JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2006**

Vol. 5, No. 8, November-December 2006

Empirical Study of Object-Oriented Metrics

K.K.Aggarwal, Yogesh Singh, Arvinder Kaur, Ruchika Malhotra
School of Information Technology, GGS Indraprastha University, Delhi 110006, India

Abstract

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 22 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, a review of the empirical study concerning chosen metrics and subset of these measures that provide sufficient information is given and metrics providing overlapping information are excluded from the set.

Key Words: Software Measurement, Object-Oriented Software, Coupling, Cohesion, Inheritance, Information-Hiding, Polymorphism

1 INTRODUCTION

A key element of any engineering process is measurement. Measures are used to better understand the attributes of the model that we create. But, most important, we use measurements to assess the quality of the engineered product or the process used to build it. Unlike other engineering disciplines, software engineering is not grounded in the basic quantitative laws of physics. Absolute measurements, such as voltage, mass, velocity or temperature, are uncommon in the software world. Instead, we attempt to derive a set of indirect measures that lead to metrics that provide an indication of the quality of some representation of software. Realizing the importance of software metrics, number of metrics have been defined for software. These metrics try to capture different aspects of software product and its process. Some of the metrics also try to capture the same aspect of software e.g., there are number of metrics to measure the coupling between different

Cite this article as follows: K.K.Aggarwal, Yogesh Singh, Arvinder Kaur, Ruchika Malhotra,: "Empirical Study of Object-Oriented Metrics", in *Journal of Object Technology*, vol. 5. no. 8, Novmeber-December 2006, pp. 149-173. http://www.jot.fm/issues/issue_2006_11/article5

classes. Software developers need to explicitly state the relation between the different metrics measuring the same aspect of software. As an example, we might be interested to know the size of a table. There can be number of metrics related to size of a table e.g., length of the table, breadth of the table, area of the table, diagonal of the table etc. But length and breadth measures are sufficient and others measures can be derived from them if required. Similarly in software, we need to identify the necessary metrics that provide useful information, otherwise the managers will be lost into so many numbers and the purpose of metrics would be lost. Since metrics are crucial source of information for decision making , a large number of OO metrics have also been proposed over the last decade to capture the structural quality of OO code and design and are related to external quality measures [Fenton96]. As the number of metrics available in literature is large, it becomes a tedious process to understand the computation of these metrics and draw inferences from their values. Secondly, as number of metrics proposed in literature are quite large as compared to number of features (e.g., coupling, cohesion, polymorphism, size and inheritance) captured by these metrics, our aim is 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 meet the above objectives, the following steps are taken:

1. Set of 22 metrics is first defined and then explained using examples. Their values are computed for three standard projects and interpretations are drawn regarding their inter-relationships from the metric values. The fault prone classes are also identified based on earlier empirical investigations.
2. To find whether all these metrics are independent or are capturing same underlying property of the object being measured, distribution of the metric values is computed and principal component analysis on these metric values is done.
3. The relationship between design measures and size of the class is analyzed to determine empirically whether the measure, even though it is declared as a coupling, cohesion, or inheritance measure, is essentially measuring size.

The paper is organized as follows: Section 2 describes the set of metrics and the data chosen for the analysis of these metrics. The explanation of the metrics and their computation on sample data is presented in section 3. Data analysis methodology is explained in section 4 and results of the analysis are presented in section 5. The discussion of work carried is presented in section 6.

2 METRICS SET AND EMPIRICAL DATA COLLECTION

The list of metrics chosen for this study is given in Table 1. To analyze metrics chosen for this work, their values are computed for three different projects. Design of these projects is presented in books “Introduction to Object-Oriented Analysis and Design” and “Object-Oriented and Classical Software Engineering”, authored by S.R Schach. Their respective codes are available on Internet [Schach04][Schach02] . The projects are developed in Java language and are referred here as Project 1, Project 2, and Project 3.

Project 1 is a software that assist a company named Martha Stockton Greengage Foundation (MSG) in making decisions whether to give loan to a couple by keeping property on mortgage. It also manages repayment of loan. Project 2 is software that helps an art dealer in making decision whether to buy a painting at the price quoted on the basis of certain criteria. Project 3 is software that assists Air Gourmet in deciding whether to continue supplying special meals (child, diabetic, low calorie, sea food etc.) to passengers who request them. The product will allow the client to enter a reservation, and generate the information needed to administer the special meal program.

S.No	Metric	Object-Oriented Attribute	Sources
1	Response for a Class (RFC)	Class	[Chidamber94]
2	Number of Attributes per Class (NOA)	Class	[Henderson96]
3	Number of Methods per Class (NOM)	Class	[Henderson96]
4	Weighted Methods per Class (WMC)	Class	[Chidamber94]
5	Coupling between Objects (CBO)	Coupling	[Chidamber94]
6	Data Abstraction Coupling (DAC)	Coupling	[Henderson96]
7	Message Passing Coupling (MPC)	Coupling	[Henderson96]
8	Coupling Factor (CF)	Coupling	[Harrison98]
9	Lack of Cohesion (LCOM)	Cohesion	[Chidamber94]
10	Tight Class Cohesion (TCC)	Cohesion	[Braind99]
11	Loose Class Cohesion (LCC)	Cohesion	[Braind99]
12	Information based Cohesion (ICH)	Cohesion	[Lee95]
13	Method Hiding Factor (MHF)	Information Hiding	[Harrison98]
14	Attribute Hiding Factor (AHF)	Information Hiding	[Harrison98]
15	Number of Children (NOC)	Inheritance	[Chidamber94]
16	Depth of Inheritance (DIT)	Inheritance	[Chidamber94]
17	Method Inheritance Factor (MIF)	Inheritance	[Harrison98]
18	Attribute Inheritance Factor (AIF)	Inheritance	[Harrison98]
19	Number of Methods Overridden by a subclass (NMO)	Polymorphism	[Henderson96]
20	Polymorphism Factor (PF)	Polymorphism	[Harrison98]
21	Reuse ratio	Reuse	[Henderson96]
22	Specialization ratio	Reuse	[Henderson96]

Table 1: Metrics for Object-Oriented Software

3 METRICS DEFINITIONS AND APPLICATIONS

In this section OO metrics chosen in this work are defined and their application on example systems is shown. The metrics chosen for analysis can be divided into 7 categories viz. size, coupling, cohesion, inheritance, information hiding, polymorphism and reuse metrics.

