Page No
Date

* Software engineering

① Introduction
② software development life cycle **
③ Requirement analysis (SRS)
④ software project managment ** (cocomo)
⑤ software design (coupling, cohesion, UML, DFD, class diagram)
⑥ coding and testing ***
⑦ Maintenance
⑧ Quality managment, Reuse

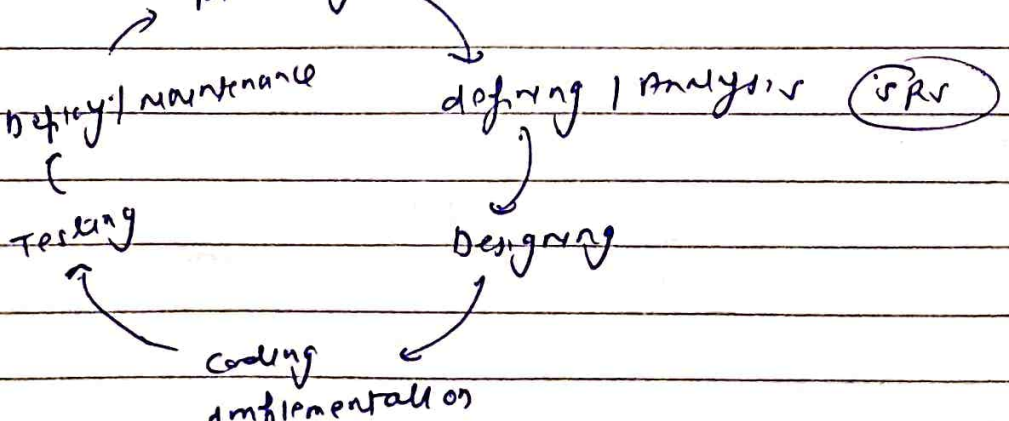
* software engineering, Definition and evolution

① It is systmatic, disciplined, cost effective techniques for software development
② Engineering Approach to develop a software
③

Evolution:

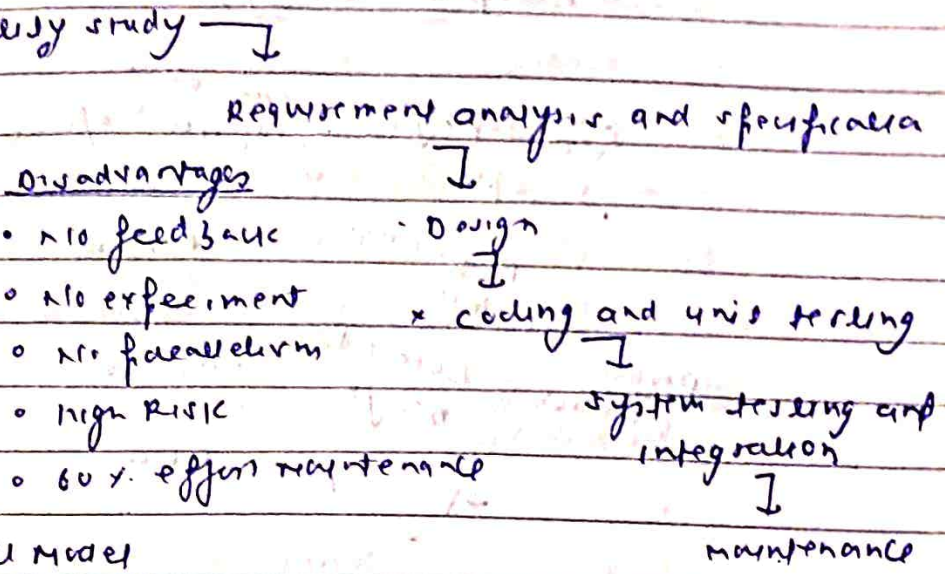
- 1945 - 1965 → origin
- 1965 - 1985 → crisis (low project → 2 work, 50 network work)
- 1985 - 2000: internet
- 2000 - 2010: Light weight
- 2010 - Till: AI, ML, DL, Cloud

* Software development



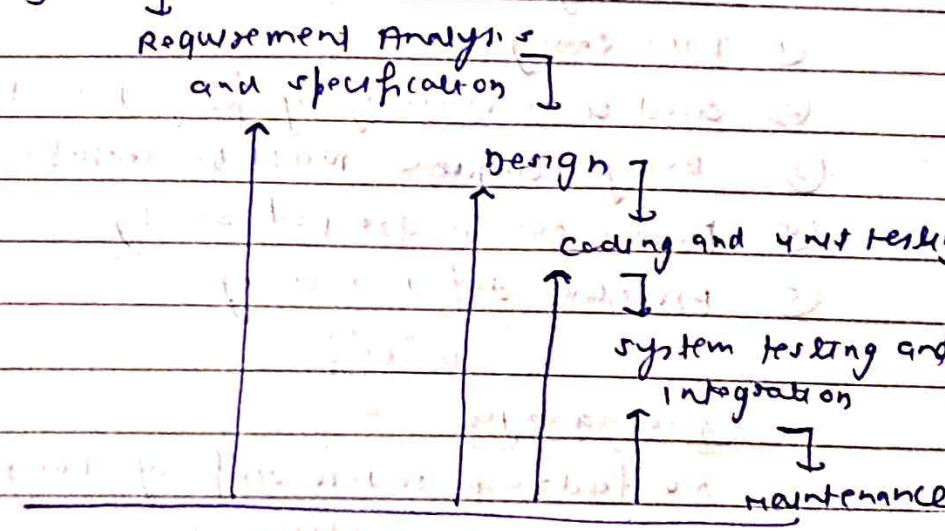
Page No.
Date

(1) Classical waterfall Model



Advantages	Disadvantages
(1) • Base Model	• No feedback
(2) simple and easy	• No refinement
(3) small project	• No parallelism
	• High Risk
	• 60% effort maintenance

(2) Iterative waterfall Model



Advantages

(1) Base Model
(2) simple and easy
(3) small project
(4) feed back.

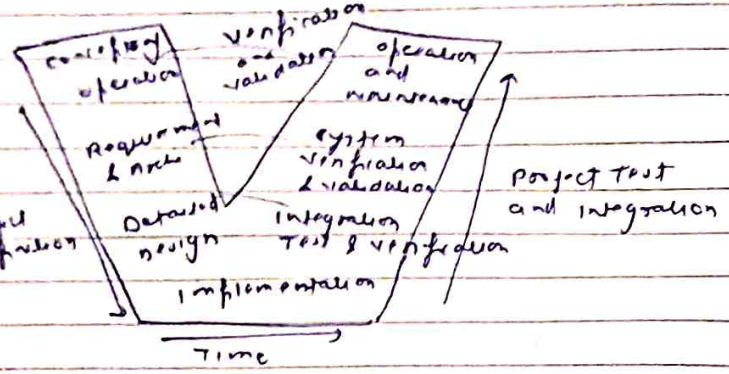
Disadvantages

(1) No phase overlapping
(2) No intermediate delivery
(3) Rigid (No change)
(4) less customer interaction

