SE - Exam & Notes

UNIT - I

* Software Crysis -
(i) Lots of fails in initial days.
(ii) many Softwares was experience.
(iii) Demand was very high.
(iv) Lack of developers
(v) Complexities were very high.

* Software Engineering -

→ It is a process of gathering requirements of user as client and then analyze, design, build and test the software application to satisfy those requirements.

* Four key Activities of SE / Process

(i) Software Specification - gather all the requirements and analyze them.

(ii) Software Development - Implimentation of requirements by coding.

(iii) Validation - Ensure that the final product meets the given requirements as mat.

(iv) Evolution - Providing versions for further refinements.

* 6 Stages of SE life Cycle-

(i) Requirement gathering and Analysis.
(ii) Feasibility Study
(iii) Designing
(iv) Implimentation or Coding
(v) Testing
(vi) maintenance

* characteristics of Software -

(i) Software does not wearing out - SW should become reliable with time instead of wearing out.

(ii) Software is not manufactured - as there is no assembly line in Software development hence it is not manufactured.

(iii) Reusability of the Components - Some standard Components are made that are may be used in new projects.

(iv) Software is flexible - Software is Capable of changes according to the new requirements.

* 6 stages in details -

(i) Requirement gathering and Analysis
- gather all req from client.
- Analyse those req.
-

(ii) Feasibility - In this process we ensure that the development is possible in available resource, expense and time as not.

→ Types of feasibility -
(a) Economic - Money.
(b) Legal - Cyber Law
(c) Operational - Can we create operations as expected by client
(d) Technical - Current Computer Support and Compatibility.
(e) Schedule - Time

(iii) Design - SDD developed (Software design Documents) according to the requirements
In this part we define the architecture of the products.
SDD include -
(i) Brief description of each modules.
(ii) Interface relationships and dependencies between modules.
(iii) Listing of error messages
(iv) input and output of every module

(v) Functionality of each module.

(iv) Coding - Implimentation and developing the software using any programming language.

(v) Testing - After coding testing team test all the functionalities and match them to the given requirements by the client.

They also find out bugs, flaws and vulnerabilities and report it to development team for fix.

* (vi) maintenance - After deployment of the final product 3 activities follows as maintenance.

(a) Bug fixing
(b) Upgrade
(c) Enhancement

* Why do we need SE
→ To

(i) Reduce the Complexity
(ii) Reduce the Cost
(iii) Reduce the time
(iv) Increase the Quality
(v) Increase the effectiveness
(vi) Handle the big projects

* Models of SDLC (Software Development Life cycle)

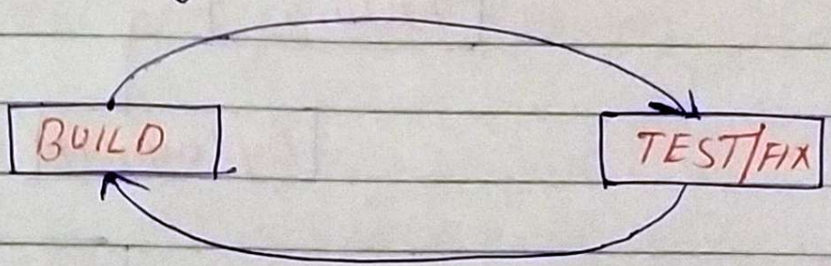
(i) Build and Fix model

In this model development proceed directly by coding without specification and designing. This model basically works on two phases -

(a) Build - In this phase we proceed directly by coding and create a project.

(b) Fix - In this phase we do testing and make the software error and bug free.

→ ~~Advan~~



→ Advantages -

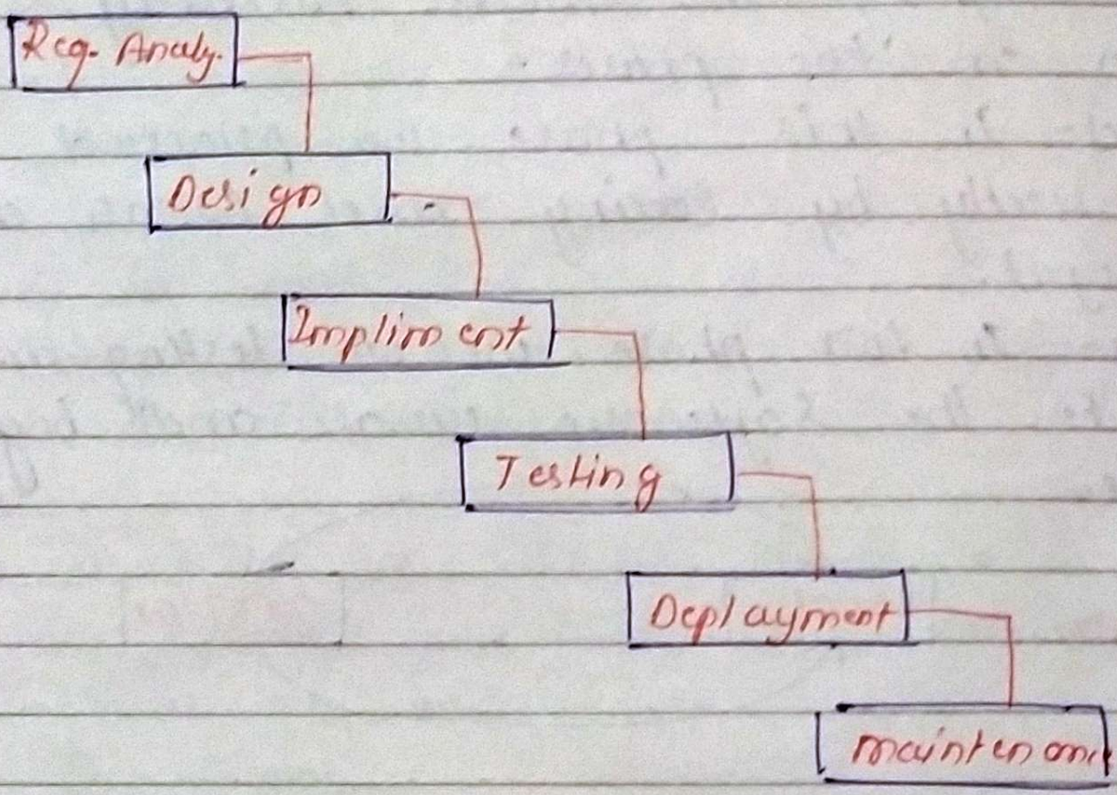
(i) Basic and Easy
(ii) Suitable for small projects.
(iii) Less planning needed.

→ Disadvantages -

(i) Can't analyse - quality, progress and risk
(ii) Problematic maintenance
(iii) Procedure is unplanned.

(iii) Waterfall Model -

This is a sequential model which is widely used in software industry. This proceeds in multiple phases in which output of one phase becomes input for next phase.