Size metrics

In this section four size metrics are discussed. These measure size of the system in terms of attributes and methods included in the class and capture the complexity of the class.

Number of Attributes per Class (NOA)

It counts the total number of attributes defined in a class. Figure 1 shows the class diagram of Book Information System. In this system, Number of Attributes (NOA) for Publication class is 2. So NOA = 2 for Publication class.

Number Of Methods per Class (NOM)

It counts number of methods defined in a class. In Figure 1, class Publication has two methods getdata( ) and display( ). Hence NOM = 2 for Publication class.

Weighted Methods per Class (WMC)

The WMC is a count of sum of complexities of all methods in a class. To calculate the complexity of a class, the specific complexity metric that is chosen (e.g., cyclomatic complexity) should be normalized so that nominal complexity for a method takes on value 1.0. Consider a class K_1, with methods M_1,........ M_n that are defined in the class. Let C_1,..........C_n be the complexity of the methods [Chidamber94].

$$\text{WMC} = \sum_{i=1}^{n} C_i \qquad (1)$$

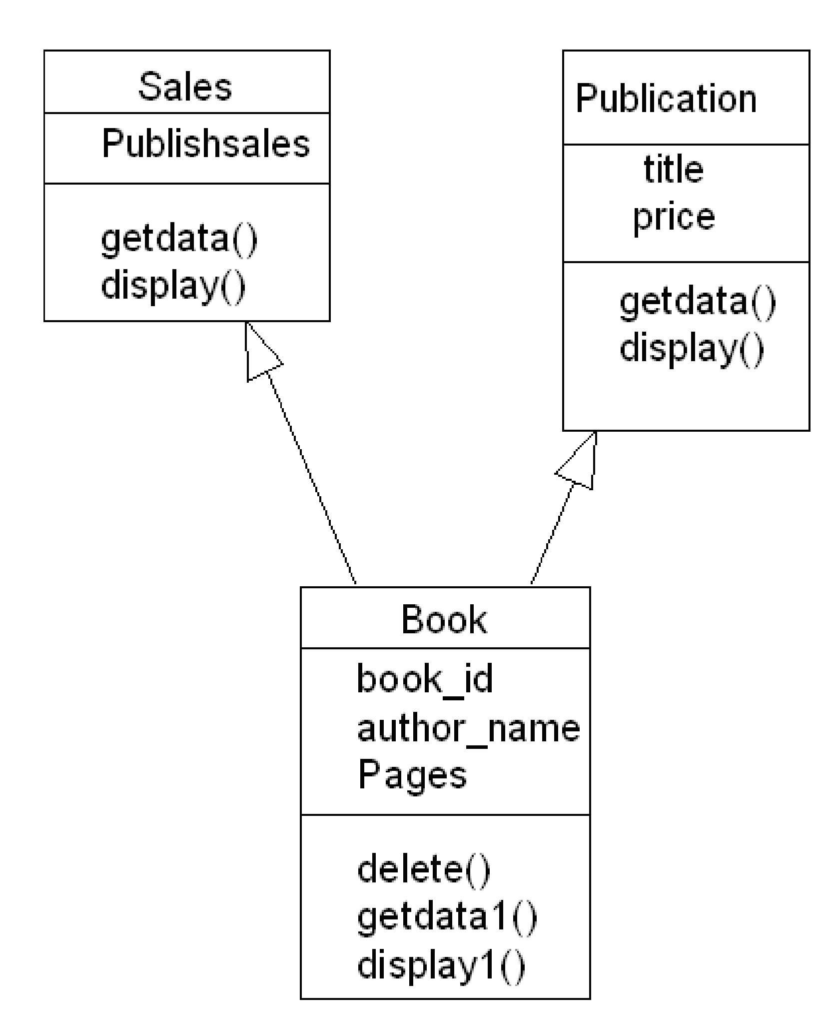


Figure 1: Class Diagram of Book Information system

If all method complexities are considered to be unity, then WMC = n, the number of methods in the class. In Figure 1, WMC for Book is 3 (considering each method complexity to be unity).

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

$$RS = M_i \cup_{all\ j} \{R_{ij}\} \tag{2}$$

where M_i = set of all methods in a class (total n) and $R_i = \{R_{ij}\}$ = set of methods called by M_i.

In Figure 1, class Book has two functions getdata1 and display1 which call methods Publication::getdata(), Sales::getdata(), Publication::display(), Sales::display().

$$RS = \{Book :: getdata1, Book :: display1, Book :: delete\} \cup \{Publication :: getdata,$$
$$Publication :: display\} \cup \{Sales :: getdata, Sales :: display\}$$
$$RFC = 7$$

Coupling metrics

Coupling relations increase complexity, reduce encapsulation, potential reuse, and limit understanding and maintainability.

Coupling Between Objects (CBO)

CBO for a class is count of the number of other classes to which it is coupled. Two classes are coupled when methods declared in one class use methods or instance variables defined by the other class. In Figure 2, Book class contains declarations of instances of the classes Publication and Sales. The Book class delegates its publication and sales issues to instances of the Publication and Sales classes. The value of metric CBO for class Book is 2 and for class Publication and Sales is zero.

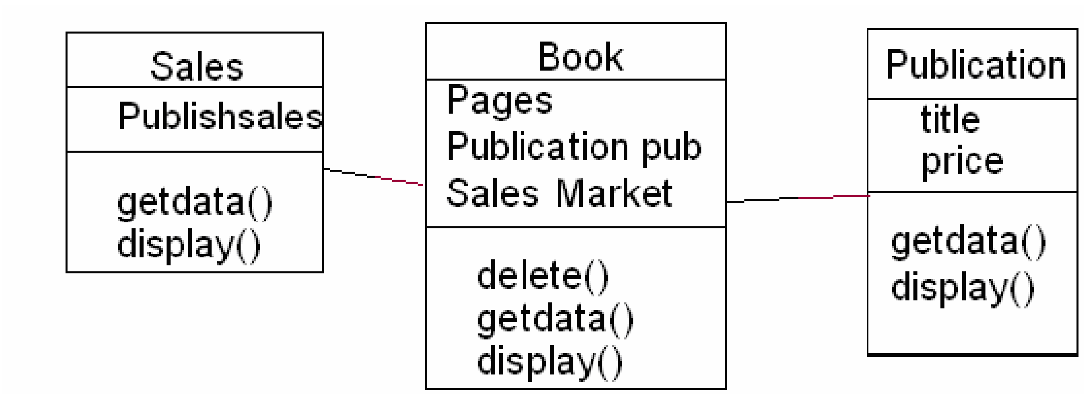


Figure 2: Class Diagram of Sales Information System

Data Abstraction Coupling (DAC)

Data Abstraction is a technique of creating new data types suited for an application to be programmed. It provides the ability to create user-defined data types called Abstract Data Types (ADTs).

