Measurement Of Software Complexity In Object Oriented Systems Abstract

Hari Om Sharan[1], Garima[1], Md. Haroon[1], and Rajeev Kumar[2]

[1]Deptt. of Computer Science, COE, Teerthankar Mahaveer University, Moradabad, (India).
[2]Department of Computer Application, Teerthanker Mahaveer University Moradabad(U.P.) India
Email ID: rajeev2009mca@gmail.com,

Abstract: Our measurement for testability and complexity also shares our thought and understanding about the complexity in the object oriented system. In this document we have explained the concept of software complexity of object oriented in very sophisticated manner. Some examples are also given to support the results. This document examines the state of art in software products measurement, with focus in the object oriented approach, which has become high popular because of his benefits: quick development, re- usability, complexity management, etc., all of this, characteristics that increase directly the quality of software products.
[Hari Om Sharan, Garima, Md. Haroon, Rajeev Kumar. **Measurement of Software Complexity in Object Oriented Systems.** *Rep Opinion* 2014;6(10):70-75]. (ISSN: 1553-9873). http://www.sciencepub.net/report. 14

Keywords: Software Testing, Software Testability, Simplicity, Complexity.

Introduction

For producing high quality object oriented applications, it is necessary to develop a strong emphasis on design aspects, especially during the early phases of software. Design metrics play an vital role in helping developers to appreciate design aspects of software to improve software quality.It is clear that software measurement is necessary for the software development process to be successful. The main goals of the software measurement are:

- Evaluate the software systems.
- Improve quality of software systems.
- Identify and correct problems early.
- Defend and justify decisions.

One side of the concept of software engineering is the idea that the software should be under control. As De Marco said:”You cannot control what you cannot measure”[De Marco, 1982]. Fenton and Pfleeger added: ”You cannot predict what you cannot measure” [Fenton and Pfleeger, 1997].

Most of the measure strategies have as the main goal to evaluate the different characteristics of software quality, such as reliability, ease of use, maintainability, robustness.

In this paper we are presenting the object-oriented software metrics proposed by Chidamber, Kemerer and several studies were conducted to validate the metrics. Chidamber, Kemerer proposed six software metrics as Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), Number of Children (NOC) , Coupling Between Object classes (CBO), Response For a Class (RFC), Lack of Cohesion in Methods (LCOM)[1,2,3].

Basic Features of object-oriented technology

Object-Oriented Technology: A way to develop and package Software that draws heavily from common experience and the manner in which real world objects relate to each other.

Object-Oriented Systems: All programming languages, tools and methodologies that support Object-Oriented Technology. The main properties of object oriented technology are following:

- ➢ Objects
- ➢ Classes
- ➢ Data abstraction and encapsulation
- ➢ Inheritance
- ➢ Polymorphism
- ➢ Dynamic binding

1. **Object:** Objects are the basic run time entities in an object oriented system. They may represent a person, a bank account or any item that the program has to handle. They may also represent user-defined data such as vector, time and lists.
2. **Classes:** The entire set of data and code of an object can be made a user-defined data type with the help of class. In fact, objects are variable of the type class. Once a class has been defined, we can create any number of objects belonging to that class. Or we can say that a class is a collection of object of similar type.
3. **Abstraction:** An essential element of object-oriented technology is abstraction. Abstraction refers to the act of representing essential feature without including the background details or explanations. Or in other word we can say that an abstraction is a mechanism that allows a complex, real-world situation to be represented using a simplified model. Object orientation abstracts the real world based on objects and their interactions with other objects. For example, one possible abstraction of a color is the RGB model.
4. **Encapsulation:** The wrapping up of data and functions into a single unit (called class) is known as encapsulation Or the process of hiding all the internal details of an object from the outside world.

5. **Inheritance:** Inheritance is the process by which one object acquires the properties of another object. This is important because it supports the concept of hierarchical classification. Most knowledge is made manageable by hierarchical (that is, top-down) classifications. For example, a Golden Retriever is part of the classification dog, which in turn is part of the mammal class, which is under the larger class animal. Without the use of hierarchies, each object would need to define all of its characteristics explicitly. However, by use of inheritance, an object need only define those qualities that make it unique within its class. It can inherit its general attributes from its parent. Thus, it is the inheritance mechanism that makes it possible for one object to be a specific instance of a more general case.

6. **Polymorphism:** Polymorphism (from the Greek, meaning "many forms") is a feature that allows one interface to be used for a general class of actions.

7. **Dynamic Binding**: Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic binding (Also known as late binding) means that a code associated with a given procedure call is not known until the time of the call at run time.

