Classification of Object Oriented Metrics by Aivosto Tool

Prepared by:Younes Abubakir Abedl ssamad

Technical Faculty Of Structural Engineering – Mesallata- Libya

Email y.onesabob@Gmail.com

Abstract

Object oriented design is becoming more popular in software development environment and object oriented design metrics is an essential part of software environment. This paper focus on a set of object oriented metrics that can be used to measure the quality of an object oriented design.

Can Summarizes this paper :

The increasing importance of software measurement has led to development of new software measures. Many metrics have been proposed related to various constructs like class, coupling, cohesion, inheritance, information hiding and polymorphism. But there is a little understanding of the empirical hypotheses and application of many of these measures. It is often difficult to determine which metric is more useful in which area. As a consequence, it is very difficult for project managers and practitioners to select measures for object-oriented systems. In this paper we investigate 15 metrics proposed by various researchers. The metrics are first defined and then explained using practical applications. They are applied on standard projects on the basis of which descriptive statistics, principal component analysis and correlation analysis is presented. Finally, classification working , evaluation for set metrics by using Aivosto tool .

KEYWORDS

Metrics- Software Metrics - Object Oriented Metrics- classification metrics.

1. Introduction .

Object oriented design is becoming more popular in software development environment and object oriented design metrics is an essential part of software environment. This chapter the focus on a set of object oriented metrics that can be used to measure the quality of an object oriented design.

The metrics for object oriented design focus on measurements that are applied to the class and design characteristics. These measurements permit designers to access the software early in process, making changes that will reduce complexity and improve the continuing capability of the design[2].

It is widely accepted that object oriented development requires a different way of thinking than traditional structured development1 and software projects are shifting to object oriented design. The main

advantage of object oriented design is its modularity and reusability. Object oriented metrics are used to measure properties of object oriented designs.

Metrics are a means for attaining more accurate estimations of project milestones, and developing a software system that contains minimal faults [1]. Project based metrics keep track of project maintenance, budgeting etc. Design based metrics describe the complexity, size and robustness of object oriented and keep track of design performance.

The increasing importance of software measurement has led to development of new software measures. Many metrics have been proposed related to various constructs like class, coupling, cohesion, inheritance, information hiding and polymorphism. But there is a little understanding of the empirical hypotheses and application of many of these measures. It is often difficult to determine which metric is more useful in which area. As a consequence, it is very difficult for project managers and practitioners to select measures for object-oriented systems. In this chapter we investigate 15 metrics proposed by various researchers. The metrics are first defined and then explained using practical applications. They are applied on standard projects on the basis of which descriptive statistics, principal component analysis and correlation analysis is presented. Finally, classification working , evaluation for set metrics by using Aivosto tool .

2. Metrics Reviews

In this section OO metrics chosen in this work are defined and their application on example systems is shown.

2.1.C.K. Metrics Model [4]

Chidamber and Kemerer define the so called CK metric suite .CK metrics have generated a significant amount of interest and are currently the most well known suite of measurements for OO software. Chidamber and Kemerer proposed six metrics; the following discussion shows their metrics.

2.1.1 Weighted Methods per Class (WMC) :

Definition : Consider a class C_1, with methods M_1, …, M_n that are defined in the class. Let c_1, …, c_n be the complexity of the methods. Then:

$$WMC = \sum_{i=1}^{n} C_i$$

If all method complexities are considered to be unity, then WMC = n, the number of methods.

Theoretical Basis : WMC definition of complexity of a thing, because methods are properties of object classes and complexity is determined by the cardinality of its set of properties. The number of methods is therefore a measure of the class definition as well as attributes of a class, because attributes correspond to properties.

High value of WMC indicates the class is more complex than that of low values. So class with less WMC is better

2.1.2. Depth of Inheritance Tree (DIT)

Definition: Depth of inheritance of the class is the DIT metric for the class. In cases involving multiple inheritance, the DIT will be the maximum length from the node to the root of the tree.

Theoretical basis: DIT relates to Bunge's notion of the scope of properties. DIT is a measure of how many ancestor classes can potentially affect this class.