→ Advantages -

(i) Simple and easy
(ii) Suitable for small projects.
(iii) Approach is systematic
(iv) Easy to maintain

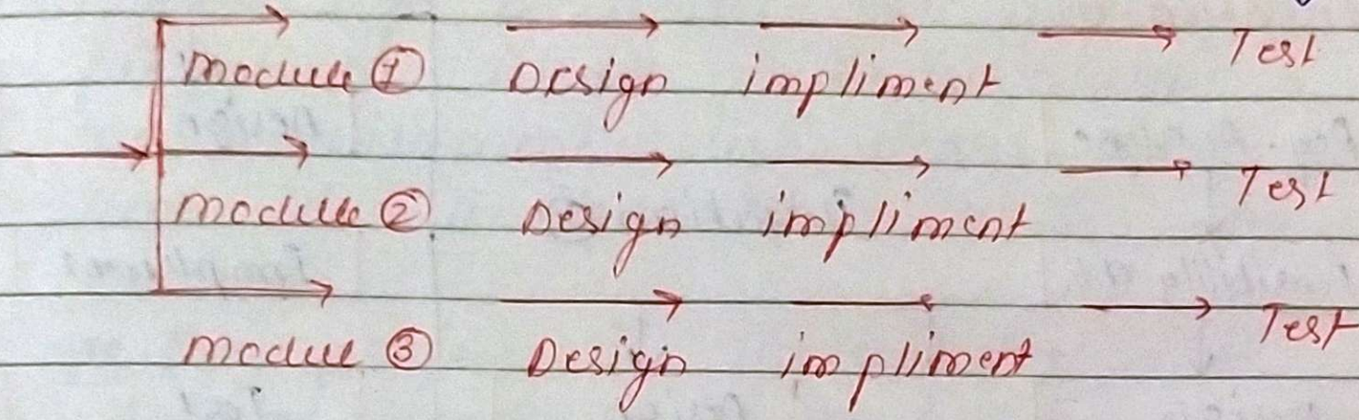
→ Disadvantages -

(i) No feedback available
(ii) High risk factor.

(iii) No requirement flexibility
(iv) maintenance is costly.

(iii) Incremental model-

In this model work is done module by module to make it possible to release the application in early phase with limited functionality.



→ Advantages-

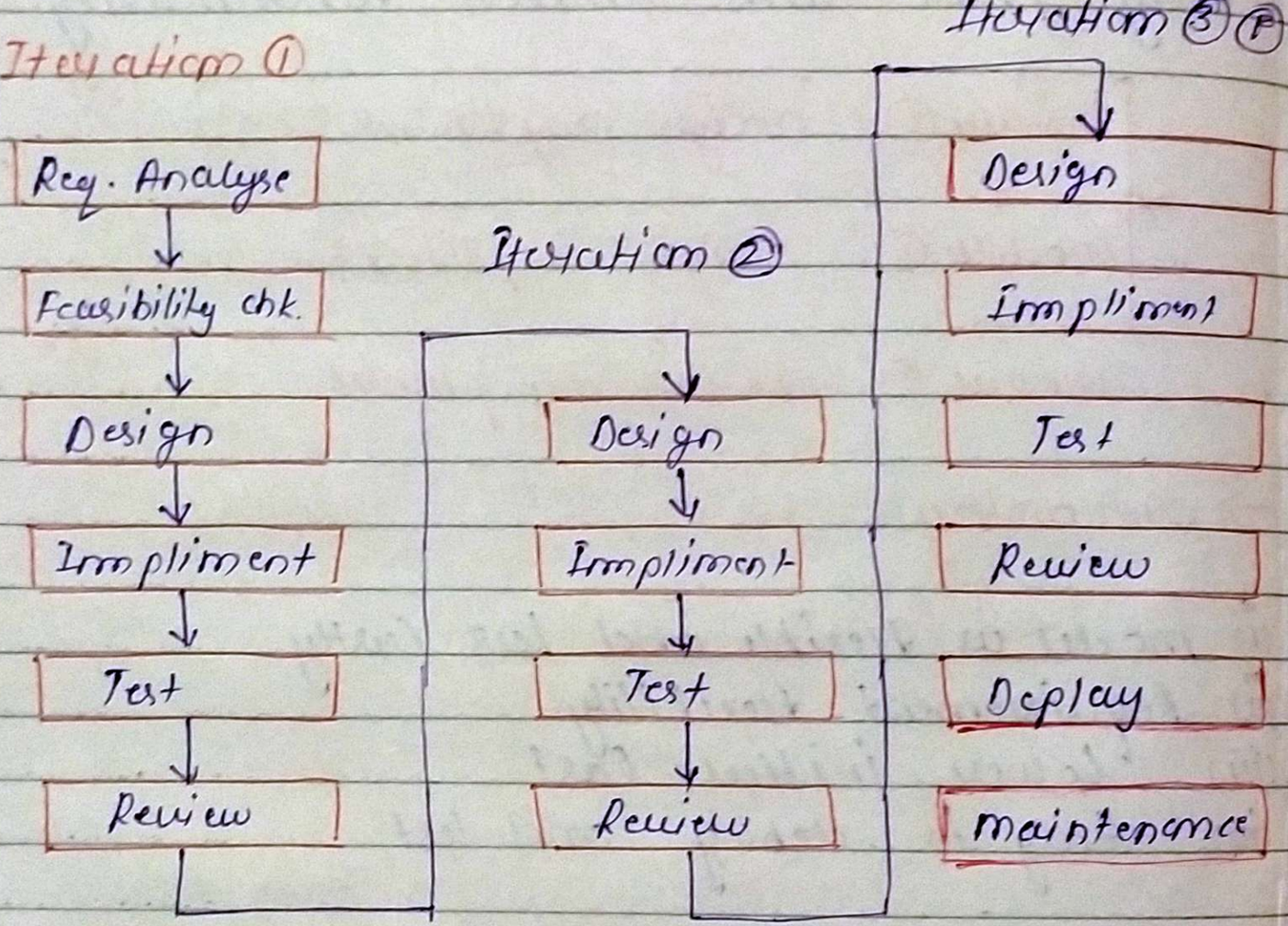
(i) model is flexible and less costly
(ii) Requirement flexibility.
(iii) Lower initial cost
(iv) Easy to debug and test.

→ Disadvantages-

(i) Not suitable for small projects
(ii) Need good planning.
(iii) Overall cost can be high
(iv) Long process.

(iv) Iterative model -

In this model we work in small iteration by iteration and quickly release the software in early stage with working condition in this model feedback is available which was absent in waterfall.



