Date / /

Software Engineering

1. Intro
2. SDLC
3. Requirement analysis
4. Software Project Management
5. Software Design
6. Coding and Testing
7. Maintenance
8. Quality Management, Reuse.

10 **Software Engineering Definition and Evolution :-**

→ Engineering approach to develop a Software.

Evolution :-

1945-65 → origin

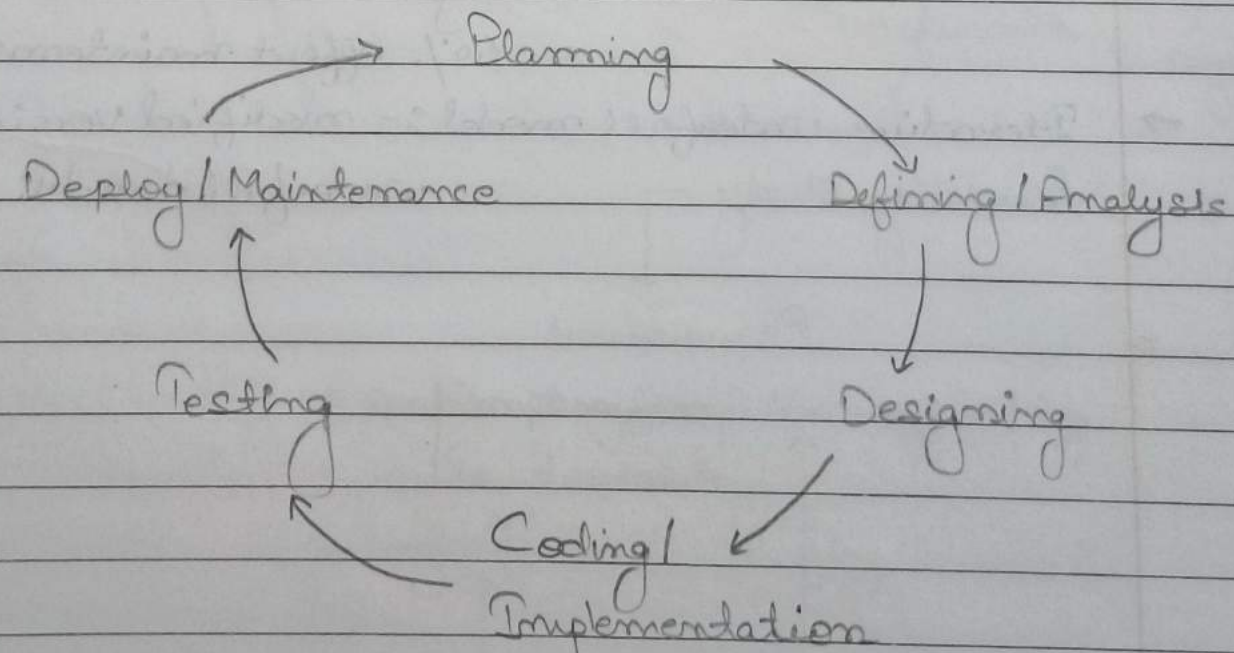
15 1965-85 → Crisis

1990-2000 → Internet

2000-2010 → Light weight.