Li and Henry defined Data Abstraction Coupling (DAC) as:

DAC = number of ADTs defined in a class

In Figure 2 there are two ADTs in class Book, pub and Market. DAC for Book class is 2.

Message passing Coupling (MPC)

Li and Henry defined Message Passing Coupling (MPC) metric as “number of send statements defined in a class”. So if two different methods in class A access the same method in class B, then MPC = 2. In Figure 2, MPC value for class Book is 4 as methods in class Book calls Sales::getdata(), Sales::display(), Publication::getdata(), Publication::display().

Coupling Factor (CF)

Coupling can be due to message passing (dynamic coupling) or due to semantic association links (static coupling) among class instances. It has been known that it is desirable that classes communicate with as few other classes and even when they communicate, they exchange as little information as possible. It is formally defined as:

$$\mathrm{CF} = \frac{\sum_{i=1}^{TC}\sum_{j=1}^{TC}\left[is\_client(C_i, C_j)\right]}{TC^2 - TC} \tag{3}$$

where TC is total number of classes

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

Couplings due to the use of the inheritance are not included in CF, because a class is heavily coupled to its ancestors via inheritance. If no classes are coupled, $\mathrm{CF} = 0\ \%$. If all classes are coupled with all other classes, $\mathrm{CF} = 100\ \%$.

Cohesion Metrics

Cohesion is a measure of the degree to which the elements of a module are functionally related. A strongly cohesive module implements functionality that is related to one feature of the software and requires little or no interaction with other modules. Thus we want to maximize the interactions within a module. Four cohesion metrics are discussed here.

Lack of Cohesion in Methods (LCOM)

Lack of Cohesion (LCOM) measures the dissimilarity of methods in a class by looking at the instance variable or attributes used by methods [Chidamber94] . Consider a class C_1 with n methods M_1, M_2…., M_n. Let (I_j) = set of all instance variables used by method M_i. There are n such sets $\{I_1\}$,…….$\{I_n\}$. Let $\mathrm{P} = \{(\mathrm{I}_i, \mathrm{I}_j) \mid \mathrm{I}_i \cap \mathrm{I}_j = 0\}$ and $\mathrm{Q} = \{((\mathrm{I}_i, \mathrm{I}_j) \mid \mathrm{I}_i \cap \mathrm{I}_j \neq 0\}$. If all n sets $\{(I_1\},.........(I_n)\}$ are 0 then P=0

$$\begin{aligned} \mathrm{LCOM} &= |\mathrm{P}| - |\mathrm{Q}|, \text{if } |\mathrm{P}| > |\mathrm{Q}| \\ &= 0 \text{ otherwise} \end{aligned} \tag{4}$$

In Figure 3, there are four methods M_1, M_2, M_3 and M_4 in class Book.

I_1={book_id, pub_id, book_name, author_name, price}

I_2={book_id}

I_3={book_name}

I_4={author_name}

$I_1 \cap I_2$, $I_1 \cap I_3$ and $I_1 \cap I_4$ are non - null but $I_2 \cap I_3$, $I_2 \cap I_4$ and $I_3 \cap I_4$ are null sets.

LCOM is 0 if numbers of null intersections are not greater than number of non-null intersections. Hence LCOM in this case is 0 [|P|=|Q|=3]. Thus a positive high value of LCOM implies that classes are less cohesive. So a low value of LCOM is desirable.

Tight Class Cohesion (TCC)

The measure TCC is defined as the percentage of pairs of public methods of the class with common attribute usage. In Figure 3, methods defined access the following attributes:
add_book = {book_id, pub_id, book_name, author_name, price}
delete = {book_id}
search_name = {book_name}
search_author = {author_name}
All methods in class Book are public. Number of pairs of methods = 6.
Methods pairs with common attribute usage = {add_book, delete}, {add_book, search_name} and {add_book, search_author}

$$\text{TCC} = \frac{3}{6} \times 100 = 50$$

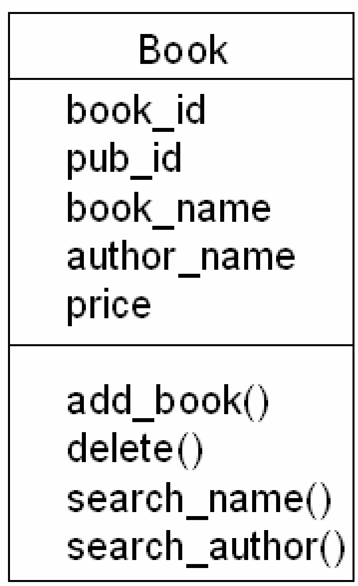


Figure 3: Class Diagram of class Book in Library Management System

Loose Class Cohesion (LCC)

In addition to direct attributes, this measure considers attributes indirectly used by a method. Method m directly or indirectly invokes a method m', which uses attribute a. LCC is same as TCC except that this metric also consider indirectly connected methods.

The measure LCC is defined as the percentage of pairs of public methods of the class, which are directly or indirectly connected. In Figure 3, LCC for class Book is same as TCC i.e.50% as there are no indirect invocations by the methods of class Book.

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.

In Figure 3, method delete calls function search_name, which is also function of the same class. Hence value for ICH is 1.

Inheritance Metrics

In this section, four different inheritance methods are considered for analysis.

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. In cases involving multiple inheritances, the DIT will be the maximum length from the node to the root of the tree. In Figure 4, DIT for Result class is 2 as it has 2 ancestor classes InternalExam/ External Exam and Student.

DIT for InternalExam and ExternalExam is 1 as it has one ancestor class Student.

Number of Children (NOC)

The NOC is the number of immediate subclasses of a class in a hierarchy. In Figure 4, NOC value for class Student is 2.

