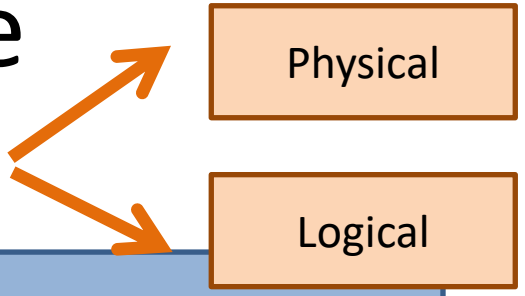
# Object Oriented Metrics

Impact on Software Quality

# Traditional metrics

- **Lines Of Code**
- **Function points**
- **Complexity**
- **Code coverage -** testing
- **Maintainability Index –** discussed later

# Lines of Code




- KLOC = 1000 Lines Of Code

*Def: A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line. This specifically includes all lines containing program headers, declarations, and executable and non-executable statements.*

*[Conte ao-Software Engineering Metrics and Models]*

- Function points = measure of develop. Resources, express amount of functionalities

Number of external inputs x 4
Number of external outputs x 5
Number of logical internal files x 10
Number of external interface files x 7
Number of external inquiries x 4

# Cyclomatic complexity

- McCabe, 1976

**CC = E - N + 2*P**

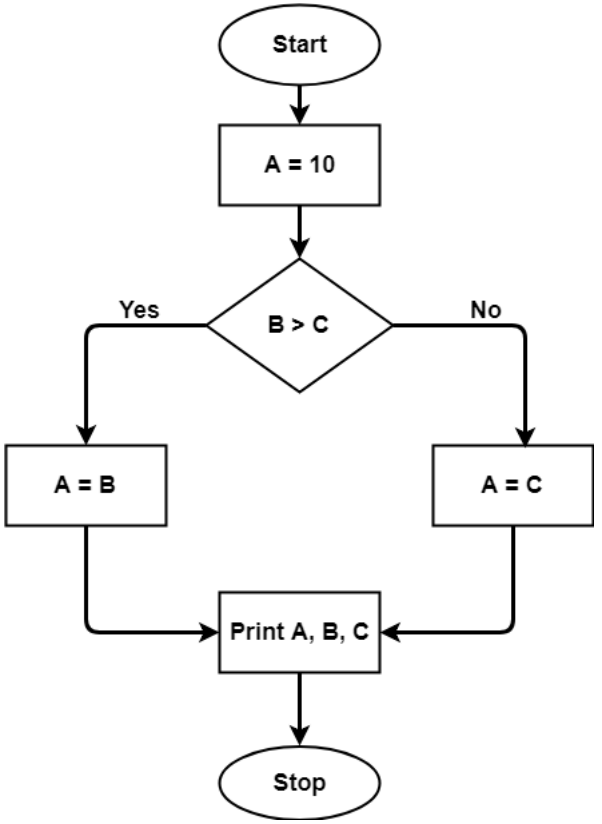
Where:

- E = number of edges in the flow graph
- N = number of nodes in the flow graph
- P = number of nodes that have exit points

- Control flow graph: graph with:
  - nodes = basic blocks
  - edges corresponding to all paths that may be traversed in a program during its execution

# Cyclomatic complexity (explained)

```
A = 10
If B > C then
                A = B
        else A = C
EndIf
Print A
Print B
Print C
```




7 nodes
7 lines
P = 1
Cc = 7-7+2 = 2

https://www.geeksforgeeks.org/cyclomatic-complexity/

# Halstead complexity/ volume

**Halstead, 1977**

**V = N x log2(n)**

*N = no of operators*

*n = no of distinct operators*

# Object oriented (OO) metrics

- Metrics - Chidamber & Kemerer – 1993
  - Available at:

  http://maisqual.squoring.com/wiki/images/5/5c/Chid_kem_metrics.pdf

- MOOD – Abreu – 1995
- [Abreu, F. B. e., "The MOOD Metrics Set," presented at ECOOP '95 Workshop on Metrics, 1995.]

