Object Oriented Metrics

Object Oriented Metrics

- Primary objectives for object-oriented metrics are no different than those for metrics derived for conventional software:
  - To better understand the quality of the product
  - To assess the effectiveness of the process
  - To improve the quality of work performed at a project level

Characteristics of object-oriented metrics

- Metrics for OO systems must be tuned to the characteristics that distinguish OO from conventional software.
- So there are five characteristics that lead to specialized metrics:
  - Localization
  - Encapsulation
  - Information hiding,
  - Inheritance, and
  - Object abstraction techniques.

Localization

- *Localization* is a characteristic of software that indicates the manner in which information is concentrated within a program.
- For example, in conventional methods for *functional decomposition* localize information around *functions* & *Data-driven* methods localize information around specific *data structures.*
- But In the OO context, information is concentrated by summarize both *data and process* within the bounds of a *class or object.*
- Since the class is the basic unit of an OO system, localization is based on objects.
- Therefore, metrics should apply to the class (object) as a complete entity.

- Relationship between operations (functions) and classes is not necessarily one to one.
- Therefore, classes collaborate must be capable of accommodating one-to-many and many-to-one relationships.

Encapsulation

- Defines encapsulation as “the packaging (or binding together) of a collection of items
- For conventional software,
  - Low-level examples of encapsulation include records and arrays,
  - mid-level mechanisms for encapsulation include functions, subroutines, and paragraphs
- For OO systems,
  - Encapsulation include the responsibilities of a class, including its *attributes and operations*, and the states of the class, as defined by specific attribute values.
- Encapsulation influences metrics by changing the focus of measurement from a *single module* to a *package of data (attributes)* and *processing modules (operations).*

Information Hiding

- Information hiding suppresses (or hides) the operational details of a program component.
- Only the information necessary to access the component is provided to those other components that wish to access it.
- A well-designed OO system should encourage information hiding. And its indication of the quality of the OO design.

Inheritance

- Inheritance is a mechanism that enables the responsibilities of one object to be propagated to other objects.
- Inheritance occurs throughout all levels of a class hierarchy. In general, conventional software does not support this characteristic.
- Because inheritance is a crucial characteristic in many OO systems, many OO metrics focus on it.

Abstraction

- Abstraction focus on the essential details of a program component (either data or process) with little concern for lower-level details.
- Abstraction is a relative concept. As we move to higher levels of abstraction we ignore more and more details.
- Because a class is an abstraction that can be viewed at many *different levels of detail* and in a *number of different ways* (e.g., as a list of operations, as a sequence of states, as a series of collaborations), OO metrics represent abstractions in terms of *measures of a class*

Class-oriented metrics

- To measure class OO metrics:
  - Chidamber and Kemerer (CK) metrics suites
  - Lorenz and Kidd(LK) metrics suites
  - The Metrics for Object-Oriented Design (MOOD) Metrics Suite

CK metrics suite

- CK has proposed six class-based design metrics for OO systems.

  1. Weighted methods per class (WMC):-

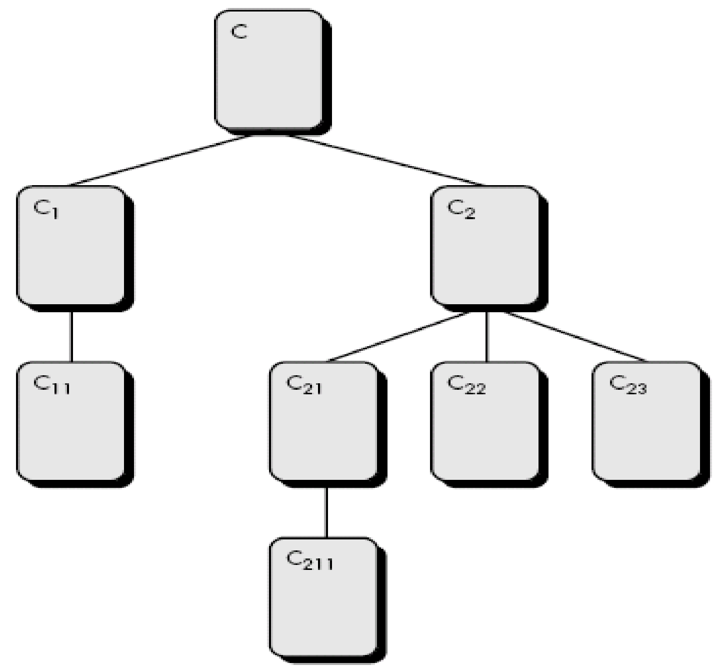
  - Assume that *n* methods of complexity $c_1, c_2, \ldots, c_n$ are defined for a class **C**.
  - The specific complexity metric that is chosen (e.g., cyclomatic complexity) should be normalized so that nominal complexity for a method takes on a value of 1.0.