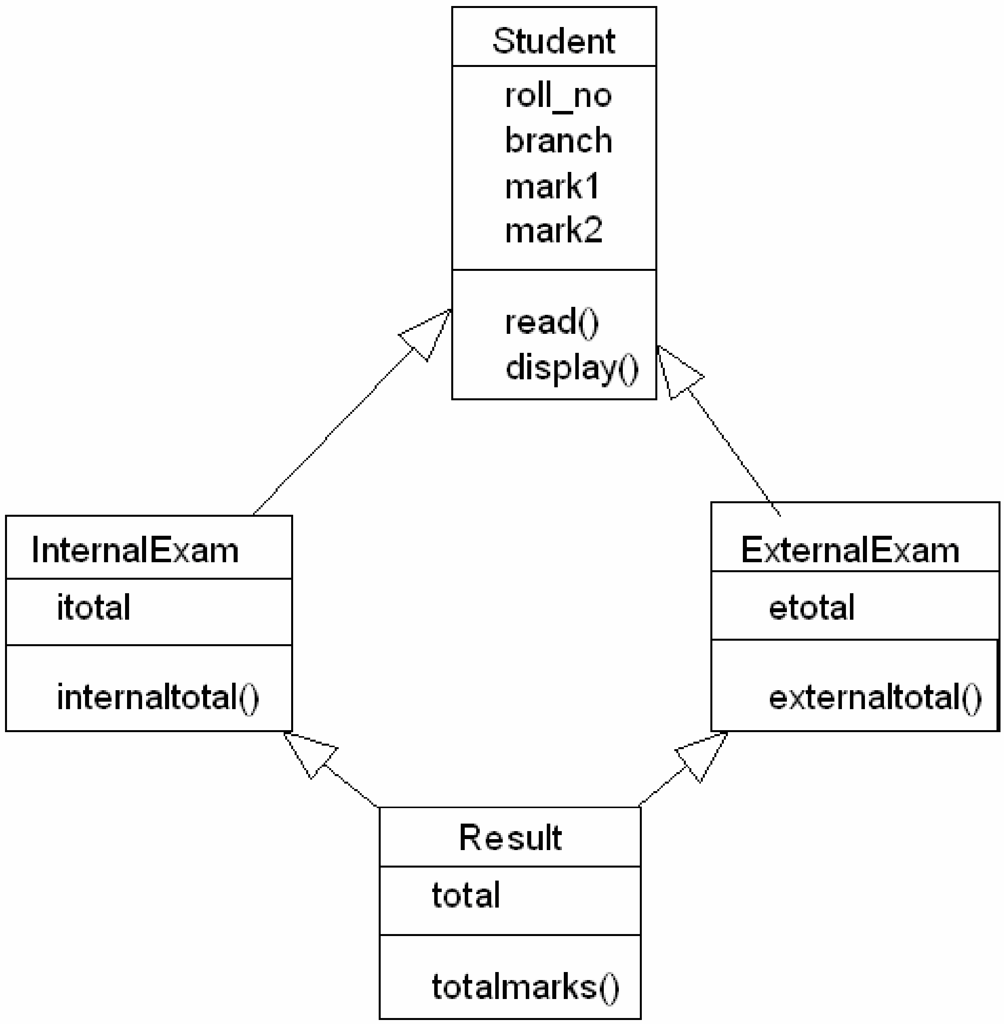


Figure 4: Class Diagram of Result Management System

Method Inheritance Factor (MIF)

It is system level metrics and is defined as follows:

$$\mathrm{MIF} = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)} \tag{5}$$

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

In Figure 4, in Student class, roll_no and branch are private attributes whereas mark1 and mark2 are protected attributes.
TC = 4 in Figure 4.
Let C_1 = Student class, C2 = InternalExam class, C3 = ExternalExam class and C4 = Result class.

$$\text{MIF} = \frac{M_i(C_1)+M_i(C_2)+M_i(C_3)+M_i(C_4)}{M_i(C_1)+M_i(C_2)+M_i(C_3)+M_i(C_4)+M_d C_1)+M_d(C_2)+M_d(C_3)+M_d(C_4)}$$

$M_i(C_1)$ = Number of Inherited Methods in Class Student = 0

$M(C_2)$ = Number of Inherited Methods in Class InternalExam = 2. Thus,

$$\text{MIF} = \frac{0+2+2+2}{11} = \frac{6}{11}$$

Attribute Inheritance Factor (AIF)

It is defined as follows:

$$\text{AIF} = \frac{\sum_{i=1}^{TC} A_d(C\ \ )}{\sum_{i=1}^{TC} A_a(C_i\ )} \qquad (6)$$

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

AIF is 0 % for class lacking inheritance

For the system in Figure 4, AIF can be calculated as follows:

Total Number of classes $(TC) = 4$

Let, C_1 = Student , C_2 = InternalExam, C_3 = ExternalExam, and C_4 = Result

$$\text{AIF} = \frac{A_i(C_1)+A_i(C_2)+A_i(C_3)+A_i(C_4)}{A_i(C_1)+A_i(C_2)+A_i(C_3)+A_i(C_4)+A_d C_1)+A(C_2)+A_d(C_3)+A_d(C_4)}$$

$A_i(C_1)$ = Number of Inherited Attributes in Class Student = 0

$A_i(C_2) = 2$, as two attributes mark1 and mark2 are inherited by sub class InternalExam

Similarly, $A_i(C_3) = 2$, $A_i(C_4) = 2$

$A_d(C_1) = 4$, as four attributes are declared in class Student.

Similarly, $A_d(C_2\ ) = A_d(C_3\ ) = A_d(C_4\ ) = 1$. Thus,

$$\text{AIF} = \frac{0+2+2+2}{13} = \frac{6}{13}$$

In Figure 4, $\text{AIF} = \frac{6}{13}$

Information Hiding Metrics

Information hiding allows a modification to be made to the internal operations of an object by hiding the implementation details behind a public interface.

Method Hiding Factor (MHF)

It is a measure of encapsulation defined as:

$$\text{MHF} = \frac{\sum_{i=1}^{TC} \sum_{m=}^{M_d(Ci)} \left(1 - V(M_{mi})\right)}{\sum_{i=1}^{TC} M_d(C_i)} \qquad (7)$$

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

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

where TC is the total number of classes, and

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

Thus, for all classes, C_1, $C_2 \ldots C_n$, 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.

