Analysis of object Oriented Metrics

1. Dr.R.V.Krishnaiah, 2. BANDA SHIVA PRASAD

Abstract

Now a days Software development plays vital role in the world. We focus on process improvement has increased the demand for software measures, or metrics with which to manage the process. Measurement is fundamental to any engineering discipline. There is considerable evidence that object-oriented design metrics can be used to make quality management decisions. This leads to substantial cost savings in allocation of resources for testing or estimation of maintenance effort for a project. The need for such metrics is particularly acute when an organization is adopting a new technology for which established practices have yet to be developed. This research addresses these needs through the development and implementation of a suite of metrics for OO design. The metric values have been calculated using a semi automated tool. The resulting values have been analyzed to provide significant insight about the object oriented characteristics of the projects.

Index Terms

Object Oriented, Design, Development, Metric, Measure, Coupling, Cohesion, Complexity, *Size*

Introduction

Object-Oriented Analysis and Design of software provide many benefits such as reusability, decomposition of problem into easily understood object and the aiding of future modifications. But the OOAD software development life cycle is not easier than the typical procedural approach. Therefore, it is necessary to provide dependable guidelines that one may follow to help ensure good OO programming practices and write reliable code. Object-Oriented programming metrics is an aspect to be considered. Metrics to be a set of standards against which one can measure the effectiveness of Object-Oriented Analysis techniques in thedesign of a system.

Five characteristics of Object Oriented Metrics are as following:

1. Localization *operations used in many classes*
2. Encapsulation *metrics for classes, not modules*
3. Information Hiding *should be measured & improved*
4. Inheritance *adds complexity, should be measured*
5. Object Abstraction *metrics represent level of abstraction*

We can signify nine classes of Object Oriented Metrics. In each of then an aspect of the software would be measured:

- Size
- Population (# of classes, operations)
- Volume (dynamic object count)
- Length (e.g., depth of inheritance)
- Functionality (# of user functions)
- Complexity

How classes are interrelated

Coupling, collaborations between classes, number of method calls, etc.Sufficiency Does a class reflect the necessary propertiesof the problem domain?Completeness Does a class reflect all the properties of the problem domain? (for reuse) CohesionDo the attributes and operations in a class achieve a single, well-defined purpose in the problemdomain?Primitiveness (Simplicity) Degree to which class operations can't be composed from other operationsSimilarity Comparison of structure, function, behavior of two or more classesVolatilityThe likelihood that a change will occur in the design or implementation of a class

Metrics

Chidamber and Kemerer's metrics suite for OO Design is the deepest research in OO metrics investigation.

They have defined six metrics for the OO design.

In this section we'll have a complete description of their metrics:

Metric 1: Weighted Methods per Class (WMC)

Definition: Consider a Class C1, with methods M1... Mn *that are defined in the class*. Let c1... cn be the complexity of the methods. Then:

$$\mathrm{WMC} = \sum_{i=1}^{n} ci$$

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

Theoretical basis: WMC relates directly to Bunge's1 definition of complexity of a thing, since 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 class definition as well as being attributes of a class, since attributes correspond to properties.

Viewpoints

- The number of methods and the complexity of methods involved is a predictor of how much time andeffort is required to develop and maintain the class.
- The larger the number of methods in a class the greater the potential impact on children, since children will inherit all the methods defined in the class.
- Classes with large numbers of methods are likely to be more application specific, limiting the possibility of reuse.

Metric 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.

1 The ontological principles proposed by Bunge in his “Treatise on Basic Philosophy” form the basis of the concept of objects.
While Bunge did not provide specific ontological definitions for object oriented concepts, several recent researchers have employed his generalized concepts to the object oriented domain.

Metric 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.

Metric 4: Coupling between object classes (CBO)

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

Theoretical basis: CBO relates to the notion that an object is coupled to another object if one of them acts on the other, i.e., methods of one use methods or instance variables of another. As stated earlier, since objects of the same class have the same properties, two classes are coupled when methods declared in one class use methods or instance variables defined by the other class.