(3) V-shaped Model :- Also known as Verification and Validation Model

- Extension of waterfall Model
- Testing is associated with every phase of life cycle
- Verification phase (Requirement Analysis, system design, Architecture Design, Module Design)

- validation focus (unit testing, integration, system, acceptance testing)



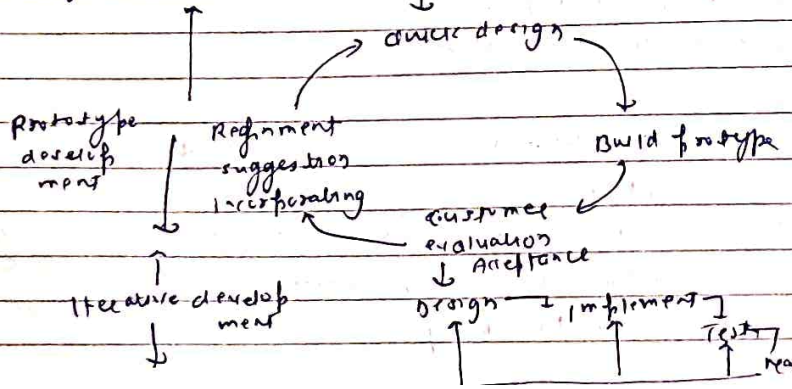
Advantages

1. Time saving
2. Good understanding of project in the beginning
3. Every component must be testable
4. Progress can be tracked easily
5. Proactive defect tracking

Disadvantages:

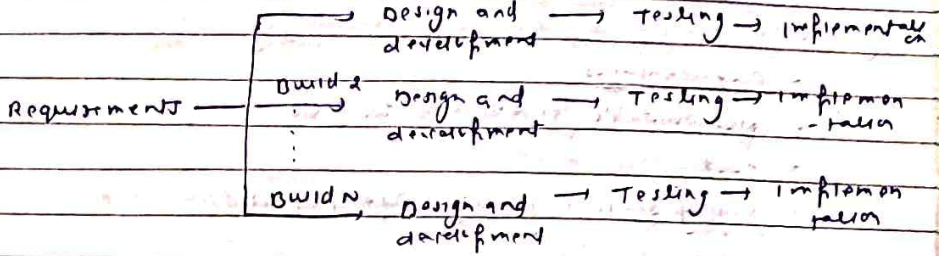
1. No feedback so less scope of changes
2. Risk Analysis not done
3. Not good for big or object-oriented projects

* Prototype Model:



* Customer not clear with idea
* Throwaway Model
* good for technical and Requirement risk
* Increase in cost of development

Incremental Model:



- Module by module working
- customer interaction maximum
- Larger the project
- Early release project demand
- flexible to changes

Evolutionary Model: - is a combination of Iterative and Incremental model of software development life cycle.

→ Incremental model first implements a few basic features and delivers to the customer. Then build the next phase and deliver it again and Repeat the steps until the desired system is fully Realized. No long term plans are made

→ Iterative model main advantage is its feedback process in every phase.

→ Also known as "design a little, build a little, test a little, deploy a little model"

- Advantage
- customer satisfaction are clearly specified
- Risk Analysis is better
- It supports changing environment
- initial operating time is less
- Better suited for a large mission - critical projects

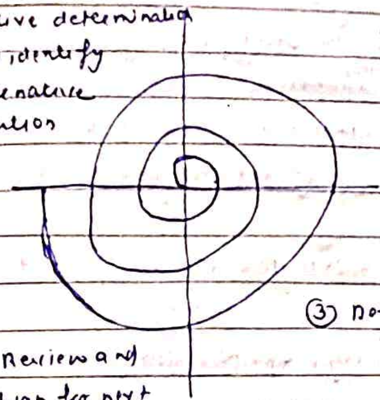
- Disadvantages
- Not suitable for smaller projects
- cost
- highly skilled resources are required

Rough Requirements specification
↓
Identify the core and the other parts to be developed incrementally
↓
Develop the core part using an iterative waterfall model
↓
Collect customer feedback and modify Requirements
↓
Develop the next identified features using an iterative waterfall model → Delivery of the next version of the complete
↓ All feature complete
Maintenance

- Spiral model : Jha pe bhi Risk likha aa jaye samaj lo its an spiral model

① Objective determination and identify Alternative solution

② Identify and resolve Risk



③ Develop next version of product

④ Review and plan for next phase

* Risk handling
* Radius of spiral = cost
* Angular dimension = progress
* Meta model

Advantages:
1) Risk handling
2) Larger projects
3) flexible
4) customer satisfaction

Disadvantages
① complex
② Expensive
③ Too much risk analysis
④ Time

Agile (more quickly)

Large projects
↓
small chunks (iteration)
↓
Release
↓
feedback
↓
enhance
↓
Re-Release

Advantages: 1) frequent Delivery
2) face to face communication
3) changes
4) Time

Disadvantages : less documentation
: Maintenance problem

- **Advantage**
- customer satisfaction are clearly specified
- Risk analysis is better
- It suffers changing environment
- initial operating time is less
- better suited for a large mission - critical projects

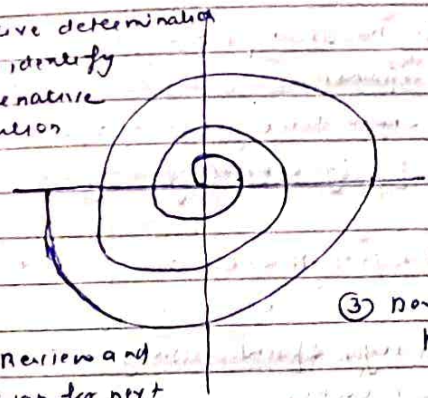
- **Disadvantages**
- Not suitable for smaller projects
- cost
- highly skilled resources are required

Rough Requirements specification
↓
Identify the core and the other parts to be developed incrementally
↓
Develop the core part using an iterative waterfall model
↓
collect customer feedback and modify requirement
↓
Develop the next identified features using an iterative waterfall model → Delivery of the next version of the complete
↓ All features complete
Maintenance

- **Spiral model** : Jha pe bhi risk likha aa jaye means it is an spiral model

① objective determination and identify alternative solution

② identify and resolve risk



③ Develop next version of product

④ Review and plan for next phase

- Risk handling
- Radius of spiral = cost
- Angular dimension = progress
- Meta model

Advantages:
1) Risk handling
2) Larger projects
3) flexible
4) customer satisfaction

Disadvantages
① complex
② Expensive
③ Too much risk analysis
④ Time

Agile (more quality)

large projects
↓
small chunks (iteration)
↓
Release
↓
feedback
↓
enhance
↓
Re-Release

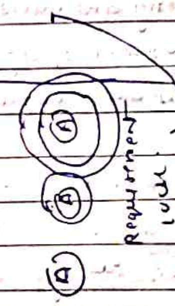
Advantages:
1) frequent Delivery
2) face to face communication
3) changes
4) Time

