Evaluation and Metrication of Object Oriented System

Kaur Amandeep, Singh Satwinder and Kahlon K. S

Abstract—**This paper presents some advances towards the quantitative evaluation of design attributes of object oriented software systems. We believe that these attributes can express the quality of internal structure, thus being strongly correlated with quality characteristics like analyzability, changeability, stability and testabilility, which are important to software developers and maintainers. In order to measure the Object Oriented design characteristics, a suite of metrics have been adopted. A motivation behind the metrics suite is the coverage of the basic structural mechanisms as encapsulation, inheritance, polymorphism, reusability, Data hiding and message-passing. Data was collected from a project based on object oriented paradigms to calculate the metrics, which was developed using a sequential life cycle model.**

I. INTRODUCTION

IN recent years we have seen the increasing use of the object oriented paradigm in software development. The use of object oriented software development techniques introduces new elements to software complexity both in software development process and in the final product [4]. The backbone of any software system is its design. It is the skeleton where the flesh (code) will be supported. The Object-Oriented (OO) paradigm includes a set of mechanisms such as inheritance, encapsulation, and polymorphism and message-passing that is believed to allow the construction of designs where those features are enforced. Many object-oriented metrics have been proposed specifically for the purpose of assessing the design of a software system. However, most of the existing approaches for measuring these design metrics involve only some of the aspects of object oriented paradigms. As a result, it is not always clear the design quality of code. Realizing the importance of software metrics, numbers of metrics have been defined for software [6].

Amandeep kaur is with Computer Science & Engineering Department, Govt Polytechnic College, Mohali, Distt. Mohali, Punjab, India (phone: +91- 94633-94617; e-mail: aman12484@gmail.com).
Satwinder Singh is Lecturer with Baba Banda Singh Bahadur Engineering College, Fatehgarh Sahib (Punjab)-India.
Dr. K.S.Kahlon is working with Guru Nanak Dev University, Amritsar.

These metrics try to capture different aspects of software product [5] and its process. Some of the metrics also try to capture the same aspects of software e.g., there are a number of metrics to measure the coupling between different classes. To analyze metrics chosen for this work, their values are computed for project.

II. OBJECTIVES

The objectives of the paper are

- To find whether each measure is independent or we can choose a subset of these metrics having equal utility as the original metric set.
- To analyze a system performance on object oriented grounds and measure the design and code quality.
- To cover the basic structural mechanisms of the object-oriented paradigm.

III. METRICS SET AND EMPIRICAL DATA COLLECTION

The list of metrics chosen for this study is given in
Figure I. Design and code of this project is available on sourceforge.net. Project is developed in C++ language and is referred as WinSCP.PROJECT is an open source SFTP client and FTP client for Windows. Its main function is the secure file transfer between a local and a remote computer. Beyond this, WinSCP offers basic file manager functionality. It uses Secure Shell (SSH) and supports, in addition to Secure FTP, also legacy SCP protocol. It was the first ever GUI SCP (and later SFTP) client for Windows. The metrics chosen for analysis can be divided into 7 categories viz. size, coupling, cohesion, inheritance, information hiding, polymorphism and reuse metrics.

S.No	Metric Object-Oriented	Attribute
1	Response for a Class (RFC)	Class
2	Number of Attributes per Class (NOA)	Class
3	Number of Methods per Class (NOM)	Class
4	Weighted Methods per Class (WMC)	Class
5	Coupling between Objects (CBO)	Coupling
6	Data Abstraction Coupling (DAC)	Coupling
7	Message Passing Coupling (MPC)	Coupling
8	Coupling Factor (CF)	Coupling
9	Lack of Cohesion (LCOM)	Cohesion
10	Tight Class Cohesion (TCC)	Cohesion
11	Loose Class Cohesion (LCC)	Cohesion
12	Information based Cohesion (ICH)	Cohesion
13	Method Hiding Factor (MHF)	Information Hiding
14	Attribute Hiding Factor (AHF)	Information Hiding
15	Number of Children (NOC) [10]	Inheritance
16	Depth of Inheritance (DIT) [10]	Inheritance
17	Method Inheritance Factor (MIF)[15]	Inheritance
18	Attribute Inheritance Factor (AIF)[15]	Inheritance
19	Number of Methods Overridden by a subclass (NMO)	Polymorphism
20	Polymorphism Factor (PF)	Polymorphism
21	Reuse ratio	Reuse
22	Specialization ratio	Reuse