→ Advantages -

(i) feedback is available
(ii) Quick project launch
(iii) Risk handling is good
(iv) Higher quality

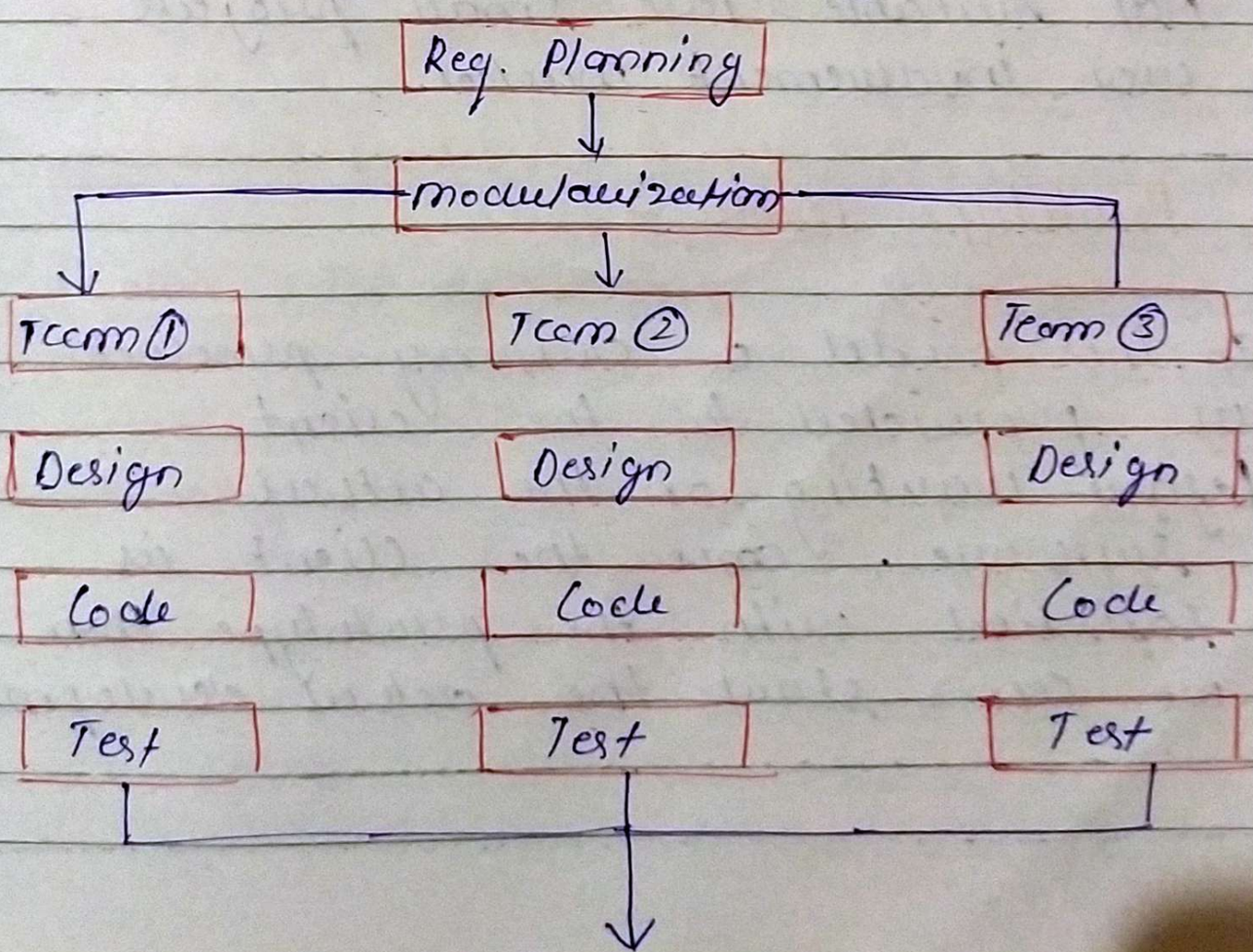
→ Disadvantages-

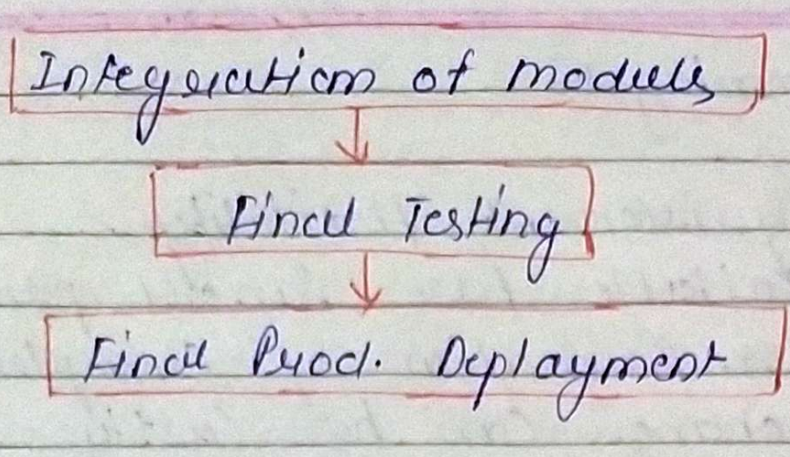
(i) No Requirement Flexibility.
(ii) Not suitable for small projects
(iii) No confirmation of completion date
(iv) Req. change can be costly.

(v) RAD - Rapid Application Development

In this type of model we make project of ~~model~~ software when the requirements is clear and time is limited like 30-90 days.

The project is distributed into several teams and each team work independently simultaneously.





→ Advantages-

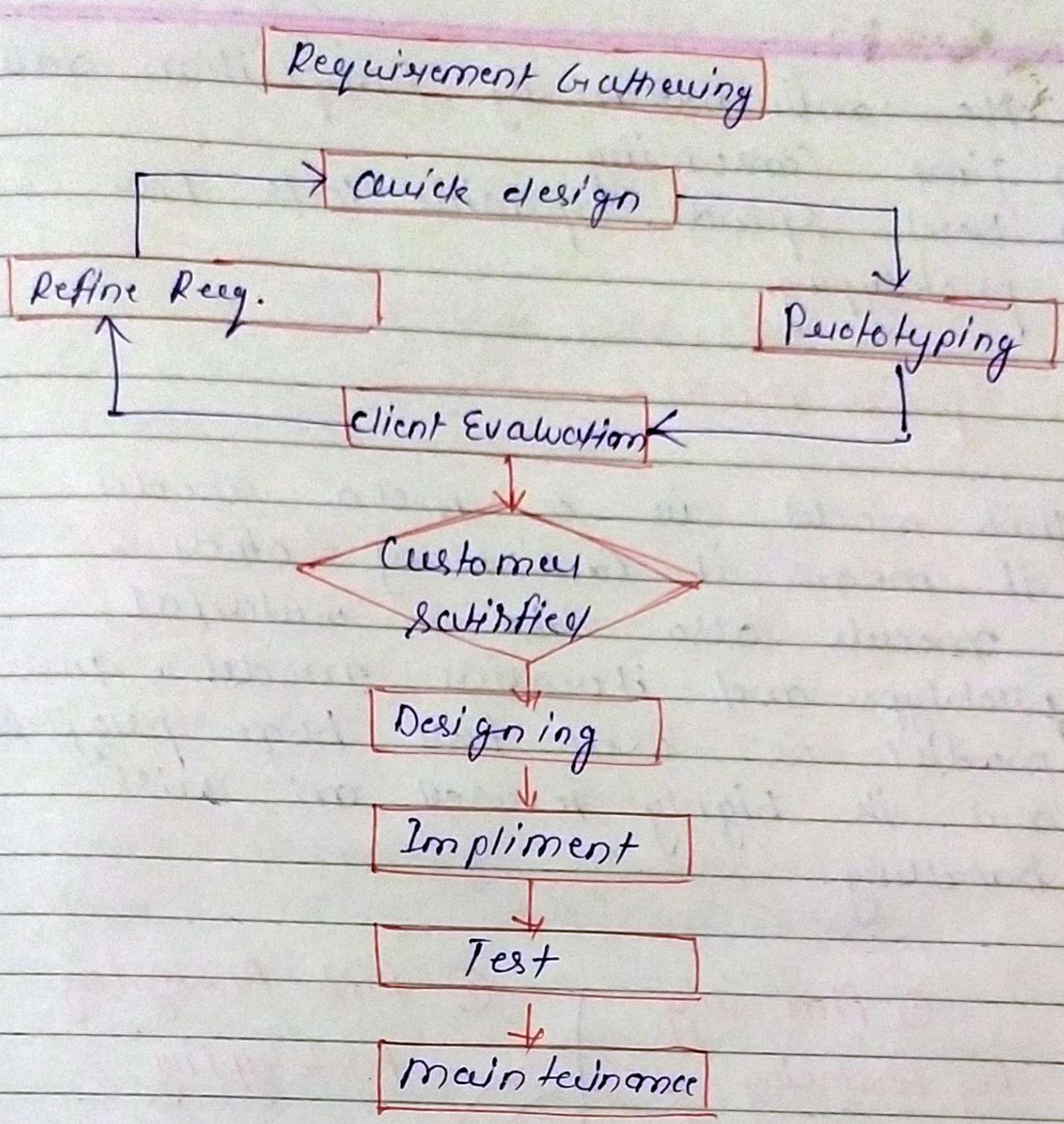
(i) Parallel working on each module
(ii) Flexible for change.
(iii) Reduce development time.
(iv) Increase Reusability of features.

