Foundation for Advancement of Science & Technology

→ monolithic Design: One module system — Not good prog

→ Modular Design: multiple modules system.
- Accomplished through decomposition
- good programming

one module for each function of the system
⇓
Functionaly independant (as much as possible)

depends on:
1. Cohesion (should be high)
2. Coupling (should be low)
⇓
objective: Create modules that are highly cohesive and have minimum coupling

Cohesion: Refers to the degree of single mindedness (perform same action)

Coupling: Refers to the degree of connectedness (b/w modules)

TYPES OF COHESION

↑ high level Cohesion (follow these)

Types:

1. Functional / Perfect Cohesion:
   - ↳ performs one action (Eg getters)
     - ↳ can have multiple steps but should be performing one task.
   - ↳ most ideal type.

2. Layered Cohesion: (slightly less ideal than functional cohesion)
   - ↳ when the system is organized into layers. High level layers can access services of lower level (✓) but not vice versa!

3. Communicational Cohesion: (very common)
   - ↳ aka informational Cohesion.
   - ↳ when all func that access the same data are made a part of same class

↓ lower level Cohesion (Should be avoided)

1. Procedural Cohesion
   - ↳ All operations that are in a sequence are bunched together in the same class
   - 1.1 Sequential Cohesion:
     - ↳ Data is also passed from one operation to the other.

2. Temporal Cohesion:
   - ↳ All operations that happen at similar time are bunched together in one class

3. Utility Cohesion:
   - ↳ aka coincidental Cohesion
   - ↳ when functions perform similar operations (come under the same umbrella)

Eg: a string library. All functions are different but are bunched together. ← bunched because they perform actions on strings

Foundation for Advancement of Science & Technology

TYPES OF COUPLING

"necessary evil"

① Content Coupling.

- ↳ ugliest type
- ↳ when one module directly access the data of another module
- ↳ happens when everything is Public
- ↳ "friend" classes can also lead to content coupling

② Common Coupling:

- ↳ if multiple classes/functions depend upon same global variables then they have common coupling
- ↳ "share the same data".
- ↳ static global variables also lead to common coupling

③ Routine Call Coupling:

- ↳ if functions need to call each other then it is routine call coupling
- ↳ 3.1 Data Coupling:
  - ↳ if they pass data along with calling each other
  - ↳ 3.1.1 Control Coupling:
    - ↳ when data being passed influences how the func2 will react.

given that func1 passes data to func2.

in case 2 we need to see the code of the function which is why this is different from stamp coupling

④ Type Use Coupling

- ↳ C1. we have 2 classes A and B and class A defines an attribute of type B in it
- ↳ C2. when a func in class A declares a local variable of type B.
- ↳ Classes are coupled. Class A's definition is dependant on class B.

⑤ Stamp Coupling

func prototype

- ↳ if the signature of a function has an attribute of another class.
  (func (class B attribute) { ... })
  ↳ of class A
- ↳ if the function of one class returns an attribute that is from another class.

Impact & External

↓

coupling types mentioned in the book. (read them)

SOLID Principles

S : Single responsibility
O : Open-Closed
L : Liskov substitution
I : Interface Segregation.
D : Dependancy Inversion

if you follow these principles then your code/design will become flexible and changing is made easier

1) S : each class should have a single responsibility (conceptually) / each class should have a single reason to change.
2) O : a class should be open for extension, but closed for modification. (it is risky to modify code)
3) L : Subclasses should be substitutable for baseClasses.
(ie. ~~when~~ if a base class obj. is replaced by a sub class obj. then the code should work fine)
a subclass should not change the overall code (preserves the semantics) of the superclass
4) I : If an interface serves 'many' clients, then it should be segregated into multiple interfaces, where each interface serves one client
5) D : Depend on abstraction, not on concretions. Make your code generic (Do not hard Code)

Object Oriented Metrics

Eg CGPA in academia
BMI in Health
quantitative data that helps to compare data.

- used to determine design quality
- used in Project Plan too. (to determine cost or effort etc)

Foundation for Advancement of Science & Technology

Diagrl use case (w/ modification)
2 ACD
3. DCD
:] — DSD
↳ DSD for UC-1 (UC name)

① Lorenz & kidd (LK)
② Chidamber & Kimerer (CK)

CoCoMo
Construction
Cost
Model

Lorenz & kidd: focuses primarily on size

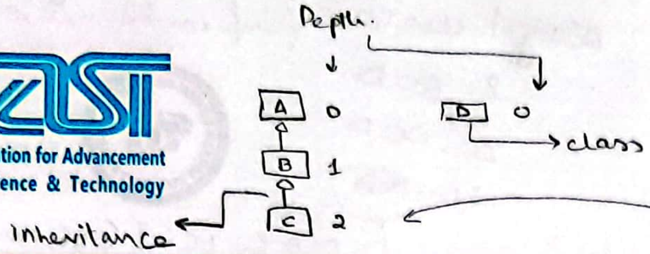
Chidamber & Kimerer: deals with both the size and quality

