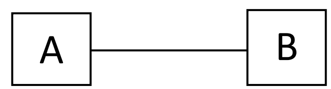
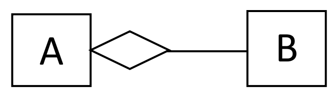
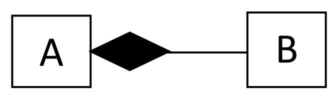
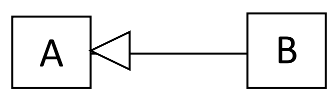
Object Oriented Software Metric

Requirements

- Understanding of class relationship
- Understanding of class diagram
  - Association


    - Aggregation


    - Composition


  - Generalization


    - Inheritance
    - Sub class – super class
    - Parent – Child

Weighted Methods per Class (WMC)

- The effort in developing a class will in some sense will be determined by the number of methods the class has and the complexity of the methods.
- Suppose a class *C* has methods *M1, M2*... *Mn* defined on it. Let the complexity of the method *M1* be *c1, then*

  WMC = $\sum_{i=1}^{i=n} c_i$

- If the ***complexity*** of each method is considered 1, WMC gives the total number *of* methods in the class.
- A separate matrix will be required to calculate complexity, like, cyclomatic complexity.
- The data based on evaluation of some existing programs, shows that in most cases, the classes tend to have only a small number of methods, implying that most classes are simple and provide some specific abstraction and operations.
- WMC metric has a **reasonable correlation** with **fault-proneness** of a class. As can be expected, the larger the WMC of a class the better the chances that the class is fault-prone.

Depth of Inheritance Tree (DIT)

- Inheritance is one of the unique features of the object- oriented paradigm.
- Inheritance is one of the main mechanisms for **reuse**
- The deeper a particular class is in a class hierarchy, the more methods it has available for reuse, thereby providing a **larger reuse potential**
- Inheritance increases coupling, which makes **changing a class harder**
- A class deep in the hierarchy has a lot of methods it can inherit, which makes it **difficult to predict its behavior**
- The DIT of a class *C* in an inheritance hierarchy is the depth from the root class in the inheritance tree
- In case of multiple inheritance, the DIT metric is the maximum length from a root to C
- Statistical data suggests that most classes in applications tend to be close to the root, with the maximum DIT metric value being around 10
- Most the classes have a DIT of 0 (that is, they are the root).
- Designers might be giving upon reusability in favor of comprehensibility
- The experiments show that DIT is **very significant** in predicting **defect-proneness** of a class: the higher the DIT the higher is the probability that the class is defect-prone

Number of Children (NOC)

- The number of children (NOC) metric value of a class *C* is the number of immediate subclasses of C
- This metric can be used to evaluate the degree of reuse, as a higher NOC number reflects reuse of the definitions in the superclass by a larger number of subclasses
- It also gives an idea of the **direct influence** of a class on other elements of a design
- The larger the influence of a class, the more important the *class* is **correctly designed**
- In the empirical observations, it was found that classes generally had a small NOC metric value, with a vast majority of classes having no children
- This suggests that in the systems analyzed, inheritance was not used very heavily
- The data *suggest* that the **larger** the NOC, the **lower** the probability of **detecting defects** in a class
- The higher NOC classes are **less defect-prone**. The reasons for this are not very clear or definite.

Coupling between Classes (CBC)

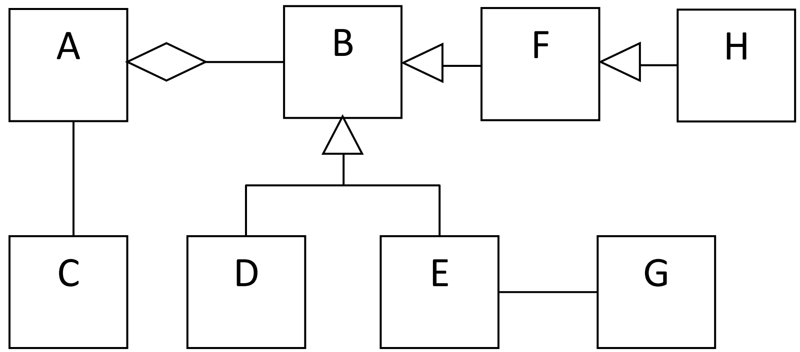
- Coupling between classes of a system reduces modularity and make class modification harder
- It is desirable to **reduce** the coupling between classes
- The less coupling of a class with other classes, the more **independent** the class, and more **easily modifiable**
- Coupling between classes (CBC) is a metric that tries to quantify coupling that exists between classes
- Two classes are considered coupled if methods of one class use methods or instance variables defined in the other *class*
- There are indirect forms of coupling (through pointers, etc.) that are hard to identify
- The experimental data indicates that most of the classes are self-contained and have low CBC value.
- Some types of classes, for example the ones that deal with managing **interfaces**, generally tend to have **higher CBC values. [Exception]**
- The data found that CBC is **significant** in predicting the **fault-proneness** of classes.