→ Disadvantages

(i) Highly skilled developers needed
(ii) only applicable for modular application.
(iii) Not suitable for small projects
(iv) user involvement needed.

(vi) Prototype Model :-

In this model a dummy product is provided to the client before working on the actual software, once the client is satisfied with the prototype then we can start the actual development.



→ Advantages-

(i) Reduce risk of incorrect user requirem.
(ii) Reduce maintenance cost.
(iii) Good where req. are not fixed.
(iv) Quality is good.

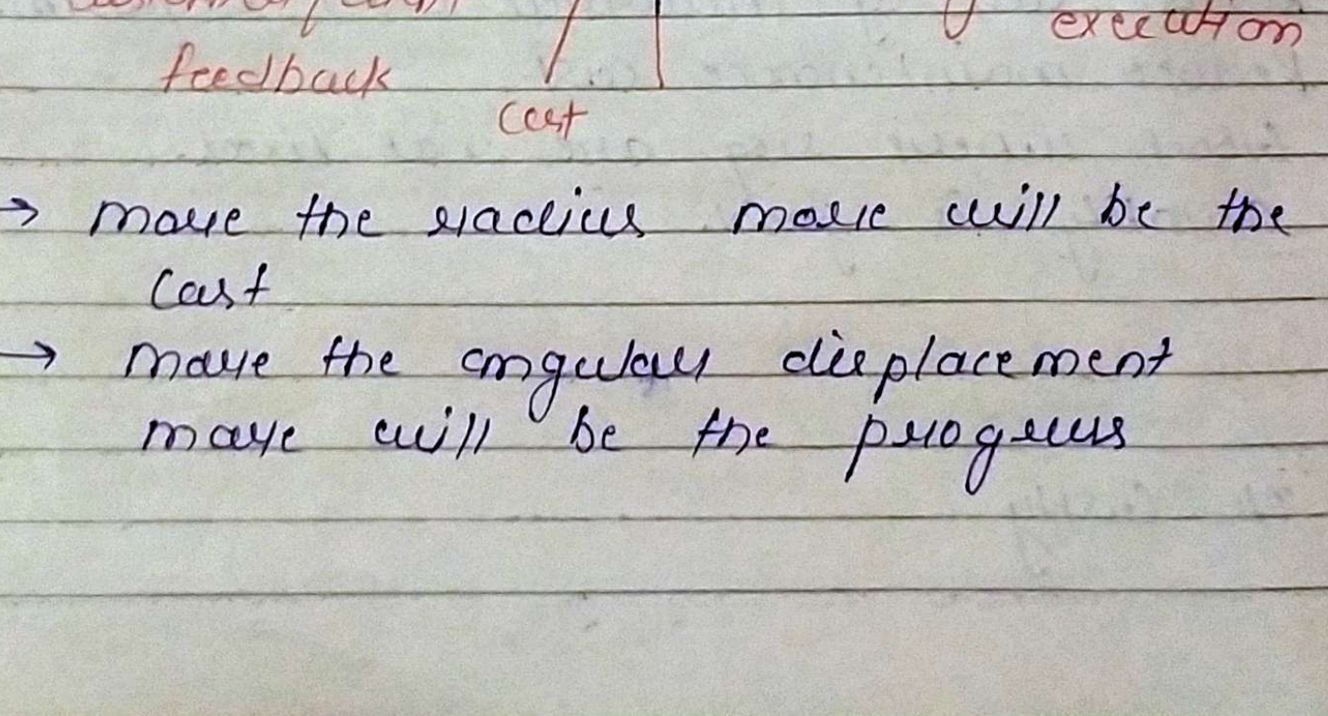
→ Disadvantages-

(i) Its costly

(ii) No Confirmation of Completion date.
(iii) Time Consuming.
(iv) Need special expensive tools for prototyping

(VII) Spiral model -

This model is a meta model it means it consist of other models also like, waterfall, prototype and iterative model. This model is used for huge projects and its highly focused on risk handling.



→ more the radius more will be the cost
→ more the angular displacement more will be the progress

→ Advantages -

(i) Risk handling is good.
(ii) meta model
(iii) Suitable for huge projects
(iv) Flexible
(v) Customer Satisfaction.

→ Disadvantages -

(i) Complexity is high
(ii) Expensive.
(iii) Risk Analysis is must.
(iv) Time Consuming
(v) Highly skilled developers needed.

UNIT – 2

Software metrics Engineering

* **Measure** - It provides a quantitative indication of extent, amount, capacity or size of some attributes of a product or process.
ex- no. of errors found in a module

* **measurement** - act of determining a measure.
ex- Collection of errors

* **Metrics** - Quantitative measure of the degree to which a system, component or process possesses a given attribute
ex- Average no. of errors found

* **Software Metrics** - A quantitative measure of characteristics which is countable of a software.

* **why do we need Software Metrics** -
→ TO
(i) Estimate the cost and schedule
(ii) Evaluate the productivity impact
(iii) Improve Software quality

(iv) Estimate the size of software.
(v) Estimate the staffing needs

* Types of metrics -

(i) Product metrics - It measures the characteristics of product after development.
like - size, complexity, performance efficiency, reliability.

(ii) Process metrics - It measures the effectiveness and development activities during the development process.
like - Effort required, Time required No of defect found during testing. maturity of process

(iii) Project metrics - It measures the characteristics of whole project like
like - no. of developers, staffing pattern cost, productivity.

