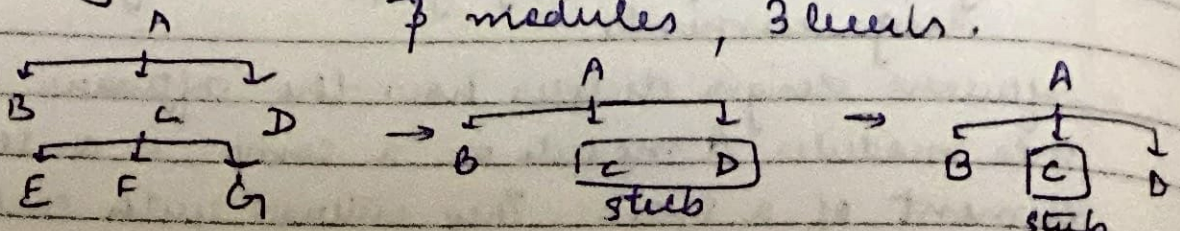
Week-4

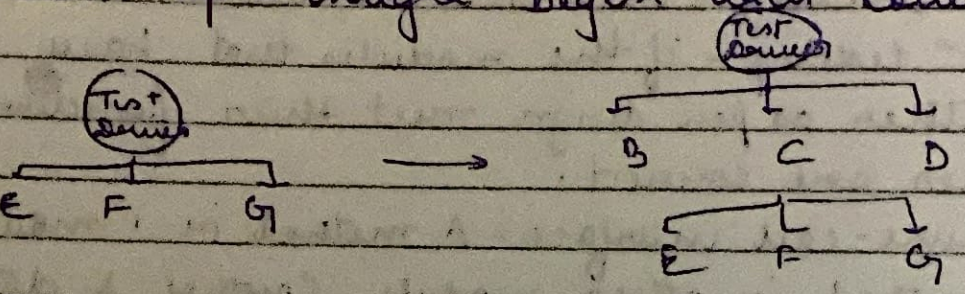
L1 Software design & Integration Testing

- Integra^n testing — appy graph coverage criteria to design elements.
- Software design dictates how the software is organized into modules. A module or a comp. is a self-contained element of a system. They interact with each other using interfaces. Interface implements a mechanism for passing control & data btw modules.
- Integra^n testing - if the modules that have been put together as per design meet their functionalities & if interfaces are correct.
- Procedure-call interface - A method in 1 modules calls a method in other module. Control & data can be passed in both direc^{ns}.
- Shared memory interface - A block of memory is shared btw 2 modules.
- Message-passing interface - 1 module prepares a msg. of a particular type & sends it to another module there this interface.
- 25% of all errors are interface errors. Types - module functionality, interface related, inadequate proc^g initializa^n errors, etc.
- When testing incomp. portions of software, we need extra software components, c/d Scaffolding.
  1. Test Stub → Spl. purpose implementa^n of a software module, used to develop/test a comp. that calls the stub.
  2. Test Driver → software comp. that replaces a comp. that takes care of the control & calling of the comp.
- 5 app. → Incremental (top-down, bottom-up), Sandwich, Big-Bang

- Top-down → works well for sys. with hierarchical design.



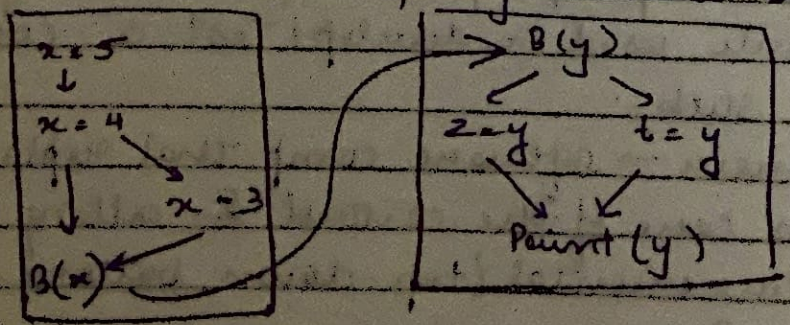
- Bottom-up → integrn begin with lowest levels.