The classes shown in Figure 5 are declared as follows:

```
class Person {
        public char name[25]; public int age;
        public void readperson(); public void displayperson();
};
class Student extends Person{
        public int roll_no[10];public float average;
        public void readstudent(); public void displaystudent(); public float getaverage();
};
class GradStudent extends Student{
        private char subject[25]; private char working[25];
        public void readit(); public void displaysubject(); public void workstatus();
};
```

n Figure 5, TC=Total number of classes=3

$$\text{MHF} = \frac{\sum_{i=1}^{3} \left[\sum_{i=1}^{Md(Ci)} \left(1\text{-}V(M_{mi})\right) \right]}{8}$$

M_{11} is method readperson() in class Person. It is visible in class Student and GradStudent

$$V(M_{11}) = \frac{\sum_{j=1}^{3} is\_visible(M_{11}, C_j)}{2}$$

$$= \frac{0+1+1}{2} = 1$$

Similarly, the visibilty of all the methods in class over other classes is calculated.

Hence, MHF = 0 as there are no private methods declared in all the classes in the system shown in Figure 5.

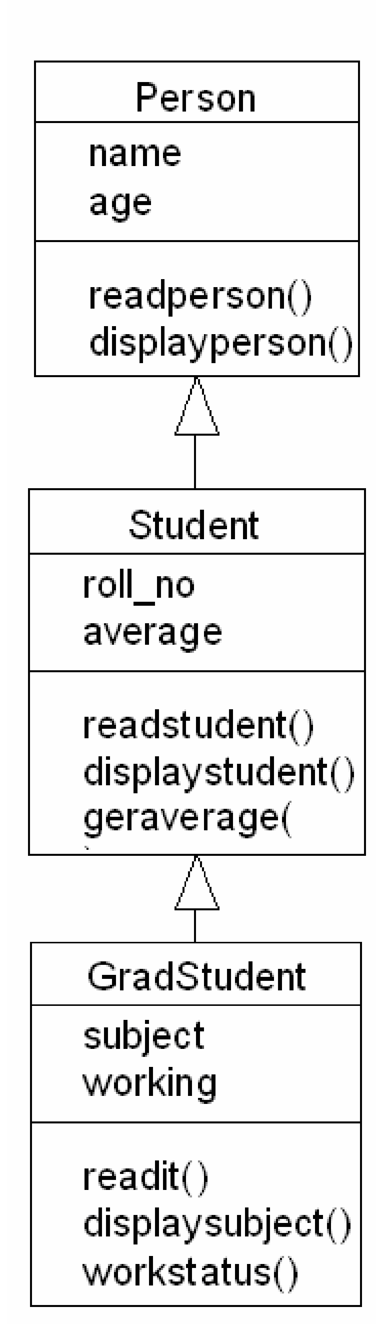


Figure 5: Class Diagram of University Management System

Attribute Hiding Factor (AHF)
It is a measure of encapsulation defined formally as:

$$\mathrm{AHF} = \frac{\sum_{i=1}^{TC}\sum_{a=1}^{Ad(Ci)}\left(1 - V(A_{ai})\right)}{\sum_{i=1}^{TC} A_d(C_i)} \tag{8}$$

where $A_d(C_i)$ is the number of methods declared in a class, and

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

where TC is the total number of classes, and

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

Thus, for all classes, C_1, $C_2 \ldots C_n$, an attribute counts as 0 if it can be used by another class, and 1 if it cannot. In Figure 5,

$$\mathrm{AHF} = \frac{\sum_{i=1}^{3}\left[\sum_{a=1}^{Ad(Ci)}\left(1\text{-}V(A_{ai})\right)\right]}{\sum_{1=1}^{TC} A_d(C_i)}$$

$$= \frac{\sum_{a=1}^{2}\left(1\text{-}V(A_{a1})\right) + \sum_{a=1}^{2}\left(1\text{-}V(A_{a2})\right) + \sum_{a=1}^{2}\left(1\text{-}V(A_{a3})\right)}{2+2+2}$$

$$= \frac{1\text{-}V(A_{11}) + 1\text{-}V(A_{12}) + 1\text{-}V(A_{13}) + 1\text{-}V(A_{21}) + 1\text{-}V(A_{22}) + 1\text{-}V(A_{23})}{6}$$

$$V(A_{11}) = \frac{\sum_{j=1}^{3} is\_visible\left(A_{11}, C_j\right)}{2}$$

$$= \frac{0+1+1}{2} = 1$$

Similarly $V(A_{12}),\; V(A_{21}),\; V(A_{22}) = 1 \; and \; for \; V(A_{23}), V(A_{13}) = 0$

As attributes 1 and 2 i.e., subject, working are private (not visible) to other classes Person and Student.

Hence, $\mathrm{AHF} = \frac{2}{6} = \frac{1}{3}$

Polymorphism Metrics

Polymorphism allows the implementation of a given operation to be dependent on the object that "contains" the operation.

Polymorphism Factor (PF)
The PF, metric is proposed as a measure of polymorphism. It measures the degree of method overriding in the class inheritance tree.

$$\text{PF} = \frac{\sum_{i=1}^{TC} M_o(C_j)}{\sum_{i=1}^{TC} [M_n(C_i) \times DC(C_i)]} \tag{9}$$

$M_n(C_i)$ = Number of New Methods

$M_o(C_i)$ = Number of Overriding Methods

$DC(C_i)$ = Descendants Count

In Figure 5, readstudent() and displaystudent() in class Student and readit() and displaysubject() in class GradStudent are renamed as readperson() and displayperson() . The classes are declared as follows:

```
class Person {
        public char name[25]; public int age;
        public void readperson(); public void displayperson();
};
class Student extends Person{
        public int roll_no[10];public float average;
        public void readperson (); public void displayperson (); public float getaverage();
};
class GradStudent extends Student{
        private char subject[25]; private char working[25];
        public void readperson (); public void displayperson (); public void workstatus();
};
```

Let, class C_1 = Person, C_2 = Student and C_3 = GradStudent

$$M_n(C_1) = M_n(C_2) = M_n(C_3) = 1$$

$$\sum_{i=1}^{TC} M_o(C_i) = 0 + 2 + 2 = 4$$

$DC(C_1) = 2$, as class Person has two descendants Student and GradStudent

Similarly, $DC(C_2) = 1$ and $DC(C_3) = 0$

$$\text{PF} = \frac{4}{2 \times 2 + 1 \times 1 + 1 \times 0} = \frac{4}{5}$$

[As readperson() and displayperson() are overridden in classes Student and GradStudent]

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. The value of metric is 2 for class Student declared for calculating PF metric.

Reuse Metrics

An object-oriented development environment supports design and code reuse, the most straightforward type of reuse being the use of a library class (of code), which perfectly suits the requirements. Yap and Henderson-sellers discuss two measures designed to evaluate the level of reuse possible within classes [Henderson96].