Object oriented metrics taxonomy

Software Engineering introduces the measures in each step in a life cycle of a software project, independently of the used model: waterfall, spiral.

Then, the metrics can be viewed from a three dimensional approach, with the next dimensions:

• Software attributes to measure (complexity, reusability)

• Step in the life cycle in which is done the measure (design, analysis)

• Granularity level in which the measure is taken (system level, program level, class level)

Metrics cannot be applied to any software attribute that want to be measured indiscriminately. The typical case of a mistaken measure is to measure the lines of code (LOC) as a complexity program measure, when this is valid as a measure of the size program, not as complexity program measure. Minkiewicz [5] considered the value of various measures of size, lines of code and function points. The model [8] estimated size, measured by function points [16] directly from a conceptual model of the system being built. A model proposed by Tan et al. estimated lines of code based on the counts of entities, relationships, and attributes from the conceptual data model [7]

In another way, all the metrics don't have to be taken in the implementation stage, although part of them are taken from this step. It should be desirable to get them in the earlier design step. The OO technology forces the growth of OO software metrics [15].

[Chindamber and Kemerer, 1994] proposed six metrics for object oriented design:

Weight Methods per Class (WMC),
Depth of Inheritance Tree (DIT),
Number of Children (NOC),
Coupling Between Object classes (CBO),
Response For a Class (RFC) and
Lack of Cohesion in Methods (LCOM)

• **Metric 1: Weighted Methods Per Class (WMC)**

This metric is an average of the number of methods per class, where each method is weighted by a complexity based on the type of method, the number of properties the method effects and the number of services this method provides to the system. The details of this weighting will be covered in more detail in later sections. This metric is the heart of the POPs count. Research indicates there are two prominent schools of thought in the determination of object-oriented metrics suitable for size estimation (remember this is size as it relates to effort and productivity). One uses a count of the total number of distinct objects [10], [12]. The other uses a count of the Weighted Methods Per Class of objects [9], [14], [11], [13]. While the number of objects has shown promise as a useful effort estimator, we favor using a WMC count for several reasons:

• Methods relate to behavior and in so doing provide a metric that has meaning to non-software savvy individuals.

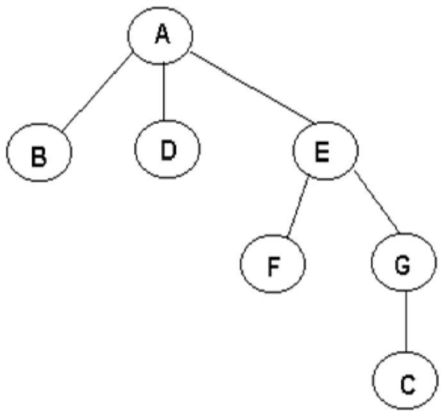
• Intended behaviors of the system are known early in the analysis, making it easier to develop a credible estimate early in the software lifecycle.

• WMC counting methods can be established to impose some rigor on the counting process.

Weighted methods per class encompass both the functionality and the inter-object communication in the POPs count.

• **Metric 2: Depth of Inheritance Tree (DIT)**

Each class described can be characterized as either a base class or a derived class. Those classes that are derived classes, fall somewhere in the class hierarchy other than the root. The DIT for a class indicates it's depth in the inheritance tree i.e. it is the length (in number of levels) from the root of the tree to that particular class. For example, in Figure 2, the DIT for Class C is 3 because there are three levels between the root, A, and class C. The average DIT, along with TLC and NOC, is used to help establish the reuse through inheritance dimension and the overall system size.



AvgDIT = ((1x0) + (3x1) + (2x2) + (1x3))/7

AvgNOC= ((1x3) + (1x2) + (1x1)) / 3 = 2

TLC=1

Sample Inheritance Tree

- **Metric 3: Number Of Children (NOC)**

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

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.

• 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 Objects (CBO)**

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

• 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 encapsulation, inter-object class couples should be kept to a minimum. 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)

The response set of a class (RFC) is defined as set of methods that can be potentially executed in response to a message received by an object of that class. No ambiguity or inadequacy is reported for this metric. RFC measures both external and internal communication, but specifically it includes methods called from outside the class [3, 6].

It is given by

RFC=|RS|, where RS, the response set of the class, is given by

$$RS = M_i \cup_{all\ j} \{R_{ij}\}$$

where Mi = set of all methods in a class (total n) and

Ri = {Rij} = set of methods called by Mi.

RFC is more sensitive measure of coupling than CB since it considers methods instead of classes

• Metric 6: Lack of Cohesion in Methods (LCOM)

This metric is a count of the number of disjoint method pairs minus the number of similar method pairs. The disjoint methods have no common instance variables, while the similar methods have at least one common instance variable.