- Sandwich uses mix of top-down & bottom up.
In big-bang, all tested modules constructs the entire sys., which is tested as whole.
- Structural Coverage → calls over interfaces
- Data flow " → exchange of data over interfa

L2 Design Integrn Testing & Graph Coverage

- Call graphs → graph models for integrn testing
Nodes → modules, Edges → call interfaces.



call interface

coupling du pair eg.

- Node coverage calls every module at least once. Edge coverage executes every call at least once. Specified path coverage tests a seq. of method calls.
- Data flow criteria for modules are most complicated.

A → caller → module that calls another module

B(x) → actual parameter → variables in the caller

call site → the node where the call appear in the code.

end A

interfaces

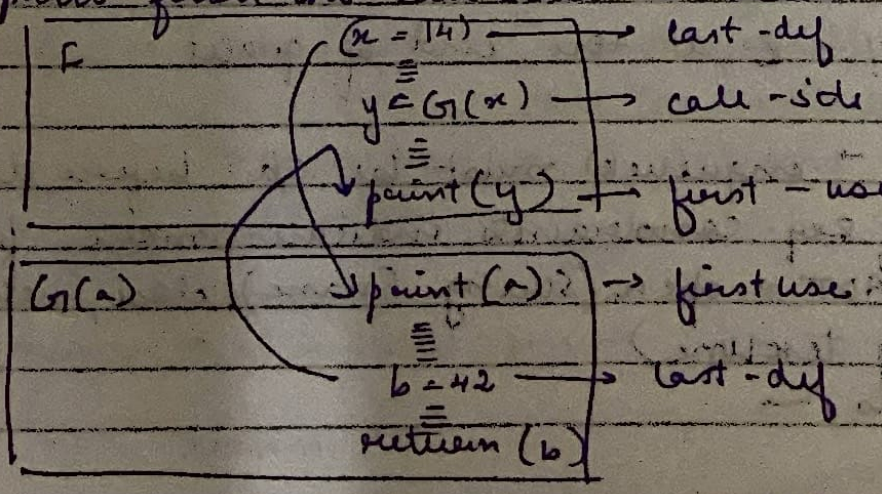
B(y) → callee → module that is called.

end B → formal parameter → variables in the callee.

- Coupling variables → Vars. that are defined in 1 unit and used in the other.

1. Parameter coup. → parameters are passed in calls.
2. Shared data " → 2 units access the same data by global vars.
3. ext. device " → 2 units access the ext. obj.
4. msg. passing interfaces.

- Last def - The set of nodes that define a var. x & has a def-clear path from the node there. a call site to a use in the other module.
- First use - The set of nodes that have uses of var. y & for which there is a def-clear path & use-clear path from the call site to nodes.



- A coupling du path is from a last def to first use
1. All-coupling def use → every last def to atleast 1 first use
2. " " use cov. → every last def to every first use
3. " " du paths " → every simple path from every last def to every first use

L3 Specifica^n Testing & Graph Coverage

- A design specifica^n describes aspects of what behaviour a software should exhibit. We have 2 types of design specifica^{ns}
1. Seq. constraints → methods
2. State behaviour → software descrip^n

- Seq. constraints → rules that impose constraints on the order in which methods will be cfd. They may be precond^{ns} on other specifica^n.

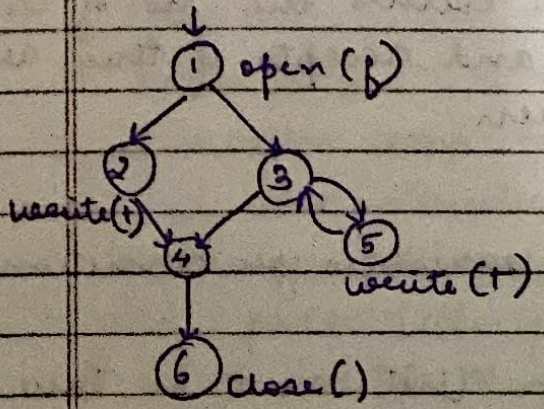
Eg. dequeue ()
{
} pre // → at least one element in queue.
enqueue (e)
{
} post // → e at the end of queue

