- **Software Engineering**
  - Programe / Instructions + Documents
  - Design / create / build
  - solution to Problems
  - systematic & disciplined
  - Development, test & maintain

Example: Making of your house.

- User Requirements
- Time & budget
- Material & labour
- Steps of construction
- Blueprint designing
- Changes / updates

- Charateristics of `S/W'
  1. It is engineered NOT manufactured
  2. Does not wear out
     It deteriorates / downgrade / degrade due to heavy changes.
  3. It is custom built.
  4. Can't be touched (untouchad)

- Software Myths → false beliefs
  - → confusion
  - → Incorrect information

1. Management Myth
2. Customer Myth
3. Practitioner Myth

(I) Management Myths

1. We have all standards & procedures available for software development.
2. Adding of the latest programs will improve s/w development.
3. Adding up of more programmers to s/w development can help meet deadlines (In case of lag).

(II) Customer Myths

① General statement of intent is enough to start writing programs, the details can be filled later

② s/w requirements continually change but change can be easily accommodated because s/w is flexible.

(III) Practitioner Myths

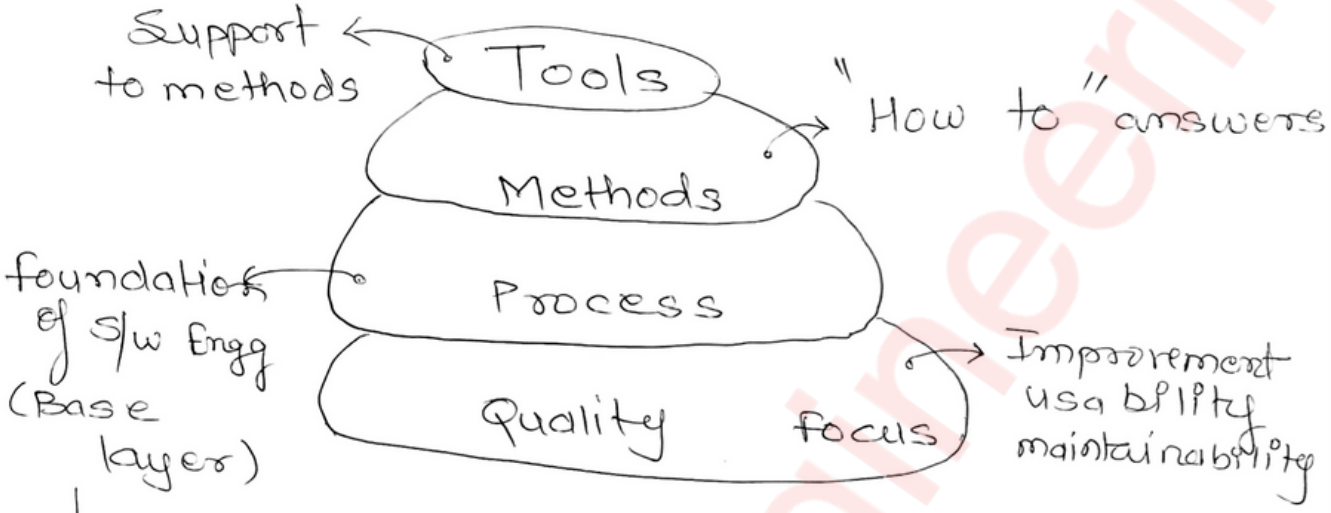
① Once the program is written, the job is done.

② There is no way to achieve or assess system quality, until it is "running"

③ The only deliverable product is the working program

④ s/w engg creates voluminous & unnecessary documentation & slow down the s/w development.

- S/w Engg :- A Layered Technology.



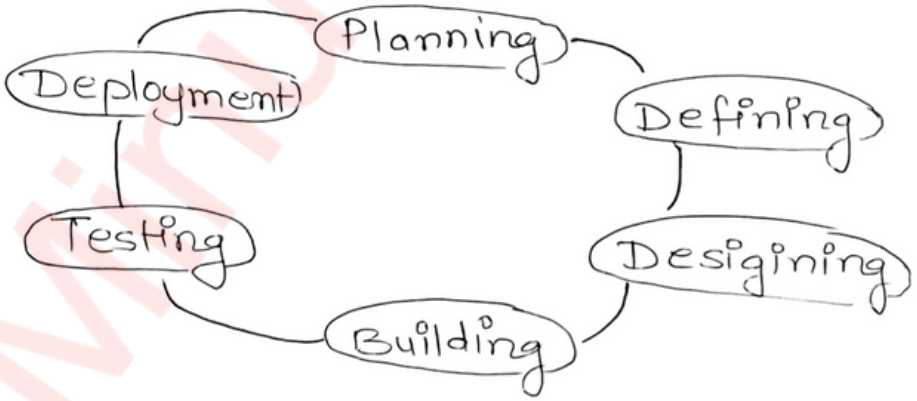
- communication
- Planning
- Modeling
- Construction
- Deployment.

- Problems / challenges in s/w development

1. Requirements: (?)
   - Incomplete
   - Ambiguous
2. Time & Budget: (✗)
   - Unrealistic
   - Jaldi ka kaam shaitan ka hota hai
3. Technical skills (↓)
4. change/updates (x)

- Software Quality Attributes
  - Correctness [As per User Requirement]
  - Usability [ease to use]
  - Reliability [failure / error proof]
  - Efficiency [Optimal / Best use of Resources]
  - Maintainability [Ease of Modification]
  - Scalability [↑↓ workload / Request / User demand]
  - Portability [Operating on different environment]
  - Security [protection against malicious behaviour]
  - Reusability [Part of one thing can be used at other]
  - Modularity [Dividing a thing into multiple module & operating on them]

- Software Development Life cycle [SDLC]



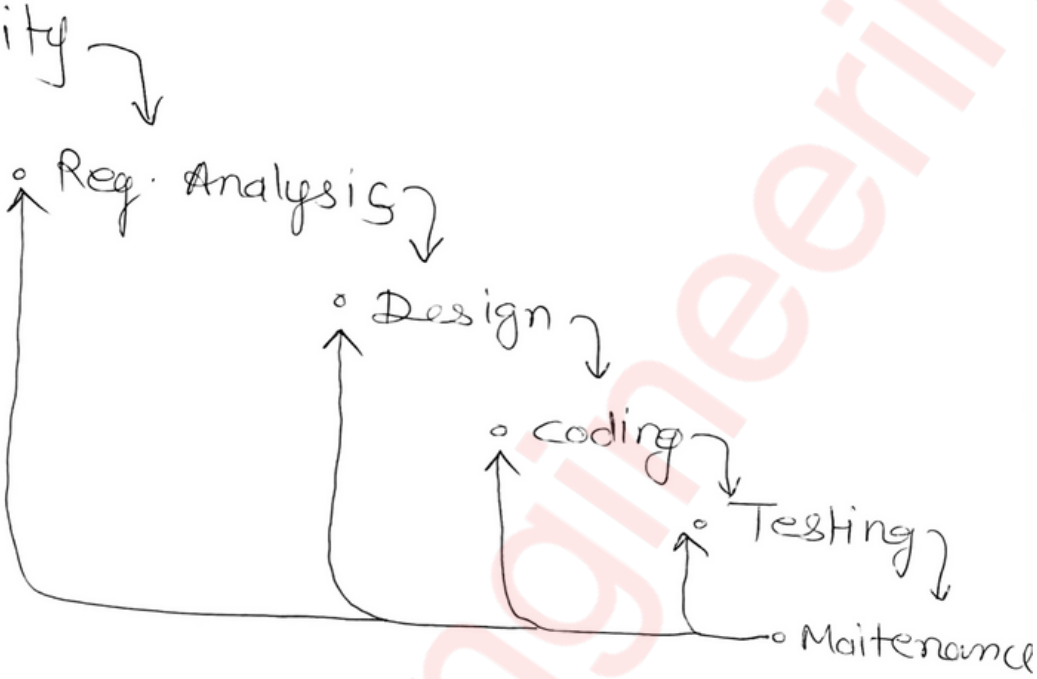
- Waterfall Model:
  - fundamental
  - Simple
  - Sequential
  - Small projects

- feasibility Study
- Requirement Analysis
- Design
- coding
- Testing
- Maintenance

- High Risk
- NO Experiment
- serial(✓)
  Parallel(x)