Reuse Ratio (U)

The reuse ratio, U, is given by

$$U = \frac{\text{Number of superclasses}}{\text{Total Number of classes}} \tag{10}$$

In Figure 5, the value of U is $\frac{2}{3}$

Specialization Ratio (S)

Specialization ratio, S, is given as

$$S = \frac{\text{Number of subclasses}}{\text{Number of superclasses}} \tag{11}$$

In Figure 5, Number of subclasses = {Student, GradStudent} and Number of superclasses= {Person, Student}. Thus, S=1

4 DATA ANALYSIS METHODOLOGY

In this section we describe the methodology used to analyze the metrics data computed for the projects. The procedure used to analyze the data collected for each measure is described in three stages: (i) data statistics, (ii) principal component analysis, and (iii) correlation to size.

I. **Descriptive Statistics**

   Within each case study, the distribution (mean, median) and variance (standard deviation) of each measure is examined. Low variance measures do not differentiate classes very well and therefore are not likely to be useful. Analyzing and presenting the distribution of measures is important for the comparison of different case studies.

II. **Principal Component Analysis**

   If a group of variables in a data set are strongly correlated, these variables are likely to measure the same underlying dimension (i.e., class property) of the object to be measured. Principal Component Analysis (PCA) is a standard technique to identify the underlying, orthogonal dimensions that explain relations between the variables in a data set. Principal components (PCs) are linear combinations of the standardized independent variables. PCs are calculated as follows. The first PC is the linear combination of all standardized variables which explain a maximum amount of variance in the data set. The second and subsequent PCs are linear combinations of all standardized variables, where each new PC is orthogonal to all previously calculated PCs and captures a maximum variance under these conditions.

   In order to identify variables with high coefficients, and interpret the PCs, we consider the rotated components. This is a technique where the PCs are subjected to an orthogonal rotation. There exist several strategies to perform such a rotation. We used the varimax rotation, which is the most frequently used strategy in the literature. Associated with each PC is its eigenvalue, which is a measure of the variance of the PC. Usually, only a subset of the PCs is selected for further analysis (interpretation, rotated components, etc.). Only PCs whose eigenvalue is larger than 1.0 are selected in this study.

The study must be replicated across different data sets to generalize the results.

III. **Correlation to size**

For each design measure, we analyze its relationship to the size of the class. This is mostly justified by the fact that size determines, to some extent, many of its external properties such as fault-proneness or effort [Braind00]. This is to determine empirically whether the measure, even though it is declared as a coupling, cohesion, or inheritance measure, is essentially measuring size. If a measure is strongly related to size, then there is no need to calculate this metric as size metrics are comparatively easy to find out. For the purpose of analyzing correlations with size, we calculated the Spearman's rho coefficient between each coupling, cohesion, and inheritance measure and the actual size measure in lines of code. A non-parametric measure of correlation was preferred, given the skewed distributions of the design measures that are usually observed.

5 ANALYSIS RESULTS

This section presents the analysis results, following the procedure described in descriptive statistics (Section 5.1), principal component analysis (Section 5.2) and correlation to size (Section 5.3). The principal component analysis and correlation to size is not done with system level measures. We have too few systems, thus a valid statistical analysis could not be performed for such measures.

Descriptive statistics

Descriptive statistics (min, max, mean, std. dev.) calculated are presented in this section. Table 2 represents descriptive statistics for class level metrics. Table 3 shows system level metric values for all three projects.

Metric	Project 1					Project 2					Project 3				
	Min	Max	μ	Medn	σ	Min	Max	μ	Medn	σ	Min	Max	μ	Medn	σ
NOA	0	9	4.5	4.5	3.6	0	11	4.5	4	4.1	0	11	4.8	4	4
NOM	1	12	6	5	4.6	1	30	10	5	10	1	24	8.8	8	7
WMC	1	12	6	5	4.6	1	30	10	5	10	1	24	8.8	8	7
RFC	1	31	20	25	13	1	54	14	10	16	2	48	14	11	15
CBO	0	2	1	1	1	0	5	2.2	2	2.2	0	6	2.2	1.5	2.6
DAC	0	1	0.5	0.5	0.7	0	0	0	0	0	0	2	1	1	1
MPC	0	4	1.7	1.5	1.7	0	13	4.7	3	5.9	0	24	8	5	9
LCOM	0	1	0.5	0.5	0.7	0	1	0.5	0.5	0.7	0	2	1	1	1
TCC	0	1	0.9	1	0.3	0	0.8	0.4	0.36	0.35	0	0.4	0.2	0.2	0.4
LCC	0	1	0.9	1	0.3	0	0.8	0.4	0.36	0.35	0	0.4	0.2	0.2	0.4
ICH	0	1	0.11	0	0.3	0	7	4	5	3.6	0	17	7.3	5	8.7
DIT	0	1	0.5	0.5	0.7	0	1	0.5	0.5	0.7	0	3	1.5	1.5	1.2
NOC	0	6	3	3	4.2	0	2	1	1	1.4	0	2	1	1	1
NMO	1	2	1.5	1.5	0.7	8	8	8	8	0	0	3	1.5	1.5	1.2

Table 2: Descriptive Statistics for Class-level Metrics

Where μ = Sample Mean ; σ =Std. Dev

Metric	Project 1	Project 2	Project3
CF	0.02	0.05	0.03
MIF	0	0.4	0.13
AIF	0.3	0.5	0.4
MHF	0	0	0
AHF	0.16	0.94	0.86
PF	0	0.8	0.4
U	0.1	0.3	0.2
S	0.5	5	2

Table 3: Metric values for System-level Metrics

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

- There are only 12.5% of the total classes that have high coupling metric values. There are 4.1% of classes with deep hierarchy. Since earlier empirical studies [Braind00] suggests that classes with more coupling and deep hierarchy are fault prone, the identified classes (16.6 %) must be thoroughly checked during testing.
- The maximum value of RFC is high for all the projects as it also counts the method invocations.
- The values for NOM and WMC are same as method complexities are generally considered to be unity.
- NOM is a subset of RFC and is easy to measure.