Viewpoints:

- The deeper a class is in the hierarchy, the greater the number of methods it is likely to inherit, making it
- more complex to predict its behavior.
- Deeper trees constitute greater design complexity, since more methods and classes are involved.
- The deeper a particular class is in the hierarchy, the greater the potential reuse of inherited methods

If DIT increases, it means that more methods are to be expected to be inherited, which makes it more difficult to calculate a class's behavior. Thus it can be hard to understand a system with many inheritance layers. On the other hand, a large DIT value indicates that many methods might be reused

2.1.3 Number of children (NOC)

Definition: NOC = number of immediate sub-classes subordinated to a class in the class hierarchy.

Theoretical basis: NOC relates to the notion of scope of properties. It is a measure of how many subclasses are going to inherit the methods of the parent class.

Viewpoints:

- Greater the number of children, greater the reuse, since inheritance is a form of reuse.
- Greater the number of children, the greater the likelihood of improper abstraction of the parent class. If a class has a large number of children, it may be a case of misuse of sub-classing.
- The number of children gives an idea of the potential influence a class has on the design. If a class has a large number of children, it may require more testing of the methods in that class.

The NOC is the number of immediate subclasses of a class in a hierarchy.

If NOC grows it means reuse increases. On the other hand, as NOC increases, the amount of testing will also increase because more children in a class indicate more responsibility. So, NOC represents the effort required to test the class and reuse

2.1.4 Coupling Between Objects *(CBO)*

Definition : CBO for a class is a count of the number of other classes to which it is coupled **.**

A class is coupled to another if methods of one class use methods or attributes of the other, or vice versa. Coupling between objects for a class is the number of other classes to which it is coupled.

some impacts on having high coupling on quality attributes in a system.

- The reusability of classes and/or subsystems is low when the couplings between these are high, since a strong dependency of an entity on the context where it is used makes the entity hard to reuse in a different context.
- Normally a module should have a low coupling to the rest of the modules. A high coupling between the different parts of a system has a negative impact on the modularity of the system and it is a sign of a poor design, in which the responsibilities of each part are not clearly defined.
- A low self-sufficiency of classes makes a system harder to understand. When the control-flow of a class depends on a large number of other classes, it is much harder to follow the logic of the class because the understanding of that class requires a recursive understanding of all the external pieces of functionality on which that class relies. It is therefore preferable to have classes that are coupled to a small number of other classes .

2.1.5 .Response for a Class (RFC)

The RFC is the cardinality of the set of all methods that can be invoked in response to a message to an object of the class or by some method in the class. This includes all methods accessible within the class hierarchy. This metric looks at the combination of the complexity of a class through the number of methods and the amount of communication with other classes. The larger the number of methods that can be invoked from a class through messages, the greater the complexity of the class. If a large number of methods can be invoked in response to a message, the testing and debugging of the class becomes complicated since it requires a greater level of understanding on the part of the tester. A worst case value for possible responses will assist in the appropriate allocation of testing time. This metric evaluations system design as well as the usability and the testability.

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

Theoretical Basis : The response set for the class can be expressed as :

$$RS = \{M\} \cup_{all\ i} \{R_i\}$$

where {Ri} = set of methods called by method i and {M} = set of all methods in the class.

2.1.6 . Lack of Cohesion in Methods (LCOM)

Definition: Lack of Cohesion (LCOM) measures the dissimilarity of methods in a class by looking at the instance variable or attributes used by methods .

Consider a class C1 with n methods M_1, M_2…., M_n. Let (I_j) = set of all instance variables used by method Mi.

There are n such sets $\{I_1\},.....\{I_n\}$.

Let

$$P = \{(I_i, I_j) | I_i \cap I_j = 0\} \text{ and } Q = \{(I_i, I_j) | I_i \cap I_j \neq 0\}$$

If all n sets $\{(I_1),.....,(I_n)\}$. are 0 then p=0

$$LOCM = |P| - |Q|, if \ |P| > |Q|$$
$$= 0 otherwise$$

2.2 MOOD Metrics Model

Abreu et at defined MOOD (Metrics for Object Oriented Design) metrics. MOOD refers to a basic structural mechanism of the object-oriented paradigm . Abreu et at proposed five metrics; the following discussion shows their metrics.