Lack of Cohesion in Methods (LCOM)

- Cohesion captures how **closely bound** the different methods of the class are
- Two methods of a class *C* can be considered "cohesive" if the set of instance variables of *C* that they access have some elements in common
- High cohesion is a **highly desirable** property for modularity
- Let I_i and I_j be the set of instance variables accessed by the methods M_1 and M_2, Q be the set of all cohesive pairs of methods, that is, all (M_i, M_j) such that I_i and I_j have a non-null intersection. Let *P* be the set of all noncohesive pairs of methods, that is, pairs such that the intersection of sets of instance variables they access is null. Then LCOM is defined as

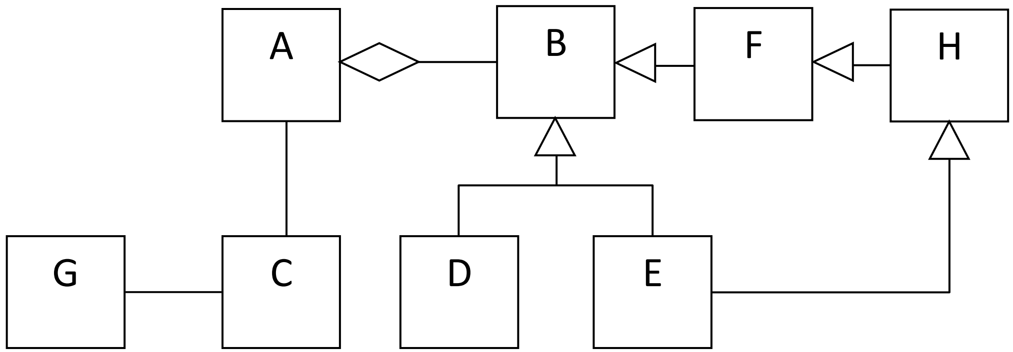
  LCOM = |P|-|Q|, if |P| > |Q|, otherwise 0.
- If there are *n* methods in a class *C,* then there are $n(n — 1)$ pairs, and LCOM is the number of pairs that are noncohesive minus the number of pairs that are cohesive
- The **larger** the number of cohesive methods, the more cohesive the class will be, and the LCOM metric will be **lower**
- A high LCOM value may indicate that the methods are trying to do different things and operate on different data entities
- If this is **validated**, the class can be **partitioned** into different classes
- The data in [BBM95] found **little significance** of this metric in predicting the **fault-proneness** of a class

Example 1 (NOC, DIT, CBC)



CLASS	NOC	Influence on Design	DIT	Reuse Potential	CBC (appx.)
A	0	Low	0	Low	Low (2)
B	3	Highest	0	Low	Highest (4)
C	0	Low	0	Low	Lowest (1)
D	0	Low	1	Moderate	Lowest (1)
E	0	Low	1	Moderate	Low (2)
F	1	Moderate	1	Moderate	Low (2)
G	0	Low	0	Low	Lowest (1)
H	0	Low	2	Highest	Lowest (1)

Example 2 (NOC, DIT, CBC)



CLASS	NOC	Influence on Design	DIT	Reuse Potential	CBC (appx.)
A	0	Low	0	Low	Low (2)
B	3	Highest	0	Low	Highest (4)
C	0	Low	0	Low	Low (2)
D	0	Low	1	Moderate	Lowest (1)
E	0	Low	3	Highest	Low (2)
F	1	Moderate	1	Moderate	Low (2)
G	0	Low	0	Low	Lowest (1)
H	1	Moderate	2	High	Low (2)

Example (LCOM)

CLASS A
a1 a2 a3 a4
O1 (a1, a2) O2 (a1) O3 (a4) O4 (a1, a4)

LCOM = |P|-|Q|, if |P| > |Q|, otherwise 0

Pairs:
(O1, O2), (O1, O3), (O1, O4), (O2, O3), (O2, O4), (O3, O4)

P = 2 (Non-Cohesive pairs)
Q = 4 (Cohesive pairs)

Q > P

LCOM = 0

Comment: The LCOM value of the class indicates that the methods of the class are cohesive, and it is a desirable design.