- RFC measures the complexity of the software by counting the number of methods in the class and also captures the information about the coupling of the class to other classes. Number of methods are measured by NOM and there are number of metrics that measure the coupling information. So NOA and NOM can be used together by project managers to provide count of number of attributes and methods in a class. These two metrics are sufficient to be used by project managers.
- CBO value is generally less in sample data, hence classes are easy to understand, reuse and maintain.
- In all the three projects DAC has less values indicating that the developers has less tendency to use data abstractions. DAC metric is of little use to developers as it only gives count of ADTs in a class, without considering number of messages passed between the classes.
- MPC is a dynamic measure. It provides more information than rest of the class coupling measures.
- TCC and LCC have exactly same values because there is no indirect connection between methods.
- LCOM values are zero because the number of pairs of methods having access to common attributes is more than the number of pairs of methods having no common attributes. It implies that the classes are cohesive.
- LCOM and TCC are providing same information. Only interpretation of values obtained is different. So project managers can use anyone of these two metrics.
- The DIT and NOC values are medium in all the projects; this shows that inheritance is used in most of the classes to optimum level.
- The value of AIF is high suggesting high use of attribute inheritance.
- The MIF value is nil for Project 1 and Project 2. It is observed that there are very less methods in superclasses; they contain only abstract methods, which are overridden in subclasses.
- MIF and AIF measures can provide overall system view about amount of information hiding incorporated by software designers.
- AHF values are high in Project1 and Project 2 as compared to project 3, which shows that attributes are declared as private in most of the classes, so information is kept hidden.
- MHF has nil values indicating that methods are declared public by developers.
- AHF metric can also be alternatively calculated as number of attributes declared private in the system divided by total number of attributes in the system.
- Similarly, MHF metric can be easily calculated as number of methods declared private in the system divided by total number of methods in the system.
- PF value is nil for Project 3, high for Project 2 and moderate for Project 1. This shows that more overloading is used in Project 3 as compared to other two projects. The relationship between various class metrics is shown in Figure 6.

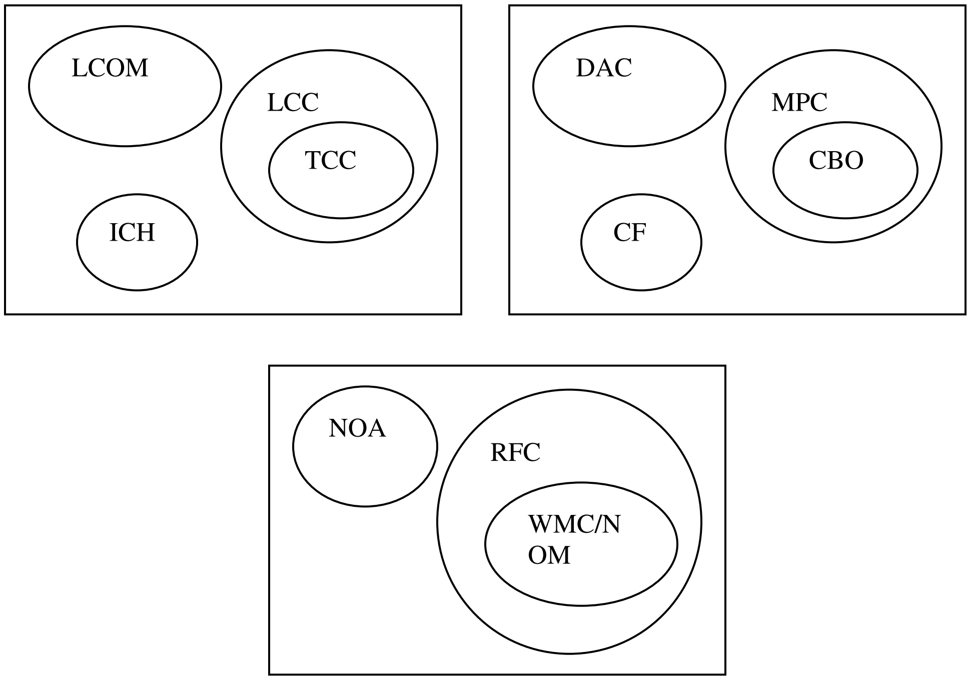


Figure 6: Relationship between Class Metrics

Principal Component (PC) Analysis

In this section the results of PC analysis are presented. The PC analysis extraction method and varimax rotation method is applied on 14 class level metrics. The data is not sufficient to apply on system level metrics. The rotated component matrix is given in Table 4. The values above 0.7 (shown in bold in Table 4) are the measures that are used to interpret the PCs. For each PC, we also provide its eigenvalue, variance percent and cumulative percent. The interpretations of PCs are given as follows:

- PC1: ICH, NOA, RFC, NOM and WMC. We have all size measures and one cohesion measure in this dimension. This shows there are classes with high attributes, internal methods (methods defined in the class) and external methods (methods called be the class). This means cohesion is related to number of methods and attributes in the class.
- PC2: LCC and TCC. These are cohesion measures.
- PC3: DAC and LCOM. The LCOM cohesion measure counts pairs of methods of classes within a class that use attributes in common. This PC cannot be easily interpreted as contain one coupling and one cohesion measure.
- PC4: DIT and NOC measure depth of inheritance and number of children.
- PC5: CBO measures coupling between objects

We can draw following observations:

- One dimension is entirely determined by cohesion measures (PC2).
- One dimension is entirely determined by inheritance measures (PC4).
- One dimension is entirely determined by coupling measures (PC5).

- CBO and MPC cannot be assigned to any of the PCs at all.
- There is one dimension that may be interpreted as size PC1.
- Numbers of dimensions captured are quite less than the total number of metrics, implying that many metrics are highly related.

Commulative %	33.79	49.44	62.8	72.97	80.34
% Variance	33.7	15.6	13.4	10.08	7.37
eigenvalues	4.731	2.1916	1.881	1.412	1.032
Component	PC1	PC2	PC3	PC4	PC5
CBO	0.051175	-0.06101	0.047826	0.164243	0.864058
DAC	0.003159	0.225027	0.864869	0.105878	0.180981
DIT	-0.25296	0.080662	-0.18067	-0.7641	-0.30287
ICH	0.833226	0.006964	-0.07167	0.106661	0.064353
LCC	-0.08988	0.981588	0.029169	-0.01219	-0.0163
LCOM	-0.05453	-0.13486	0.842833	0.041382	-0.12783
MPC	0.643017	0.132814	-0.15363	0.199787	0.487224
NMO	0.082303	0.177845	-0.14835	-0.66096	-0.21268
NOA	0.773894	-0.11178	0.130222	0.142889	-0.36484
NOC	0.110619	0.342395	-0.1606	0.745171	-0.21276
NOM	0.984645	-0.06097	-0.01496	-0.02356	0.051122
RFC	0.88602	-0.10624	-0.01313	0.044426	0.116636
TCC	-0.08988	0.981588	0.029169	-0.01219	-0.0163
WMC	0.984645	-0.06097	-0.01496	-0.02356	0.051122

Table 4: Results of Principal component Analysis

Correlation with size

In this section we analyze the correlation of coupling, cohesion, and inheritance measures with size measure as size measure is highly related to fault proneness [Braind00]. In Table 5, we indicate for each measure the correlation coefficient with size. The correlation coefficient is well below 0.5 for all measures indicating that none of the measures is strongly related to size. However RFC, ICH and NOM have high values as compared to other measures. NOM is a measure of number of methods in a class so the result is as expected.