* Size oriented metrics -

(i) LOC - Line of Code

(ii) FPA - Functional Point Analysis

(iii) Halstead's Software metrics

(iv) Data Structured metrics

(v) Cyclometric Complexity

(i) LOC metrics- It calculate number of lines in a source code

- It is size oriented metrics
- It is old and traditional.
- Language dependent
- Not for GUI based languages

→ Advantages-
- Easy and Simple

→ Disadvantages -
- Language dependent
- No proper industry standards
- No Consistency - A program can be written differently in some programming language

(ii) Function Point Analysis -
This metric is used to measure functionalities and size of the software. it estimate the size and cost of the system.

- It independent of language

- Estimate development effort in early phase.
- Directly linked with requirements

→ Functional units of a system

External Input - EI
External Output - EO
External Enquiry - EQ
Internal Logical Files - ILF
External ~~Logical~~ Files - EIF
Interface

→ Table -

Function units	Low	Avg	High
EI	3	4	6
EO	4	5	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10

Trick - Learn first line (EI) 3, 4, 6
→ put the same values in 3rd line (EQ)
→ in second line add one (+1) in each column of first row (3+1) (4+1) (6+1)
→ Learn 4th line 7, 10, 15 (ILF)
→ Put 7 and 10 diagonally in last row and put 5 in first column → Done

→ Formulae

$$FPA = UFP \times CAF$$

$$CAF = 0.65 + (0.01 \times \Sigma Fi)$$

$$\Sigma Fi = 14 \times \text{Scale given}$$

-Scales-

0 - No influence
1 - Incidental
2 - Moderate
3 - Average
4 - Significant
5 - Essential

* Halstead's Software metrics-

In this metrics we calculate the no. of tokens, which is classified as operators and operands.

n_1 = No. of unique operators
n_2 = No. of unique operands
N_1 = Count of total no. of operators
N_2 = Count of total no. of operands

Total no. of tokens = $n = n_1 + n_2$

Size of the program = $N = N_1 + N_2$

Difficulty $D = \frac{n_1}{2} \times \frac{N_1}{n_2}$

Effort $E = D \times V$

→ Advantages -
- Simple to calculate
- Predict rate of error.
- " maintenance effort
- Program language independent.

→ Disadvantages -
- Depends on Complete Code
- No use of predictive estimate model.

* Data Structure metrics -

It measures the amount of data processed input and output in a software. It Compute effort and time required.

It used to Calculate -
(i) The Amount of data

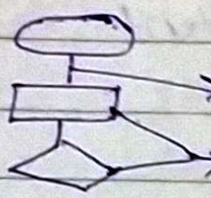
(ii) usage of data within a module

(iii) Program weakness

(iv) Sharing of data among modules

* Cyclomatic Complexity-

It used to calculate the complexity of a program by calculating no of nodes and edges of the flow graph of some code. by using the formulae



$$m = E - N + 2$$

E = Edges

N = Nodes

→ Advantages-
- It is a quality metrics
- Faster than Halstead.
- Easy to apply

→ Disadvantages-
- Not measure data Complexity
- Nested structure hard to understand
- Sometimes gives misleading values.

UNIT - 3:

Planning -

* Activities of Project planning -

(i) Estimating the project size, duration, cost and effort.
(ii) Scheduling manpower and other resources.
(iii) Staff organisation and staffing plans
(iv) Risk identification, analysis and ending planning.
(v) miscellaneous plans such as quality assurance plan, Configuration management plan, etc.

* Senior Management Team	Project management Team
1. Approve prjkt. and provide resource for project.	1. Review the prjkt plan and implim-ent them.
2. Review the project plan to ensure that it meets the objective.	2. manage all prjkt activities.
3. Resolve Conflict b/w team members	3. Prepare budget

4. Consider risks and manage them.

4. understand the objectives and find a way to to accomplish them

* Project Planning Activities-

(i) Identification of Project requirements

(ii) Identification of Cost estimates

(iii) Identification of the risk.

(iv) Identification of Critical Success factors.

(v) Preparation of project charter.

(vi) Preparation of project Plan

(vii) Commencement of the project

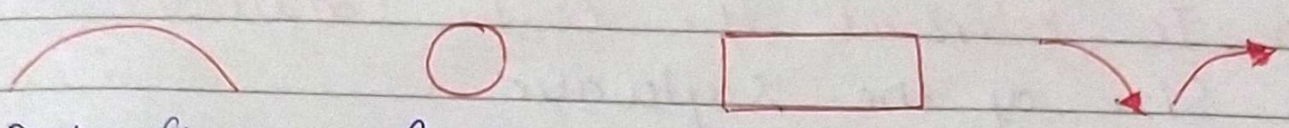
(viii) Organization Structure

* DFD - Data Flow Diagram

A DFD shows how data is processed in terms of input and output as how data store, when it come from and where it goes.

→ characteristics of DFD-

(i) All names should be unique
(ii) Arrow shows flow of Data
(iii) No logical decision must be made in command box.
(iv) No loss of details



→ Levels of DFD-

(i) 0-Level- It includes single bubble of process with input and output data

(ii) 1-Level- It includes multiple bubbles of process and we break down 0-Level process into multiple subprocess.

(iii) 2-Level- It is a deeper version of 1-level DFD it used to plan necessary details about the system

* DBMS ~~~ Attribute, Entity, Primary key, Candidate key, Super key foreign key, 1-1R, 1-mR, m-1R m-mR etc.

UNIT - 4

Software Cost Estimation

A process of predicting the effort required to develop a software.

→ To estimate the cost consider-
- Size of the software
- Software quality
- Hardware
- Tools
- Developers
- Travelling
- Communication
- Training and Support.

* Cocomo model (Constructive Cost model)

- It used to predict effort and schedule of software product based on the size of the S/w.

→ Types of Cocomo model-

(i) Basic - A single Valued Static model that calculate effort on the bases of size found by LOC (Lines of Codes)

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

$$D = c(E)^d$$

No. of Developers = $\frac{E}{D}$

* Table -

Software Project	a	b	c	d
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

Trick - a ⟶ 2.4 + 0.6 , + 0.6,
b ⟶ 1.05 + 0.7 , + 0.8
c ⟶ 2.5 ⟶ Same
d ⟶ 0.38 - 0.03 , -0.03

(ii) Intermediate - It is extension of basic cocomo model that use functional size to calculate the effort.

(iii) Detailed - It is same as Intermediate but detailed as it considers each step of SDLC for effort estimation.

- Advantages-

(i) It is transparent
(ii) more accurate
(iii) Easy to Impliment
(iv) Easy to estimate total cost

- Disadvantages-

(i) It ignores the requirement and documentation.
(ii) It ignore customer skills and Coperation, knowledge.
(iii) change in time effects the accuracy.
(iv) Hardware failure is cut by control

* Software maintenance-

— SE-2 —

UNIT - 5

* Requirement engineering - The process of gathering software requirements from the client, analyze, document them.

It is required to maintain a proper SRS (System Requirement Specification) documents.

It help to understand the developer better that what client actually wants and get a clear image what to proceed further accordingly.

→ Steps -

(i) Inception - at the very begining like initiate communication, identify problem and find solution. Scope.

(ii) Elicitation - [extraction] - In this step we extract out all the needs and requirements from client and stack-holders

(iii) Elaboration - In this part we discuss, elaborate, expand, refine the details we get from Inception and Elicitation and try to develop (RRM) Refined Requirement model.