FIGURE I: METRICS FOR OBJECT ORIENTED SOFTWARE

IV. METRICS DEFINITIONS AND APPLICATIONS

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 [2] [3]. It is often difficult to determine which metric is more useful in which area. .A few metrics are explained using practical applications.

SIZE METRICS

a) Number of Attributes per Class (NOA): It counts the total number of attributes defined in a class[16]. In figure 1, Number of Attributes (NOA) for TexternalConsole class is 7. So NOA = 7 for TExternalConsole class

.b) Number of Methods per Class (NOM): It counts number of methods defined in a class[16]. In fig 1 (NOM) for TExternalConsole class is 4.

c) Weighted Methods per Class (WMC): WMC is a count of sum of complexities of all methods in a class[10].

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

TExternalConsole

Print()
Input()
choice()
SetTitle()

bool FPendingAbort;
HANDLE FRequestEvent;
HANDLE FResponseEvent;
HANDLE FCancelEvent;
HANDLE FFileMapping;
bool FLimitedOutput;
static const int PrintTimeout = 5000;

FIGURE II: CLASS DIAGRAM FOR WinSCP

For complexities to be unity, the WMC = *n*, (number of methods in the class). In Fig 1, WMC for TExternalConsolet is 4.

d) Response For a Class (RFC): It is number of methods in the set of all methods that can be invoked in response to a message sent to an object of a class. It includes all methods accessible within the class hierarchy. It looks at the combination of the complexity of a class through the number of methods and the amount of communication with other classes. In class TSFTPPacket, there are 79 methods which can be invoked in response to a message sent to an object of a class[10].

COUPLING METRICS

e) Coupling Between Objects (CBO): Two classes are coupled when methods declared in one class use methods or instance variables defined by the other class [10]. In Fig 2, TConsole class contains declarations of instances of the classes TConsoleRunner and TexternalConsole.

The value of metric CBO for class TConsole is 2 and for class TConsoleRunner and TExternalConsole is zero.

f) Data Abstraction Coupling (DAC): It provides the ability to create user-defined data types called Abstract Data Types (ADTs) [16]. Li and Henry defined Data Abstraction Coupling (DAC) as:

DAC = no. of ADTs defined in a class

In Fig 2 there is one DAC= ADTs in class TConsoleRunner (FSynchronizeController)

g) Message passing Coupling (MPC): Li and Henry defined Message Passing Coupling (MPC) metric as “number of send statements defined in a class” [16]. So if two different methods in class A access the same method in class B, then MPC = 2. In fig 2, MPC value for class TConsole is 4 as methods in class TConsole call TExternalConsole **::** Print() , TExternalConsole **::** Input(), TConsoleRunner**::** Print(), TConsoleRunner**::** Input().

h) Coupling Factor (CF): Coupling can be Dynamic Coupling or Static Coupling among class instances. It is desirable that classes communicate with as few other classes and exchange as little information as possible [15]. It is formally defined as:

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

Where TC is total number of classes

$$(C_c, C_s) = \begin{cases} 1 \; iff \; C_c \Rightarrow C_s \wedge C_c \neq C_s \\ 0 \; oherwise \end{cases}$$

Couplings due to inheritance are not included in CF, because a class is heavily coupled via inheritance. If no classes are coupled, CF = 0 %. If all classes are coupled with all other classes, CF = 100 %.

COHESION METRICS

i) Lack of Cohesion in Methods (LCOM): LCOM = number of different methods within a class with reference to a given instance variable [10]. It measures the degree of similarity of methods by instance variable or attributes. Consider a class C1 with n methods M1,…., Mn. Let (Ij) = set of all instance variables used by method Mi. LCOM = | P | - | Q |, if | P | > | Q |

= 0 otherwise

In Fig1, there are four methods M1, M2, M3 and M4 in class Book.