The appearance of the Unified Modelling Language (UML) [Booch et al., 1999] as a standard of modelling object oriented information systems have provided a great contribution toward building quality object oriented systems. In [Genero et al., 2000] propose a set of metrics in order to assess the complexity of UML class diagrams from the relations in UML, such as association, aggregation. 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 [3, 4].

Consider a class C1 with n methods M1, M2,….,Mn. Let (Ij) = set of all instance variables used by method Mi. There as n such sets {I1},….{In}. Let $P = \{(I_i, I_j) \mid I_i \cap I_j = 0\}$ and $Q = \{(I_i, I_j) \mid I_i \cap I_j \neq 0\}$. If all n sets

{(Ii),….(In)} are 0 then P=0

LCOM=|P| - |Q|, if |P|>|Q|

= 0 otherwise

Benefits of object-oriented system

The Advantage or benefits of object oriented system are following:

- The use of objects as basic modules assists the designer to model complex real-world systems (Model Complexity).
- The flexibility of object-oriented code allows a rapid response to changes in their requirements.
- The reuse of standard components reduces both the development time for new applications and the volume of code generated.
- The increased maintainability of software makes it more reliable and reduces maintenance costs.
- Improve Productivity
- Designed for Change

Complexity Measurement

Cyclomatic complexity is software metric (measurement). It was developed by Thomas J. McCabe [7] and is used to measure the complexity of a program. It directly measures the number of linearly independent paths through a program's source code. Cyclomatic complexity is computed using the control flow graph of the program: the nodes of the graph correspond to the commands of a program, and a directed edge connects two nodes if the second command might be executed immediately after the first command. A method with a low cyclomatic complexity may imply that decisions are deferred through message passing, not that the methods is not complex. The 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 [3].

The cyclomatic complexity of a flow graph is as follows

$$M = E - N + 2P$$

Where

M = Cyclomatic complexity

E = Number of edges of the graph

N = Number of nodes of the graph

P = Number of connected components.

Step 1. Testability Analysis

Example:

For understanding and the analysis the role of complexity in the software testability we are taking examples of vending machine.In this example, in the first step we measure the testability by using the method of John McGregor and S. Srinivas . Than in the second step we draw the Control flow graph and find the complexity of the program.

Vending Machine

```
1. public class VendingMachine
2. {
3. final private int COIN = 25;
4. final private int VALUE = 50;
5. private int totValue;
6. private int currValue;
7. private Dispenser d;
8. public VendingMachine()
9. {
10.totValue = 0;
11.currValue = 0;
12.d = new Dispenser();
13.}
14. public void insert()
15. {
16. currValue += COIN;
17. System.out.println("Current value = " + currValue );
18. }
19. public void return()
20. {
21. if ( currValue == 0 )
22. System.err.println( "no coins to return" );
23. else
24. {
25. System.out.println("Take your coins");
26. currValue = 0;}
27. }
28. public void vend( int selection )
29. {
30. int expense;
31. expense = d.dispense( currValue, selection );
32. totValue += expense;
33. currValue -= expense;
34. System.out.println( "Current value = " + currValue );
35. }
36.}
```

Table 4.1

S.No	Method Name	Visibility Component(ζ)	Method Testability(ή)	Class Testability (θ)
1	VendingMachine()	3/3=1	2*1=2	2
2	void insert()	3/3=1	2*1=2	
3	void return()	3/3=1	2*1=2	
4	void vend()	4/4=1	2*1=2	

Cyclometic Complexity

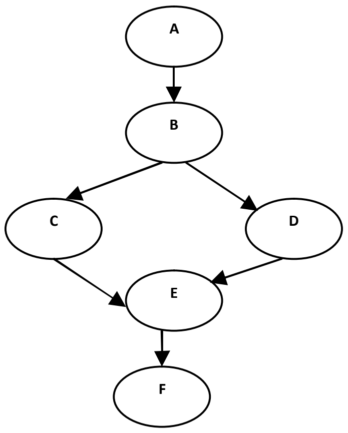


Fig shows the flow graph of the vending machine, its complexity is **2.**

Implimentation
Base Converter:

This program enables a user to convert from numbers of different bases to numbers of different bases. The number bases supported are decimal, binary, hexadecimal, octal, and a user defined base. This means that you can theoretically convert from any base to any base if you so choose.

