC | SC |
| Reuse | MR | CR | SR |
| Productivity | MP | CP | SP |
| Quality | MQ | CQ | SQ |

However TAPROOT cannot be considered as a final proposal. Looking in depth at each metric abstraction it is observed that values across these metrics cannot be obtained till date in the implementation phase. So it cannot be considered as a framework to be used in the early design phase. Also at the granularity level packages has not been given any consideration. Therefore a framework called Framework for Predicting Reliability of Object Oriented Design FPROOD was proposed which is useful in the early design phase for assessing the design quality and predicting the reliability of the object oriented software.

**Table 9-** FPROOD Framework

| | Design Metric | Size Metric | Complexity Metric |
|---|---|---|---|
| Class | a. DIT (Depth of iheritance tree<br>b. RFC (Response for class)<br>c. CBO (Coupling between objects | a. Number of methods<br>b. Number of children (NOC) | a. WMC (Consider number of method in class) if (activity diagram across key methods available then consider CC |
| Package | Coupling Metric<br>a. Instability (I=Ce/(Ca+Ce))<br>b. Abstractness (A=Na/N)<br>a. Distance from main sequence line.<br>Cohesion Metric:<br>a. Relational cohesion (H=(R+1)/N) | a. Number of classes in a package (N)<br>b. Number of abstract classes in a package (Na) | a. Number of relationship between classes in a package (R)<br>b. Afferent coupling (Ca)<br>c. Efferent coupling (ce) |
| System | a. Average number of methods per class | a. Number of class<br>b. Total number of methods<br>c. Total number of package | a. Total length of inheritance chain |

## Conclusion

The object-oriented approach naturally inclined towards early assessment and evaluation. To accomplish this we need a proper set of metrics. Design metrics play an important role in helping developers understand design aspect of software and, hence, improve software quality and developer productivity. Although, many object-oriented metrics has been proposed, but there is as yet no consensus on which are best, and most have not been well-validated. Also, many of the metrics and quality models currently available for object oriented software analyses can be applied only after a product is

complete or nearly complete. They rely upon information extracted from the implementation of the product. This provides information late to help in improving internal product characteristics prior to the completion of the product. Thus, there is a need for metrics and models that can be applied in the early stages of development (requirements and design) to ensure that the analysis and design have favorable internal properties that will lead to the development of a quality end product. RFQMOOD model and FPROOD framework are useful in assessing quality attributes in early design phase but it can be further explored for various aspects of object oriented design parameters which contribute towards object oriented design metrics.

**Conflicts of Interest:** None declared.

**References**


[1] Booch G. (2006) *Object Oriented Analysis & Design with Application*, Pearson Education India.

[2] Bansiya J. and Devis C.G. (2002) *IEEE Transaction on Software Engineering*, 28(1), 4-17.

[3] Subramanyam R. and Krishna M.S. (2003) *IEEE Transaction on Software Engineering,* 24(4), 297-310.

[4] Abounader J.R. and Lamb D.A. (1997) *A Data Model for Object Oriented Design Metrics*.

[5] e Abreu F.B. and Carapua R. (1994) *Journal of Systems and Software*, 26(1), 87-96.

[6] Cortellessa V. and Pompei A. (2004) *ACM SIGSOFT Software Engineering Notes,* 29(1), 197-206.

[7] McCall J.A., Richards P.K. and Walters G.F. (1977) *Factors in Software Quality*, 1,2 and 3.

[8] Dormey G.R. (1995) *IEEE Transaction Software Engineering,* 21(2), 146-162.

[9] Dormey G.R. (1996) *IEEE Software,* 13(1), 33-43.

[10] Shepperd M. (1992) *Information and Software Technology*, 34 (10), 674-680.

[11] Moreau D.R. and Dominick W.D. (1989) *Journal of Systems and Software*, 10, 23-28.

[12] Chidamber S.R. and Kemerer C.F. (1994) *IEEE Transactions on Software Engineering*, 20(6), 476-493.

[13] Apvrille L., de Saqui-Sannes P., Lohr C., Sénac P. and Courtiat J.P. (2001) *The Unified Modeling Language: Modeling Languages, Concepts and Tools,* 287-301.

[14] Churcher N.I. and Shepperd M.J. (1995) *ACM SIGSOFT Software Engineering Notes*, 20(2), 69-75.

[15] Hitz M. and Montazeri B. (1996) *IEEE Transactions on Software Engineering*, 22.

[16] Henderson-Sellers B. (1996) *Software Metrics,* Prentice Hall, Hemel Hempstead, UK.

[17] Li W., Henry S., Kafura D. and Schulman R. (1995) *Journal of Object Oriented Programming*, 48-55.

[18] Basili V.R., Briand L.C. and Melo W.L. (1996) *IEEE Transactions on Software Engineering*, 22(10), 751-761.

[19] Martin R.C. (2000) *Object Mentor,* 1-34.

[20] e Abreu F.B. (1992) *Pragmatic and Theoretical Directions in Object-Oriented Software Metrics*, 78-80.

[21] e Abreu F.B. and Melo W. (1996) *Proc. METRICS' 96*, Berlin, Germany, 90-99.

[22] Saini S. and Salaria R.S. (2004) 2nd *ACIS International Conference on Software Engineering Research*, *Management and Applications*, Los Angeles, 5-7.

[23] Mayer T. and Hall T. (1999) *IEEE Technology of OO Languages and Systems*, 108-117.

[24] Lorenz M. and Kidd J. (1994) *Object Oriented Software Metrics*, Prentice Hall.

[25] Desai C. (2008) *International Journal of Computational Intelligence and Telecommunication Systems,* 11-16.