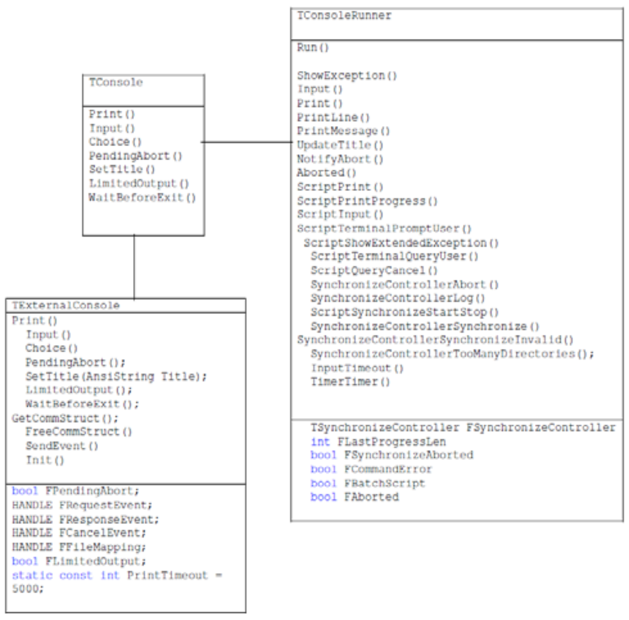

FIGURE III: CLASS DIAGRAM

I1 = {Str, From Beginning},I2 = {Str, Echo, Timer},I3 = {Option, Cancel, Break, Timeout},I4 = {Title}

$I_1 \cap I_2$ = str,$I_1 \cap I_3$ = null,$I_1 \cap I_4$ = null,$I_2 \cap I_3$ = timer,$I_2 \cap I_4$ = null,$I_3 \cap I_4$ = null

$I_1 \cap I_2$, $I_2 \cap I_3$ are non null but I1∩I3, I1∩I4, $I_2 \cap I_4$, $I3 \cap I_4$ are null sets.

LCOM is 2 if numbers of null intersections are not greater than number of non-null intersections. Hence LCOM in this case is 0 [|P|= 4 |Q|= 2]. Thus a positive high value of LCOM implies that classes are less cohesive. So a low value of LCOM is desirable.

j) Tight Class Cohesion (TCC): The measure TCC is defined as the percentage of pairs of public methods of the class with common attribute usage [16]. In Fig.1, methods defined in class TExternalConsole access the following attributes:

Print = {Str, FromBegining}

Input = {Str, Echo, Timer}

Choice = {Option, Cancel, Break, Timeout, Timer}

SetTitle = {Title}

All methods in class Book are public. Number of pairs of methods = 28.

Methods pairs with common attribute usage = {Print, Input}, and {Input, Choice}

TCC = 2 /28 * 100 = 7.142

k) Loose Class Cohesion (LCC): The measure LCC is defined as the percentage of pairs of public methods of the class, which are directly or indirectly connected [16]. In fig1, LCC for a class TExternalConsole is same as TCC that is 7.142 % as there is no direct invocation.

l) Information flow based Cohesion (ICH): ICH for a class is defined as the number of invocations of other methods of the same class, weighted by the number of parameters of the invoked method [16]. In Fig1, ICH is zero as no method is called by function of same class TExternalConsole.

INHERITANCE METRICS

m) Depth of Inheritance Tree (DIT): It is defined as the maximum length from the node to the root of the tree and measured by the number of ancestral classes. In Figure 4, DIT for TSFTPLoadFilesPropertiesQueue class is 2 as it has 2 ancestral classes TSFTPFixedLenQueue and TSFTPPacket. DIT for is TSFTPFixedLenQueue 1 as it has one ancestral class TSFTPPacket.

n) Number of Children (NOC): It is defined as the number of immediate subclasses. In Figure 4, NOC value for class TSFTPPacket is 2.

o) Method Inheritance Factor (MIF): MIF is defined as the ratio of the sum of inherited methods in all classes of the system under consideration to the total number of available methods for all classes.