(iv) Negotiation - Agree on a deliverable system that is realistic for developers and clients and create a win-win situation.

(v) Specification - Describe the requirement formally as SRS.

(vi) Validation - Elimination of errors, ambiguities, un-realistic specification, and conflicts.

(vii) Requirement management - Manage changing requirements as well as monitor, track, control the changing req.

* SRS - System Requirement Specification

It is a document that ~~doesn't~~ Capture Complete description about how System is expected to perform. It signed off at the end of Requirement engineering phase

(iii) Elaboration - In this part we discuss, elaborate, expand, refine the details we get from Inception and Elicitation and try to develop (RRM) Refined Requirement Model.

(iv) Negotiation - Agree on a deliverable system that is realistic for developers and clients and create a win-win situation.

(v) Specification - Describe the requirement formally as SRS.

(vi) Validation - Elimination of unclear, ambiguities, un-realistic specification, and conflicts.

(vii) Requirement management - Manage changing requirements as well as monitor, track, control the changing req.

* SRS - System Requirement Specification

It is a document that ~~desc~~ Capture Complete description about how system is expected to perform. It signed off at the end of Requirement engineering phase

* characteristics of SRS →

(i) Consistency - No req. conflicts

(ii) Completeness - Should have all the complete details inside. SRS includes functional and non functional Req.

(iii) Unambiguousness - every requirement should have a unique and clear interpretation or meaning. and easy to understand

(iv) Correctness - it should cover all the needs that are truly expected from the system.

(v) Ranking of importance and Stability - Ranking should be correct according to the priority and desirability of the requirements.

(vi) Modifiability - Should be capable of any changes occured during the process.

(vii) Verifiability - All the detail in the SRS should be verified that final Software meet the all req.

(VIII) Traceability - All req. should be traceable that we should be very clear about the origin of every req. in SRS

UNIT - 6

* Software Design and Implimentation-

→ Disigning An activity by which we identify software component and their relationships according to client requirement.

→ Implimentation - Process of converting design into an actual software by coding

→ Software desin and implimentation is an process of SE at which we develop an executable System.

* Software design Concepts -

(i) Abstraction
(ii) Refinement
(iii) Modularity
(iv) Software Architecture
(v) Control Hierarchy
(vi) Structural Partitioning
(vii) Data Structure
(viii) Software Procedure
(ix) Information hiding

* Features of good design -

(i) Correctness
(ii) Completeness
(iii) Efficiency -
(iv) Flexibility
(v) Consistency
(vi) maintainability

* Cohesion and Coupling

→ Cohesion is measure of function strength of a module. relationship of function within the module

For good software design cohesion should be high.

module have cohesion called intra-module

→ Types of cohesiveness -

(i) Functional - in which the parts of module group together because they contribute to module a single well defined task.

(ii) Sequential - in which parts group together due to output of one part is Input to other.

(iii) Communicational - grouped because they operate on same data.

(iv) Procedural - group because they operate on same algorithm.

(v) Temporal - group when the all tasks must be executed in the same time span

(vi) Logical - group because all parts fall into the same logical class of functions.

(vii) Coincidental - group when have little or no relationship to one another.

* Coupling →

→ It is the measure of interdependence between the module

→ For good software design Coupling should be low

modules has Coupling are called inter-modules

* Types of Coupling -

(i) Data Coupling - When two modules have dependencies of data on each other.

(ii) Stamp Coupling - when multiple modules have dependencies of some data structure and work on different part of it.

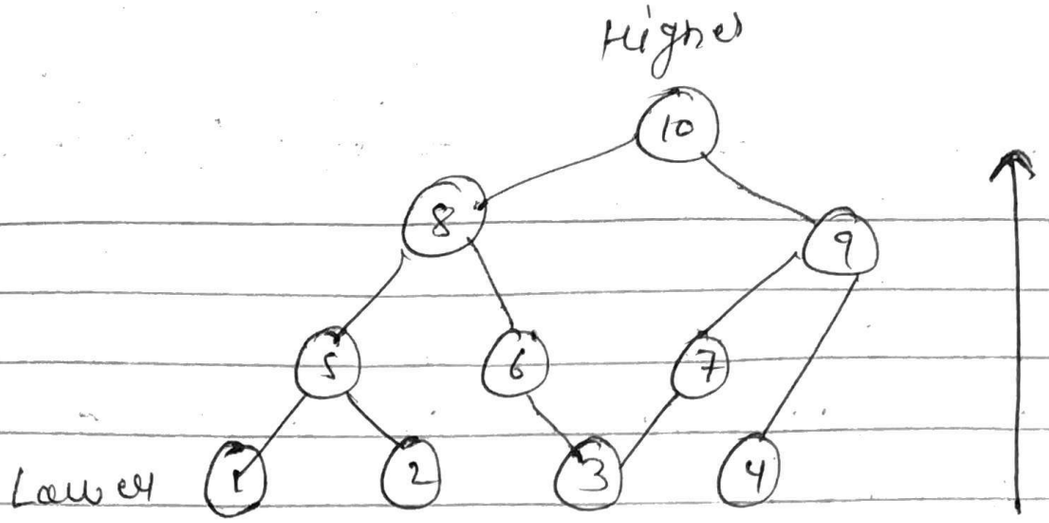
(iii) Control Coupling - when dependencies of functions and flow of execution

(iv) Common Coupling - when multiple module have read-write access to global data

(v) Content Coupling - when a module can directly, access, modify or refer to the content of other modules

* Design strategies

(i) Bottom-up Approach — In this method the multiple subsystems merge into fewthers subsystems and again more system into fewther subsystem till we get a whole system.



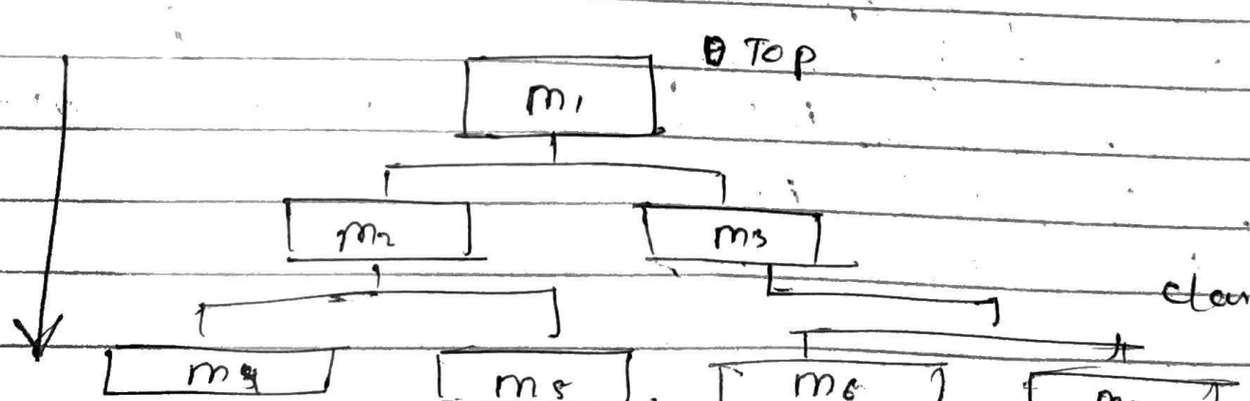
Advantages-

(i) Reusability of functions or modules is possible.
(ii) Data hiding is possible and use in the systems where needed.