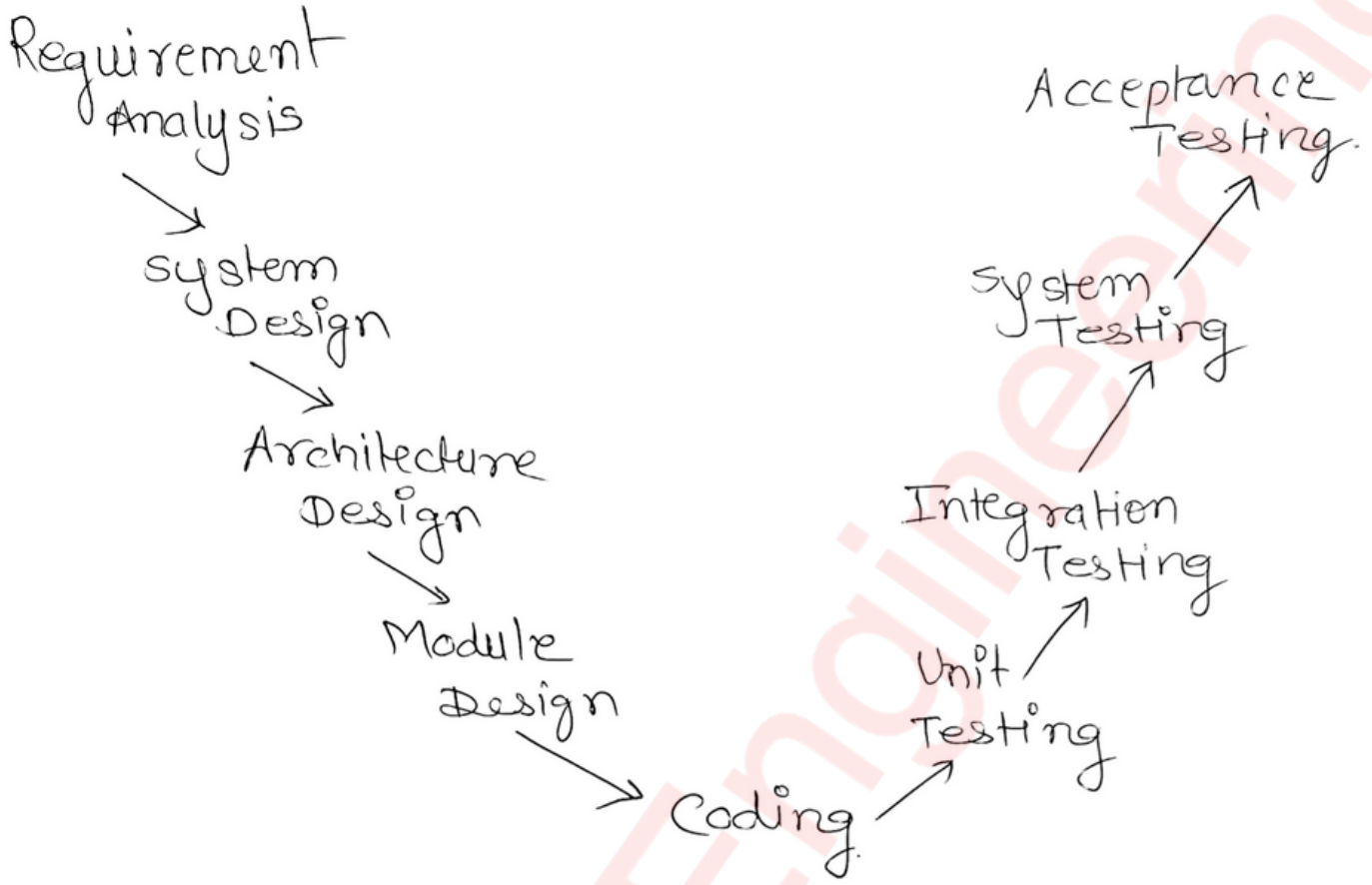
- Communication
- Planning
- Modeling
- Construction
- Deployment

- Iterative waterfall Model



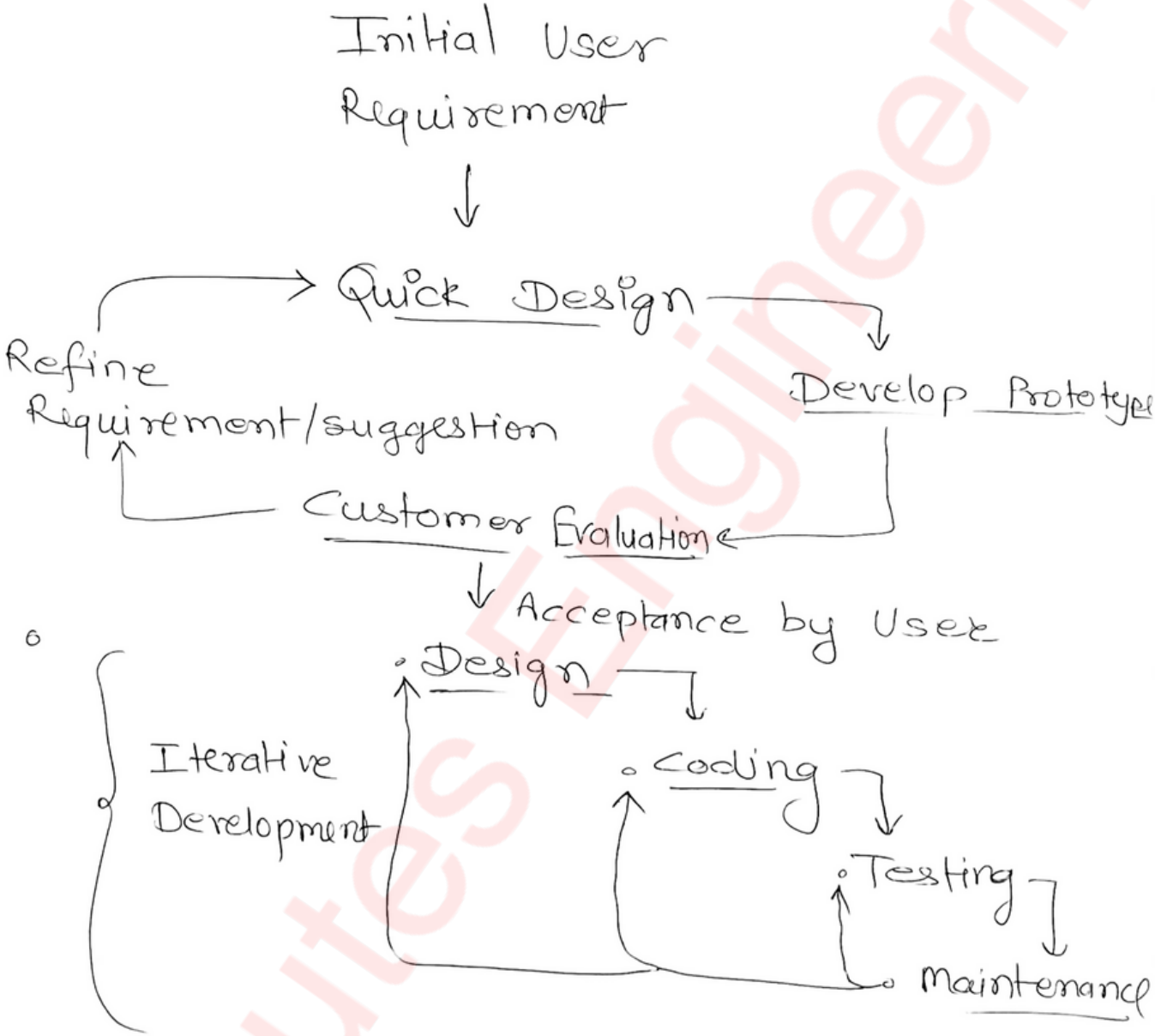
⇒ Here you can "go back".
⇒ Small projects
⇒ serial (✓)
⇒ fixed Requirement Natured

- V Shaped Model (verification & validation)
  - Testing in every phase/stage
  - verification
    - Req. Analysis
    - system, architecture, Module Design
  - validation
    - Testing (Unit, Integration, system)



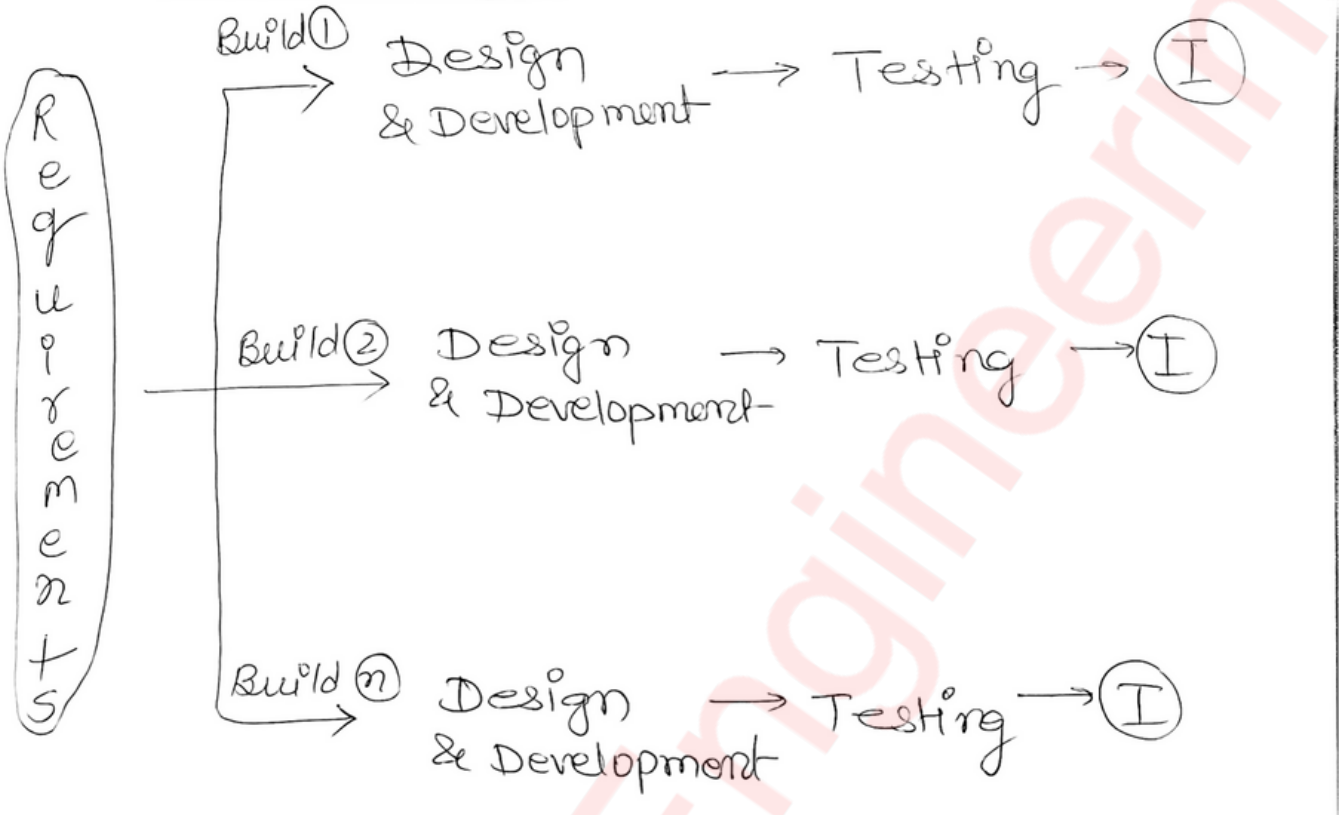
- Use when requirements are clear & fixed.
- Small to medium projects
- Simple & easy
- Time saving.
- 'Testing' is emphasis

- Prototyping Model

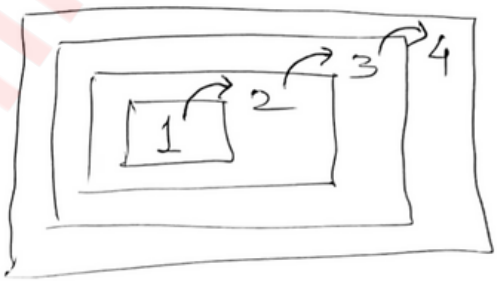


- Requirements / Idea not clear
- Cost of development ↑↑
- Can cause delays.