2010 - Till date → AI, ML, DL

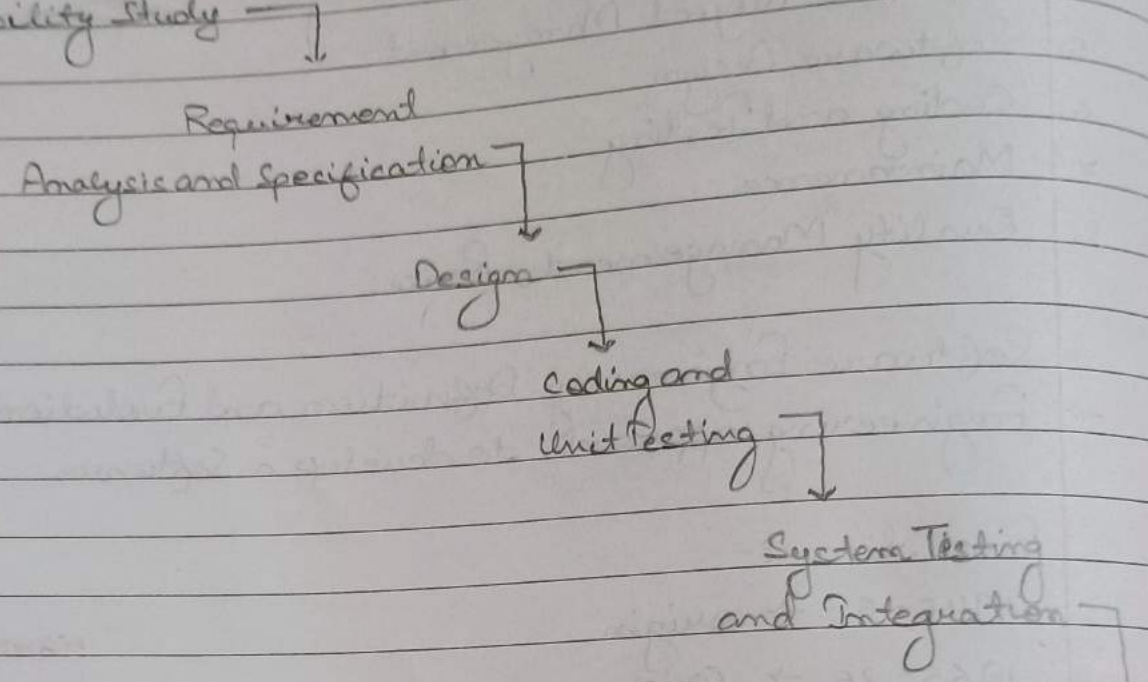
→20 SDLC :-



25

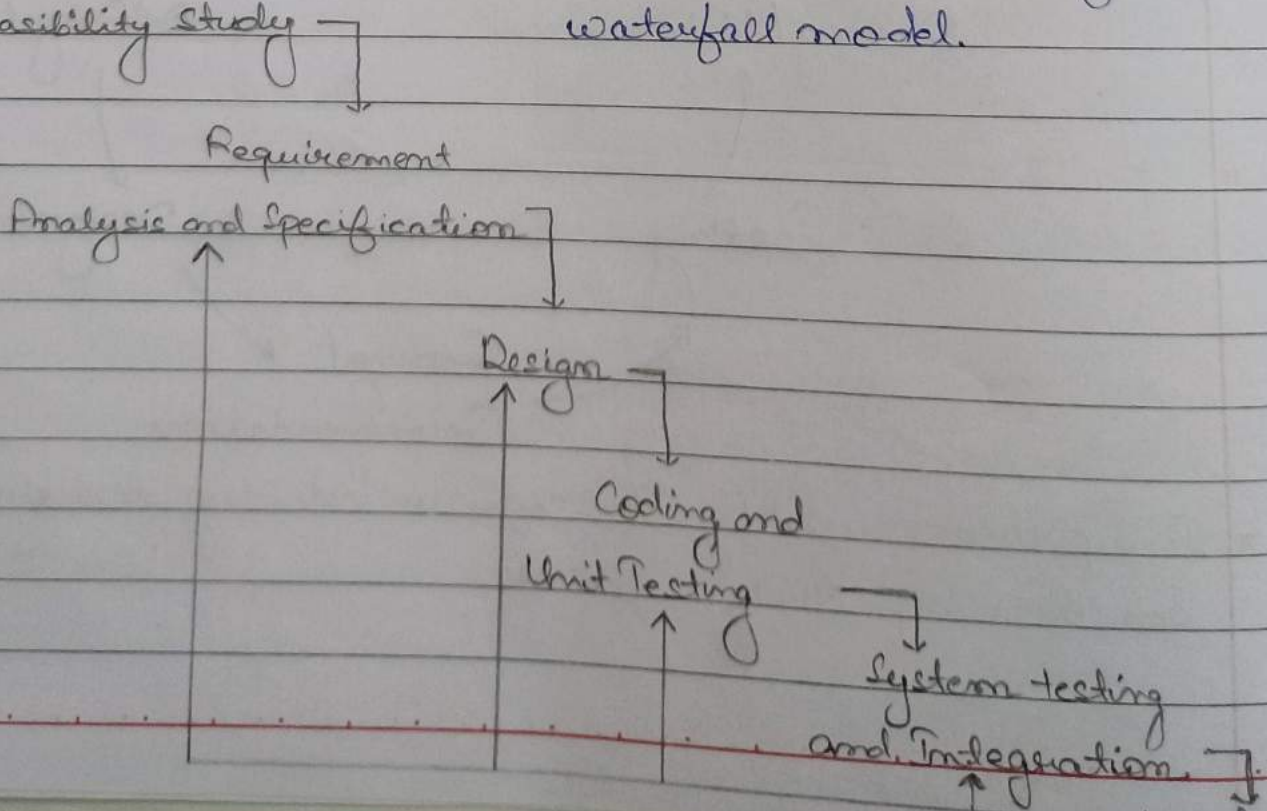
30 Is a process used by the software industry to design, develop and test high quality software.