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

Where, *Ma(Ci) = Mi(Ci) + Md(Ci)*

TC= total number of classes

Md(Ci) = the number of methods declared in a class

Mi(Ci) = the number of methods inherited in a class

p) 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 for all classes. AIF is suggested to express the level of reuse in a system.

$$\mathrm{AIF} = \frac{\sum_{i=1}^{TC} A_d(C\ )}{\sum_{i=1}^{TC} A_a(C_i)}$$

$$\text{where, } A_a(C_i) = A_i(C_i) + A_d(C_i)$$

TC= total number of classes

Ad (Ci) = number of attribute declared

Ai (Ci) = number of attribute inherited

INFORMATION HIDING METRIC

q) Attribute Hiding Factor (AHF): This metric is the ratio of hidden (private and protected) attributes to total attributes and is for the measurement of encapsulation and information hiding [15].

r) Method Hiding Factor (MHF): This metric is the ratio of the total inherited methods and total methods defined [15].

POLYMORPHISM METRICS

s) Polymorphism Factor (PF): It measures the degree of method overriding in the class inheritance tree[15].

$$\mathrm{PF} = \frac{\sum_{i=1}^{TC} M_o(C_j)}{\sum_{i=1}^{TC} [M_n(C_i) \times DC(C_i)]}$$

$$M_n(C_i) = \text{Number of New Methods}$$
$$M_o(C_i) = \text{Number of Overriding Methods}$$
$$DC(C_i) = \text{Descendants Count}$$

In this project no method or function is extended from existing class and function. So PF is zero.

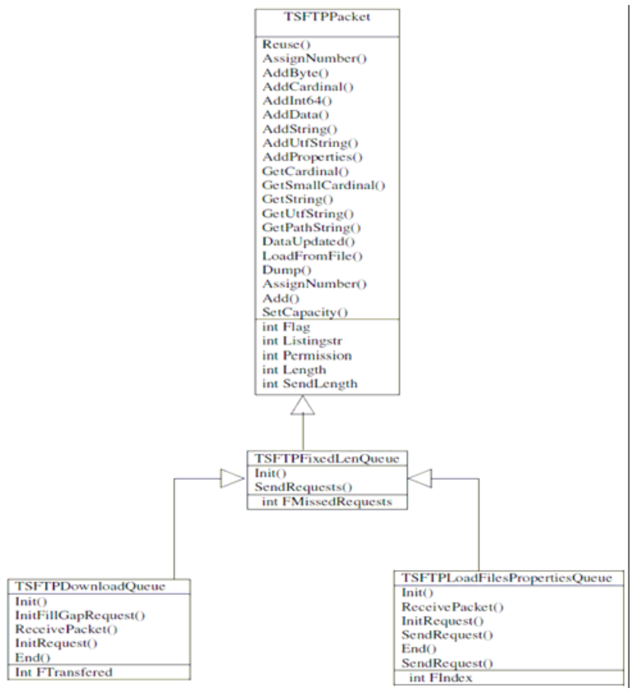

FIGURE IV: CLASS DIAGRAM

*t) Number of Methods Overridden by a subclass (NMO):*When a method in a subclass has the same name and type signature as in its superclass, then the method in the superclass is said to be overridden by the method in the subclass[16]. The value of metric is 2 for class TConsole.

REUSE METRIC

u) Reuse ratio: U, is given by [16]

U=No. of superclasses/ Total no. classes 13/59 = 0.220

v) Specialization Ratio (S): S = [16]Number of subclasses/ Number of superclasses

46/13 = 3.53

V. RESULTS

In this section Figure V gives results of class level metrics and Figure VI gives results of system level metrics. The metrics chosen for analysis can be divided into 7 categories viz. size, coupling, cohesion, inheritance, information hiding, polymorphism and reuse metrics.