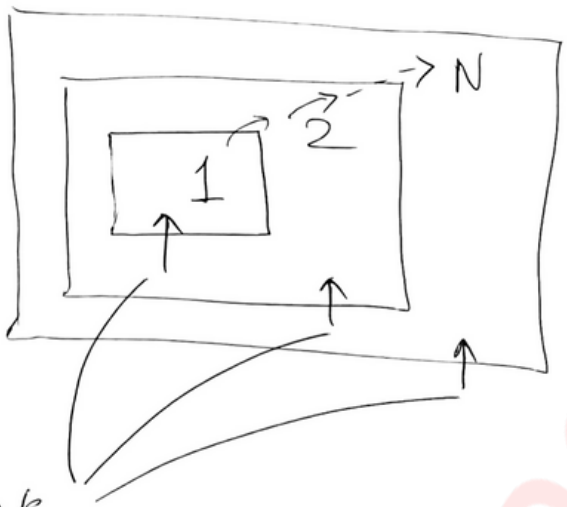
- Incremental Model



- Req. are divided into multiple modules
- Each module goes through design, development Testing & Implementation.
- module by module [function adding]
- Can be used for medium - Large projects
- Total cost is ↑↑



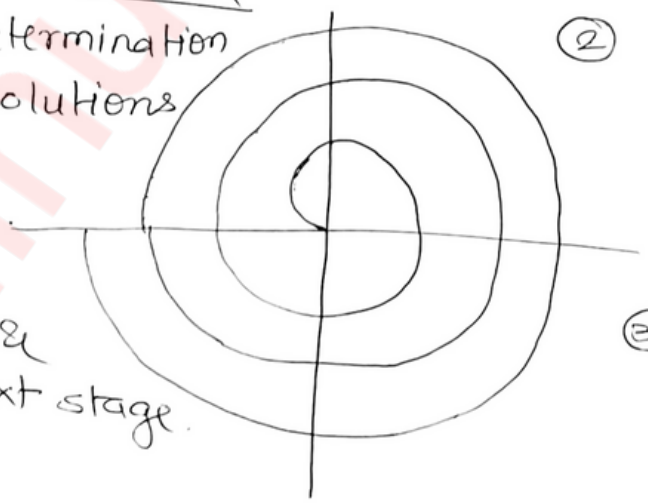
- Evolutionary model
  - ↳ Iterative + Incremental



feedback

- for large Projects
- Risk analysis is good
- Cost is ↑↑
- feedback is available

- Spiral model



① Obj. determination & Alt. solutions

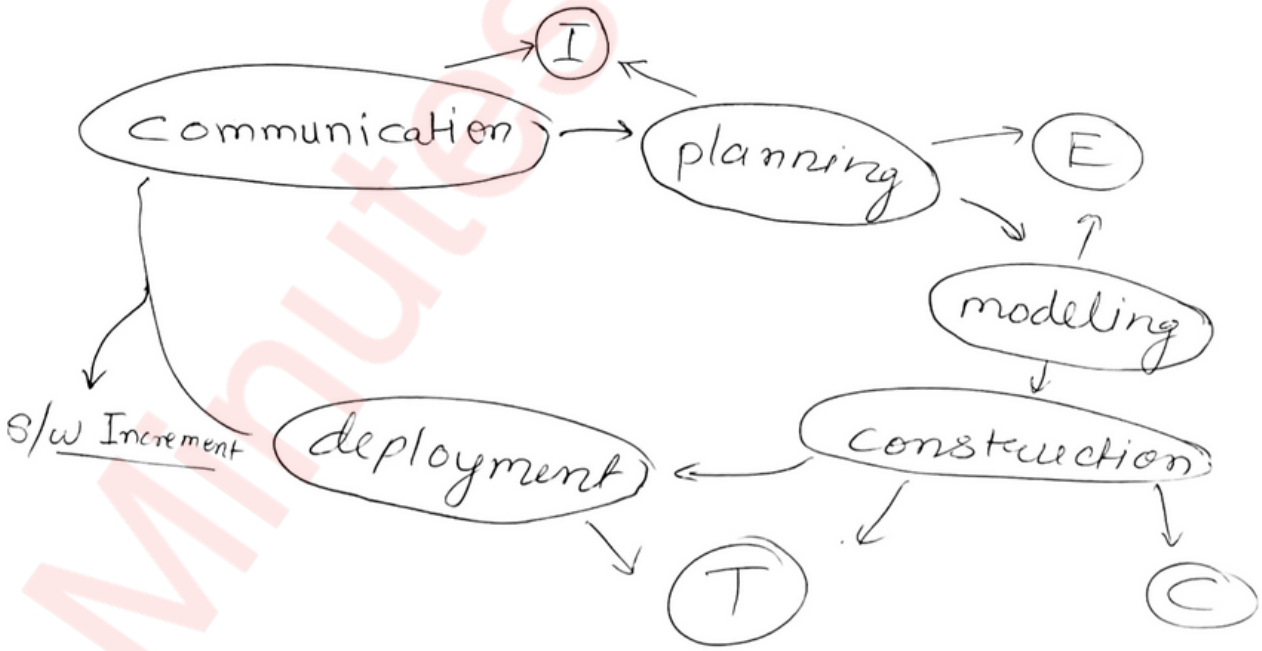
② Identify & Resolve Risk

③ Develop Next version of Product

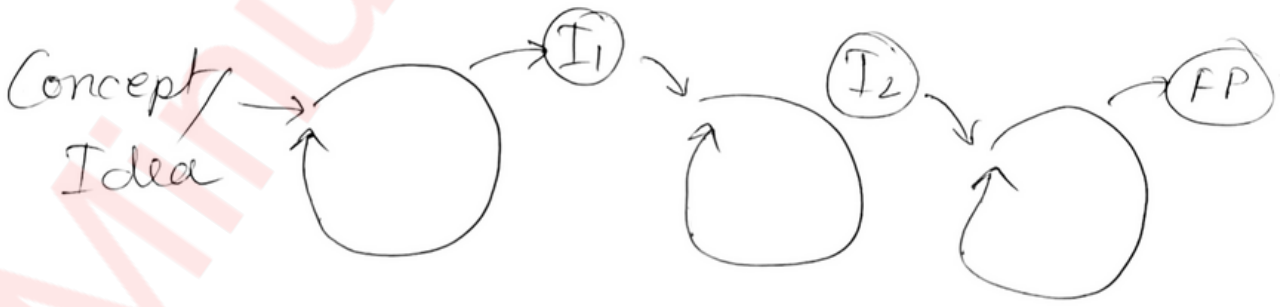
④ Review & Plan Next stage

- Risk Mgmt / Handling / Analyze
- Cost ↑↑ ∝ Radius of spiral model.
- Can be used in Large Projects.
- Not easy & simple, its complex.
- Custome satisfaction.

- The unified Process
  - Inception
  - Elaboration
  - Consteuction
  - Transition.



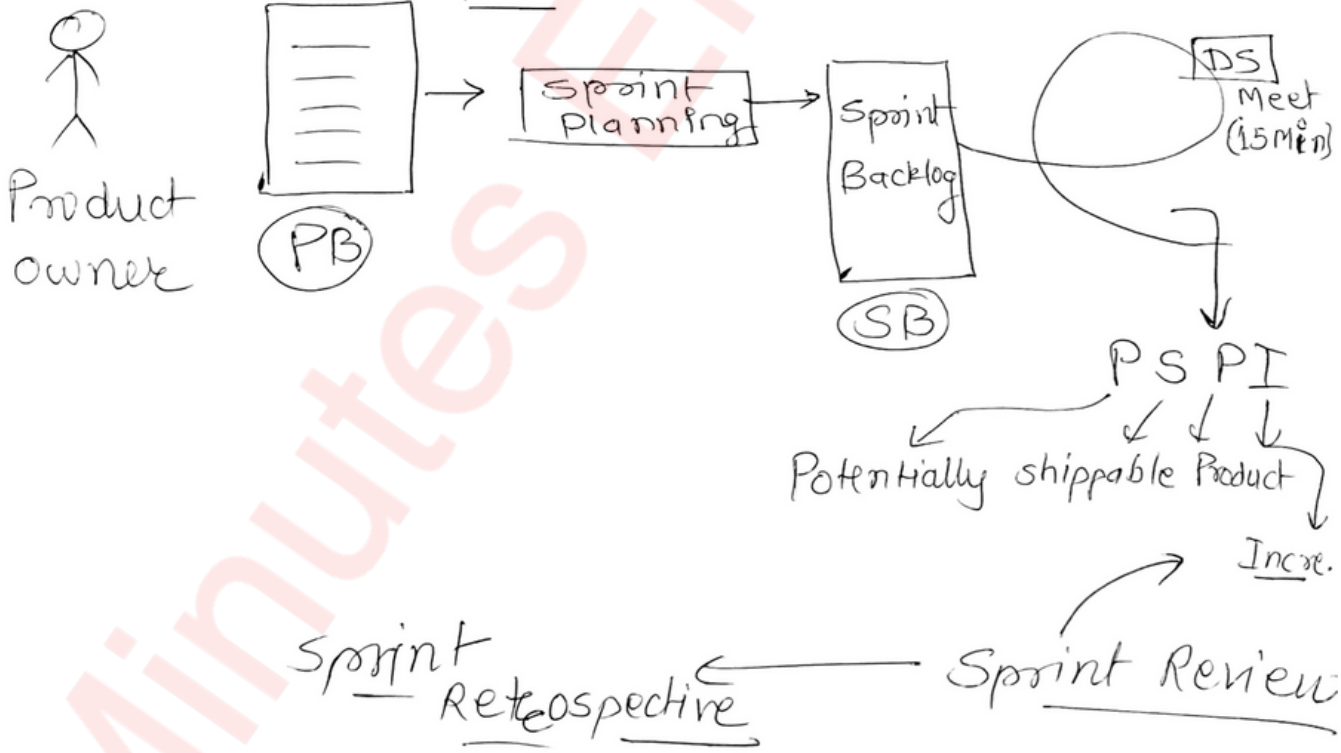
- "Agile" S/w development.
  - Quick & accurate
  - flexibility & collaboration
  - Principles
    - Communication & Collaboration [Do'nt just run behind formality & Procedure]
    - focuse on incremental product solution first then go for detailed lengthy documentation
    - Prioritize clients ideas and requirements.
    - Accept changes & be flexible.