→ Classical Waterfall model :- It is a linear model sequential approach to the software development lifecycle that is popular in SE and product development.



Advantage	Disadvantage
• Base model	• No feedback
• Simple and easy	• No experiment
• Small and Project	• No parallelism
	• High Risk
	• 60% effort maintenance

→ Iterative waterfall model :- modified version of classical waterfall model.

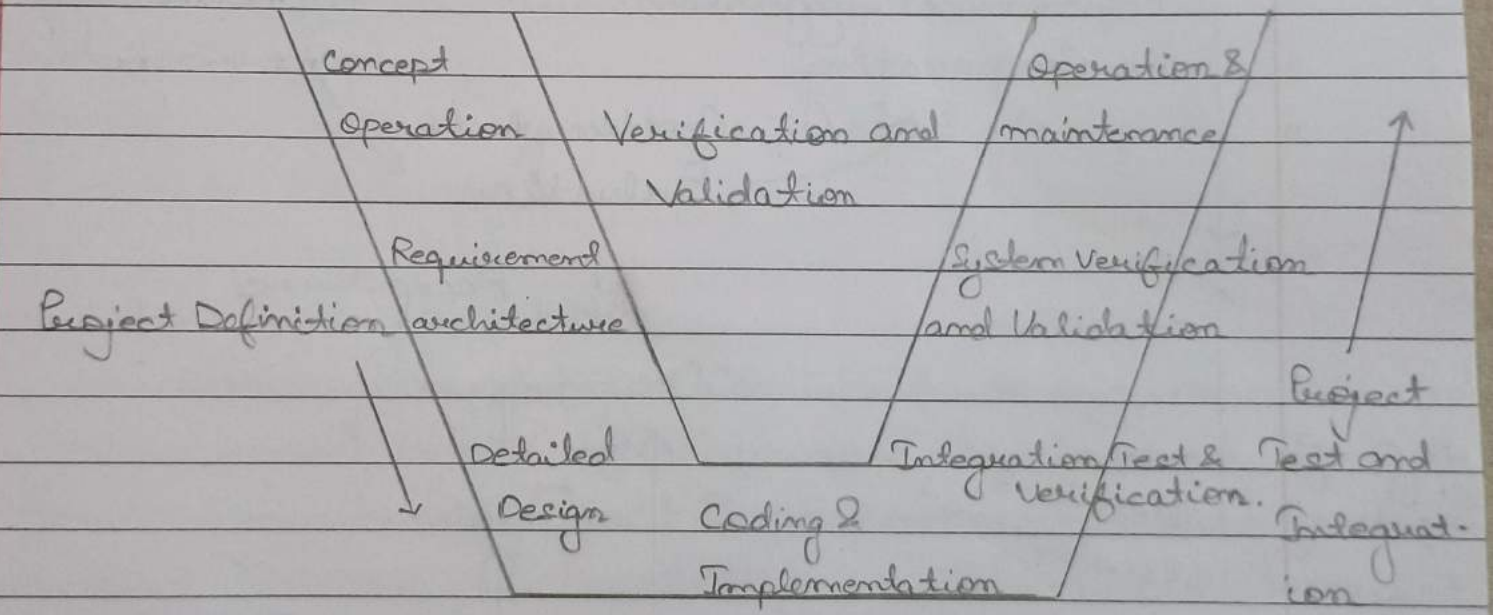


Camlin Page
Date / /

Advantage :-
- Base model
- Simple easy
- Small Project
- Feedback

Dis Disadvantage :-
- No phase over lapping
- No intermediate delivery
- Rigid
- ~~less~~ less Customer Interaction.

