TOPIC NAME : DAY : TIME : DATE : / /

Software Quality

- attribute to imply if software meets specs
- Subjective process = management team has to decide if acceptable.

ex: safety, sequritrily, reliability, portability

3 approach to ensure quality of software

1. Organization wide policy, procedure, standards
   ↳ ex:- all codes in Java
   - data encryption

2. Project specific policies
   ↳ ex: secure login system - banking
   fast - game

3. Quality control for each project.

- External entities can varify the standards

TOPIC NAME :

DAY :
TIME : DATE : / /

	Q C (Reactive)	Q Assurance (proactive)	Testing
Purpose	1. Devs developing software w/o worrying about software quality (at last) -Making sure all req, specs complete	1. Before development Predict error and code in a way to prevent it	1. Detecting and solving error and concerned w quality
Focus	as a whole	Process	source code
what	verify and fix error	Prevent	Detect and solve
who	Team work	Team	Developers
Time	Before release verify ↳ making sure of quality before release only	Throughout process	Along w dev

TOPIC NAME : DAY :
TIME : DATE : / /

Software quality Assurance ;

Apply tecnical ways :

+ software testing T test cases to detect error .

+ Enforcing standards . ex security standards check

+ Measurement metrics to track software quality .

+ continuesly recording the mesures and improving it .

Control flow graph

↳ representantion of source code

① Node : All statements become node

② Directed Edge / Arc : execution path of statements
↳ represents a branch

③ path : collection of nodes linked w edges

TOPIC NAME : DAY : TIME : DATE : / /

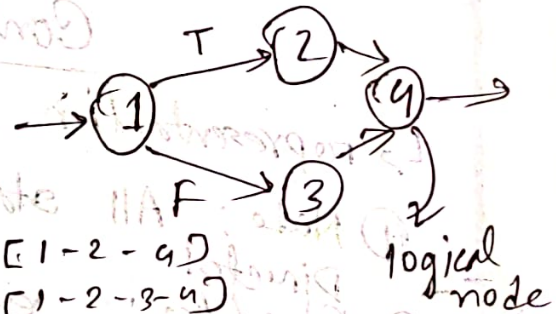
① A CFG should have :
① 1 entry arc : directed edge
② exit arc

② All nodes should have
① ~~At~~ least 1 entry arc
1 exit arc

③ Logical node :
- doesn't represent any actual statement
- joining point of several edge
↳ if-then-else

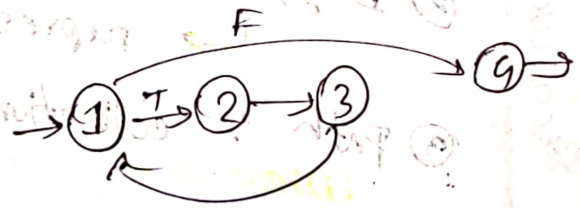
ex: ①

```
if x>0 : } 1
    S1   } 2
else :
    S2   } 3
```



[1-2-4]
[1-2-3-4]

```
while x<10 : } 1
    S1  } 2
    x++ } 3
```



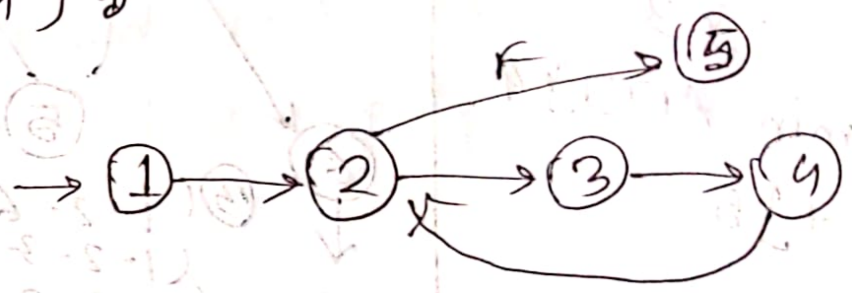
[1-2-3-1-4]
[1-4]

TOPIC NAME : DAY : TIME : DATE : / /

$$\overset{1}{\text{for (I=0}} , \overset{2}{I<10} , \overset{4}{I{+}{+}}) :$$

s1 } 3
s2

s4 } ~~5~~



From CFG, paths list that start at start node and end at end node, cover all arcs and edges → covering graph / path based white box test.

cyclomatic complexity measure : tecniowe to find minum of min paths that cover all arc and nodes in CFG.