- SCRUM Modul

  - → Agile Methodology
  - → Light weight, simple to understand
  - → Team work & Collaboration.
  - → fast, Quick & money efficient
  - → Sprint: It a duration or Time periods of a month or less.

- Prioritized list



Requirements analysis & specification

→ Understand $\xrightarrow{\text{exact}}$ Req. of client
$\xrightarrow{\text{what}}$ to built

→ Ambiguities & inconsistencies (x) incompleteness

"Communication is the key"

→ Requirement changes also occur
→ Time & Budget constraints
→ Understanding gap.
[Different client, different field, different expectation]

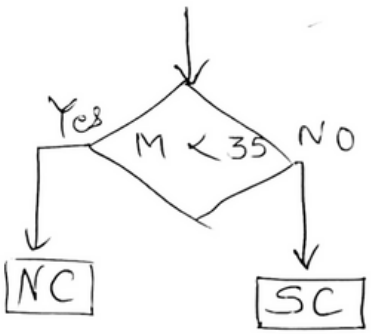
Questions:
1. what's the problem ?
2. Why its important to solve it ?
3. what are the possible solutions ?
4. what's the I/p & O/p exactly should be ?
5. what issues/difficulties may arise ?
6. Budget & deadline of the project ?

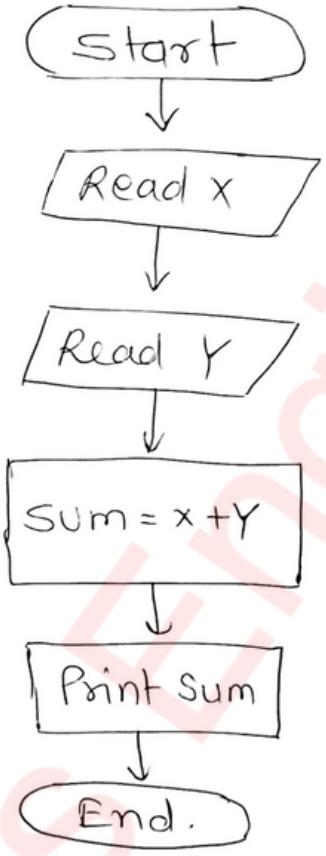
- Steps of Analysis & Specifications.
  - → Req. gathering
  - → Req. Analysis
  - → Req. Documentation
  - → Req. Review

① Req. Gathering
- → Interview / meet
- → Group Discussion [from Both sides]
- → Delphi technique [written GD, iterative nature]
- → facilitated Workshop [team, collaborative & creative]
- → Quality functional deployment [VIMP, IMP, NIMP]
- → Role Play [Different user types]
- → Usecase Approach [visual / Graphical Representation]