① LK Metrics:

i) Number of Scenario Scripts (NSS): Number of use cases
ii) Number of key Classes (NKC): ACD tells the key classes (the classes that are important for the problem domain)
iii) Number of supporting Classes (NSC): DCD tells the supporting classes that are needed for implementation
iv) Average Number of Supporting Classes per Key Class (ASC): gotten by ratio of points ii and iii $= \frac{NSC}{NKC}$
v) Number of sub systems (NSuS): Number of packages in your SW.
vi) Class Size (CSize) = sum of Number of Attributes (NA) and number of Operations (NOp); all attr. and all oper. are counted (even inherited ones)

applicable to subclasses only.
if a class does not participate in inheritance then the value of both of these will be N/A (not applicable)

vii) Number of Operations Added (NOA): The number of operations added by a subclass — inheritance.
(if a class has no parent then it's NOA will be equal to NOp)
viii) Number of Operations Overwritten (NOpO): if a class is not part of an inheritance heirarchy then NOpO will be zero. if a class is a part of inheritance, and overwrites any functions then it will have a non-zero value.



Tells how deep you are in the inheritance.

Depth

IX) Specialization Index (SI) = $\frac{NOpO \times D}{NOp}$: The higher the SI, the complex the class.

higher than class level.

NOTE: LK metrics (i) to (v) are applicable on a higher level

LK metrics (vi) to (IX) are applicable on a class level

② CK Metrics:

i) Weighted Methods per Class (WMC) : sum of complexities of all methods. $= \sum_{i=1}^{n} c_i$ (where m_i has complexity c_i)

(taking all complexities to be 1 :) $WMC = n$

(in the cases when the class does not have inheritance, we say that $WMC = NOp = n$)

ii) Number of Children (NOC) : this counts all immediate children (no grandchildren are included). (if a class is not part of inheritance (or is the leaf class) then NOC = zero)

iii) Depth of Inheritance Tree (DIT) : same as Depth of LK.

iv) Coupling Between Objects (CBO) : Total number of associations (all associations apart from self associations)

aggregation, simple, etc. ⇒ (Ternary associations are counted as 2)

v) Response for a Class (RFC) : (code is needed to determine the value)

All operations in a class (Level 0) + all operations called by those oper (Level 1)

count each operation only once!

vi) Lack of Cohesion Methods (LCoM) : (code is needed)

(details on next page)

cppforschool.com
↓
super market billing

Foundation for Advancement of Science & Technology

LCom is calculated to find out if non cohesive functions (methods) exist in a class. If func A uses a set of instances (class variables that are not static) that func B doesn't, Then there will be more lack of cohesion.

⇒ we have :

I = all instances of class = $\{ i_1, i_2, \ldots, i_x \}$

local variables of any func are NOT part of this.

if a constructor/destructor is explicitly written Then it will be part of methods.

m = all methods of class = $\{ M_1, M_2, \ldots, M_n \}$

$|M| = n$.

length of set m.

⇒ we say

$M_1 \longrightarrow I_1$: M_1 uses a set of instances I_1 subset of I

$M_2 \longrightarrow I_2$

⋮

$M_n \longrightarrow I_n$.

$\therefore |P| + |Q| = \frac{n(n-1)}{2}$

⇒ we take two sets :

$P = \{ (I_r, I_s) \mid I_r \cap I_s = \phi \}$: All pairs (I_r, I_s) who have nothing in common with eachother

$Q = \{ (I_r, I_s) \mid I_r \cap I_s \neq \phi \}$: All pairs (I_r, I_s) who have something in common with eachother

$\boxed{\text{where } r \neq s.}$

⇒ we calculate LCom by

1) if $|P| > |Q|$ then $LCoM = |P| - |Q|$

2) Otherwise , $LCoM = 0$

NOTE : The higher The value of P, the greater the LCoM, The lower the quality. (because less communicational cohesion)

Foundation for Advancement of Science & Technology

on GC

The following data is for "UML-class-diagram-metrics-tool.pdf"

Metric	User	Student	Administration
NOp	9	14	10
NA	4	7	7
Csize	13	21	17
NOpA	N/A	5	1
NOpO	N/A	0	1
NOC	2	0	0
D/DIT	0	1	1
CBO	2	4	3
SI	0	0	$\frac{1 \times 1}{10} = 0.1$

10 → 9 + 2 - 1 = 10

own functions

parent functions

repeated function (getName())

= NOpO