# Classification

- [Abreu]
- 5 aspects:
  - System
  - Coupling
  - Inheritance
  - Class
  - method

# Classification

- [Marinescu - Object-Oriented Software Metrics]
- Available at: http://www2.informatik.hu-berlin.de/swt/intkoop/jcse/workshops/risan2007/0206-Marinescu-jcse-metrics.pdf
- 4 aspects:
  - Size & Structural Complexity
  - Inheritance
  - Coupling
  - Cohesion

# Useful for:

Evaluate, predict, improve software quality

Detecting design flaws

# *Weighted Methods Per Class (WMC)[CK]*

- ***Definition**: Consider a Class C1, with methods M1,... Mn that are defined in the class. Let $c_1$,... $c_n$ be the complexity of the methods.* Then :

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

# *Weighted Methods Per Class (WMC)*

- *Indicator for how much time and effort is required to develop and maintain the class.*
- Higher values $\Rightarrow$ higher impact on descendants
- Higher values $\Rightarrow$ more application specific, low reusability & maintainability
- low values $\Rightarrow$ greater polymorphism

# *Weighted Methods Per Class (WMC)*

- **Criterion**: $\in$ [1, 50]  or max 10% of classes may have WMC over 25
- **Remark**: counts
  - Constructors and event handlers
  - Property accessor (Get, Set, Let)
- **Advice**:
  - Refactor classes with high WMC
- **Impact**: maintainability, reusability

# *Depth of Inheritance Tree (DIT)[CK]*

***Definition:*** maximum length from the node to the root of the tree.

- measure of how many ancestor classes can potentially affect this class.

# *Depth of Inheritance Tree (DIT)*

- Higher DIT $\Rightarrow$ higher complexity + difficult to predict behavior + difficult to maintain
  - But higher reusability
- **Values**:
  - $\leq 10$
  - Rec. $\leq 5$ (RefactorIT & Visual Studio)
  - In java min = 1
- **Advice**:
  - If $\leq 2$ then poor OO design
  - Not just per class, entire distribution

# *Number of Children (NOC)[CK]*

- ***Definition**: Number of immediate sub-classes subordinated to a class in the class hierarchy.*

- It is a measure of how many sub-classes are going to inherit the methods of the parent class.

# *Number of Children (NOC)*

- high NOC $\Rightarrow$
  - High reuse of base class
  - Base class may require more testing
  - Improper abstraction of the parent class
  - Misuse of sub-classing
  - + high WMC $\Rightarrow$ complexity at the top of the class hierarchy - poor design
- Upper part of class hierarchy – higher NOC than lower part

# *Number of Children (NOC)*

- **Advice**:
  - $\in$ [0,10] (RefactorIT)
  - $\geq$ 10 => restructure class hierarchy

# COUPLING

**Interdependencies between modules**

# Coupling between Objects (CBO)[CK]

$CBO(c) = |\{d \in C - \{c\} \mid uses(c, d) \text{ or } uses(d, c)\}|$

$CBO'(c) = |\{d \in C - (\{c\} \cup Ancestors(c)) \mid uses(c, d) \text{ or } uses(d,c)\}|$

- Where:
  - item C: an OO software system
  - Ancestors(c): generalizations of c
  - uses(c, d) true if method (re)defined by d invoked by method of c

# Coupling between Objects (CBO)

- through method calls, field accesses, inheritance, arguments, return types, and exceptions
- High coupling ⇔ many dependencies
- Low coupling ⇔ few dependencies
- **High CBO** ⇒
  - Low modularity and reusability
    (independent classes - easier to reuse)
  - Low maintainability
  - High complexity ⇒ difficult testing

# Coupling between Objects (CBO)

- **Advice**:
- If high CBO and high NOC re-evaluate design
- Used by senior designers and project managers to track
  - whether the class hierarchy is losing its integrity
  - whether different parts of a large system are developing unnecessary interconnections in inappropriate places.