② Requirement Analysis

(I) flow chart

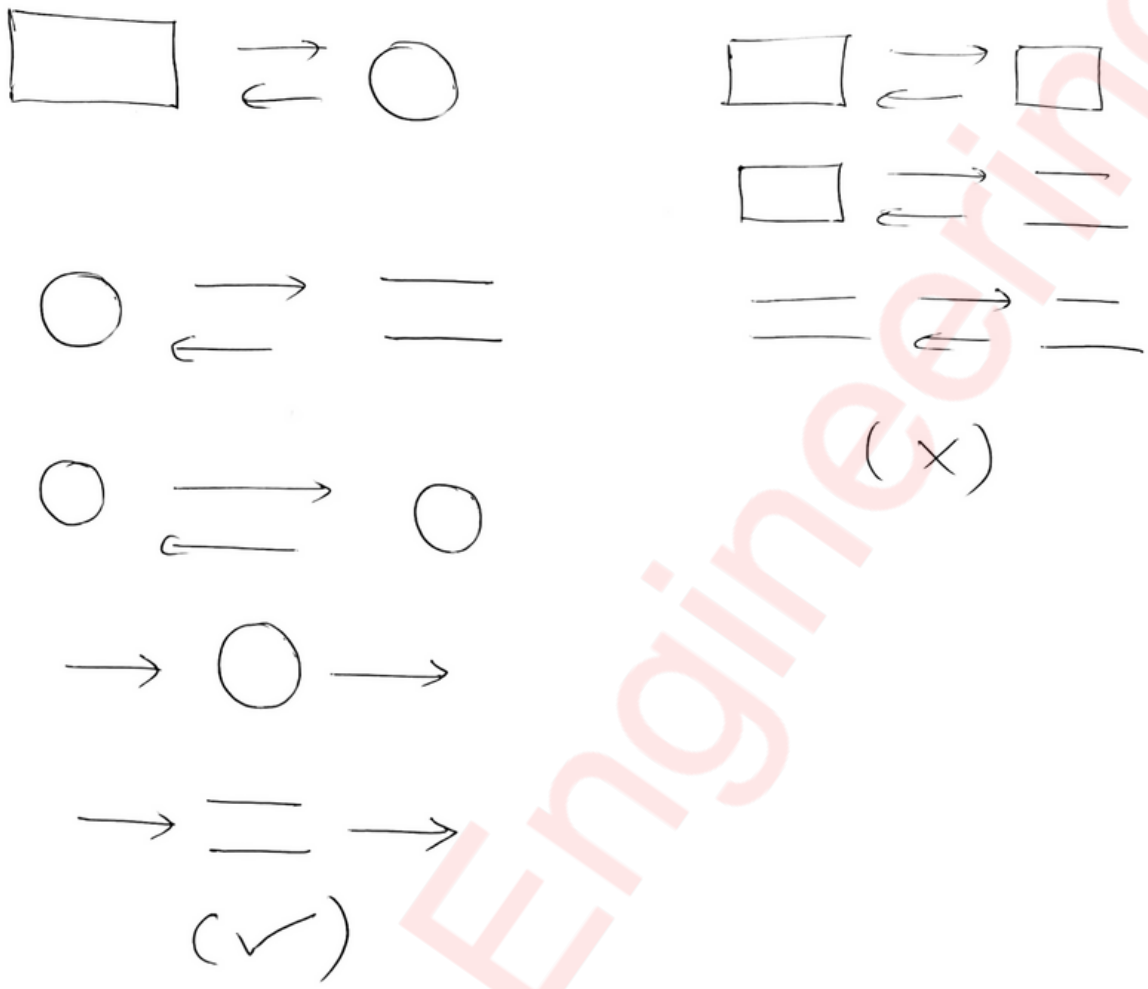


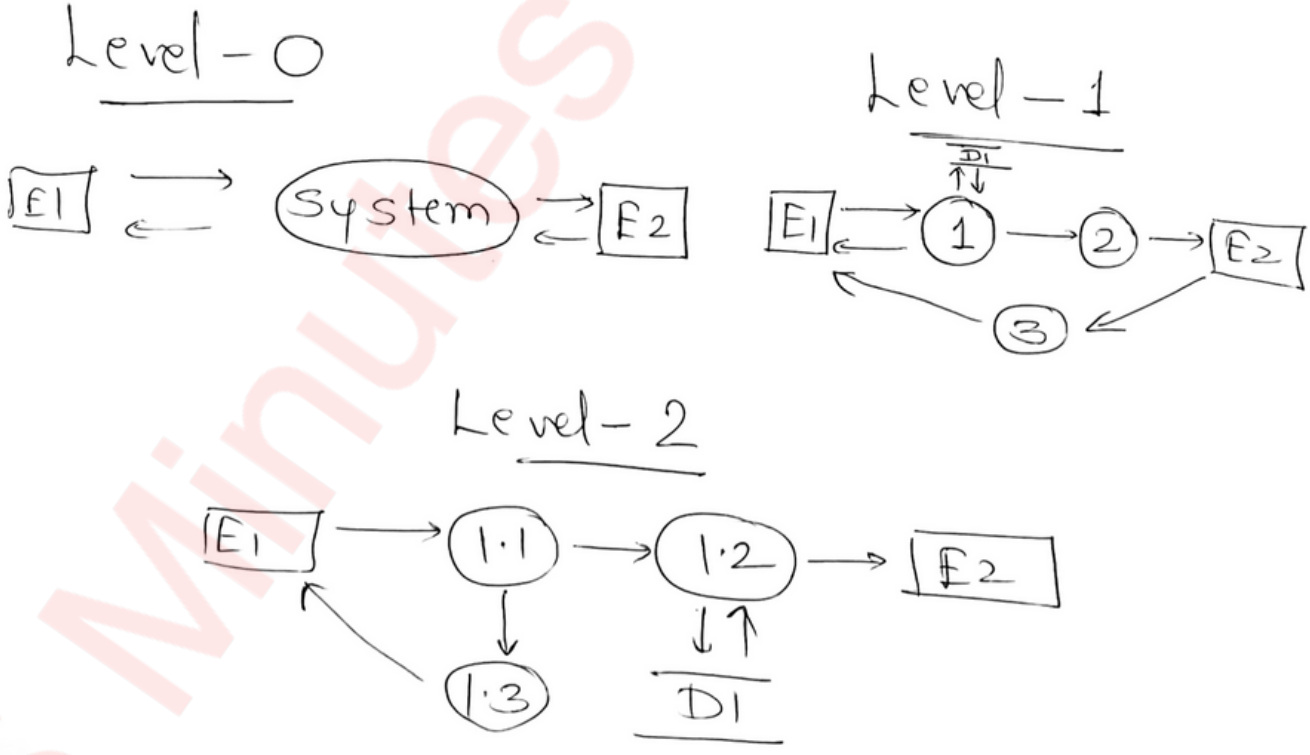


Control flow during execution of program

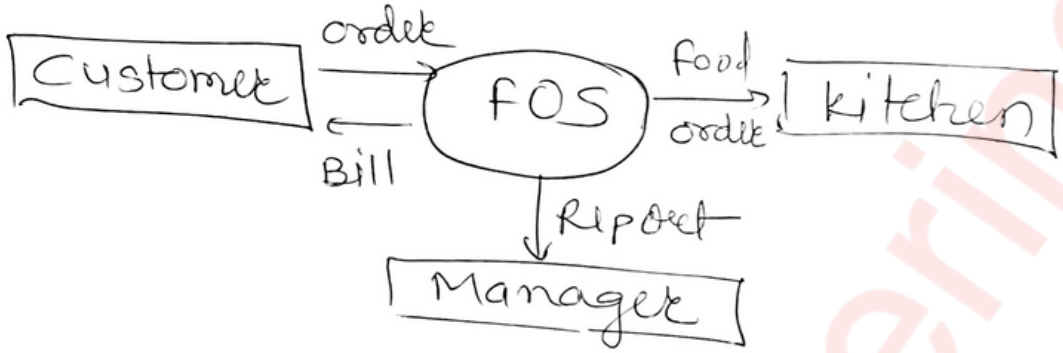
(II) Data flow diagram

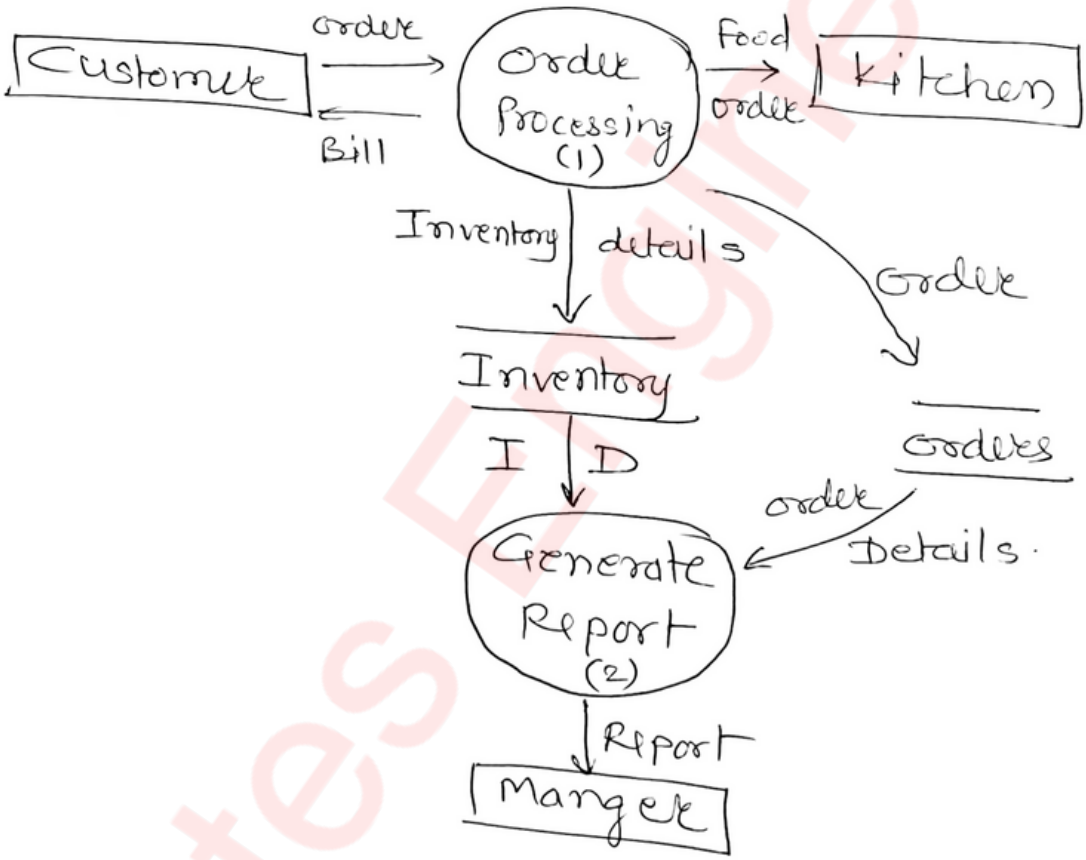
- Process
- External Entity
- Data store
- Data flow

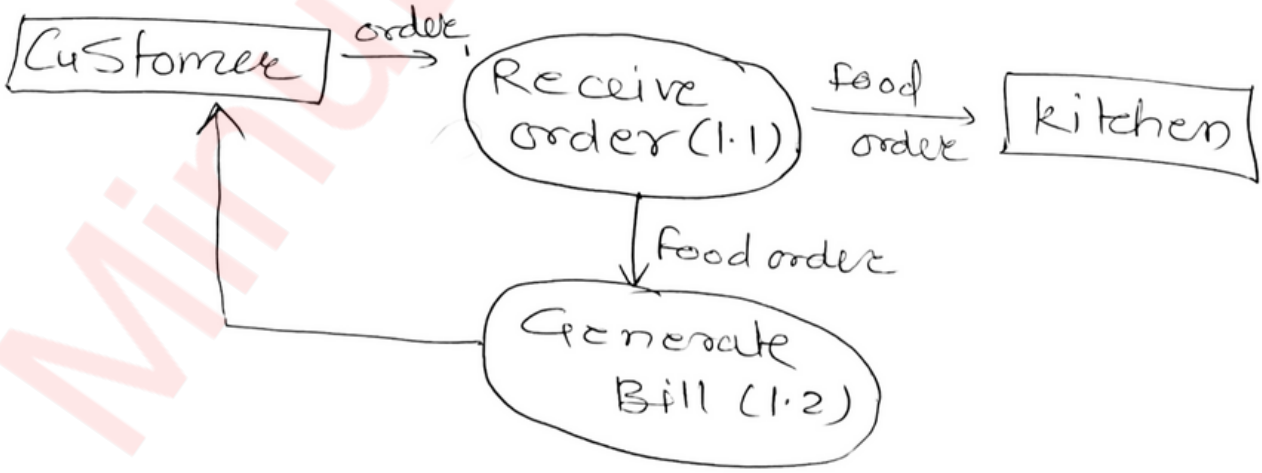




Eg:







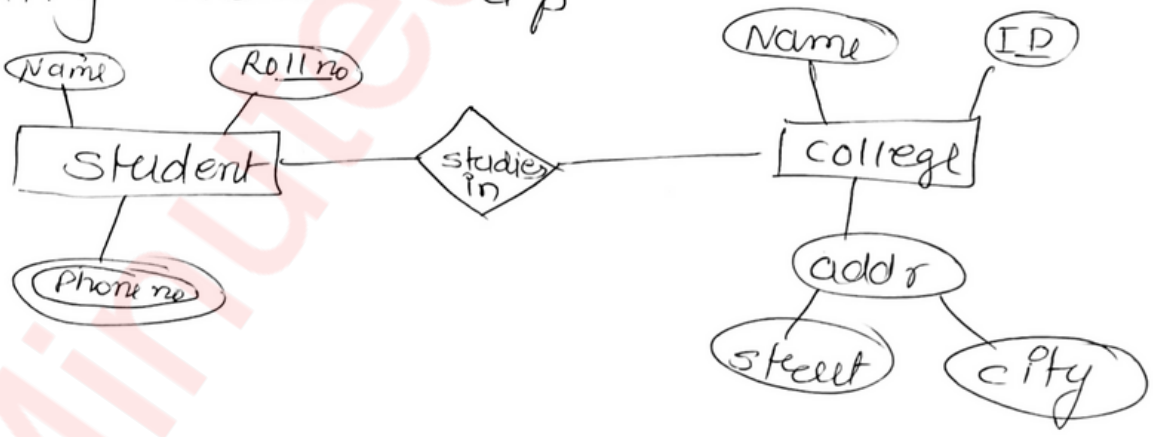
(III) Data Dictionary [lists all data items in DFD]

- Name of item
- Aliases
- Description
- Range of values. [1-10, 1-50, 1-100]

→ It clears out the ambiguity
→ Provides standard terminology for data.

(IV) ER Diagram

Entity Relationship



(V) Decision Table

⇒ Cause - effect table

↙ ↘

I/p condition → O/p action

" If - then - else "

eg:

Conditions	R1	R2	R3
C1			
C2			
C3			
Actions			
A1			
A2			

(3) Requirement Documentation:

- Req. Gathered ✓
- Req. Analyzed ✓
- Now we need to document them in some sandard format (IEEE 830)
- Here our SRS will be as per the gobaly accepted standard.

- Benefits of IEEE 830
  - Promotes → clarity
    - consistency
    - completeness

I) Introduction
II) Overall Description
III) Specific Requirements
IV) Update Mgmt
V) Document Approval.