2.2.1 Method Hiding Factor (MHF)

It is a measure of encapsulation defined as:

$$MHF = \frac{\sum_{i=1}^{Tc} \left[\sum_{m=1}^{Md(ci)} (1 - V(M_{mi}))\right]}{\sum_{i=1}^{Tc} M_d(ci)}$$

Where $M_d(C_i)$ is the number of methods declared in a class, and

$$V(M_{mi}) = \frac{\sum_{j=1}^{Tc} is\_visible(M_{mi,} C_j)}{(Tc - 1)}$$

where *TC* is the total number of classes, and

$$\text{is\_visible}(M_{mi}, C_j) = \begin{cases} 1 & \text{if } j \neq i \text{ may call } M_{mi} \\ 0 & \text{otherwise} \end{cases}$$

Thus, for all classes, C1, C2…Cn, a method counts as 0 if another class, can use it and 1 if it cannot be used by another class. The total for the system is divided by the total number of methods defined in the system.

If the value of MHF is high (100%), it means all methods are private which indicates very little functionality. Thus it is not possible to reuse methods with high MHF. MHF with low (0%) value indicate all methods are public that means most of the methods are unprotected.

2.2.2 Attribute Hiding Factor (AHF)

It is a measure of encapsulation defined formally as:

$$AHF = \frac{\sum_{i=1}^{Tc} \left[\sum_{a=1}^{Ad(ci)} (1 - V(A_{ai}))\right]}{\sum_{i=1}^{Tc} A_d(c_i)}$$

Where $A_d(C_i)$ is the number of attribute declared in a class, and

$$V(A_{ai}) = \frac{\sum_{j=1}^{Tc} is\_visible(A_{ai,} C_j)}{(Tc - 1)}$$

where *TC* is the total number of classes, and

$$\text{is\_visible}(A_{ai}, C_j) = \begin{cases} 1 & \text{if } j \neq i \text{ may call } A_{ai} \\ 0 & \text{otherwise} \end{cases}$$

Thus, for all classes, C_1, C_2…C_n, an attribute counts as 0 if it can be used by another class, and 1 if it cannot .

If the value of AHF is high (100%), it means all attributes are private. AHF with low (0%) value indicate all attributes are public

2.2.3 Method Inheritance Factor (MIF)

Definition: The MIF metric states the sum of inherited methods in all classes of the system under consideration. The degree to which the class architecture of an object oriented system makes use of inheritance for both methods and attributes .

MIF is defined as the ratio of the sum of the inherited methods in all classes of the system as follow :

$$MIF = \frac{\sum_{i=1}^{Tc} M_i(C_i)}{\sum_{i=1}^{Tc} M_a(C_i)}$$

where,

$M_a(C_i) = M_i(C_i) + M_d(C_i)$

TC= total number of classes

$M_d(C_i)$ = the number of methods declared in a class

$M_i(C_i) =$ the number of methods inherited in a class

If the value of MIF is low (0%), it means that there is no methods exists in the class as well as the class lacking an inheritance statement.

2.2.4 Attribute Inheritance Factor (AIF)

Definition: AIF is defined as the ratio of the sum of inherited attributes in all classes of the system. AIF denominator is the total number of available attributes for all classes.

It is defined in an analogous manner and provides an indication of the impact of inheritance in the object oriented software . AIF is defined as follows :

$$AIF = \frac{\sum_{i=1}^{Tc} A_i(C_i)}{\sum_{i=1}^{Tc} A_a(C_i)}$$

where,

$A_a(C_i) = A_i(C_i) + A_d(C_i)$

TC= total number of classes

$A_d(C_i)$ = number of attribute declared in a class

$A_i(C_i) =$ number of attribute inherited in a class

If the value of AIF is low (0%), it means that there is no attribute exists in the class as well as the class lacking an inheritance statement

2.2.5 Coupling Factor (COF)

Definition : The COF is defined as the ratio of the maximum possible number of couplings in the system to the actual number of coupling is not imputable to inheritance .

The COF is defined as follows:

$$COF = \frac{\sum_{i=1}^{TC}\left[\sum_{j=1}^{TC} is\_client(C_i, C_j)\right]}{\left(TC^2 - TC\right)}$$

Here is_ client (C_i ,C_j) = 1 if and only if, a relationship exists between the client class, Cc and the server class Cs and Cc not equal to Cs.

And is_ client (C_i ,C_j) = 0, otherwise

Where the summation occurs over i=1 to TC. TC is defined as total number of classes.