step : ① Draw CFG ⑳

② M: ~~How~~ How many paths possible
↳ cyclomatic complexity number

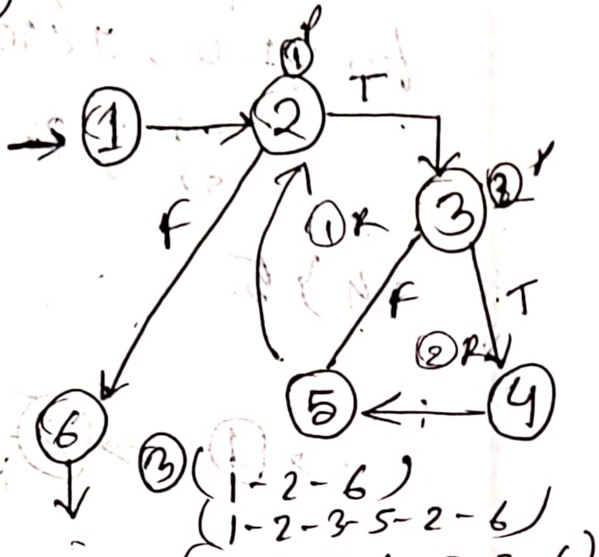
③ Basic path set : c paths

④ Design test cases to cover all statements

TOPIC NAME : DAY : TIME : DATE : / /

①

```
min = A[0] } 1
I = 1
while : } 2
   if : } 3
      min = A[I] } 4
   I++ } 5
print } 6
```



②

② calc M → cyclomatic complexity

① $M = R + 1$ ↳ num of regions → block covered by edges → as a whole $= 2 + 1 = 3$

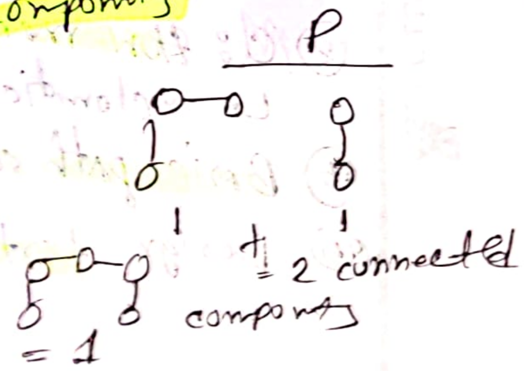
② $M = P + 1 = 2 + 1$ ↳ num of predicate node
[if else - loop
[two paths possible

③ $M = E - N + 2P$

↓ num of edge ← Discard entry, exit
↓ nodes
→ num of connected components

$= 7 - 6 + 2 \times 1$

$= 3$



TOPIC NAME : ③ DAY :
TIME : DATE : / /

Path : independent
- start to end traverse .
- At least one arc that not traverse before
- cover all nodes , arcs at least once

independent paths total <= M
↳ set of these paths → Basic path set

ex!

$$\begin{tikzcd}
 & & & 3 \arrow[dr] & & \\
{} \arrow[r] & 1 \arrow[r] & 2 \arrow[ur] \arrow[dr] & & 5 \arrow[ll, bend right] \arrow[r] & {} \\
 & & & 4 \arrow[ur] & &
\end{tikzcd}$$

$M = \underset{R+1}{2+1} = 3$

1 - 2 - 3 - 5 , 1 - 2 - 4 - 5 , $\underbrace{1 - 2 - 3 - 5 - 2 - 4 - 5}$

① enough to cover all paths
So can be < 3

② After this all paths will have been covered

TOPIC NAME : DAY : TIME : DATE : / /

Find path: [1-2-6]

~~A~~ = {5...}

```
A = 5 , N = 1 } 1
I = 1
while  I<N { } 2

        }

print  min { 6 }
```

Specialization Index

SIX

```
class person{
    read()
    display() }
class student ex Person{
        read ()
        dis ()
        avg ()
class   gradu ex stu{
    read ()), dis(), work()
```

SIX =

TOPIC NAME : ______________ DAY : ______________

TIME : DATE : / /

$$SIX = \frac{NMO \times DIT}{NMO \cancel{+} NMA + NMI}$$

NMO = num of overloaded ops
read() , display() in graduate = 2

DIT = depth of inheritance (2)
↳ person , class , graduate

$$\begin{tikzcd} person \arrow[r, bend right, "1"'] & class \arrow[r, "2"'] & graduate \end{tikzcd}$$

NMA = ~~no~~op added for inheritance
work~~arg~~ () → 1

NMI = inherited but not overloaded
arg () = 1

TOPIC NAME : ______ DAY : ______ TIME : DATE : / /

Refactoring

+ change internal structure. w/o changing external - take small steps.

ex: ① Delete duplicate code

② class too big - remove extra portion

~~No~~ ↳ Doesn't external bhaviour

Not refactoring : ① using another Database

② login system add

Need : ① Buisness value

② better design ③ development speed

④ easy to understand code

⑤ error identify

Use when:

① Before adding new func

+ make understandable code

+ make it simple

② Find error - understand

③ code review -

GOOD LUCK™

TOPIC NAME : ______ DAY : ______
TIME : DATE : / /

Refactor doesn't add new func but makes existing one understandable and simple

Process :

① codesmell - suspected things to cause issues are codesmells.