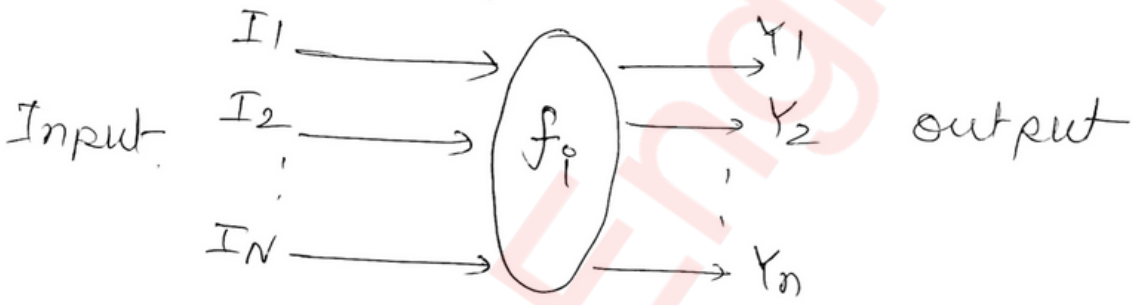
(4) Requirement Review

→ "final call / step / phase"

eg: final check after completely writing the answer sheet.

- Imp parts of SRS Documents
  - → functional Req. of system
  - → Non functional Req. of system
  - → Goals of Implementation

① functional Req.



" specifies the functionalities req. in system"

② Non-functional Req.

→ Deals with characteristics of system that cannot be expressed as fn.
eg:- Usability, maintainability of system

③ GOI. (Guidelines / suggestions regarding development)

→ Develop the s/w in such a way that future Goals are also meet.

- Properties of Good SRS

① Complete
② Consistent
③ Unambiguous
④ Verifiable
⑤ Modifiable
⑥ Testable
⑦ Relevant
⑧ Understandable & Readable

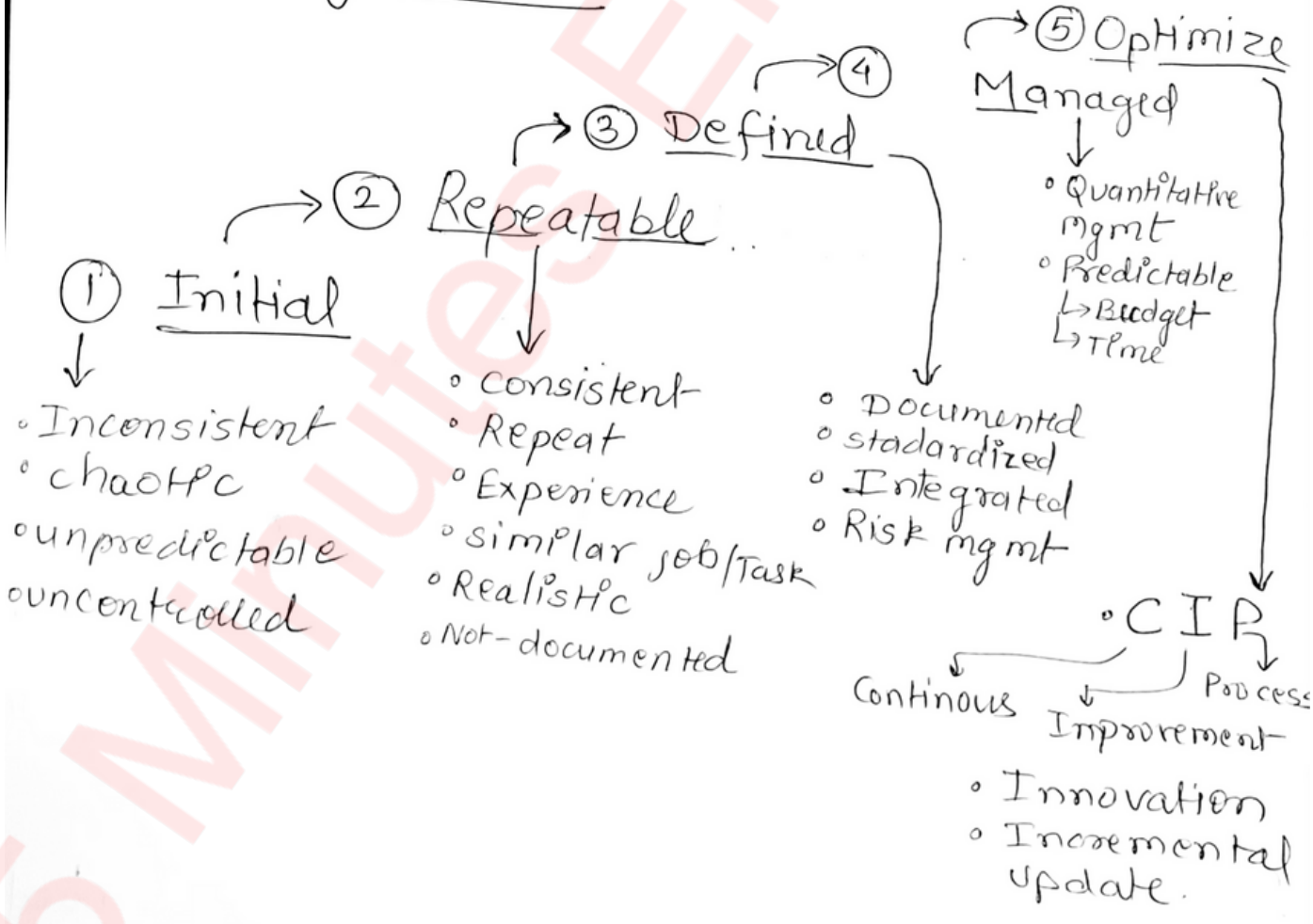
- Without SRS

→ Wrong system developed
→ Doubts at every step
→ Difficulty in flow understanding.
→ Time & Budget waste.

- Capability maturity model (CMM)
  - → Improve / Enhance software Process
    To get best/optimal Quality s/w

"Dhoni's mindset"
- → focus on process
  as depends on process

- Maturity Levels



- Software design → I/p : SRS
  → o/p : S/w design doc
  - → Interface Design
  - → Architectural Design
  - → Detailed Design

- Good S/w design doc characteristics
  - → ① Must have all client Req. as per SRS
  - → ② Must be free from conflicts, ambiguity and incorrectness.
  - → ③ Must explain each & every feature of s/w clearly.

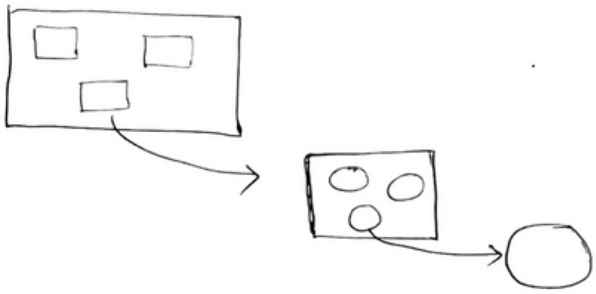
(I) Interface Design : Interaction betn System & environment
- ⇝ System Internal is ignored
- ⇝ Just focus on I/p to sytem & final o/p.

(II) Architectural Design : components of sytem
- ⇝ their interfaces, responsibilities
- ⇝ Interaction betn them
- ⇝ Internal of components are ignored.

(III) Detailed Design

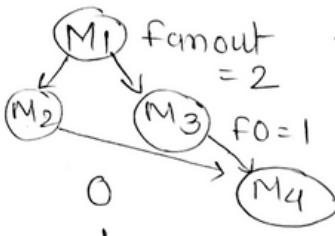
~> focus is on the internal elements of all components of the system

~> Internal Algorithms & data structure



• Modularity:

Module { functions | Attributes }



- Independent
(Complexity is Reduced)
(Its easy to manage)
(Not much)

0 ——————————— n

← Degree of Modularity →

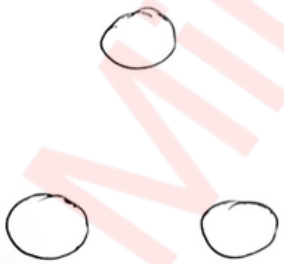
(inter Module) Coupling ↓ low

(intra Module) Cohesion ↑ high

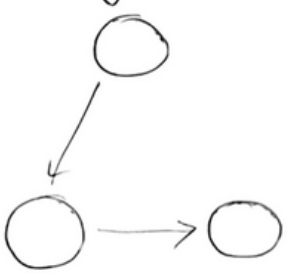
war

Parameters using which we try to understand the quality of modularity in design

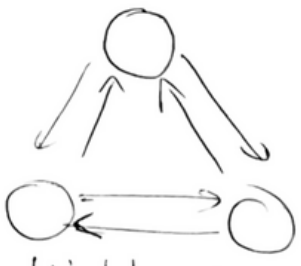
① Coupling: The measure of Interdependence of one module on other.



uncoupled



Loosely coupled



Highly coupled.

Types of Coupling.

- Data (Optimal / Best)
- Stamp
- Control
- External
- Common
- Content (worst)

① **Data**: Data passing communication in betn 2 modules. (Call value)

② **Stamp**: Communicate / interact by passing data structure (Call by reference)

③ **Control**: Control info or logic or statements One module is trying to control functionality of other.

④ **External**: Coupling in modules on external matter like H/W, CPU, Buses.

⑤ **Common**: Shared / common data. [Critical section, race condition (OS)]

- **Content**: Subset / part / portion.
  when one module is a part of other module.
  → Data can be modified.
  Its worst.

* **Cohesion**: The measure of the functions steingth.
  (Intra module).

Types of Cohesion

- function (High / Best)
- sequence
- Communicational
- Procedural
- Temporal
- logical
- Coincidental (low / worst)

① Functional: different fn's come together Collaborate & work to perform a single task/fn in a Module.

② Sequential: $f_1 \rightarrow f_2 \rightarrow f_3$
- One's O/p is I/p for other
- flow & control dependency.

③ Communicational: 2 fn operate on same I/p data / data structure

④ Procedural: flow control, it is part of a procedure to be followed.

⑤ Temporal: (Time) A module having temporal C, all the fn must be executed in same time span.

⑥ Logical: Similar operation performed. (logically related) (functionally different)

⑦ Coincidental: Random or some kind of coincidence (worst).