Disadvantages-
(i) Final output does not match as requirements because of reusability
(ii) Its complicated to implement.

* Top- Down Approach

- In this approach we divide a whole system into subsystem and again devide these subsystem into further subsystems till we get the lowest level.

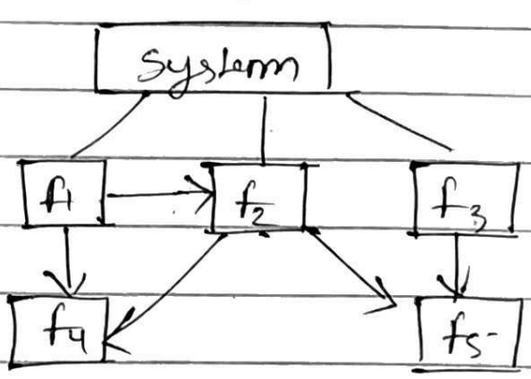


* Advantages-
(i) The final out would be same as requirements.
(ii) Irregularities are less

* Dis-advantages-
(i) Data hiding is not possible
(iii) Time Consuming
(iii) Coplexity increase with each branch

* Functional technique - In this technique shows what the system does in this method a system devide into subsystems called functions

oriented



* object oriented - In this we design on the basis of class, objects and relationship bet classes

UNIT - 7, 8

* Software Testing - It is a process of identify correctness, completeness and quality of developed computer software.

* Methods of Testing

(i) Static Testing - In this method of testing used to check defect in the software without executing the code.

→ Programming tool / editor check Syntax and code structure.

→ Code review and code walkthrough also comes in this category.

(ii) Dynamic Testing - In this method executing program code with a given set of test case and analyse input values and output values.

Static	V/s	Dynamic
(i) Perform in early stage		(i) Perform in later stage
(ii) Code is not executed		(ii) Code is executed
(iii) less costly		(iii) High costly
(iv) Discover variety of bugs		(iv) limited types bugs

(v) Prevents the defects } (v) Find and fix the defects

* Verification	V/s	Validation
(i) Are you building it right?		(i) Have you build it right?
(ii) Checking document design, Code and program		(ii) testing the actual final product.
(iii) Its static testing		(iii) Its Dynamic testing
~~(iv) include execution~~		
(iv) Do not includes execution of code		(iv) Includes execution of code.
(v) Done by developer		(v) Done by testers
(vi) It comes before validation		(vi) It comes after verification

* Black Box testing - Testing functionality of an application without looking into the internal structure or working.

In this method gives the input value and check the result is desired or not.

Black Box testing techniques ⟶

(i) Boundary value - in Input higher and lower values.

(ii) Equivalence class - divide into similar class if one element of class passes the test → ok

(iii) Cause effect graphing - combination value in input in systematic way.

(iv) Pair wise - multiple parameters pair wise with different values.

(v) State based - Based on state Active or in-active.

* White Box testing - tests internal structure and working of pgm at code level

→ Techniques-

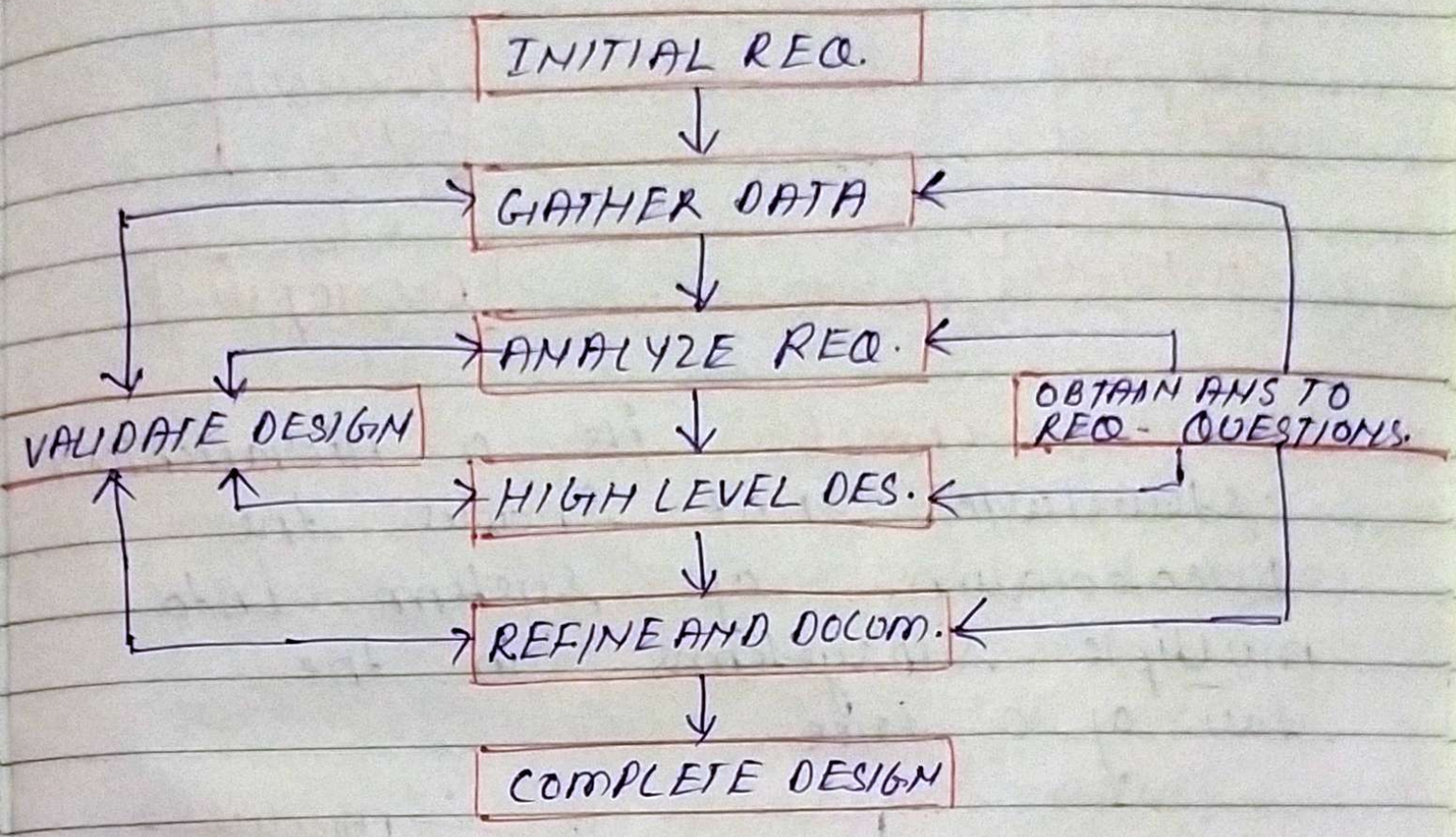
(i) Control flow testing - branch condition True/False

(ii) Data flow " = where data declared and defined and where used to change.