Disadvantages : less documentation
: maintenance problem

- Scrum: one of the most software Agile methodology
- scrum is a light weight, iterative and incremental framework
- scrum breaks down the development phase into stages or cycle called "sprints".
- The development ~~team~~ time for each sprint is maximised and dedicated, thereby managing only one sprint at a time
- scrum Team has scrum master and product owner with constant communication on the daily basis
- Keywords: Backlog, sprint, Daily scrum, scrum master, product owner

Advantages: ① freedom and Adaptation
② high quality, low risk product
③ Reduce the development time up to 40%
④ scrum customer satisfaction is very important
⑤ Reviewing the current sprint before moving to new one

Disadvantages: ① more efficient for small team size
② No changes in the sprint

Classical waterfall	Iterative waterfall	Prototype waterfall Model	Incremental Model	Evolutionary Model	RAD Model	Spiral Model	Agile Model
Basic, Rigid, inflexible, Not for Real Project	Basic: problem is well defined and understood	user Requirement not clear, costly, No early lock on Requirements, → highly user involvement, → Reusability	Module by Module delivery, Easy to test and de-bug  Requirement lock	Larger project	Time and cost constraints, → user at all levels, → Reusability	→ Risk, → Not for small project, → No early lock on Requirement, → less Experience can work	flexible, Advanced, parallel. Process divide into sprints

Page No.
Date

* Software Requirement: It is the description of features and functionalities of the target system
→ It is the description of what the system should do
→ Requirement engineering (RE) Refers to the process of defining, documentation, maintaining Requirement in the engineering design process
→ It is a four steps process which includes:
(1) feasibility study
(2) Requirement gathering and elicitation
(3) software Requirement specification
(4) software Requirement validation.

* Tools suppliers for Requirement engineering
→ observation Reports (users observation)
→ Questionnaires (interviews, surveys, and polls)
→ use cases
→ Requirement workshop
→ Mismapping
→ prototyping

* Functional Requirement: Requirement which are Related to functional / working aspects of software fall in to this category
→ Non-functional Requirement are reflected characteristics of target software; (security, storage, configuration, performance, cost, interoperability, flexibility, Disaster, privacy, Accessibility)

→ software Requirement specification: is a description of a s/w system to be developed. It lays out functional and non-functional Requirement of the software to be developed

Page No.
Date

→ It may include a set of use cases that describe user interaction that the s/w must provide to the perfect interaction.

SRS structure;

1 Introduction
- 1.1 purpose
- 1.2 Intended Audience
- 1.3 Scope
- 1.4 Definitions
- 1.5 References

2 overall description:
- 2.1 user interfaces
- 2.2 system interfaces
- 2.3 constraints assumption, and dependencies
- 2.4 user characteristics

3 system features and Requirements
- 3.1 functional Requirement
- 3.2 use cases
- 3.3 External interface Requirement
- 3.4 Logical db Requirement
- 3.5 Non-functional Requirement

4 delivered for Approval.

* User Requirement with Real life example.
- easy and simple to operate
- quick response
- effectively handling operational error
- customer support

* User Requirement specification :- The (URD) or (URS) is a is a document usually used in software engineering that specifies what the user expects the s/w to be able to do

→ It is a contractual agreement

Bubble chart

* **Data flow diagram**
  - a graphical tool useful for communicating with users, managers and other personnel
  - useful for analyzing existing as well as proposed systems
  - focus on the movement of data b/w external entities and processes, and b/w processes and data stores.
  - a relatively simple technique to learn and use.

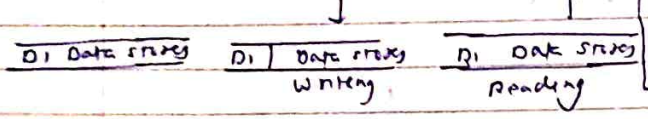
* **DFD elements**:- used
  - source / sinks (external entities) ▭
  - Data flow ⇄
  - process ◯
  - Data store
    ↓
    DB

* **External entities**: ▭
  - either supply or receive data
  - They do not process data

① **Source**: Entity that supplies data to the system

② **Sink**: Entity that receives data from the system

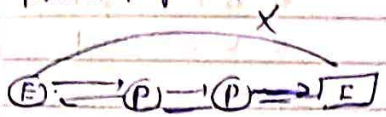
* **Data stores.**



- A data store is a Repository of data
- Data can be written in to a data store. This is depicted by an incoming arrow
- Data can be Read from a data store. This is depicted by an outgoing arrow.
- External entity can't Read or write to the data store

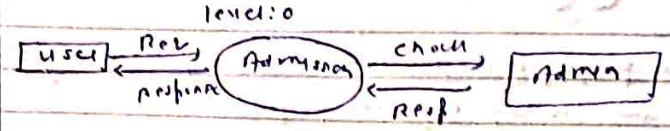
* **Rules of data flow:**
  - Data can flow from:
    - External entity to process
    - process to external entity
    - process to store and back
    - process to process

  Data can't flow from
  - External entity to external entity
  - External entity to store
  - store to external entity
  - store to store.



* **DFd for university admission system:**

level: 0



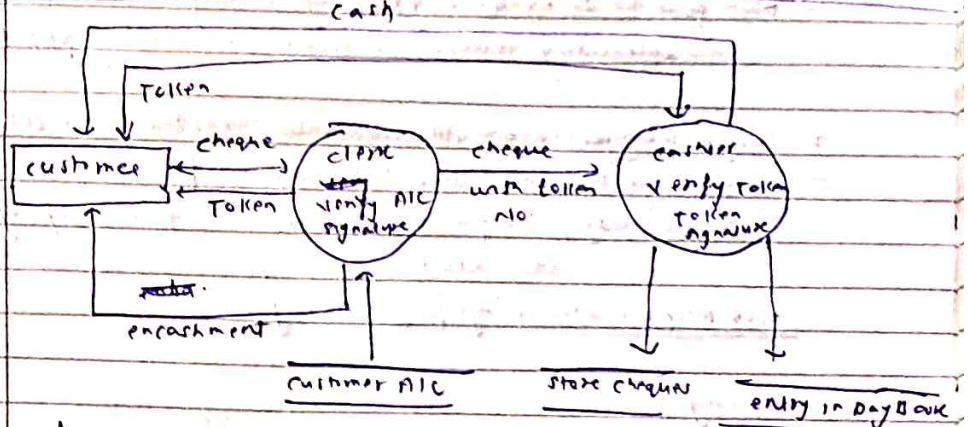


fig: Physical DFD for cheque encashment

① Which flexibility among the following is invalid in case of a data flow diagram

(a) A process having in-bound data flows more than out-bound data flow

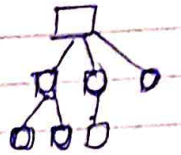
(b) A data flow b/w two processes

(c) A data flow b/w two data stores

(d) A data store having more than one in-bound data flows

* software design approaches:

functional oriented design	object oriented design
• system is designed from a functional view point	→ system is viewed as a collection of objects (i.e entities)
→ Top down decomposition	→ Bottom up approach
- divide and conquer approach	→ UML is used
→ DFD is used	