Viewpoints:

- Excessive coupling between object classes is detrimental to modular design and prevents reuse. The more independent a class is, the easier it is to reuse it in another application.
- In order to improve modularity and promote ncapsulation, inter-object class couples should be kept to aminimum. The larger the number of couples, the higher the sensitivity to changes in other parts of the design, and therefore maintenance is more difficult.
- A measure of coupling is useful to determine how complex the testings of various parts of a design are likely to be. The higher the inter-object class coupling, the more rigorous the testing needs to be.

Metric 5: Response for a Class (RFC)

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 }È all i { Ri }where { Ri } = set of methods called by method i and { M } = set of all methods in the class The responseset of a class is a set of methods that can potentially be executed in response to a message received by anobject of that class26. The cardinality of this set is a measure of the attributes of objects in the class. Since itspecifically includes methods called from outside the class, it is also a measure of the potentialcommunication between the class and other classes.

Viewpoints:

- If a large number of methods can be invoked in response to a message, the testing and debugging of the class becomes more complicated since it requires a greater level of understanding required on the part of the tester.
- The larger the number of methods that can be invoked from a class, the greater the complexity of the class.

• A worst case value for possible responses will assist in appropriate allocation of testing time.

Metric 6: Lack of Cohesion in Methods (LCOM)

Definition: Consider a Class C1 with *n* methods M1, M2..., Mn. Let {Ij} = set of instance variables used by method Mi. There are *n* such sets {I1},... {In}. Let P = { (Ii,Ij) | Ii Ç Ij = Æ } and Q = { (Ii,Ij) | Ii Ç Ij _ Æ }.
If all n sets {I1},... {In} are Æ then let P = Æ.
LCOM = |P| - |Q|, if |P| > |Q|
= 0 otherwise28Example: Consider a class C with three methods M1, M2 and M3. Let {I1} = {a,b,c,d,e} and {I2} = {a,b,e}and {I3} = {x,y,z}. {I1} Ç {I2} is non-empty, but {I1} Ç {I3} and {I2} Ç {I3} are null sets. LCOM is the(number of null-intersections - number of non-empty intersections), which in this case is 1.*Theoretical basis:* This uses the notion of degree of similarity of methods. The degree of similarity for twomethods M1 and M2 in class C1 is given by:s() = {I1} Ç {I2} where {I1} and {I2} are the sets of instance variables used by M1 and M2
The LCOM is a count of the number of method pairs whose similarity is 0 (i.e. s() is a null set) minus thecount of method pairs whose similarity is not zero. The larger the number of similar methods, the morecohesive the class, which is consistent with traditional notions of cohesion that measure the interrelatednessbetween portions of a program. If none of the methods of a class display any instance behavior,i.e. do not use any instance variables, they have no similarity and the LCOM value for the class will be zero.The LCOM value provides a measure of the relative disparate nature of methods in the class. A smallernumber of disjoint pairs (elements of set P) implies greater similarity of methods. LCOM is intimately tiedto the instance variables and methods of a class, and therefore is a measure of the attributes of an objectclass.

Viewpoints:

• Cohesiveness of methods within a class is desirable, since it promotes encapsulation.
• Lack of cohesion implies classes should probably be split into two or more sub-classes.
• Any measure of disparateness of methods helps identify flaws in the design of classes.
• Low cohesion increases complexity, thereby increasing the likelihood of errors during the development process.

3. MOOD (Metrics For Object Oriented Design)

The MOOD metrics set refers to a basic structural mechanism of the OO paradigm as *encapsulation* (MHF and AHF), *inheritance* (MIF and AIF), *polymorphisms* (PF) , *message-passing* (CF) and are expressed as quotients. The set includes the following metrics:

Method Hiding Factor (MHF)

MHF is defined as the ratio of the sum of the invisibilities of all methods defined in all classes to the total number of methods defined in the system under consideration.The invisibility of a method is the percentage of the total classes from which this method is not visible.*note :* inherited methods not considered.