# Afferent Coupling (Ca, Fan-in) [R. Martin 94]

- determines the number of classes and interfaces from other packages that depend on classes in the analyzed package

- indicator of the level of responsibility:
  - If the package is relatively abstract then a large number of incoming dependencies is acceptable and not if the package is more concrete.
- Indicator of maintainability and testability

# Efferent Coupling (Ce, Fan-out) [R. Martin 94]

- Determines the number of other packages that the classes in the package depend upon is an indicator of the package's independence.
- High **Ce**:
  - package is unfocussed
  - unstable since it depends on the stability of all the types to which it is coupled.
- max 20 (RefactorIT) recommends an upper limit of 20.
- Advice: extract classes from the original class so the class is decomposed into smaller classes.
- Referred by R.C Martin (Uncle Bob) - *Clean architecture : a craftsman's guide to software structure and design, 2018*

## Afferent Coupling

- measure of how many other classes use the specific class

## Efferent Coupling

- measure of how many different classes are used by the specific class

```
class Foo {
     Qclass q;
}
class Bar {
Qclass q;
}
class Qclass {
 // ...
}
```

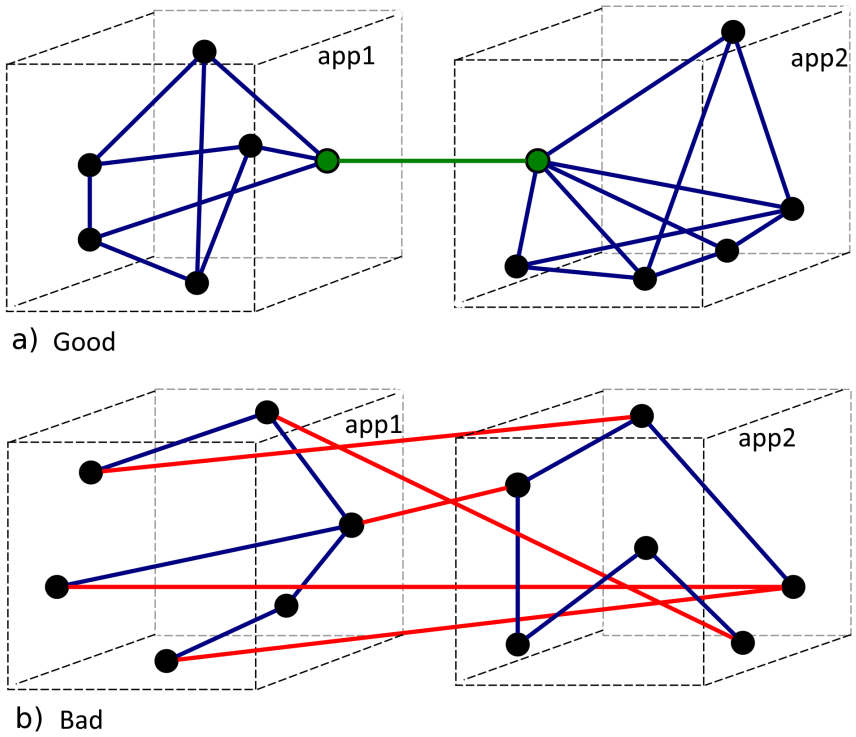
Ce(Foo) = Ce(Bar) = 1
Ca(Qclass) = 2

# COHESION

**how related the functions within a single module are**

**High cohesion:**

- Reduced complexity
- Increased maintainability
- Increased reusability




[Wikipedia]