Design Approach

- Top down
- Bottom Up

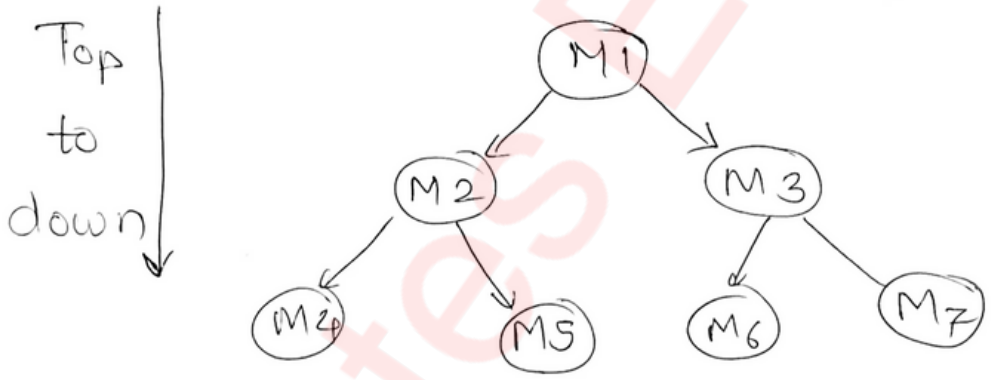
Top down design approach

→ Divide & conque.

→ Dividing a big problem into small problem further more repeatedly till we solve it easily.

→ You need to have entire problem requirements Problem statement

→ Small or medium size project.



Bottom Up degin approach

→ Initially no need to have understanding of entire s/w req. / product req.

→ Go from low level fn / modules to higher ones.

→ Eg: Engineering exam last night study.

- Estimation

→ Project Planning Process
- Before development
- focus on activities req. for successful completion of project.

- Estimating
  - Size of Project
  - Cost
  - Time / Duration
  - Effort.

Precise & accurate Estimates → leads to → successfull completion or mgmt of Project.

for cost estimation
- size of s/w
- H/w
- Tools & licenses
- skilled personnel
- Travel & communication
- Training & support.

Resource Mgmt

- Human resources
- Tools
- libraries of s/w
- H/w

Available Resources ⟷ Balance Mgmt. ⟷ S/w Project Demand

① Imp / Immediate Resources at each phase must be determined. to make them available.

② Resoure requeston demand & de allocating them whey they are no lo gee required.

③ Reuse the S/w Resources

- Requirements
- Design
- codl
- Test cases.

Estimation Techniques

- Empirical
- Heuristic
- Analytical

(1) Empirical :

- Guesses or Assumption
- Prior experience on similar projects
- historical/previous product/project Data.
- Common sense based.
- Expert Judgment [one or pair]
  - Delphi cost estimation. [Group or team work]

(2) Heuristic

- COCOMO → Constructive cost model.
  - Mathematical eqns involving the project parametes
  - Independent parametek first. then dipendent parametek.

→ COCOMO

- Primary factor : size of project
- (✓) Line of code (LOC) = size/no of lines
- (x) function Point = features/functions of S/W.

3 categories

① organic : small size, simple, less innovate

② semi detached : medium size, difficult, medium innovate

③ Embedded : Large size, Very complex, most creative & innovative.

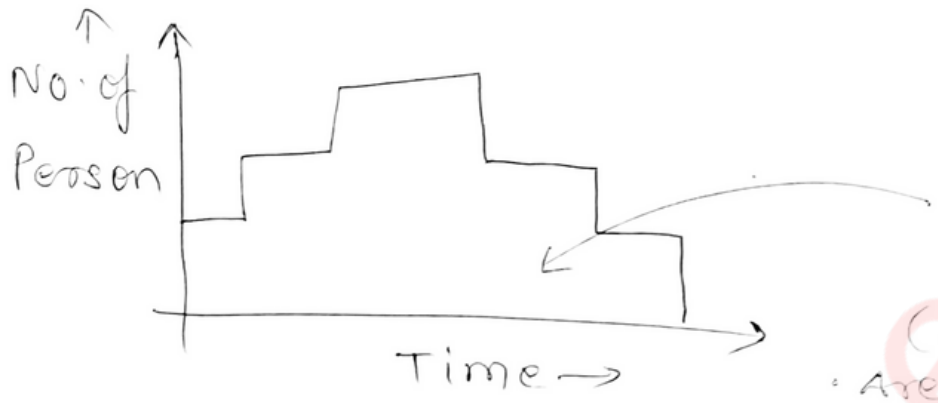
* Basic Cocomo

↳ focus on 'size' (kLOC)
↑ kilo

Time → $Effort = a_1 (KLOC)^{a_2}$ PM

$T_{dev} = b_1 \times (Effort)^{b_2}$ months ↑ Person month

- effort is expressed in units of PM.
estimation



Person month curve/plot

So, effort estimate a_1 → a_2 ↓

- organic : $2.4(KLOC)^{1.05}$ PM
- semi-detached : $3.0(KLOC)^{1.12}$ PM
- Embedded : $3.6(KLOC)^{1.20}$ PM

2. Time estimate b_1 → b_2 ↓

- organic : $2.5(E)^{0.38}$ Months
- sem-detached : $2.5(E)^{0.35}$ Months
- Embedded : $2.5(E)^{0.32}$ Months

Q:- Project type: semi-detached.

LOC = 50,000

= 50 KLOC

∴ ① Effort estimate

$= a_1 (KLOC)^{a_2}$

$= 3.0 (50)^{(1.12)}$

= 239 PM

② $T = b_1 (E)^{b_2}$

$= 2.5 (239)^{0.35}$

= 17 months.

③ Productivity (P) $= \frac{KLOC}{E}$

$= \frac{50}{239}$

$= 0.209$

° Intermediate COCOMO

- Basic cocomo assumes that (E) & (T) are fn of size done.
- But besides size other project parameters.

eg: LOC = 30,000
Type = Embedded.
Parameter: Reliability must be high

- Product Attr
- Project Attr
- Personal Attr
- Computer Attr.

(very low, low, normal, high, very high)

So modified formula $= a_1 (KLOC)^{a_2} \times (EAF)$ (Effort Adj. Factor)

$= 2.8 (30)^{1.20} \times 1.15$

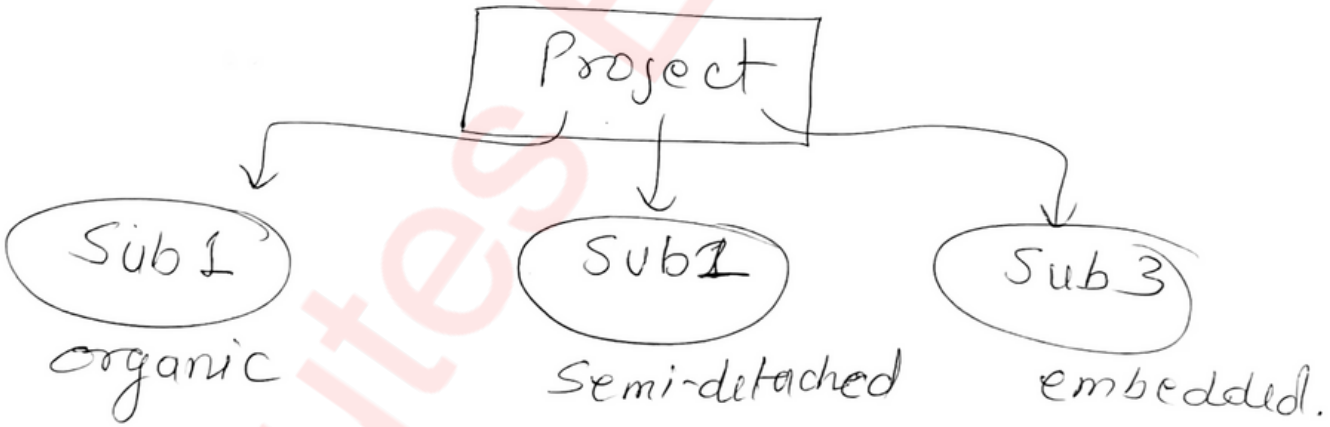
$= 191$ PM

$T = b_1 (E)^{b_2} = 2.5 (191)^{0.32} = 13$ Month

Complete COCOMO

→ Intermediate COCOMO considers a s/w project as a single homogeneous entity.

→ But large projects/products/systems are made up of several small sub-systems.



E: Estimate of sub-systems.

$E(Sub1) + E(Sub2) + E(Sub3)$

- **Software Testing**
  - To find Bugs / faults / Errors in Code.
  - With min resources, cost & time we can Identify max bugs.

Verification	Vs	Validation
1st		2nd
"Are we building the product / thing right"		Are we building the right product / thing.
Prevent errors		Detect errors
Phase by Phase		final Product.
By Developer		By Testers
Includes Unit testing & Integration testing & Reviews		Includes Black Box & White Box testing.
Static nature		Dynamic nature

- Principles of S/W Testing
  - → Identify as many bugs as possible
  - → Early testing saves your resources
  - → start with a proper test plan
  - → As per user requirements
  - → focus more on heavy Bug Areas.
  - → Go for functional as well as Non-functional Testing.

- Unit Testing : Each Source code Module is Tested.
  - → Hence also called as module testing.
  - → Early phase testing.

- Integration Testing
  - → Incremental (Top-down & Bottom Up)
  - → Non-Incremental. (Big Bang)
  - → Interaction, connectivity, & Data flow betn unit tested modules

- System Testing
  - Whole / Complete System
  - Don't focus on internal work
  - focus on I/p & O/p
  - Black Box Testing
  - Verifies the overall functionality of system.