Attribute Hiding Factor (AHF)

AHF is defined as the ratio of the sum of the invisibilities of all attributes defined in all classes to the total number of attributes defined in the system under consideration.

Method Inheritance Factor (MIF)

MIF is defined as the ratio of the sum of the inherited methods in all classes of the system under consideration to the total number of available methods (locally defined plus inherited) for all classes.

Attribute Inheritance Factor (AIF)

AIF is defined as the ratio of the sum of inherited attributes in all classes of the system under consideration to the total number of available attributes (locally defined plus inherited) for all classes.

Polymorphism Factor (PF)

PF is defined as the ratio of the actual number of possible different polymorphic situation for class Ci to the maximum number of possible distinct polymorphic situations for class Ci.

Coupling Factor (CF)

CF is defined as the ratio of the maximum possible number of couplings in the system to the actual number of couplings not imputable to inheritance.

4. Some Traditional Metrics

There are many metrics that are applied to traditional functional development. The SATC2, from experience, has identified three of these metrics that are applicable to object oriented development: Complexity, Size, and Readability. To measure the complexity, the cyclomatic complexity is used.
2 Software Assurance Technology Center (SATC) at NASA Goddard
Space Flight Center

Metric 1: Cyclomatic Complexity (CC)

Cyclomatic complexity (McCabe) is used to evaluate the complexity of an algorithm in a method. It is a count of the number of test cases that are needed to test the method comprehensively. The formula for calculating the cyclomatic complexity is the number of edges minus the number of nodes plus 2. For a sequence where there is only one path, no choices or option, only one test case is needed. An *IF* loop however, has two choices, if the condition is true, one path is tested; if the condition is false, an alternative path is tested.

Figure 1 shows a method with a low cyclomatic complexity is generally better. This may imply decreased testing and increased understandability or that decisions are deferred through message passing, not that the method is not complex. Cyclomatic complexity cannot be used to measure the complexity of a class because of inheritance, but the cyclomatic complexity of individual methods can be combined with other measures to evaluate the complexity of the class. Although this metric is specifically applicable to the evaluation of Complexity, it also is related to all of the other attributes

The SATC's approach to identifying a set of object oriented metrics was to focus on the primary, critical constructs of object oriented design and to select metrics that apply to those areas. The suggested metrics are supported by most literature and some object oriented tools.

The metrics evaluate the object oriented concepts: methods, classes, coupling, and inheritance. The metrics focus on internal object structure that reflects the complexity of each individual entity and on external complexity that measures the interactions among entities. The metrics measure computational complexity that affects the efficiency of an algorithm and the use of machine resources, as well as psychological complexity factors that affect the ability of a programmer to create, comprehend, modify, and maintain software.

We support the use of three traditional metrics and present six additional metrics specifically for object oriented systems. The SATC has found that there is considerable disagreement in the field about software quality metrics for object oriented systems.

Some researchers and practitioners contend traditional metrics are inappropriate for object oriented systems. There are valid reasons for applying traditional metrics, however, if it can be done. The traditional metrics have been widely used, they are well understood by researchers and practitioners, and their relationships to software quality attributes have been validated .

Table 1 presents an overview of the metrics applied by the SATC for object oriented systems. The SATC supports the continued use of traditional metrics, but within the structures and confines of object oriented systems. The first three metrics in Table 1 are examples of traditional metrics applied to the object oriented structure of methods instead of functions or procedures. The next six metrics are specifically for object oriented systems and the object oriented construct applicable is indicated.

Object-Oriented Specific Metrics

As discussed, many different metrics have been proposed for object oriented systems.

The object oriented metrics that were chosen by the SATC measure principle structures that, if improperly designed, negatively affect the design and code quality attributes.

The selected object oriented metrics are primarily applied to the concepts of classes, coupling, and inheritance. Preceding each metric, a brief description of the object oriented structure is given. For some of the object-oriented metrics discussed here, multiple definitions are given; researchers and practitioners have not reached a common definition or counting methodology. In some cases, the counting method for a metric is determined by the software analysis package being used to collect the metrics.