Pressman [34] argue that, although many factors affect software complexity, understandability, and maintainability. It is reasonable to conclude that as “the COF value” increases, the complexity of object

oriented design will also increase, and as a result the understandability, maintainability, and the potential for reuse may suffer. The value of COF can be varies between 0% and 100%. 0%COF indicates no class are coupled and 100% COF indicates all class are coupled with all other classes. High values of COF should be avoided.

2.3 Other Metrics Models

Several researchers propose object oriented metrics from different point of view. These metrics helps the designers to know which metrics are found at which level of decision. , the following discussion shows their metrics.

2.3.1 Number Of Methods per Class *(NOM):*

Definition : It counts number of methods defined in a class
High value of NOM indicates the class is more complex than that of low values. So class with less NOM is better.

2.3.2 Message passing Coupling *(MPC)*

Definition : (MPC) is "number of send statements defined in a class". So if two different methods in class A access the same method in class B.

2.3.4 Tight Class Cohesion (TCC) is defined to be a ratio of the number of pairs of directly connected methods in a class, NDC(C), to the maximum possible number of connections in a class, NP(C).

$$TCC(C) = \frac{NDC(C)}{NP(C)}$$

2.3.5 Loose Class Cohesion *(LCC)* is defined to be a ratio of the sum of the number of pairs of directly connected methods, NDC(C), and number of pairs of indirectly connected methods, NIC(C), in a class C to the maximum possible number of connections in C,

$$LCC(C) = \frac{NDC(C) + NIC(C)}{NP(C)}$$

Bieman and Kang identified a problem with constructor methods for TCC and LCC. A class constructor is an initialization function. It generally accesses all attributes in the class, and thus, shares attributes with virtually all other methods. Constructors create connections between methods even if the methods do not have any other relationships.

Therefore, the presence of a constructor method artificially increases cohesion as measured by TCC and LCC. Bieman and Kang have therefore recommended excluding constructors (and also destructors) from the analysis of cohesion [6].

3. Classification OO Metrics

Object oriented metrics can be collected in different ways. Although different writers have described different metrics, according to object oriented design, there are some similarities found in different metrics model. The following table shows similar OO metrics.

The metrics chosen for analysis can be divided into 5 classification. class ,information hiding , coupling , cohesion , inheritance**.**

category because most of the object oriented metrics are defined in above mention categories. In this table discussed CK metrics suite, MOOD metrics model, and metrics defined by Chen & Lu, Li & Henry.

Table 1**:** Classification OO Metrics

CLASS METRICS	INFORMATION HIDING METRICS	INHERITANCE METRICS	COUPLING METRICS	COHESION METRICS
Weighted Methods per Class (WMC)	Method Hiding Factor (MHF)	Depth of Inheritance Tree (DIT)	Coupling Between Objects (CBO)	Lack of Cohesion in Methods (LCOM)
Response for a Class (RFC)	Attribute Hiding Factor (AHF)	Number of children (NOC)	Coupling Factor (CF)	Tight Class Cohesion (TCC)
Number Of Methods per Class (NOM):		Method Inheritance Factor (MIF)	Message passing Coupling (MPC)	Loose Class Cohesion (LCC)
		Attribute Inheritance Factor (AIF)		

EXPERIMENTAL STUDY

Metrics have a number of interesting characteristics for providing development support.

Some of them are simple, precise, general and scalable to large size software systems. Abreu et al. state a set of metrics for evaluating the use of the mechanism that support the main concepts of the object-oriented paradigm and the consequent emphasis on reuse, that are believed to be responsible for the increasing in software quality and development productivity .

4.1 Analysis results by descriptive statistics for class level metrics and system level metrics :

In this section, we analyzed some metrics by using Avisto tool. In our analysis we use three projects to measure object oriented metrics

Descriptive statistics include calculating (Min, Max ,Average , Std.dev ,Median) in this section.