S.NO.	METRIC OBJECT-ORIENTED	VALUE
1	NOA	5
2	NOM	20
3	WMC	20
4	RFC	79
5	CBO	2
6	DAC	1
7	MPC	4
8	LCOM	2
9	TCC	7.142
10	LCC	7.142
11	ICH	0
12	DIT	2
13	NOC	2
14	NMO	2

FIGURE V: CLASS LEVEL METRICS

S.NO	METRICS	VALUE
1	CF	50
2	MIF	0.491
3	AIF	0.676
4	MHF	0.305
5	AHF	0.375
6	PF	0
7	U	0.220
8	S	3.53

FIGURE VI: SYSTEM LEVEL METRICS

VI. CONCLUSION AND FUTURE WORK

In this paper, work has been done to explore the quality of design of commercial software components using object oriented paradigm. A number of object oriented metrics have been proposed in the literature for measuring the design attributes such as inheritance, coupling, cohesion, polymorphism, reusability etc. In this paper, metrics have been used to analyze various features of software component. The number of methods and the complexity of methods involved is a predictor of how much time and effort is required to develop and maintain the class. 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 on the part of the tester. This metric set can be applied on various projects and evaluate and compare the performance of the code using object oriented paradigm.

VII. REFERENCES

[l] F.B. Abreu and R. Carapuca, ”Candidate Metrics for Object- Oriented Software within a Taxonomy Framework,”]. System and Software, vol. 26, no. l, pp. 87-96, Jan. 1994.

[2] L. Briand, S. Morasca, and V. Basili, De$ning and Vdidating High- Level Design Metrics, Techtucal Report CS-TR-3301, Univ. of Maryland, Dept. of Computer Science, College Park, Md., 1994.

[3] L. Briand, S. Morasca, and V. Basili, ”Property Based Software Engineering Measurement,” IEEE Trans. Software Eng., vol. 22, no. 1, p. 68-86, Jan. 1996.

[4] I. Brooks, “Object-Oriented Metrics Collection and Evaluation with a Software Process,” Proc. OOPSLA ’93 Workshop Processes and Metrics for Object-Oriented Software Development, Washington, D.C., 1993.

[5] S.R. Chidamber and C.F. Kemerer, “A Metrics Suite for Object- Oriented Design,” IEEE Trans. Software Eng., vol. 20, no. 6, pp. 476493, June 1994.

[6] S.R. Chidamber and C.F. Kemerer, ”Authors Reply,” lEEE Trans. Software Eng., vol. 21, no. 3, p. 265, Mar. 1995.

[7] L.Briand , W.Daly and J. Wust, Unified Framework for Cohesion Measurement in Object-Oriented Systems. Empirical Software Engineering, 3 65-117, 1998.

[8] L.Briand , W.Daly and J. Wust, A Unified Framework for Coupling Measurement in Object-Oriented Systems. IEEE Transactions on software Engineering, 25, 91 121,1999.

[9] L.Briand , W.Daly and J. Wust, Exploring the relationships between design measures and software quality. Journal of Systems and Software, 5 245-273, 2000.

[10] S.R.Chidamber and C.F.Kamerer, A metrics Suite for Object-Oriented Design. IEEE Trans. Software Engineering, vol. SE-20, no.6, 476-493, 1994.

[11] N.Fenton et al, Software Metrics: A Rigorous and practical approach. International Thomson Computer Press, 1996.

[12] R.Harrison, S.J.Counsell, and R.V.Nithi, An Evaluation of MOOD set of ObjectOriented Software Metrics. IEEE Trans. Software Engineering, vol. SE-24, no.6, pp. 491-496 June1998.

[13] B.Henderson-sellers, Object-Oriented Metrics, Measures of Complexity.Prentice Hall, 1996.

[14] Lorenz, Mark & Kidd Jeff, Object-Oriented Software Metrics, Prentice Hall, 1994.

[15] F.B. Abreu and W. Melo, “Evaluating the Impact of Object-Oriented Design on Software Quality”,1996

[16] Simple class-level OO metrics (http://www.aivosto.com/project/help/pm-oo-misc.html)