→ V model in SDLC :-
Also known as verification and validation model
Extension of waterfall model
Testing in every phase.
Verification phase
Validation phase

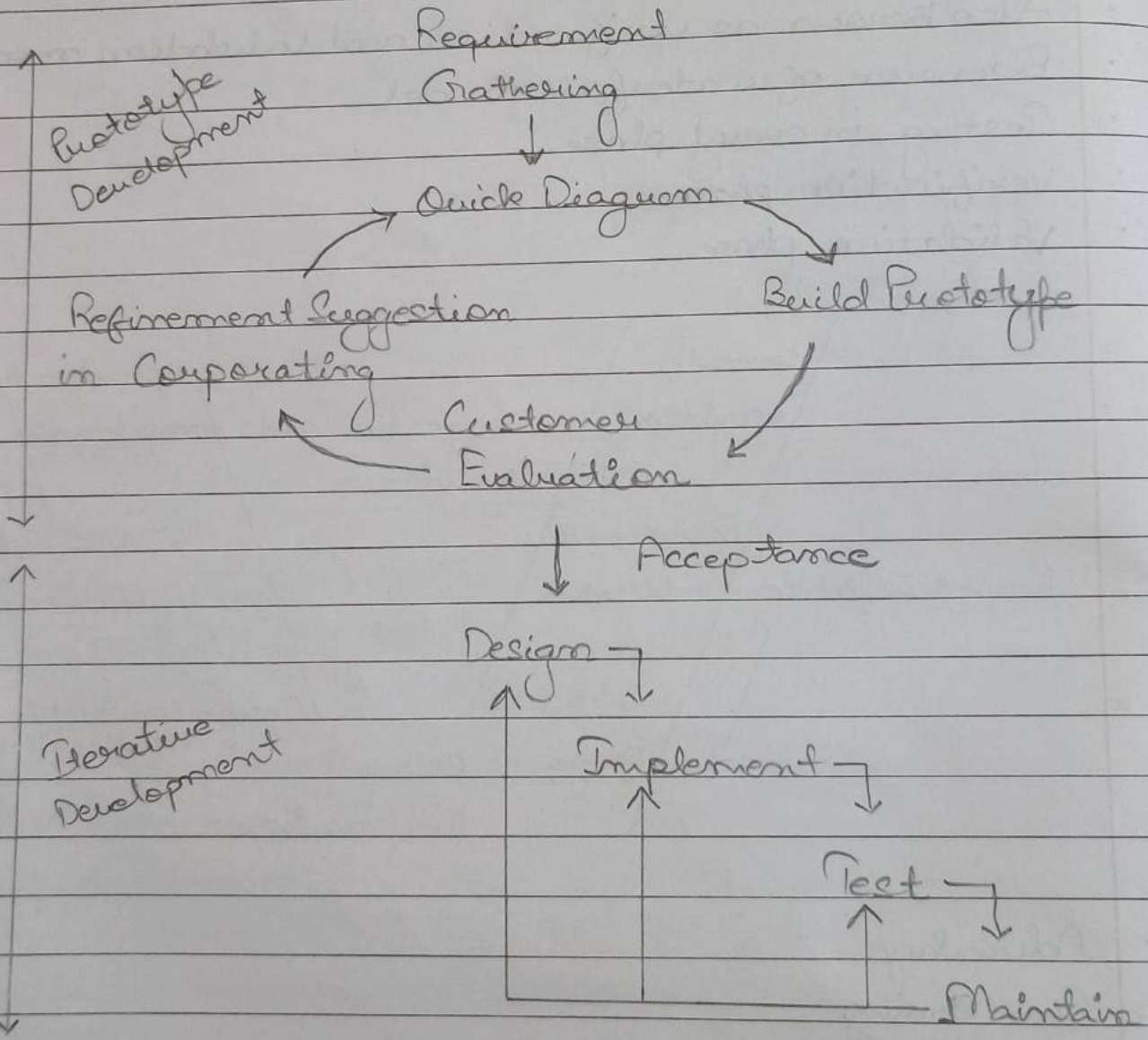


Advantage
Time Sharing
Good understanding of ~~people~~ project in the beginning
Every Component must be testable
Progress can be tracked easily.
Proactive defect tracking.