• Software project management: is an art of science of planning and leading software projects

→ Main goal is to enable a group of developers to work effectively towards successful completion of project

→ Project manager is an administrative leader of the team

→ various factors make this job very complex. (eg ~~chaotic~~ changeability, complexity, uniqueness, possibility of multiple solution etc)

Job Responsibilities of project manager:

- planning
- organizing
- staffing
- directing
- monitoring
- controlling
- innovating
- delegating

* skilled required for project manager:

- Managerial skills
- Technical skills
- Problem solving skills
- coding skills
- conceptual skills
- leadership skills
- communication skills

Project planning:

- Estimation (cost, duration, effort)
- staffing (staff organization, staff plans)
- scheduling manpower and other resources
- Risk management
- miscellaneous plans (quality, assurance plan, configuration and installation plans)

→ Risk management: A risk is an uncertain event or condition that, if it occurs has a positive or negative effect on a project objective

→ A Risk management plan is a document that a project manager prepare as to foresee risks, estimate impacts and define responses to risks

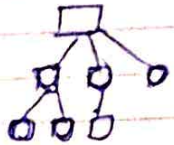
* Reactive vs proactive Risk management strategies

→ Reactive Risk management tries to reduce the damage of potential threats and speed an organization's recovery from them, but assumes that those threats will happen eventually

→ Proactive Risk management identifies threats and aims to prevent those events from ever happening in the first place

* software design approaches:

Functional oriented design	Object oriented design
• system is designed from a functional view point	→ system is viewed as a collection of objects (i.e entities)
→ Top down decomposition	→ Bottom up approach
• divide and conquer approach	→ UML is used
→ DFD is used	



• Software project management: is an art of science of planning and leading software projects

→ Main goal is to enable a group of developers to work effectively towards successful completion of project

→ Project manager is an administrative leader of the team

→ various factors make this job very complex - (eg changeability, complexity, uncertainty, possibility of multiple solutions etc)

Job Responsibilities of project manager:

- planning
- organizing
- staffing
- Directing
- Monitoring
- controlling
- Innovating
- Representing

- Skilled Required for project manager:
  - Managerial skills
  - Technical skills
  - Problem solving skills
  - coding skills
  - conceptual skills
  - leadership skills
  - communication skills

Project planning:

- Estimation (cost, duration, effort)
- staffing (staff organization, staff plans)
- scheduling manpower and other Resources
- Risk management
- miscellaneous plans (Quality, Assurance plan, configuration and installation plans)

→ Risk management: A Risk is an uncertain event or condition that, if it occurs has a positive or negative effect on a project objective

→ A Risk management plan is a document that a project manager prepare as to foresee risks, estimate impacts and define responses to risks

x Reactive vs proactive Risk management strategies

→ Reactive Risk management tries to reduce the damage of potential threats and speed an organization's Recovery from them, but assumes that those threats will happen eventually

→ Proactive Risk management identifies threats and aims to prevent those events from ever happening in the first place

- Types of Risk:
- Business Risk: Building a product that no one wants or loosing budgetary commitments
- Technical Risk: - concerned with quality, design, implementation interface, maintenance problems
- project Risk: concerned with schedule, cost, Resources, customer-related issues

Ex: • In 2005 Denver International Airport set out to create the most sophisticated luggage handling system in the world. The project was soon deemed to be far from more complex than any one who anticipating and delayed for 16 months at a cost of $560m → project Risk

→ ford was Ready to Release the edsel, the market had already moved on to compact cars and edsel was a flop
↳ Business Risk

→ on January 28, 1986, the challenger space shuttle exploded just 76 second after take off
↳ TR

* Risk Assessment: RM — RA = Identify, Access, planning
RC

→ It is a process to Rank the Risk in terms of their damage
→ Determine the Average probability of occurence value for each Risk.
→ Determine the impact for each component based on impact Assessment matrix (severity value).
→ Risk Assessment values are determined by multiplying the score for the probability and severity value together

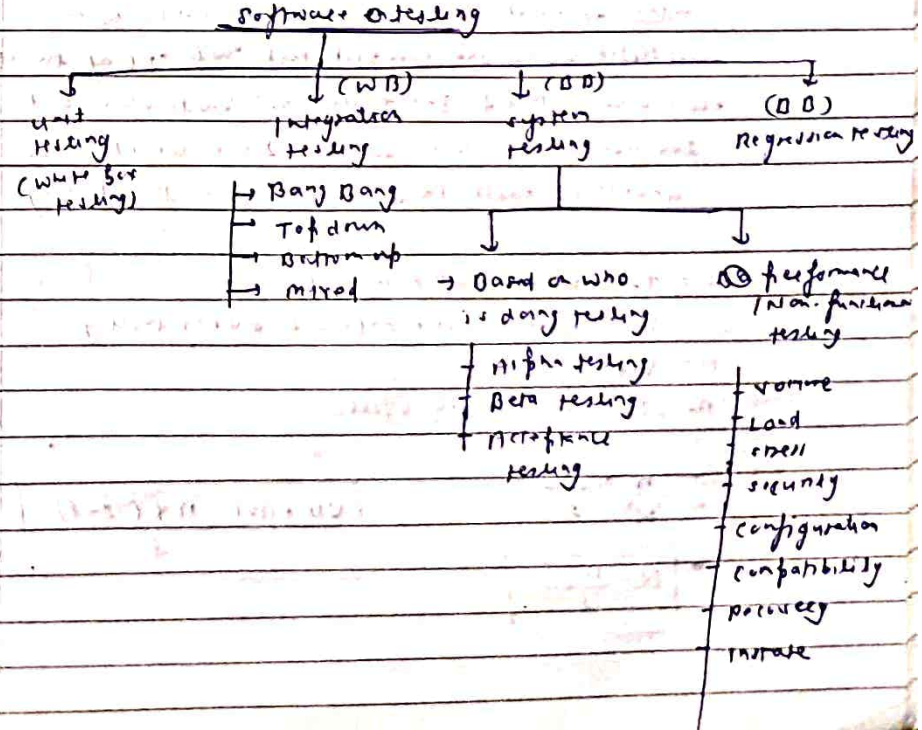
IRA: PXS

Risk Assessment Steps:

* Risk control vs Risk mitigation:

→ Risk control is basically the specific Action to Reduce a Risk event's probability of happening. for ex: correct inspection and maintenance of the car Reduce the likelihood of Mechanical failure
→ whereas Risk mitigation is the set of Actions to Reduce the impact of a Risk event. For ex., Air bags are used to Reduce the impact of An accident on the passengers and driver

* Risk mitigation and control strategies:
- Risk Avoidance
- Risk Reduction
- Risk transfer
- Risk monitoring



*	Verification	Validation
(1)	Are you building it right?	(1) Have you built the right thing?
(2)	Check whether an artifact is from your artifact	(2) Check the final product against specification
(3)	Done by developer	(3) done by tester
(4)	concerned with final containment of error	(4) aim is to make a final product error free
(5)	Methods involved Review, inspection, unit testing and integration testing	(5) involved system testing
(6)	static and dynamic activities	(6) only dynamic

Q A software was testing using the error seeding strategy in which 20 error were seeded in the code. when the code was tested using the complete test suite 16 of the seeded error were detected. The same test suite also detected 200 non-seeded error. what is the estimated no. of undetected error in the code after the testing.