Recall, a class is a template from which objects can be created. This set of objects shares a common structure and a common behavior manifested by the set of methods. A method is an operation upon an object and is defined in the class declaration. A message is a request that an object makes of another object to perform an operation. The operation executed as a result of receiving a message is called a method. Cohesion is the degree to which methods within a class are related to one another and work together to provide well-bounded behavior. Effective object oriented designs maximize cohesion because cohesion promotes encapsulation. Coupling is a measure of the strength of association established by a connection from one entity to another.

Classes (objects) are coupled when a message is passed between objects; when methods declared in one class use methods or attributes of another class. Inheritance is the hierarchical relationship among classes that enables programmers to reuse previously defined objects including variables and operators. [2, 3, 5, 8]

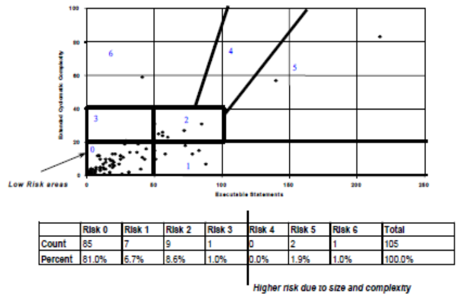
METRIC: Depth of Inheritance Tree (DIT)

The depth of a class within the inheritance hierarchy is the maximum number of steps from the class node to the root of the tree and is measured by the number of ancestor classes.

The deeper a class is within the hierarchy, the greater the number 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, but the greater the potential for reuse of inherited methods.

For many of the metrics, it is more effective to analyze the modules using two metrics.

In Figure 12 the methods are plotted based on size and complexity. The SATC has done extensive applied research to identify the preferred values. The “risk regions” shown indicate where methods have the potential for poor quality that will effect maintain ability, reusability and readability. (These regions of risk were developed for non object oriented code and are expected to decrease in size with further research.) The table below the graph summarizes the diagram.



	Risk 0	Risk 1	Risk 2	Risk 3	Risk 4	Risk 5	Risk 6	Total
Count	85	7	9	1	0	2	1	105
Percent	81.0%	6.7%	8.6%	1.0%	0.0%	1.9%	1.0%	100.0%

Figure 12: Size to Complexity Comparison

SUMMARY

Object oriented metrics help evaluate the development and testing efforts needed, the understandability, maintainability and reusability. This information is summarized in Table 4.

Metrics	Objective	Testing Efforts	Understan -dability	Maintain -ability	Develop Effort	Reuse
Complexity	↓	↓	↑	↑		
Size (LOC)	↓	↓	↑	↑		
Comment %	↑	↓	↑	↑	↓	
WMC	↓			↑	↓	↑
RFC	↓	↓				↑
LCOM	↓		↑	↑	↓	↑
CBO	↓	↓	↑	↑		↑

Table 4 : Object Oriented Metrics Effects

<u>Existing work:</u>

Structural metrics are calculated from the source code such as ***references*** and ***data sharing*** between methods of a class belong together for cohesion.

1. It define and measure relationships among the methods of a class based on the number of ***pairs of methods*** that share instance or class variables one way or another for cohesion.

Disadvantage

- Lacking of high cohesion

<u>Proposed System:</u>

1. In proposed System unstructural information is retrieved from the source code like ***comments*** and ***identifiers.***
2. Information is retrieved from the source code using ***Latent Semantic Indexing.***
3. With the help of C3 and existing metrics we are achieving the ***high cohesion*** and ***low coupling***.

Advantage

- We can predict the ***fault prediction*** using high cohesion

<u>Approaches</u>

- Retrieving the structured information.
- Check the availability of structured information foryour source code.
- Apply the LCOM5 formula for structured information.
- Analyze about the comments i.e. unstructured information.
- Index Searching
- Apply the Conceptual similarity formula.
- Comparison

<u>Approach</u>-1:

In this Approachwe are going to take the structured information like identifiers, (Example Variables). Invocation of declared methods and declared constructors. Here the Java program should be well compiled and it should be valid comments.