Table 2 : Descriptive statistics for class level metrics

Metrics	Project1					Project2					Project3				
	Avg	St.dev	Median	Min	Max	Avg	Stdev	Median	Min	Max	Avg	Stdev	Median	Min	max
WMC	10.1	3.7	9	8	21	11.3	3.4	11.5	7	18	8.3	5.2	7	3	16
DIT	2	0	2	2	2	2	0	2	2	2	1.8	0.4	2	1	2
CBO	0.9	0.4	1	0	2	1.8	1.6	1	1	6	2	0.7	2	1	3
RFC	10.6	9.6	10	8	21	13.1	3.7	13	7	19	10.8	6.1	10	3	20
LCOM	5.1	1.1	5	2	6	2	0.7	2	1	3	1.8	0.4	2	1	2
LCC	0.12	0.25	0	0	0.69	0.54	0.27	0.63	0	0.86	0.59	0.43	0.68	0	1
MPC	0.5	0.5	0	0	1	4.5	3	4.5	0	13	3	3	3	0	6
TCC	0.09	0.19	0	0	0.25	0.41	0.24	0.42	0	0.78	0.33	0.47	0	0	1

ISBN: 978-0-9891305-5-4 ©2014 SDIWC

Table 3 : Metric values for System-level Metrics

Metrics	Project1	Project2	Project3
CF	0.03	0.13	0.33
MIF	0	0	0
AIF	0	0	0
MHF	0.3	0.9	0.57
AHF	0.21	0.24	0.19
PF	0	0	0

Following are the observations made from applying these metrics on projects:

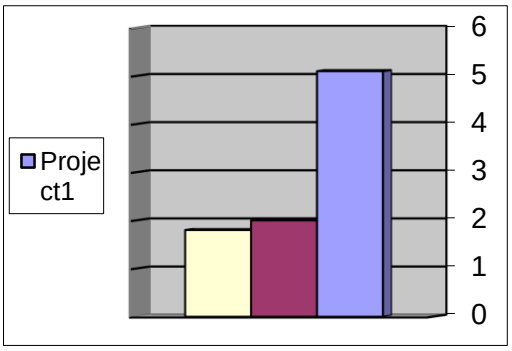

Figure 1 : Graphical representation of LCOM from table2

- LCOM values are low in project2 and project3 . It implies that the classes are cohesive

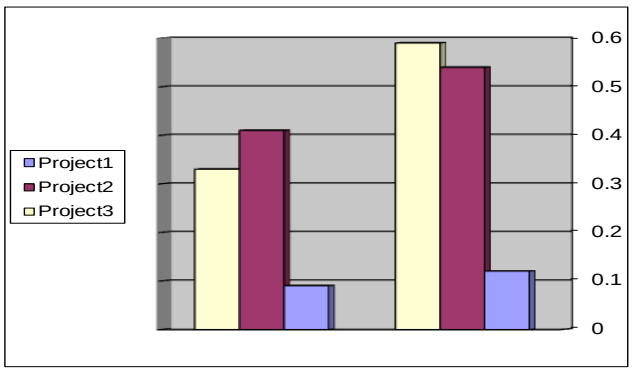

Figure 2 : Graphical representation of LCC AND TCC from table2

- TCC and LCC have Different values because there is indirect connection between methods
- The WMC look quite similar for project1 and project 3 this result indicates that most of the classes have more polymorphism and less complexity.
- The DIT look for all projects . there is a majority of DIT values = 2, it may represent poor exploitation of the advantages of OO design and inheritance.
- The CBO look for all projects the value is generally less in sample data, hence classes are easy to understand, and reuse
- The RFC look quite similar for all projects this result indicates that most of the classes have more low complexity

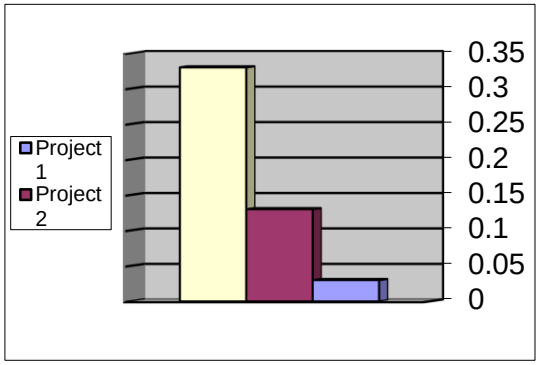

Figure 3 : Graphical representation of CF from table3

ISBN: 978-0-9891305-5-4 ©2014 SDIWC

- Coupling relations increase complexity, reduce encapsulation and potential reuse, and limit understandability and maintainability. Very high values of CF should be avoided. , and CF is expected to be lower bounded. CF should not exceed 12%.