This project has only one class. The testability and complexity analysis of this project is as follows:

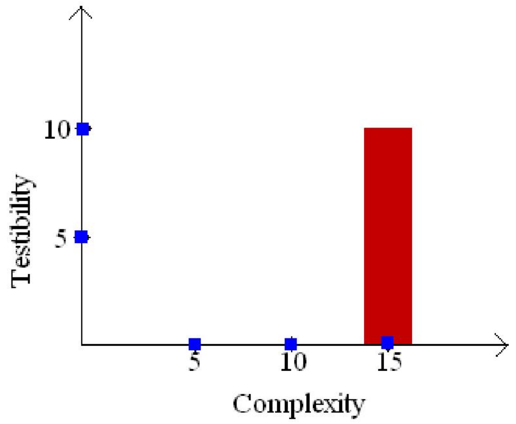


Fig. Base converter testability and complexity graph

S. No	Class No.	LOC	Testability	Complexity
1	1	589	10	15

Complexity of Base Converter is = 15

References
1. Arti Chhikara, R.S.Chhillar, Sujata Khatri,Evaluating The Impact Of Different Types Of Inheritance On The Object Oriented Software Metrics, International Journal of Enterprise Computing and Business Systems,Volume 1 Issue 2 July 2011 ISSN (Online): 223-8849 http://www.ijecbs.com.
2. Dr. Rkesh Kumar, Gurvinder Kaur, Comparing Complexity in Accordance with Object Oriented Metrics, International Journal of Computer Applications (0975 –8887) Volume 15– No.8, February 2011.
3. Amjan Shaik, C. R. K. Reddy, Bala Manda, Prakashini. C, Deepthi. K, An Empirical Validation of Object Oriented Design Metrics in Object Oriented Systems Journal of Emerging Trends in Engineering and Applied Sciences (JETEAS) 1 (2): 216-224,2010 (ISSN: 2141-7016)].
4. Dr. M.P.Thapaliyal and Garima Verma."Software Defects and Object Oriented Metrics" - An Empirical Analysis. International Journal of Computer Applications 9(5):41–44, November 2010.
5. [MIN09] Minkiewicz A., "The evolution of software size: A search for value," CROSSTALK, Vol. 22, No. 3,2009 pp. 23-26.
6. Ms Puneet Jai kaur, Ms Amandeep Verma , Mr. Simrandeep (2007) Thapar3,"Software Quality Metrics for Object-Oriented Environments, Proceedings of National Conference on

Challenges & Opportunities in Information Technology (COIT-2007), RIMT-IET, Mandi Gobindgarh. March 23.
7. [TAN06]Tan H. B. K., Y. Zhao, and H. Zhang, "Estimating LOC for information systems from their conceptual data models," Proceedings of the 28th International Conference on Software Engineering, Shanghai, China, ACM Press, New York, 2006, pp. 321-330.
8. [FRA06] Fraternali P., M. Tisi, and A. Bongio, "Automating function point analysis with model driven development," Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research, Toronto, Canada, ACM Press, New York, 2006, pp. 1-12.
9. [BRA06]Braz M. R. and S. R. Vergilio, “Software Effort Estimation Based on Use Cases”, Proceedings of 30th Annual International Computer Software and Applications Conference (COMPASAC ’06), IEEE Computer Society, September 2006, pp. 221-228.
10. [COSO5] Costagliola G., F. Ferrucci, G. Tortora, and G. Vitiello, “Class Point: An Approach for the Size Estimation of Object-Oriented Systems”, IEEE Transaction on Software engineering, Vol. 31, No. 1, January 2005, pp. 52-74.
11. [CHU95]Chucher N.I. and M.J. Shepperd, “Comments on a metrics Suite for Object-oriented Design” IEEE Transaction on Software Engineering, Vol. 21, No.3, 1995, pp. 263-265.
12. [CHI94]Chidamber S. R. and C. F. Kemerer, “A Metrics Suite for Object Oriented Design”, IEEE Transactions on Software Engineering, Vol. 20, No. 6, June 1994, pp. 476-493.
13. [GRA92]Grady R.B., “Practical Software Metrics for Project Management and Process Improvement” (Prentice Hall, Englewood Cliffs, NJ, 1992; ISBN: 0-13-720384-5).
14. [CHI91]Chidamber S. R. and C. F. Kemerer, “Towards a Metrics Suite for Object Oriented Design”, Proceeding on Object Oriented Programming Systems, Languages and Applications Conference (OOPSLA’91), ACM, Vol. 26, Issue 11, Nov 1991, pp. 197-211.
15. [CA 1991] Booch.G,” Object-Oriented Design and Application”, Benjamin/Cummings, Mento Park, CA, 1991.
16. [ALB83] Albrecht A. and J. Gaffney, "Software function, source lines of code and development effort prediction," IEEE Transactions on Software Engineering, Vol. 9, 1983,pp.639-648.

10/17/2014