Sol: S = Total errors seeded in code
s = Total seeded error detected during testing
N = Total errors
n = Total non-seeded error

$$\frac{n}{N} \times \frac{s}{S}$$

$$N = \frac{n \times S}{s}$$

$$\text{UD error} = \frac{n \times (S-s)}{s}$$

S = 20
s = 16
n = 200

$$N = \frac{n \times S}{s}$$

$$= \frac{200 \times 20}{16} = \frac{4000}{16}$$

$$= 250$$

∴ Undetected error: $\frac{n \times (S-s)}{s}$

$$= \frac{200 \times (20-16)}{16}$$

$$= \frac{200 \times 4}{16}$$

$$= 50$$

Q Perfect risk factor is considered in?
(a) spiral model (b) waterfall model (c) prototyping model
(d) Iterative model

Q The no. of function points to by a proposed system is calculated as 500. suppose the system is planned to develop in java if so estimate the effort (E) required to complete the project using effort formula of basic cocomo model. is given as: $E = a(KLOC)^b$. assume a and b are 2.5 and 1.0

Sol: Size estimation → LOC / FP
↓
effort
↓
cost, schedule time

$$\frac{LOC}{FP} = 50$$

$$\frac{LOC}{500} = 50$$

LOC = 25000

LOC = 25K

$$E = a(KLOC)^b$$

$$= 2.5(25)^{1.0}$$

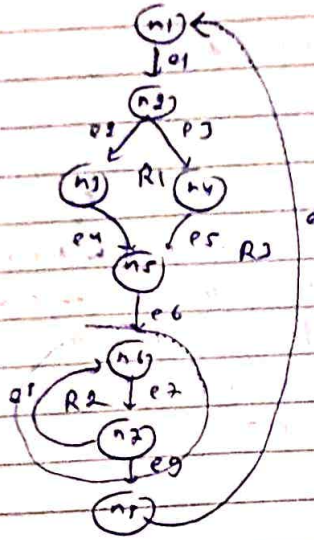
$$= 62.5$$

Q1 Which of the following is white box testing techniques?

(a) path coverage (c) cause-effect graphing

(b) equivalence class testing (d) state based testing

Q2 find the cyclomatic complexity of following flow graphs of software code?