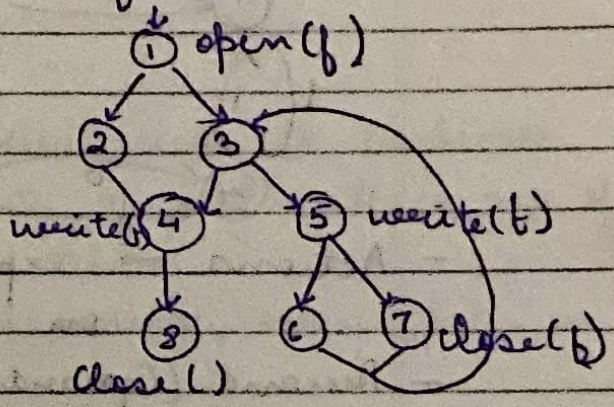
Seq. const. → enqueue(e) must be cfd before dequeue.

- Absence of seq. constraints indicates more fault.

Eg. class FileADT → open (String Fname), close, write (String textline)
Constraints →

(i) open(f) must be executed before write(t).
(ii) open(f) " " " " close().
(iii) write(t) may not be executed after a close() unless there is an open(f) in btw.
(iv) write(t) should be executed before close().





[1,3,4,6] violates 4

[1,3,5,7,3,5] violates 3.

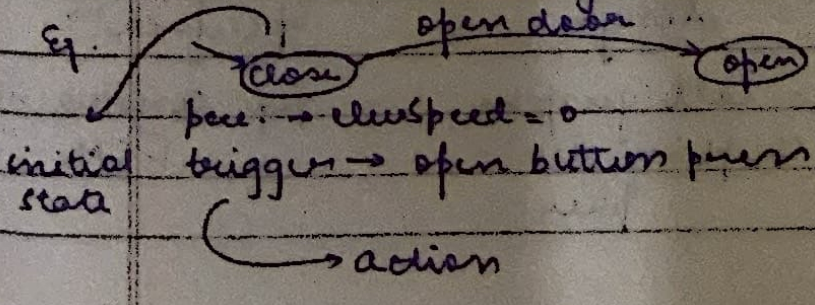
- Not all seq. constraints can be captured using simple constraints. Hence, we use FSMs (Finite State Machines)

L4 Graph Coverage & FSMs

- A FSM is a graph that describes how software variable are modified during execn.

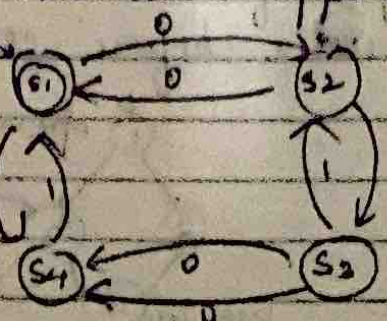
Nodes → States (set of values for key variables)
Edges → Transitions (model possible changes from 1 state to another).
↳ have guards & actions with them.

Eg.