$$WMC = \sum c_i$$

  - So if no. of methods are increase, complexity of class also increase.
  - Objects with large number of methods are likely to be more application specific, limiting the possible reuse

2. Depth of the inheritance tree (DIT):-

- This metric is “the maximum length from the node to the root of the tree (base class)”
- Referring to Figure, the value of DIT for the class-hierarchy shown is 4.
- Lower level subclasses inherit a number of methods making behavior harder to predict
- Deeper trees indicate greater design complexity
- On the positive side, large DIT values imply that many methods may be reused.



3. Number of children (NOC):-

- The subclasses that are immediately subordinate to a class in the class hierarchy
- Referring to previous figure, class **C2** has three children—subclasses **C21**, **C22,** and **C23**.
- As the NOC increases, reuse increases, but the abstraction represented by the parent class can be diluted.
- Depth is generally better than breadth in class hierarchy, since it promotes reuse of methods through inheritance
- Classes higher up in the hierarchy should have more sub-classes then those lower down
- As NOC increases, the amount of testing (required to exercise each child in its operational context) will also increase.

4. Coupling between object classes (CBO):

- CBO is the number of collaborations between two classes (fan-out of a class C)
  - The number of other classes that are referenced in the class C (a reference to another class, A, is a reference to a method or a data member of class A)
- As collaboration increases reuse decreases
- High fan-outs represent class coupling to other classes/objects and thus are undesirable
- High fan-ins represent good object designs and high level of reuse
- Not possible to maintain high fan-in and low fan outs across the entire system
- As CBO increases, it is likely that the reusability of a class will decrease.
- If values of CBO is high, then modification get complicated.
- Therefore, CBO values for each class should be kept as low as is reasonable.

1. ‘FAN IN’ is simply a count of the number of other Components that can call, or pass control, to Component A.
2. ‘FANOUT’ is the number of Components that are called by Component A.

5. Response for a class (RFC)

- RFC is the "Number of Distinct Methods and Constructors invoked by a Class" (local + remote)
- As RFC increases
- testing effort increases
- greater the complexity of the object
- harder it is to understand