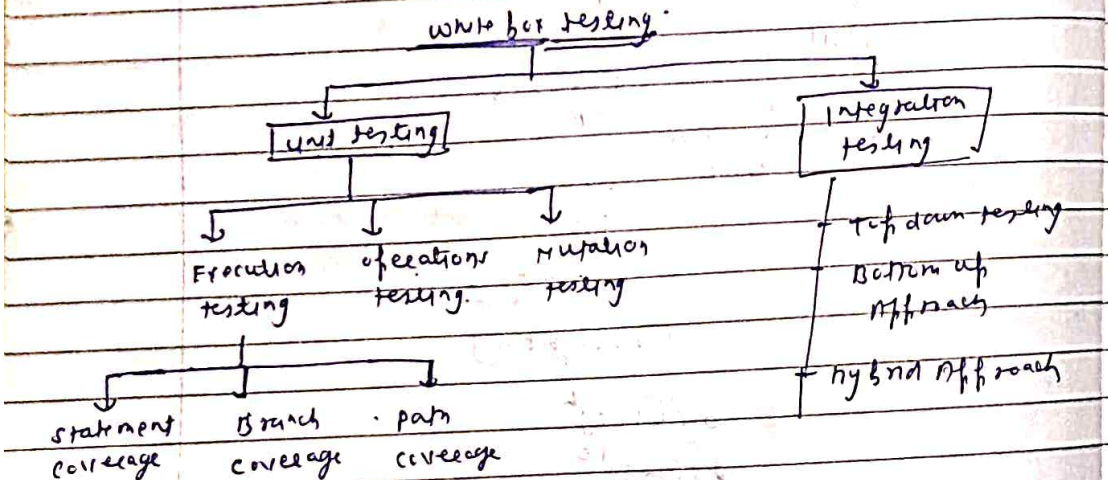
cyclomatic complexity = R+1
= 3+1
= 4

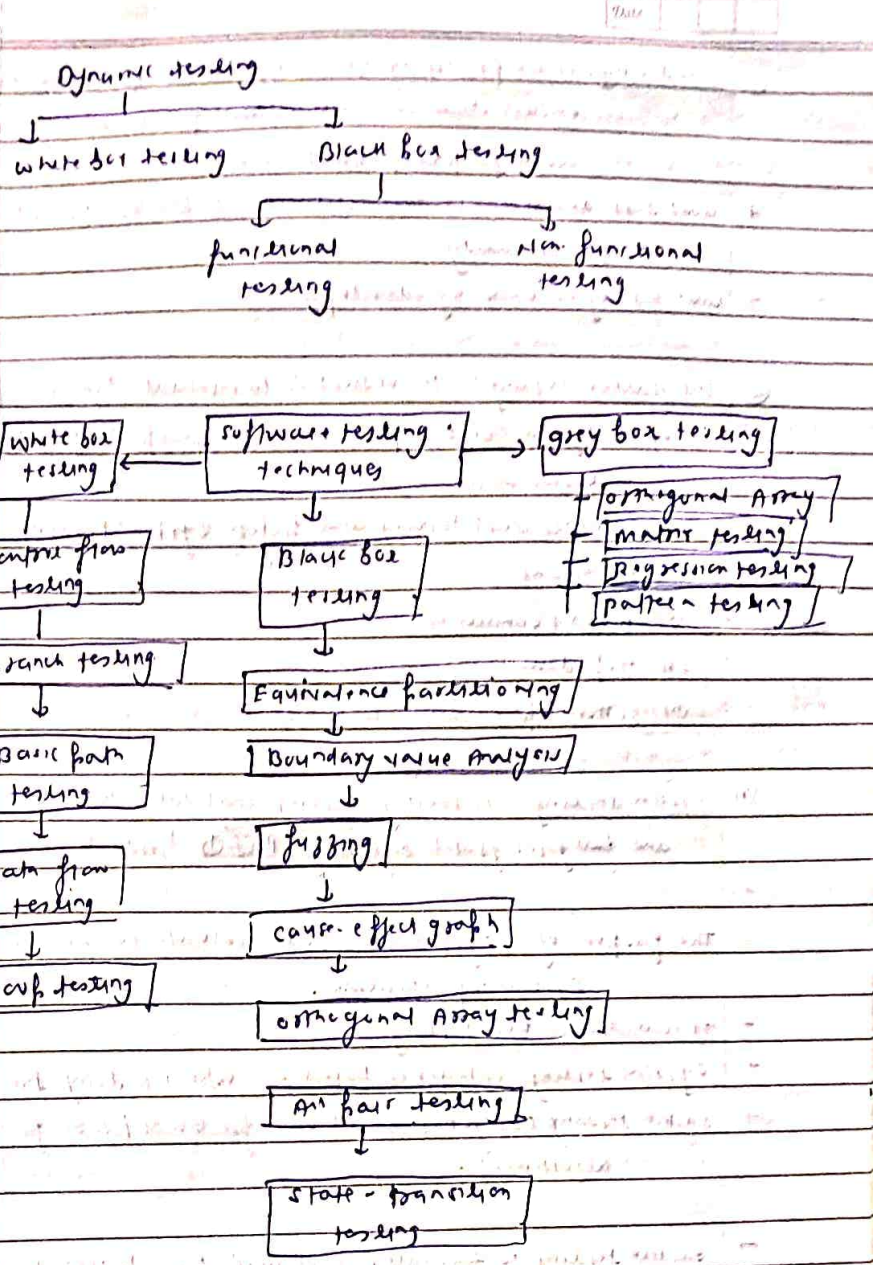
R=3

R+1: Bounded Region

R → 10 edges to reach from one node to another

= i.e R+1
= 3+1
= 4 Ans





① Unit testing: is the process of checking small piece of code to ensure that the individual part of a program work properly on their own

→ Unit test are used to test individual blocks (units) of a functionality.

→ Unit testing is done by developer

② Integration testing:- is conducted to evaluate the compliance of a system or component with specified functional Requirements.

→ It occur after unit testing and before system testing

(1) Big-Bang

(2) Mixed (sandwich)

(3) Top-down

(4) Bottom-up

③ System testing:- is level of testing that validates the complete and fully integrated software product

→ The purpose of a system test is to evaluate the end to end system specification.

→ It is a black-box testing

→ system testing categories based on: who is doing the testing?

→ system testing categories based on: functional / Non functional Requirement.

→ system testing is basically performed by a testing team that is independent of the development team. that helps to test the quality of the system impartial. It has both functional and non-functional testing. system testing is a black box testing

Types of system testing:

① Performance testing: It is a type of sw testing that is carried out to test the scalability, stability and reliability of the sw product or application.

② Load testing:- type of sw testing that is carried out to determine the behavior of the system or sw product under extreme load

③ stress testing:- performed to check the Robustness of the system under varying loads

④ scalability testing:- check the performance of sw application or system in terms of its capability to scale up or scale down the no. of user Request loads

* White box testing (also known as clear-box testing, glass box testing, transparent box testing, and structural testing) is a method of sw testing that tests the internal structure or working of an application.

→ It can be applied at the all the unit, integration, and system levels of the sw testing process.

→ There are some of the top white box testing tools to use:- Veracode, cppunit, Nunit, Rcunit etc

* White box test design techniques:

- control flow testing
- Data flow testing
- Branch testing
- statement coverage testing
- Decision coverage testing
- path testing.

* Statement coverage ~~testing~~ technique:- is used to design white box test cases
* This technique involves execution of all statements of the source code at least once
- It is used to calculate the total no. of executed statements in the source code out of the total statements present in the source code
- This technique covers dead code, unused code, and branches

```
1  main()
2  {
3  int n1, n2, n3;
4  if(n1>=n2 && n1>=n3)
5  printf("%d is the largest no", n1);
6  else if(n2>=n1 && n2>=n3)
7  printf("%d n2 is the largest no", n2);
8
9  else
10 printf("%d n3 is the largest number", n3);
11 }
```

① case: n1 n2 n3 — 30, 20, 10

5/10 × 100 = 50%

② case: 10, 20, 30 (n1, n2, n3) — 10>20 false, 20>=10 T, 20>30

$\frac{P}{10} \times 100 = P\%$

* Condition coverage:- techniques
① It is used to cover all conditions
② Condition coverage is also known as predicate coverage in which each one of the boolean expressions have been evaluated to both true and false.

```
read a,b,c;
if(a==c || b==c)
{
print 1;
}
```

a=0 b=0 c=0 d=0

it covers all condition, but it is condition coverage

a=1 b=c, c=0, d=0

```
else
{
if(c==0 && d==c)
{ print 2;
}
}
```

③ Data flow testing:- It is one of the white-box testing techniques
* Data flow testing focuses on two points:
(i) in which statement the variables are defined
(ii) in which statement the variables are used.

→ To design the test cases that covers control flow and variable definition and their uses in the modules.

Ex:

Line	Code	Var	def line	use
1	read a,b,c;	a	1	2,3
2	if(a>b)	b	1	2,5
3	x=a+1	c	1	NA
4	print x;	x	3,5	4
5	else	z	X	6
5	x=b-1			
6	print z;			

* unused variable is detected

* Advantages of data flow testing
① A variable that is declared but never used within the program
② A variable that is used but never declared
③ A variable that is declared multiple times before it is used
④ Deallocating a variable before it is used

* **Boundary Value Analysis:** The Boundary value testing is a type of testing techniques are helpful for detecting any errors or threats that happened at the boundary values of valid and invalid partition rather than focuses on the center of the I/P data

Test case Number #1

let us Assume a test case that takes the speed of a car from 40-80 to get the best fuel efficiency

40-80
Min Max

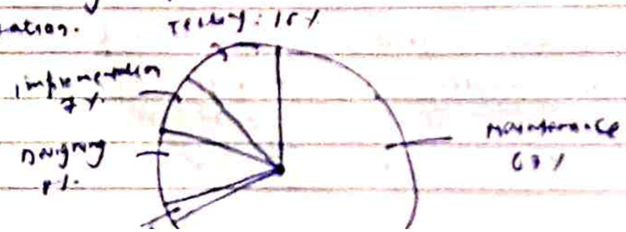
Boundary value → max within a value less than the [?]

Boundary Value Test Case		
Invalid Test case	Valid Test case	Invalid Test case
(min value - 1)	(min, +min, -max, max)	(max+1)
= 40-1	(40, 41, 79, 80)	(80+1)
= 39		= 81

* Software Maintenance

* Its primary goal is to modify and update software application after delivery to correct error and improve performance

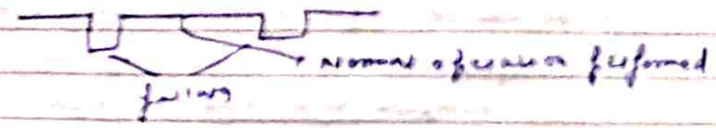
* Software maintenance is an inclusive activity that include:
(1) error corrections
(2) deletion of obsolete capabilities.
(3) enhancement of capabilities.
(4) optimization.



Types:
1. corrective maintenance
2. Adaptive maintenance
3. preventive " "
4. perfective " "

* Maintenance metrics:-

(1) Mean Time b/w failures (MTBF): It is the average time available for a system or component to perform normal operation b/w failures



(2) Mean Time To Repair: (MTTR): It is the average time required to repair a failed component

MTBF = (sum of operational time / no. of failures)
MTTR = sum of down time period / no. of failures

Availability = MTBF / (MTBF + MTTR)

Q. A machine that runs for 24 hours. During this time, it failed twice and each time it took [illegible] an hour to get it back up and running. find the MTBF and MTTR.