→ Disadvantage :-

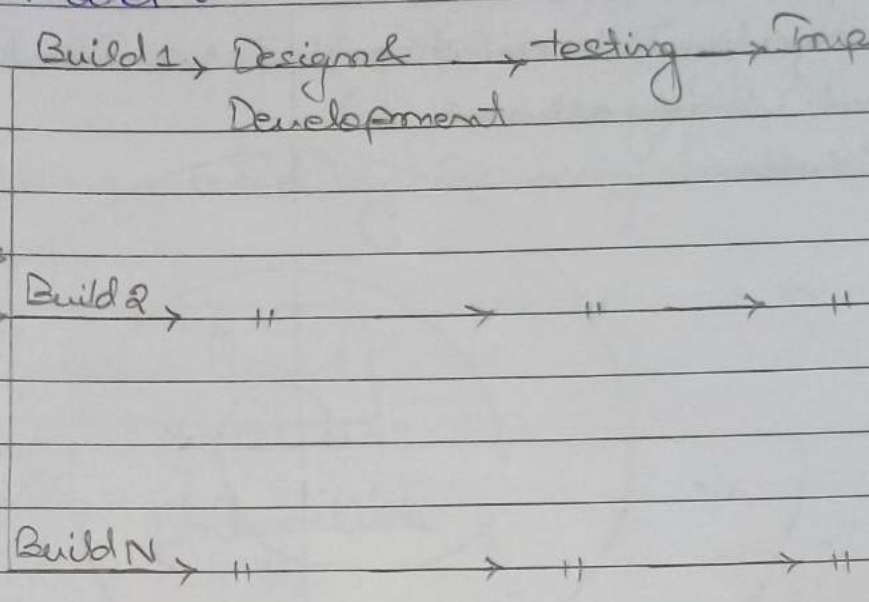
- No feedback so less scope of changes
- Risk analysis not done
- Not good for big or Object Oriented project.

→ Prototyping model :- It is a dummy model.



The prototyping model is a systems development in which a prototype is built, tested and then reworked as necessary until an acceptable outcome is achieved from which the complete system can be developed.

→ Incremental model :-



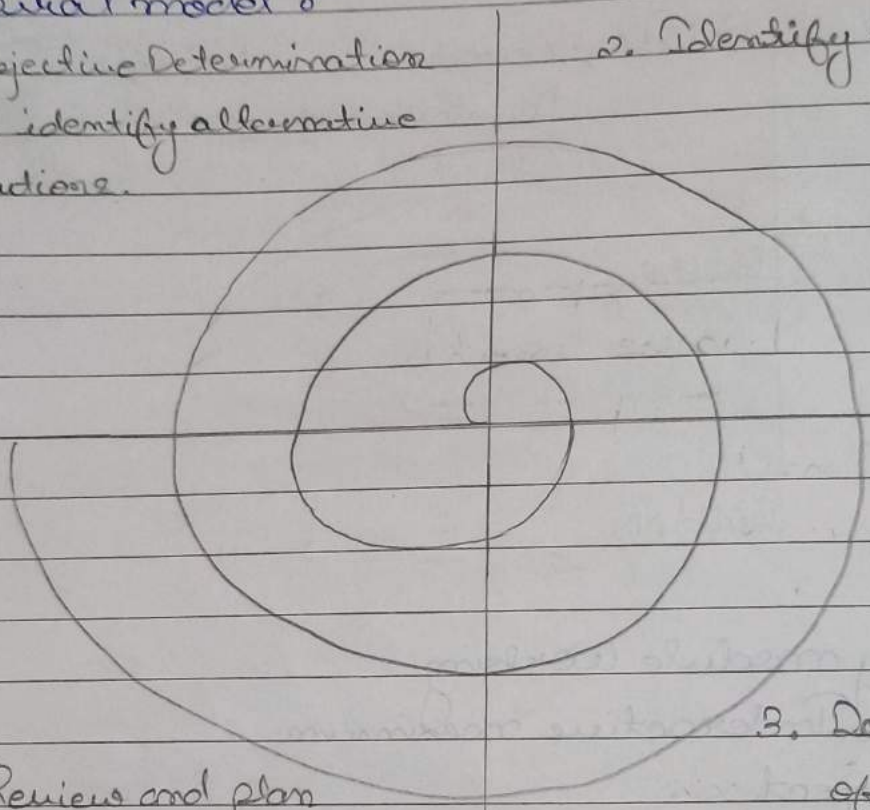
- Module by module working
- Customer Interactive maximum
- Large Project
- Beneficial for early release project demand.
- Flexible to change.