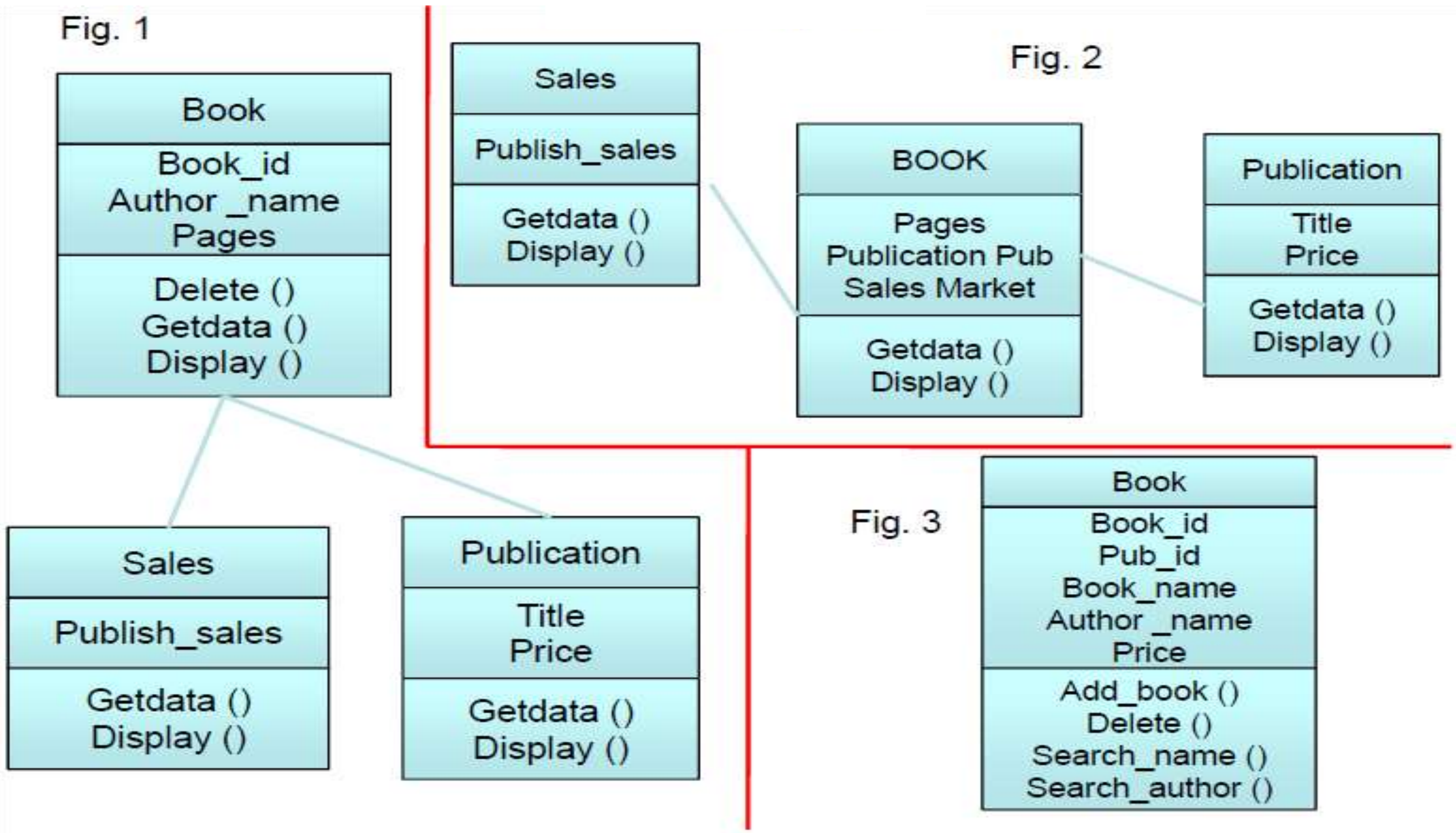
6. Lack of cohesion in methods (LCOM).

- This is a notion of degree of similarity of Methods
  - LCOM is the number of methods that access one or more of the same attributes.
    - If no methods access the same attributes, then LCOM = 0.
- If LCOM is high, methods may be coupled to one another via attributes. This increases the complexity of the class design.
- In general, high values for LCOM imply that the class might be better designed by breaking it into two or more separate classes.
- It is desirable to keep cohesion high; that is, keep LCOM low.

- Take class *C with M1, M2, M3*
- $I1 = \{a, b, c, d, e\}$
- $I2 = \{a, b, e\}$
- $I3 = \{x, y, z\}$
- $P = \{(I1, I3), (I2, I3)\}$
- $Q = \{(I1, I2)\}$
- Thus LCOM = 1
- There are *n such sets I1 ,…, In*
  - $P = \{(Ii, Ij) \mid (Ii \cap Ij) = \emptyset\}$
  - $Q = \{(Ii, Ij) \mid (Ii \cap Ij) \neq \emptyset\}$
- If all *n sets Ii are* $\emptyset$ *then* $P = \emptyset$
- LCOM = |*P*| - |Q|, *if |P| > |Q|*
- LCOM = 0 otherwise

Example

- Compute WMC, RFC, CBO, LCOM. Consider complexity to be 1

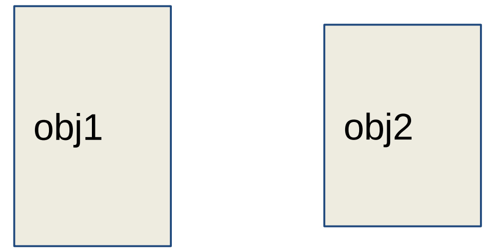


Solution

1. WMC for book is 3, sale is 2 and publication is 2
   - Weighted Number Methods in a Class (WMC)
     - Methods implemented within a class or the sum of the complexities of all methods
2. RFC = 3+2+2 = 7
   - Response for a Class (RFC)
     - Number of methods (internal and external) in a class.
3. CBO = 2 (class book) and 0 (class publication and sales)
   - Coupling between Objects (CBO)
     - Number of other classes to which it is coupled.

```
class book
{
 int a, b;
book (int a, int b)
{this.a=a;
this.b=b;
}
book(book ref)
{
a=ref.a;
b=ref.b;
}
```

```
class A
{
psvm()
    {
    int a=10; int b=20;
    book obj1=new book(a,b);
    book obj2= new book(obj1);
    }
    }
```