Sol: 2 x 1 = 2

MTBF = $\frac{22}{2}$ = 11

MTTR = $\frac{2}{2}$ = 1

Availability = $\frac{11}{11+1}$ = $\frac{11}{12}$ = 91.6%

- **Reverse engineering / Backward engineering**

→ It can help to improve the understanding of the underlying source code for the maintenance and improvement of the software.

→ In some cases the goal of the reverse engineering process can simply be a redocumentation of legacy systems

Code → Module → Design → Requirement specification specification

Usage

- Malware developers often use reverse engineering techniques to find vulnerabilities in an operating system to build a computer virus that can exploit the system vulnerabilities

- Reverse engineering is also being used in cryptoanalysis to find vulnerabilities in substitution cipher symmetric key, algorithm or public key cryptography

Computer aided software engineering

→ It is the domain of the s/w tools used to design and implement applications.

→ Case tool were used for developing high quality, defect-free, and maintainable s/w

→ It help to improve s/w development as well as a maintenance

- Improves quality at lower cost and increases productivity.

Case tool example:

(1) Planning and Requirement phase (flow chart maker)
(2) Design phase (UML, Animated s/w design)
(3) Coding (Cscope, eclipse)
(4) Testing (selenium, cucumber)
(5) Web development tools (Fireworks, Adobe edge inspect, foundation 3)
(6) Quality Assurance tools (soap test, apps watch)
(7) Maintenance (Bugzilla: for defect tracking, HP quality center)

* **Smoke testing**: is the initial testing phase executed to check whether the s/w under the test is ready/stable for further testing. It is also known as Build Acceptance testing

* Sanitary testing comes after smoke testing

* **Regression testing**: Means hamara koi existing software hai jo ki kaam kr rha hai but new bug aa gya hai aur hum chahte hai ki wo bug remove ho jaye tou hum new updation late hai aur new updation se kya jo humara existing s/w hai uski functionalities koi s/w changes na aaye ye Regression testing

→ Regression testing is a type of s/w testing that verifies the changes made to the system, such as bug fix or new features, do not impact previously working functionality.

- There are several types of Regression testing including:

(1) **Full regression testing**: Testing the entire application from start to finish after after changes have been made

(2) **Partial regression testing**: Testing only those parts of the application that were affected by the changes

- performance / non-functional testing can determine how a system performs in terms of responsiveness and stability under a particular workload

- A performance test measures scalability, speed, reliability, and throughput of your product

① Load testing: It evaluate the product's ability to perform under anticipated load. for ex max capacity of a web server

② Stress testing: It is normally used to understand the upper limit of a capacity within the system. for ex capacity of a web site or an e-commerce platform during peak traffic hours such as during a major sale events

③ Volume:

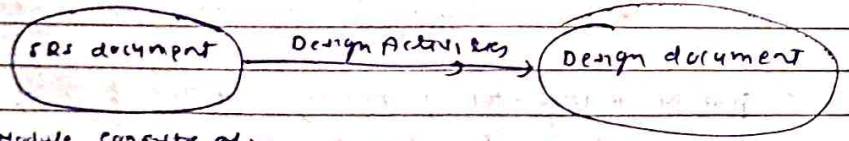
④ Endurance:

⑤ Spike testing: test the product reaction to sudden increase and decrease in the load.

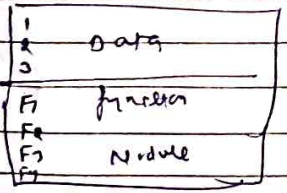
---

* cohesion and coupling:

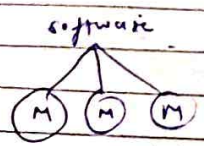
→ Design phase transform SRS document in to a form easily implementable in some programming language



* A Module consists of:
  1. several functions
  2. Associated data structures



→ Modularity: It a fundamental attributes of any good design
Decomposition of a problem cleanly in to modules:
1 Modules are almost independent of each other
2 divide and conquer principles



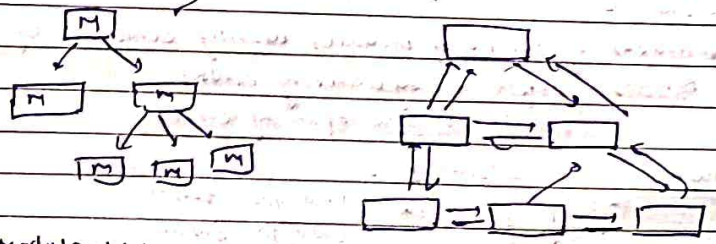
Meta data / function
↓
attributes

- Modularity:

if Modules are independent:
* Module can be understood separately
* Reduces the complexity greatly

To understand why this is so,
→ Remember that it is very difficult to break a bunch of sticks but it is very easy to break the sticks individually

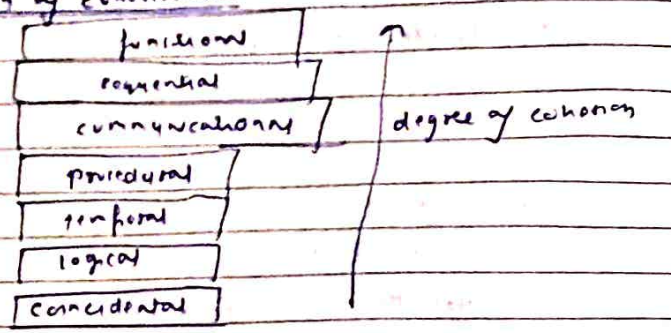
→ Example of cleanly and Non-cleanly Decomposed Modules



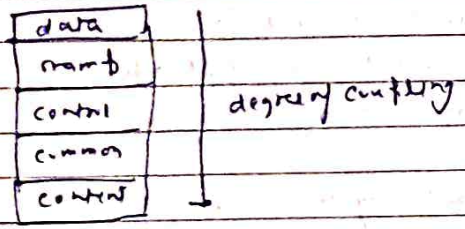
Modularity:
In technical terms, Modules should display
1. High cohesion
2. Low coupling

cohesion and coupling

classification of cohesiveness



coupling:

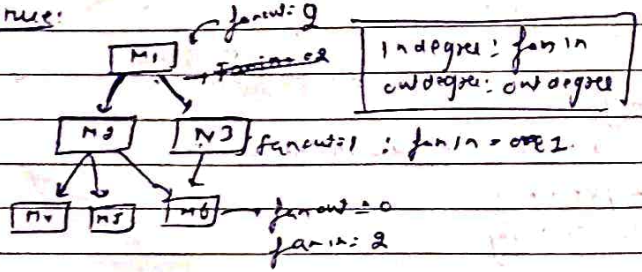


characteristics of module structures:

① Fan-in: indicates how many modules directly invoke a given module
high fan-in Represent code Reuse and is in general encouraged

② Fan-out:
↳ A measure of the no of modules directly controlled by given module
depth: no of levels of control
width: overall span of control