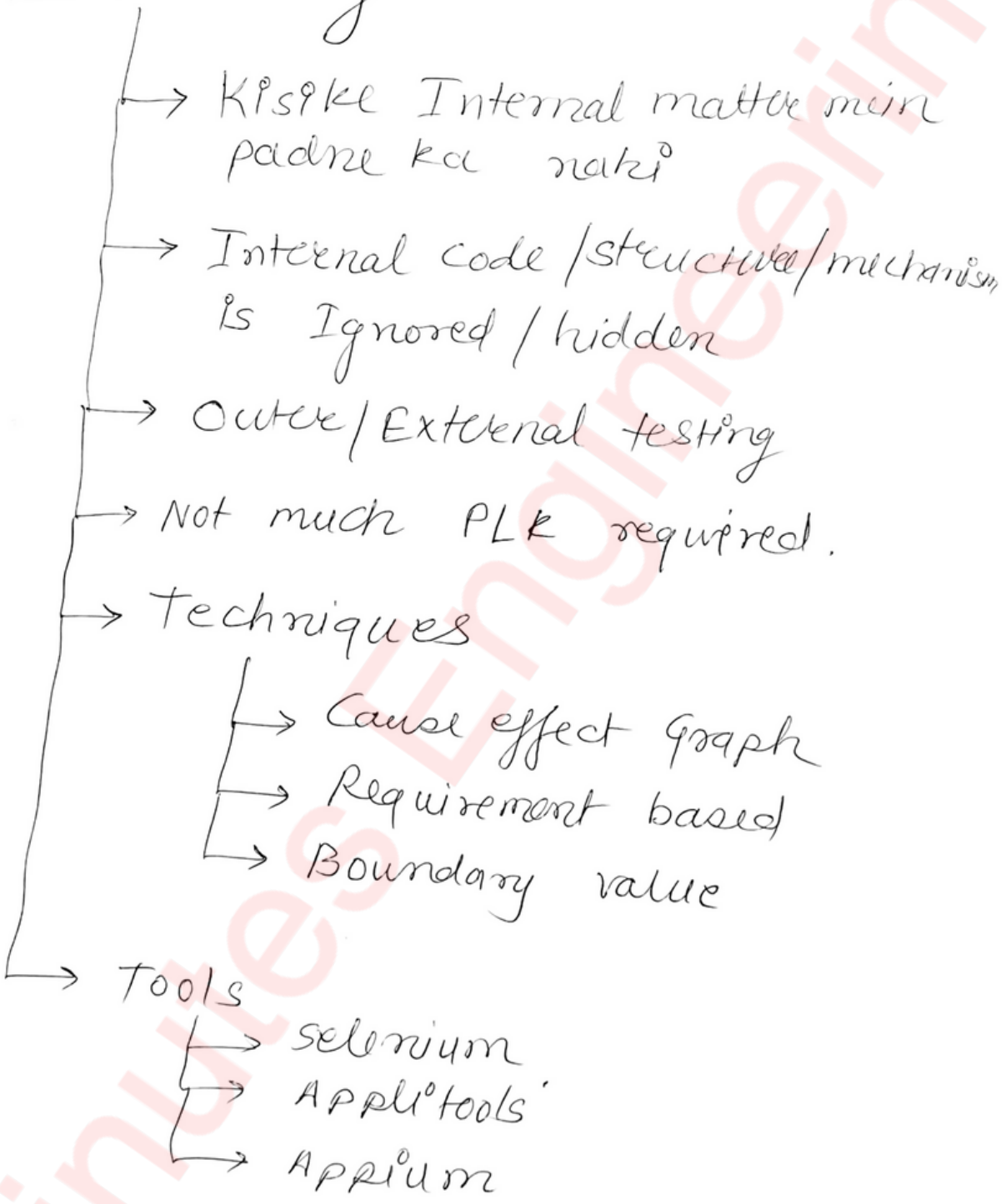
- Acceptance Testing (UAT) α, β
  - final Testing phase Before s/w deployment
  - focus on ease of use for Best user experience
  - performed by clients or endusers.

→ Performance Testing
→ Load Testing
→ stress Testing
→ Scalability Testing

} Non functional

- White Box Testing
  - Glass, transparent & clear Testing.
  - focus on internal structure Testing.
  - Used / applied on unit & integration level testing.
  - Techniques
    - statement coverage
    - Branch coverage
    - Dataflow testing
    - Path coverage.
  - Tools : Sqlmap, Nunit, VeraUnit

- Black Box Testing



- Software Maintanance
  - Starts once the S/w product is deployed & users are using it.
  - Update / Upgrade / improve.
  - Bug fixing
  - Optimization
  - Light weight
  - delay / lag elimination
  - It covers majority time / efforts in SDLC.
  - Types
    - fixing / corrective
    - Perfective
    - Adaptive
    - Preventive

- Software Re-Engineering
  - Examine & Update the existing s/w to enhance its performance.
  - Something more than maintenance.
  - Reverse / Backward Engineering.
    Product code → module/ Component → Design → Requirement.
  - Restructuring.
    - Preserve the core function But reorganize/refactor the existing code to make it more optimal to use & understand.
  - forward Engineering
    - follow the normal SDLC for new/improved Requirements

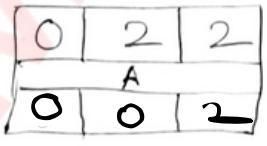
- Critical Path method:
  - Critical task [Can't be delayed]
  - Critical Path
    - Min. time required to complete the project that path covering the critical task.

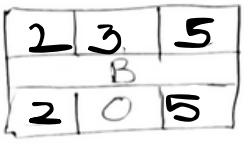
→ finding critical Path.
- Identify
- Sequencing.
- Estimation
- Visualization / Diagram
- selection.

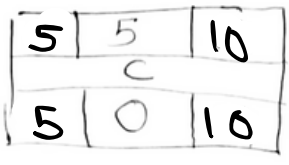
Eg:

Task	Predecessor	Time / Duration
A	—	2
B	A	3
C	B	5

Float = LF - EF
LS = LF - D







- **Regression Testing**
  - ↳ Bug fixing or Updation or new feature add ons
    - Do not change the core system function/natwel.

→ full RT : Test the entire system after update.

→ Partial RT : Test only that module wheel update is done.

Risk Management

→ • Problem → Uncertain & Impact/loss
  • event
  • Condition

→ Identification
→ Analysis
→ Planning
→ monitoring.
→ Resolution

} Risk Assessment
} Risk control.

• Reactive & Proactive RM

↓ React to Risk/problem
"Kya gadbad hogayi"
→ Problem occured
→ Risk that has ↑↑↑ high chances to occure

↓ Prevent the Risk
"kya gadbad hosakti hai".
→ future Risk
→ Prevention is better than cure.

Types of Risk

- Budget Risk
- Schedule / Time Risk
- Technical Risk
- Operational Risk
- Business Risk.

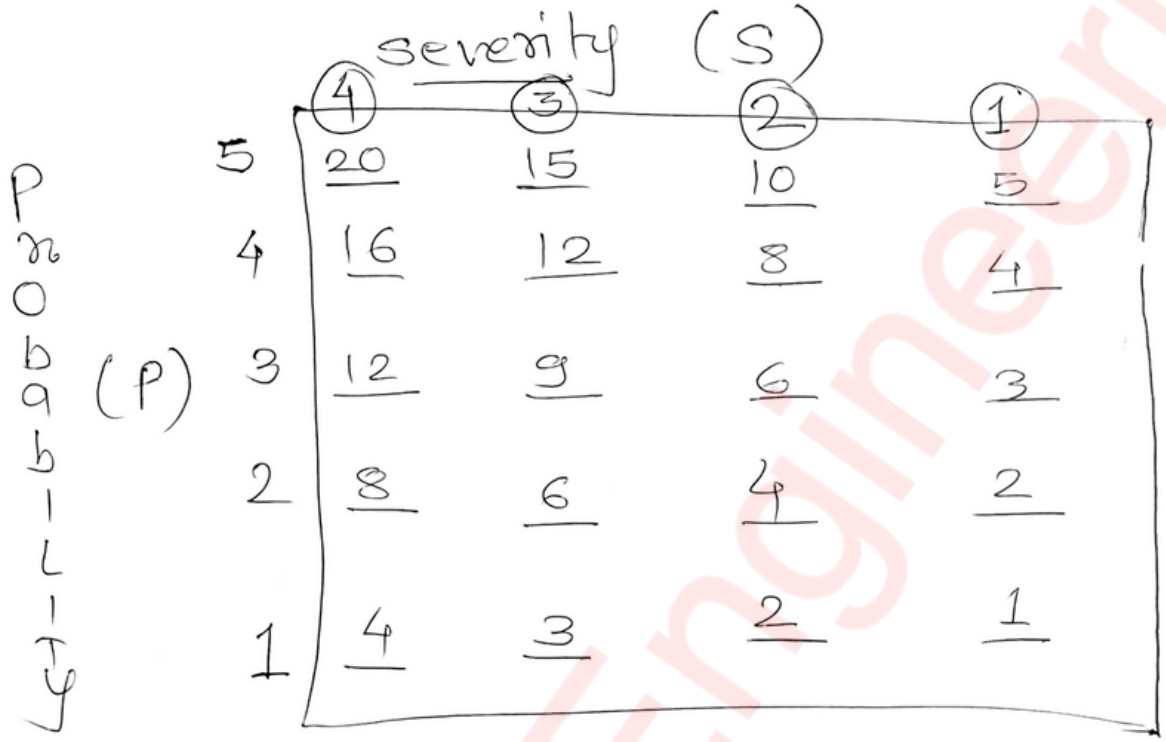
- **Risk Assessment**
  - Identify, analyse & plan or prioritize the risks.

↑ more damage → ↑ High priority (severity)

↓ less damage → ↓ less priority.

Risk → frequently occuring (Probability of occurence)
Rare occurence.

- Impact Assessment Matrix.



- Risk Control
  - → Try to Reduce the risk occurence probability by some actions.
  - → PLAN (A) PLAN (B) PLAN (C).
  - → Avoid, Transfer or Reduce.
  - → Monitoring is a process that is carried throughout the project. (Continuous evaluation.
  - → Risk mitigation: Try to Reduce the impact of Risk on system.