② → take baby step

③ Do not harm code

codesmell : Duplicates, complex etc are codesmells

ex: dup code, large class, long method, Dead code,

Inappropiate Naming

- var, method name has to be meaningful
  ↳ refactor by giving proper name

comments

Code should be self explanatory so we don't need comments.

↳ if can't understand code w/o comment - codesmell

TOPIC NAME : ______ DAY : ______ TIME : DATE : / /

Refactor comments : ① ~~Extract method~~
~~HRO assertion~~

// comment

{ code } $\longrightarrow$ Divide code into methods

② Rename method

getinvc $\longrightarrow$ get invoice ( )

③ Extract method

extract lines and put in a method

① Introduce Assertion

// remove comment of if-else

add assert method for checking

Long Method

+ Duplicate code + hard to manage

+ hard to comprehend

Refactor :

TOPIC NAME : ______________ DAY : ______________
TIME : DATE : / /

① Extract method : use functions
↳ seperate codes

```
main {
  x1 ()
  x2 ()
  x3 ()
}
```

```
define x, {
  code }
```

② Replace temp with query

```
get val ()
  ret x+a
```

use get_val () in code

③ Introduce parameter Object

func (x, y, z) → replace with obj (values)

```
class values {
  set x
      y
      z
}
```

④ Preserve whole obj

```
low = get obj () . get low ()
```

pass obj directly

```
w = P. main ( get obj() )
```

⑤ Decompose conditional

if (........)
↳ bring in a method

TOPIC NAME : DAY: TIME: DATE: / /

Feature envy

When a method is more intresented in other class than it's own.

+ when a method calls too many class

Sol: ① replace class to the class it calls most

① More field

+ send variable to class 2 if it or method calls class 2 more

②

Duplicated code

extract, pull up field, form temp, substitute algo.

① Literal duplicacy - word same exact

② Symantic - not same by looks - but doing same thing

↳ level ① for auto vs for each i++ loop

② Loop (5) vs lines repeat 5 lines

TOPIC NAME : DAY : TIME : DATE : / /

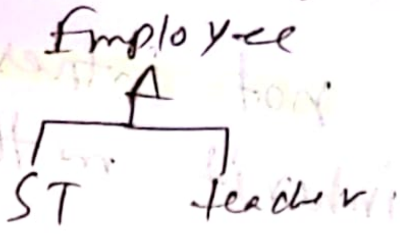
conceptual dup: $\rightleftarrows$ qruick sort / bubble sort > same using diff algo

Logical dup : changed order . ngt diff

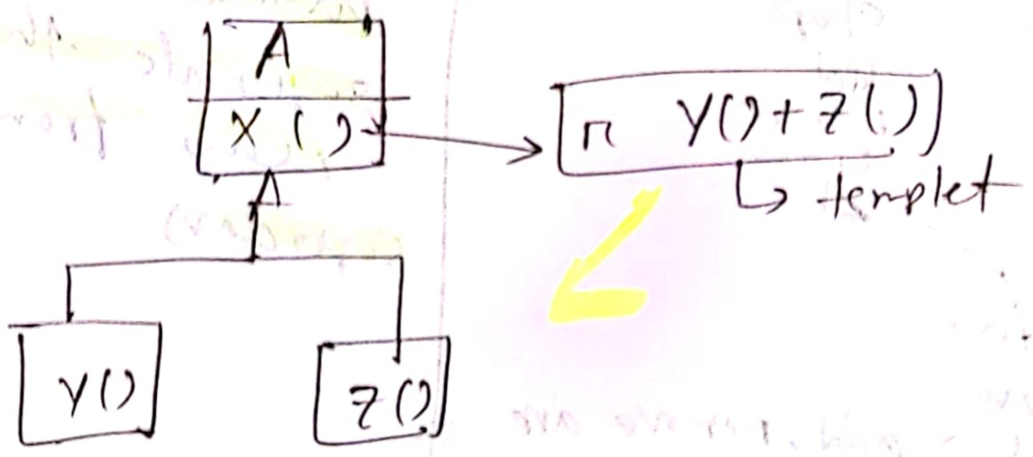
ex: login if email | if pass
if pass | if email

Solve

① pull up method : bring under one superclass



② Form templet method :



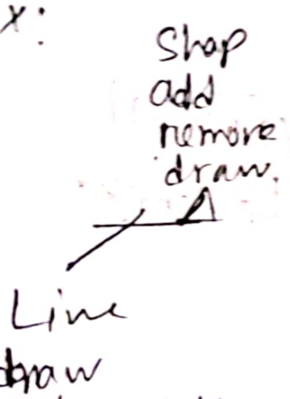
③ Substitute algo :

```
if Jhon :      |  condits = [Jhon, Aunto]
if Aunto :     |→ check in arr !
   ⋮           |
20 times       |
```

Refused Beqruest

subclass not intrested to do assigned task / inherit method they don't want

ex:



↳ add, remove are redundant for this class

pull down, seperate the class from superclass