4. LCOM: Lack of cohesion in methods

- I_1 {add_book ( )} = { book_id, Pub_id, Book_name, Author_name, Price}
- I_2 {delete ( )} = { book_id}
- I_3 {search_name ( )} = {Book_name}
- I_4 {search_author( )} = {Author_name}

$I_1 \cap I_2$, $I_1 \cap I_3$, $I_1 \cap I_4$ are non null sets

$I_2 \cap I_3$, $I_2 \cap I_4$ and $I_3 \cap I_4$ are null sets

Thus LCOM = 0, if no of null interactions are not greater than number of non null interactions.

Hence, LCOM = 0 [$|P| = |Q| = 3$]

The MOOD Metrics Suite

1. Method inheritance factor (MIF).

- The degree to which the class architecture of an OO system makes use of inheritance for both methods (operations) and attributes is defined
- Value of MIF indicates impact of inheritance on the OO Software

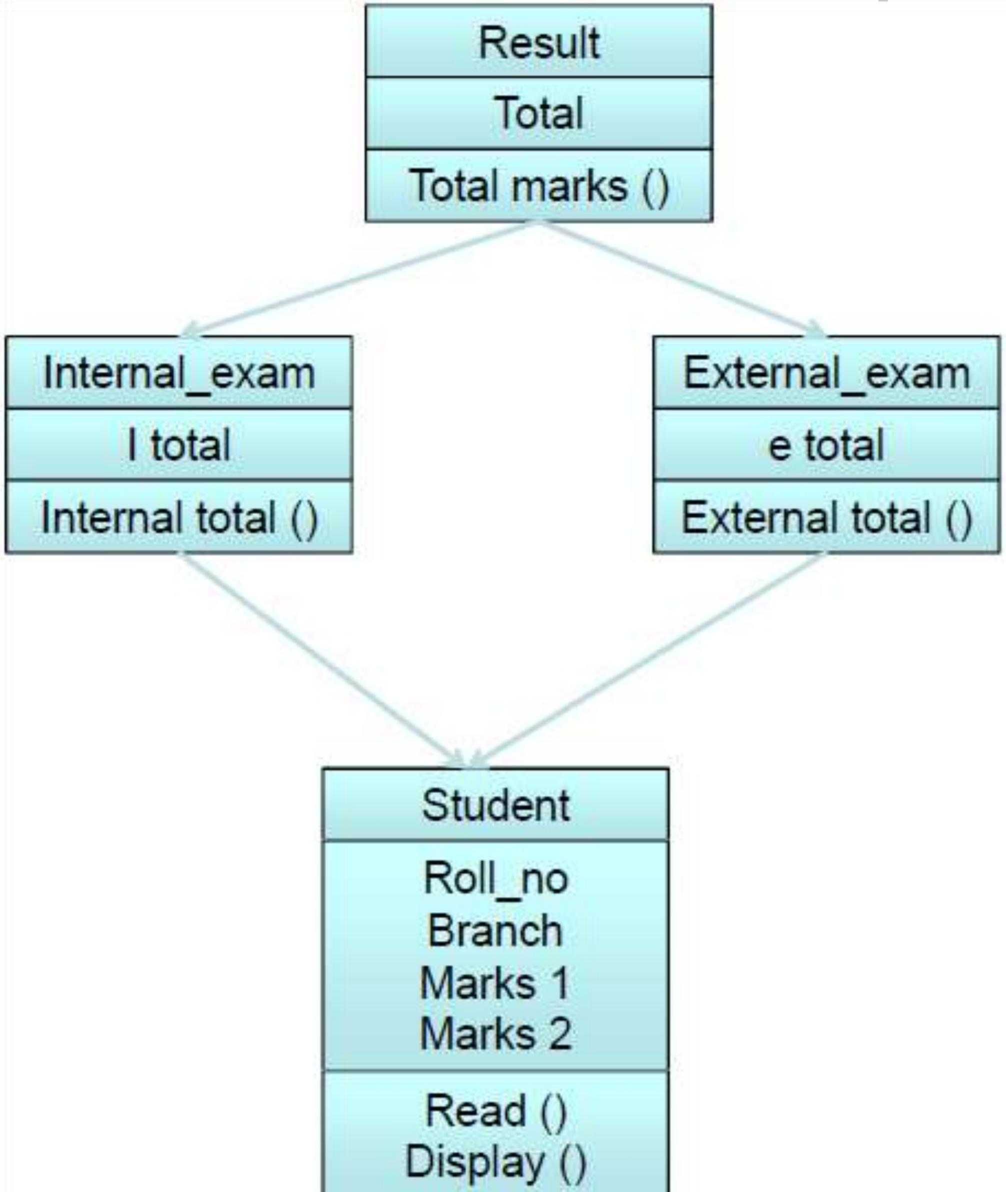
$$MIF = \frac{\sum_{i=1}^{n} M_i(C_i)}{\sum_{i=1}^{n} M_a(C_i)}$$