Metric	Spearman's rho
CBO	0
DAC	0.122703
DIT	-0.37221
ICH	0.312448
LCC	0.067275
LCOM	-0.10095
MPC	0.266573
NMO	-0.41379
NOA	0.209879
NOC	0.208223
NOM	0.400092
RFC	0.487952
TCC	0.067275
WMC	0.400092

Table 5: Correlation of Class-level metrics with size

6 CONCLUSION AND FUTURE WORK

There are only 12.5 % of the total classes that have high coupling metric values. There are 4.1% classes with deep hierarchy. Since earlier empirical studies suggests that classes with more coupling and deep hierarchy are fault prone, the identified classes (12.5%) must be thoroughly checked during testing.

Theoretical analysis of these metrics suggest that out of 14 class level metrics, 6 metrics (NOA, NOM, MPC, DAC, LCOM and LCC) provide sufficient information for usage and other metrics are either subset of these metrics or are providing same information in different format. The number of dimensions captured in PC analysis is only 5 which are much lower than the number of measures. This simply supports the conclusions drawn from theoretical analysis i.e. many of the metrics proposed are based on comparable ideas and therefore provide somewhat redundant information. The correlation analysis between different metrics and size shows that hardly any of the metric is strongly correlated to size.

The measures could not be evaluated over a large data set but this is a problem that has plagued much of empirical software engineering research. More similar type of studies must be carried out with different data sets to give generalized results across different organizations. We plan to replicate our study on large data set and industrial object-oriented software system.

REFERENCES

[Briand98] L.Briand , W.Daly and J. Wust, Unified Framework for Cohesion Measurement in Object-Oriented Systems. *Empirical Software Engineering*, 3 65-117, 1998.

[Briand99] 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.

[Briand00] 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.

[Chidamber94] 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.

[Fenton96] N.Fenton et al, Software Metrics: *A Rigorous and practical approach*. International Thomson Computer Press, 1996.

[Harrison98] 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.

[Henderson96] B.Henderson-sellers, *Object-Oriented Metrics, Measures of Complexity.* Prentice Hall, 1996.

[Lee95] Y.Lee, B.Liang, S.Wu and F.Wang, *Measuring the Coupling and Cohesion of an Object-Oriented program based on Information flow*, 1995.

[Lorenz94] M.Lorenz, and J.Kidd, *Object-Oriented Software Metrics*. Prentice-Hall, 1994.

[Schach04] S.R.Schach, *Introduction to Object-Oriented Analysis and Design*. Tata McGraw-Hill,. WWW Site: http://highered.mcgraw-hill.com/sites/0072826460/student_view0/case_studies.html.

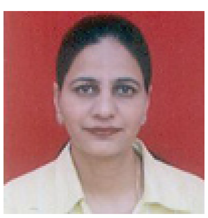
[Schach02] S.R Schach, *Object-Oriented and Classical Software Engineering*. Tata McGraw-Hill, 2002. WWW Site: http://www.mhhe.com/engcs/compsci/schach5/student/airgourmet.java.java.

[Venugopal97] K.R. Venugopal, Rajkumar, T.Ravishankar, *Mastering C++*, Tata McGraw Hill,1997.

About Authors

K. K. Aggarwal is vice chancellor at the Guru Gobind Singh Indraprastha University, India. He received his doctorate from Kurushetra University. He was president of the Institution of Electronics and Telecommunication Engineers (IETE) from 2002 through 2004. Recently he was awarded "Delhi Ratan Bhuddhijeevi Samman" by the All India Conference of Intellectuals (AICI). Prof. Aggarwal has written few books and many of his articles have appeared in several books published by IEEE of USA. He is coauthor of a book on software engineering and has published more than 300 publications in national and international journals and conferences. He can be reached by e-mail at kka@ipu.edu.

Yogesh Singh is a professor with the University School of Information Technology and the School of Engineering and Technology, Guru Gobind Singh Indraprastha University, Kashmere Gate, India. He received his master's degree and doctorate from the National Institute of Technology, Kurukshetra. His area of research is Software Engineering focusing on Planning, Testing, Metrics and Neural Networks. He is coauthor of a book on software engineering, and is a Fellow of IETE and member of IEEE. He has more than 150 publications in international and national journals and conferences. Singh can be contacted by e-mail at ys66@rediffmail.com.



Arvinder Kaur is a Reader with the University School of Information Technology. She obtained her doctorate from Guru Gobind Singh Indraprastha University and her master's degree in computer science from Thapar Institute of Engg. and Tech. Her research interests include software engineering, object-oriented software engineering, software metrics, microprocessors, and operating systems. She is also a lifetime member of ISTE and CSI. Kaur has published more than 30 research papers in national and international journals and conferences. Her paper titled "Analysis of object oriented Metrics" was published as a chapter in the book *Innovations in Software Measurement* (Shaker -Verlag, Aachen 2005). She can be reached by e-mail at arvinderkaurtakkar@yahoo.com.

Ruchika Malhotra is a research scholar with the University School of Information Technology, Guru Gobind Singh Indraprastha University, India. She is working as a visiting faculty with Amity Institute of Information Technology, India. She received her master's degree in software engineering from the University School of Information Technology, Guru Gobind Singh Indraprastha University, India. Her research interests are in improving software quality, statistical and adaptive prediction models for software metrics, neural nets modeling, and the definition and validation of software metrics. She has more than 10 publications in international journals and conferences. Her paper titled “Analysis of object oriented Metrics” was published as a chapter in the book *Innovations in Software Measurement* (Shaker -Verlag, Aachen 2005). She can be contacted by e-mail at ruchikamalhotra2004@yahoo.com.