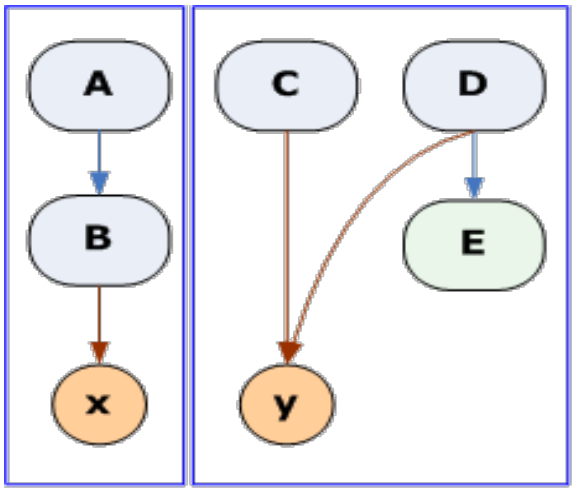
Discuss

# Lack of Cohesion of Methods (LCOM)

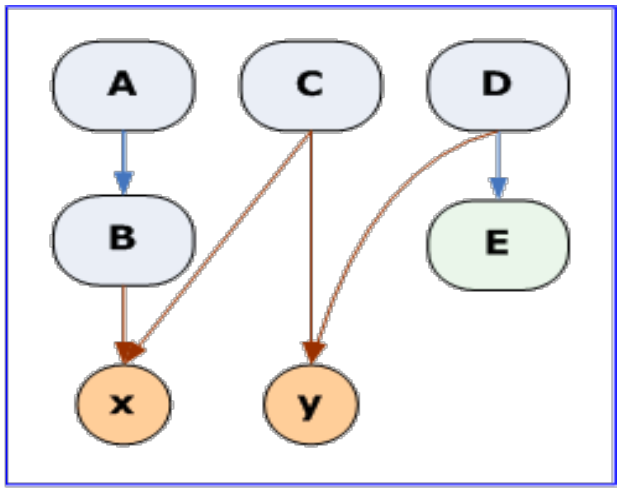
- LCOM1, LCOM2, LCOM3 and **LCOM4** (Hitz & Montazeri)
- ***Definition***: Let C be a class
  - IC is the set of instance variables of C
  - MC is the set of methods of C
  - GC is a graph with
    - vertices V = MC and
    - edges $E \subset (V \times V)$ where $(m, n) \in E \Leftrightarrow$ m and n share at least one common instance variable

**LCOM = number of connected subgraphs**

# Lack of Cohesion of Methods (LCOM4)




LCOM4 = 2




LCOM4 = 1

Methods A and B are related if:

-they both access the same class-level variable

- A calls B, or B calls A.

**LCOM4=1** - cohesive class =>"good" class

**LCOM4>=2 –** problem => split class into smaller classes

**LCOM4=0** - no methods in a class => "bad" class

# Lack of Cohesion of Methods (LCOM4)

- Cohesiveness promotes encapsulation
- Lack of cohesion - split into two or more sub-classes
- Any measure of disparateness of methods helps identify flaws in the design of classes
- Low cohesion increases complexity and maintainability

# TCC (Tight Class Cohesion) [Bieman, Kang 95]

- measures the cohesion of a class: *the relative number of method-pairs that access an attribute of the class*
- Example: TCC = 2/10, resp. 4/10
- TCC ∈[0..1]
- Higher TCC => higher cohesion

- LCC – Loose Class Cohesion

# **MOOD** and **MOOD2** metrics [F. Brito & Abreu]

1. MHF - Method Hiding Factor
2. AHF - Attribute Hiding Factor
3. MIF - Method Inheritance Factor
4. AIF - Attribute Inheritance Factor
5. PF - Polymorphism Factor
6. CF - Coupling Factor

- **MOOD2 – extension of MOOD**

# OO metrics Tools

- Metrics plugin Eclipse - Java
- IntelliJ MetricsReloaded plugin – Java
- SourceMeter
  - Java, Python, C#, C++
- Ndepend + Metrics - .NET
- PhpMetrics

- Very few: SonarQube
- Python – few metrics/tools: Radon, Understand