- $M_i(C_i)$ is the number of methods inherited and not overridden in C_i
- $M_a(C_i)$ is the number of methods that can be invoked with C_i
- $M_d(C_i)$ is the number of methods declared in C_i
- n is the total number of classes

- $M_a(C_i) = M_d(C_i) + M_i(C_i)$
- All that can be invoked = new or overloaded + things inherited

- MIF is [0,1]
- MIF near 1 means little specialization
- MIF near 0 means large change

Example



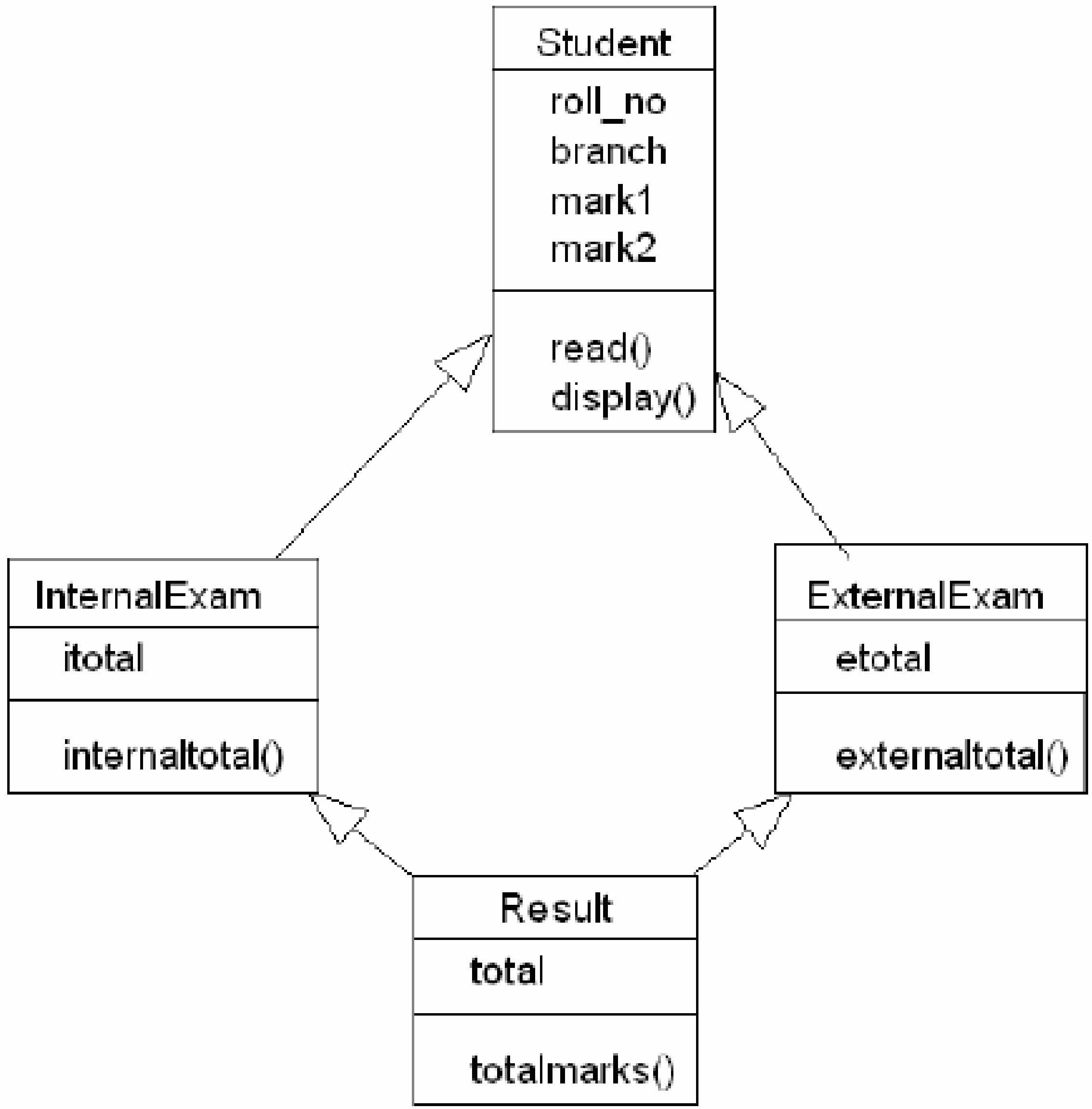
Private attributes:
Roll_no
Branch

Protected Attributes:
Marks 1
Marks 2

n = 4

Let,

C_1 =Student class,
C_2 =Internal exam class
C_3= External exam class
C_4 = Result class


Student
roll_no
branch
mark1
mark2
read()
display()
InternalExam
itotal
internaltotal()
ExternalExam
etotal
externaltotal()
Result
total
totalmarks()

Solution

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

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

Where, $M_i(C_1)$ = number of inherited methods in class student =0

$M_i(C_2)$ = number of inherited methods in class internal exam=2

MIF = 0+2+2+2 / 11 = 6/11

2. 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.

- $CF = [\sum_i \sum_j \mathit{is\_client}(C_i, C_j)]/(TC^2 - TC)$

- *is_client* = 1, *if and only if* a relationship exists between the client class, *Cc,* and the server class, *Cs,* and $Cc \neq Cs$
  = 0, otherwise

- $(TC^2 - TC)$ is the total number of relationships possible, where TC= Total number of classes in the system under consideration.
- CF is [0,1] with 1 meaning high coupling
- As the value for CF increases,
  - the complexity of the OO software will also increase and
  - understandability, maintainability, and the potential for reuse may suffer as a result.

```
class A
{