(iii)

UNIT - 5 XTRA

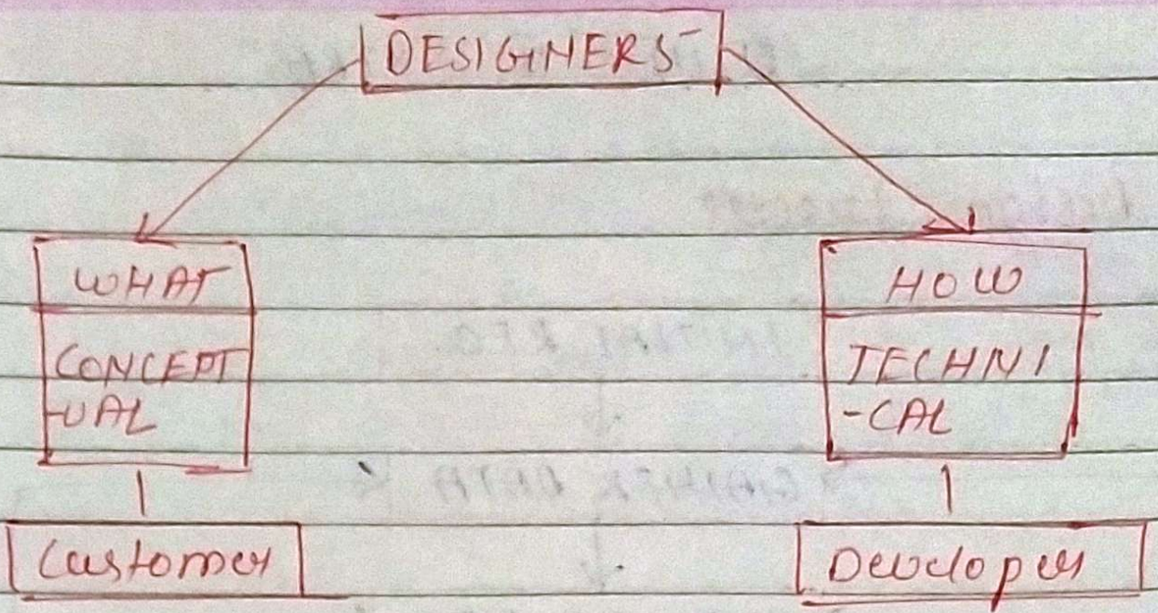
* Design Process -



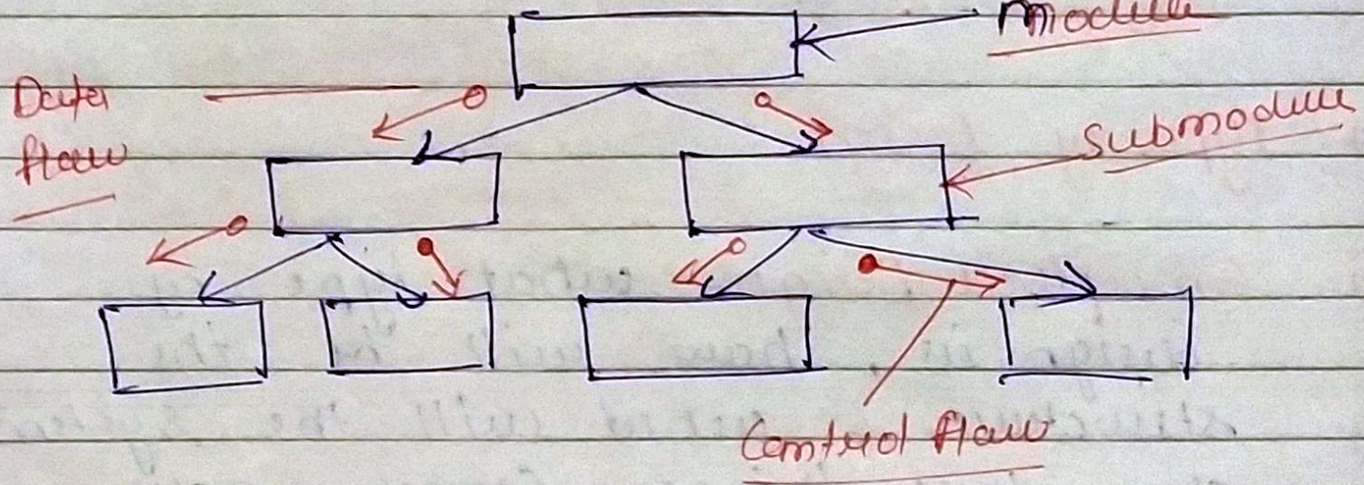
* Types of Design

(i) Conceptual Design - what type of design is, how will be the structure, what will the system do. these things comes under Conceptual design.

(ii) Technical design - How the project is impliment, what is the software and hardware needs all comes under technical design. like - Hardware, software, Network I/O, etc.



* Structure Chart- its a Hierarchical structure that shows the breakdown of system into multiple subsystems in the form of a tree



UNIT 7, 8 XTRA

* Testing Levels -

(i) UNIT Testing - This testing performed by developers while software is in development phase unit by unit under white box testing technique.

(ii) Integration Testing - This testing performed in testing phase when software is ready to test, in this testing we unite all the units and test the functionality and find out bugs, this is done by testers.

Example - Bang Bang, Top down, Bottom up, mixed.

(iii) System testing - This testing performed when final product is ready the we test the software as whole

Example - Alpha testing, Beta testing Acceptance

(iv) Regression Testing - This testing is performed in maintenance phase for upgradation with time to check that all features are working after upgradation are not.

* (v) Acceptance Testing - This testing is performed to check the acceptability, that the final product is acceptable by client according to their requirements or not

* Software quality assurance - This method is used to ensure that the final product is meet all the standards and objectives for quality.

* Software quality Control - It is a set of procedure used to ensure that software meets its quality goals at the best value to the client.

* Software Audit - A type of review in which one or more members of who are not part of development team conduct an independent examination of software

* Software maintenance - Software maintenance is a very important part of SDLC that occurs after deployment of final software it is used to increase the performance, functionality and reliability of the software with time.

→ Need of maintence -

→ To
- Correct faults
- Improve performance
- Improve reliability
- Improve functionality
- Provide long term support

* Types of maintenance -

(i) Corrective maintenance - It occurs when fault is already arrived in order to fix that.

(iii) Adaptive maintenance- This program when we need to update the software in order to make it adaptable of new changes.

(iii) Perfective maintenance- It program to give a software long term support with new functionalities.

(iv) Preventive maintenance- This program aimed to prevent the software from in future problems.

* Software Reverse Engineering-
It is process or reverse process to obtain the source code from actual final product - with the help of documentation and design.

→ Need
→ To
- Learn
- Security auditing
- Addition of features
- Get idea for new product

* Software Configuration management

In this process we systematically organize, manage and control the changes in code, document and other entities by SDLC

→ why we need SCM

→ because

1. Continously updating software
2. change in user requirements
3. there are multiple machines and OS.
4. It also control the cost invoked in making changes.