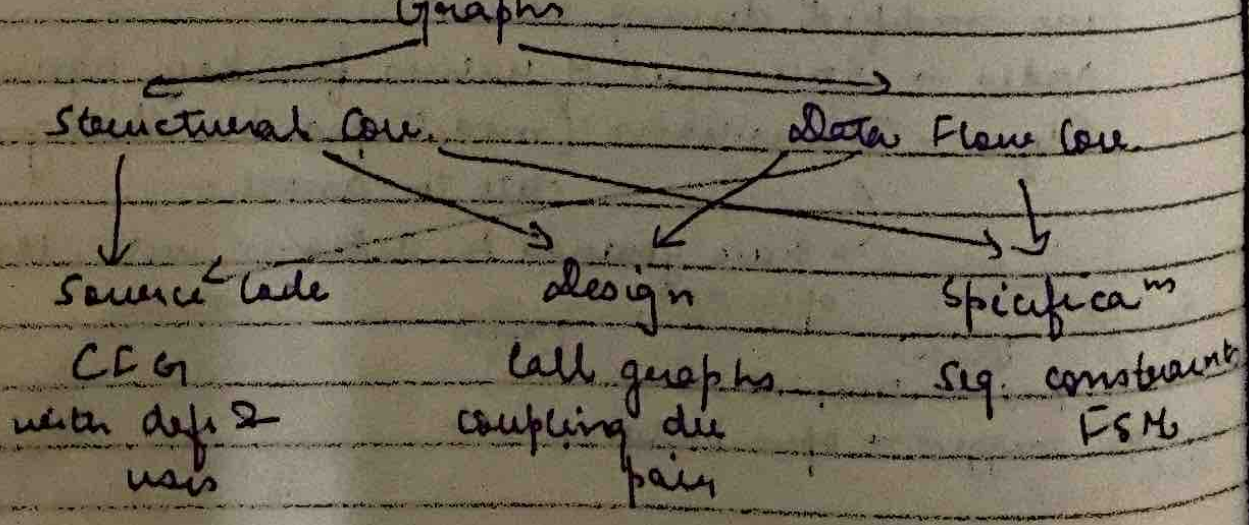
- many modelling notations support FSMs - UMLs, state tables, boolean logic, etc. They are good for modelling control intensive applications, not for data intensive applications.

Eg.



FSM counts the no. of 0s & 1s and accepts if they are even.

- Actions → express Δs to variables or conditions on variables.
- Precond. (Guards) → Conditions that must be true for transitions to be taken.
- Triggering Events → Δs to variables that cause transition to be taken.
- Node cov. → execute every state (State cov.)
- Edge " → " every transition (transition ")
- Edge pair " → " " pair of transitions
- Data Flow cov. → nodes do not include defs/uses of var., defs of variables are used immediately.
- CFGs are not FSMs representing software. Call graphs are not FSMs.



L5 Testing Source Code – Classical Coverage Criteria

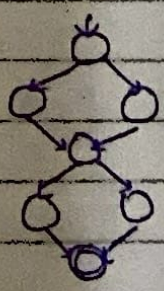
- Most common graph model for source code is CFG.
- Code coverage, Cyclomatic complexity, data flow testing, DD-Path.
- Node coverage = Statement cov.
  Edge " = Branch "
  Prime path " = Loop "
- Cyclomatic comp. → software metric used to indicate the structural complexity of prog., It is no. of linearly independent paths in CFG.
- Basis path testing → tests each linearly indep. paths in the CFG of prog.
- Linearly indep. path → A path that does not contain other paths in it.

$$M(CP) = E - N + 2P$$ → no. of connected comp.

- For graphs which correspond to single program, w/o any method has only 1 comp.

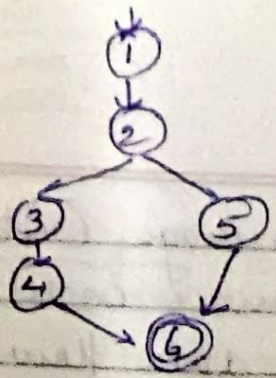
$$M = E - N + 2$$ → cyclomatic no.

Eg:



$M = 8 - 7 + 2 = 3$

- Basis path testing → tests a code w.r.t its M. CFG is analysed to find a set of linearly indep. paths of execution. This becomes TR.
  → This subsumes branch (edge) cov., comp. path cov. subsumes this.
- A DD path → is a path of execution btw. 2 decisions in the CFG.

Eg. 

- A chain is a path in which →
  (i) initial & final nodes are diff.
  (ii) all the int. vertices have in & out degree as 1.

- A maximal chain → which is not part of other chain.
- DD path (set of vertices) that satisfies 1 of the conditions —
  (i) 1 initial vertex with in-degree 0.
  (ii) 1 final " " out-degree 0.
  (iii) 1 decision vertex with in/out-degree ≥ 2.
  (iv) Vertices with in-degree & out as 1
  (v) maximal chain of len ≥ 1.

Eg. [1], [2], [3,4], [5], [6]