B obj;
}

Class B{}
```

3. Polymorphism factor (PF).

- PF as “the number of methods that redefine inherited methods, divided by the maximum number of possible distinct polymorphic situations

$$PF = \frac{\sum_i M_o(C_i)}{\sum_i [M_n(C_i) * DC(C_i)]}.$$

- $M_n()$ is the number of new methods
- $M_o()$ is the number of overriding methods
- DC() number of descendent classes of a base class

4. Attribute Hiding Factor (AHF)

- attribute hiding factor measure how variables and methods are encapsulated in a class.
- An attribute is called visible if it can be accessed by another class or object. Attributes should be "hidden" within a class. They can be kept from being accessed by other objects by being declared a private.
- AIF = (sum of the invisibilities of all attributes defined in all classes) / (total number of attributes defined in the project)
- Ideally, all attributes should be hidden, and thus AHF=100% is the ideal value. Very low values of AHF should trigger attention.

5. Method Hiding Factor (MHF)

- The Method Hiding Factor measures the invisibilities of methods in classes.
- An attribute is called visible if it can be accessed by another class or object. Attributes should be "hidden" within a class. They can be kept from being accessed by other objects by being declared a private.
- AIF = (the sum of the invisibilities of all methods defined in all classes.) / (total number of methods defined in the project)
- Ideally, The Method Hiding Factor should have a large value.

6. Attribute inheritance factor (AIF)

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

Where:

A_i: inherited Attributes

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

A_d: defined Attributes

TC: Total number of Classes.