<u>Approach</u>-2:

In this Approachdeals we are going to search the declared variables among all the classes. Because the main theme of the declaring class variable is, it should be used in all methods. So that the declared variables are found among all the methods.

<u>Approach</u>-3:

In this Approachwe are going to apply the LCOM5 (Lack of cohesion in methods) formula. If the result is equal to one means, the class is less cohesive according to the structured information.

<u>Approach</u>-4:

Here we are going to retrieve the index terms based on that comments which are present in all the methods. Comments are useful information according to the software engineer. In concept oriented analysis we are taking the comments. Based on the comments we are going to measure the class is cohesive or not.

<u>Approach</u>-5:

In this Approachwe are going to check the index terms among the comments which are present in all the comments.

<u>Approach</u>-6:

In this Approachwe are going to apply the conceptual similarity formula. Based on the result we can say the class is cohesive or less cohesive according to concept oriented.

<u>Approach</u>-7:

In this Approachwe are going to compare the two results. Based on the results we can say that cohesion according to structure oriented and unstructured oriented.

Conclusion

We conclude that the Object oriented metrics exist and do provide valuable information to object oriented developers and project managers. The SATC has found that a combination of "traditional" metrics and metrics that measure structures unique to object oriented development is most effective. This allows developers to continue to apply metrics that they are familiar with, such as complexity and lines of code to a new development environment. However, now that new concepts and structures are being applied, such inheritance, coupling, cohesion, methods and classes, metrics are needed to evaluate the effectiveness of their application. Metrics such as Weighted Methods per Class, Response for a Class, and Lack of Cohesion are applied to these areas. The application of a hierarchical structure also needs to be evaluated through metrics such as Depth in Tree and Number of Children. At this time there are no clear interpretation guidelines for these metrics although there are guidelines based on common sense and experience.

References

1. Booch, Grady, *Object Oriented Analysis and Design with Applications*, The Benjamin/Cummings Publishing Company, Inc., 1994.
2. Chidamber, Shyam and Kemerer, Chris, "A Metrics Suite for Object Oriented Design*", IEEE Transactions on Software Engineering*, June, 1994, pp. 476-492.
3. Hudli, R., Hoskins, C., Hudli, A., "Software Metrics for Object Oriented Designs", IEEE, 1994.
4. acobson, Ivar, *Object Oriented Software Engineering, A Use Case Driven Approach*, Addison-Wesley Publishing Company, 1993.
5. Lee, Y., Liang, B., Wang, F., "Some Complexity Metrics for Object Oriented Programs Based on Information Flow", *Proceedings: CompEuro*, March, 1993, pp.
6. Lorenz, Mark and Kidd, Jeff, *Object Oriented Software Metrics*, Prentice Hall Publishing, 1994.
7. McCabe & Associates, *McCabe Object Oriented Tool User's Instructions*, 1994.
8. Rosenberg, Linda H., "Metrics for Object Oriented Environments", EFAITP/AIE Third Annual Software Metrics Conference, December, 97.
9. Sommerville, Ian, *Software Engineering*, Addison-Wesley Publishing Company, 1992.
10. Sharble, Robert, and Cohen, Samuel, "The Object Oriented Brewery: A Comparison of Twoobject oriented Development Methods", *Software Engineering Notes*, Vol 18, No 2., April1993, pp 60 -73.

AUOTHERS PROFILES:

1.Dr.R.V.Krishnaiah
M.Tech(EIE),M.Tech(CSE), PhD,MIE,MIETE,MISTE
Principal,
DRK INSTITUTE OF SCINCE & TECHNOLOGY, Hyderabad.

BANDA SHIVA PRASAD
M.TECH -CSE

DRK INSTITUTE OF SCINCE & TECHNOLOGY, Hyderabad.