Module structure:



Cocomo Model: constructive cost Model

used for calculate the effort, development time, Average staff size, and productivity

constructive cost Model include:

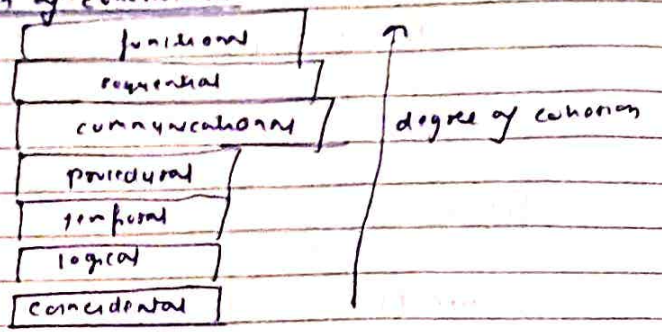
① Basic cocomo Model
② Intermediate cocomo Model
③ complete / detailed cocomo Model

⇒ cocomo Applied on 3 classes of sw project
① organic Mode ② semi detached Mode ③ embedded

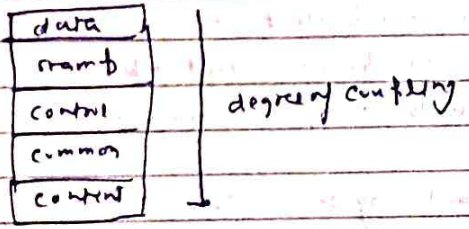
	Project size	Nature of project	Innovation	Deadline for a project
x organic	⇓ 2-50 KLOC	⇓ small size project experienced developers ex: payroll inventory project	little	not tight
x semi-detached	50-300 KLOC	Medium size project and team ex: OS system	Medium	Medium
x embedded	over 300 KLOC	Medium large project Real time system ex: ATM Airtraffic control	significant	tight

cohesion and coupling

classification of cohesiveness



coupling:

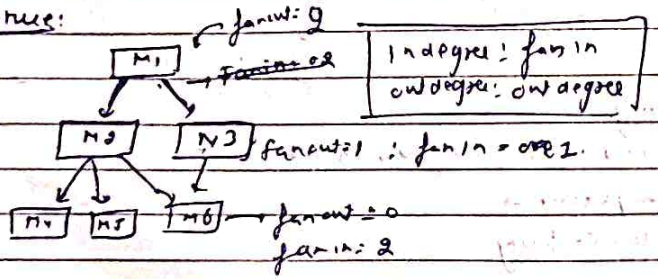


characteristics of modwe structure:

① Fan in: indicates how many modwes directly invoke a given modwe
high fan-in Represent code Reuse and is in general encouraged

② Fan-out:
↳ A measure of the no of modwes directly controlled by given modwe
depth: no. of levels of control
width: overall span of control

Modwe structure:



cocomo Model: constructive cost Model

Used for calculate the effort, development time, Average staff size, and productivity

Constructive cost Model include:

① Basic cocomo Model
② Intermediate cocomo Model
③ complete / detailed cocomo Model

cocomo Applied on 3 classes of sw project
① organic Mode ② semi detached Mode ③ embedded

	Project size	Nature of project	Innovation	deadline for a project
organic	⇓ 2-50KLOC	⇓ small size erp developer ex: payroll inventory project	little	not tight
semi-detached	50-300KLOC	Medium size project and team ex: db system	Medium	Medium
embedded	over 300KLOC	large project Real time system ex: ATM Airtraffic control	significant	tight

• **COCOMO Model:**

Basic COCOMO Model

$E = a_b (KLOC)^{b_b}$ — E = effort applied to the project

$D = c_b (E)^{d_b}$ — (d = development time)

People required $P = E / D$

a_b, c_b, d_b, b_b are coefficients

s/w project	a_b	b_b	c_b	d_b
→ organic	2.4	1.05	2.5	0.38
→ semi-detached	3.0	1.12	2.5	0.35
→ embedded	3.6	1.20	2.5	0.32

eg suppose that a project was estimated to be 400 KLOC. Calculate the effort and development time for each of three modes i.e. organic –
semi-detached –
embedded –

Sol. The basic COCOMO equation takes the form

$E = a_b (KLOC)^{b_b}$: effort applied

$D = c_b (E)^{d_b}$: development time

estimated size of project: 400 KLOC

(1) organic mode: $E = a_b (KLOC)^{b_b}$

$= 2.4 (400)^{1.05}$

$= 1295.31$ PM (person month)

$D = 2.5 (1295.31)^{0.38}$ — $D = c_b (E)^{d_b}$

$= 38.07$ PM

(2) Embedded mode: $E = 3.6 (400)^{1.20}$

$= 4772.81$ PM

$D = 2.5 (4772.8)^{0.32}$

$= 38$ PM

(3) Semi detached

$E = 3.0 (400)^{1.12}$

$= 2462.79$ PM

$D = 2.5 (2462.79)^{0.35}$

$= 38.45$ PM

• Intermediate COCOMO Model – It is an extension of basic COCOMO model.

→ uses a set of 15 additional predictors (cost drivers)

→ cost drivers adjust nominal cost of project to actual project environment

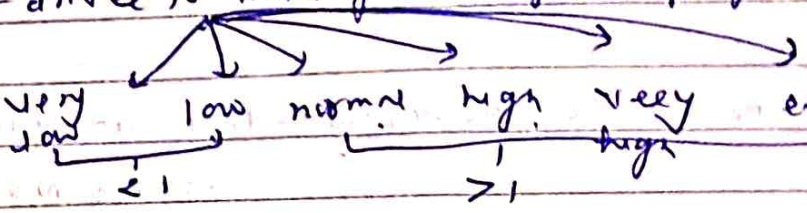
(1) Product attributes:
→ required s/w reliability
→ db size (DATA)
→ product complexity (CPLX)

(2) Computer attributes
→ execution time constraints (TIME)
→ main storage constraints
→ virtual m/c volatility (VIRT)
→ computer turn around time (TURN)

③ personal Attributes → Analyst capabilities (ACAP)
→ App experience
→ programmer capability
→ virtual M/C experience
→ programming lang experience.

④ project attributes:
→ Modern programming practices
→ used of SW tools
→ Required development schedule

→ Each cost drive is Rated for the given project environment



Equation for intermediate COCOMO is:

$E = a_i (KLOC)^{b_i} * EAF$

$D = c_i (E_i)^{d_i}$

$N = E/d$

↳ Effort adjustment factor
↓
It can be calculated by multiplying all the values that have been obtained after categorising each cost drive.

	Project	a_i	b_i	c_i	d_i
1.	organic	3.2	1.05	2.5	0.38
2.	semi-detached	3.0	1.12	2.5	0.35
3	embedded	2.8	1.20	2.5	0.32

organic: $E = 2.4 (KLOC)^{1.05}$ PM

$E = 3.0 (KLOC)^{1.12}$ PM

$E = 2.8 (KLOC)^{1.20}$ PM