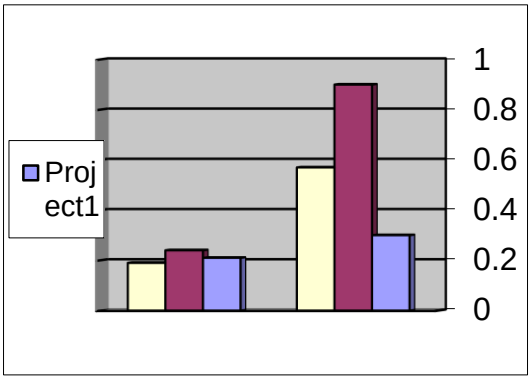

Figure 4 : Graphical representation of MHF and AHF from table3

- MHF values are high in Project2 and project3 which shows that methods are declared as private in most of the classes, so information is kept hidden
- AHF values are low in all Projects which indicating that methods are declared public by developers.
- The value of MIF is low (0%), it means that there is no methods exists in the class as well as the class lacking an inheritance statement.
- The value of AIF is low (0%), it means that there is no attribute exists in the class as well as the class lacking an inheritance statement
- PF value of 0% may indicate projects uses no polymorphism.

Conclusion

Measurement can help to improve the software process, assist in the tracking and control of a project and asses the quality of a product. By analyzing metrics, a developer can correct those areas of software process that are the cause of software defects.

Several measures have been defined so far in order to estimate object oriented design.

Coupling and cohesion are used to measure a system's structural complexity, and can be used to assess design quality and to guide improvement efforts. The application of object oriented design principles for example modularity, abstraction lead to bette MHF nability and reusability. Designers always have to be considering the bad symptoms of designs. Because, design with bad symptom needs more measurements .

A wide variety of object oriented metrics have been proposed to assess the testability of an object oriented system. Most of the metrics focus on encapsulation, inheritance, class complexity and polymorphism.

Class cohesion is considered as one of most important object-oriented software attributes Cohesion refers to the degree of the relatedness of the members in a class. Members in a class are attributes and methods. Several metrics have been proposed in the literature in order to measure class cohesion in object-oriented systems.

These metrics have been defined to capture class cohesion in terms of connections among members within a class. However, several studies have noted that they fail in many situations to properly reflect the cohesion of classes. According to many authors, they do not take into account many characteristics of classes.

References

1- Alkadi Ghassan, Carver Doris L.: Application of Metrics to Object-Oriented Designs, -Proceedings of IEEE Aerospace Conference, Volume 4, pages 159 - 163, March 1998.

2- McCabe Thomas J.: A Complexity Measure, IEEE Transactions on Software Engineering, Volume SE-2, September, Number 2, pages 308 - 320, December 1976.

3-Weyuker Elaine J.:Evaluating Software Complexity Measures,IEEE Transactions on Software Engineering, Volume 14,Number 9, pages 1357 - 1365, September 1988.

4- Chidamber Shyam R., Kemerer Chris F.: A Metrics Suite for Object Oriented Design, IEEE Transactions on Software Engineering, Volume 20, Number 6, pages 476 - 493, June 1994.

5 - k.k.aggarwal, school of information technology ' empirical study of object-oriented metrics ' , journal of object technology, november-december 2006

6 - Bieman J. M, “Cohesion and Reuse in an Object-Oriented System”,

Appendix

Aivosto Tool

Aivosto is a tool for Visual Basic language developers. A developer can take source code of any size and complexity and rework it into well-designed code.

In addition, it provides a comprehensive set of smart query functions, a graphical dependency analyzer, and over 100 quality metrics, audits, and corrective actions, that make it possible to analyze and track large volumes of code. It may be used as a stand-alone tool.

Web: http://www. **Aivosto**.com

Metrics collocation :

OOP Object Oriented Programming

WMC Weighted Methods Per Class

DIT Depth Of Inheritance Tree

NOC Number Of Children

CBO Coupling Between Objects

RFC Response For Class

LCOM Lack Of Cohesion In Methods

MOOD Metrics For Object Oriented Design

MHF Method Hiding Factor

AHF Attribute Hiding Factor

MIF Method Inheritance Factor

AIF Attribute Inheritance Factor

NOM Number Of Methods Per Class

TCC Tight Class Cohesion

LCC Loose Class Cohesion