→ Evolutionary model :-

- Evolutionary model is a combination of iterative and Incremental model of software development life cycle
- Incremental model first implement a few basic features and deliver to the customer. Then build the next part and deliver it again and repeat the step until the desired system is fully realized. No longer 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".

This method used for Risk management that combines the iterative development process model with elements of waterfall model.

→ Spiral model :-



- Risk Handling
- Angular Dimension = Progress
- Meta model
- Radius of Spiral = Cost

→ Advantage :-

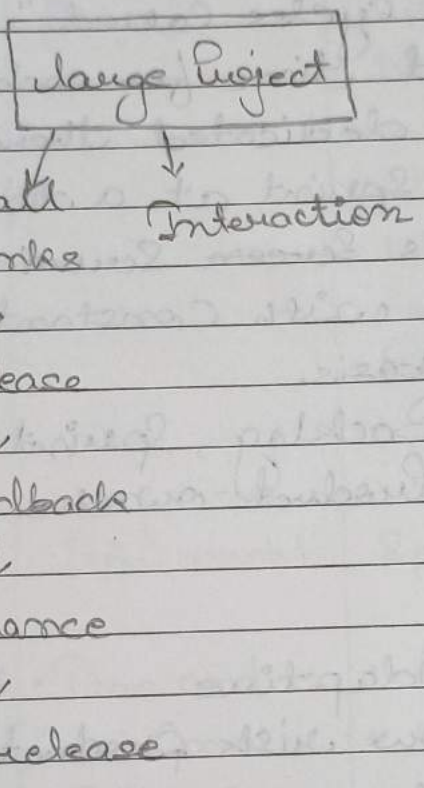
- Risk Handling
- Large Projects
- Flexible
- Customer Satisfaction.

→ Disadvantage :-

- Complex
- Expensive
- Too much Risk analysis
- Time.

Date / /

→ Agile model :-
Agile (move quickly) → meaning
latest technology used by company to develop software.



→ Advantage :-
- Frequent Delivery
- Face to face Communication with client
- changes
- Time less

→ Disadvantage :-
- less documentation
- maintenance Problem.

→ Scrum model :-

- One of the most popular agile methodology.
- So Scrum is a light weight, iterative and incremental framework.
- Scrum breaks down the development phases into stages or cycles called "Sprints".
- The development time for each Sprint is maximized and dedicated there by managing only one Sprint at a time.
- Scrum team has ~~Screen~~ Scrum master and product owner with constant communication on the daily basis.
- Keywords :- Backlog, Sprint, Daily Scrum, Scrum master, Product owner.

→ Advantage :-

- Freedom and Adaptive
- High Quality, low risk product.
- Reduce the development time up to 40%.
- Scrum Customer Satisfaction is very important.
- Reviewing the current Sprint before moving to new one.

→ Disadvantage :-

- More ~~eff~~ efficient for small team size.
- No Changes in the Sprint.

Classical Waterfall	Iterative Waterfall	Prototype Model	Incremental model
• Basic • Rigid • Inflexible • Not for real Project	• Basic • Problem is well under-Stood.	• User requirement is not clear. • Costly. • Not early on requirement. • High user involvement. • Reusability.	• Module by module delivery. • Faulty to test debug.

Evolutionary model	RAD model	Spiral model	Agile model
• Large project	• Time and Cost Constraint. • User at all level. • Reusability	• Risk • Not for Small project • No Faulty lock on requirement • Less Experience can work.	• Flexible • Advanced • Parallel • Process divided into Sprints

→ Software Requirements :-

- It is the description of features and functions of the target System.
- It is is the description of what the System should do.
- Requirement Engineering refers to the process of defining, documenting and maintaining requirements in the engineering design process.

→ It has four step :-

- Feasibility Study
- Requirement Gathering.
- Software Requirement Specification. SRS
- Software Requirement Validation. SRV

* Software Management :-

Tools Support for Requirement Engineering :-

- Observation reports
- Questionary.
- Use Case
- User Stories
- Requirement Workshops
- Mind Mapping
- Role playing
- Prototyping

→ Functional Vs Non-functional requirement :-

Requirement, which are related to functional / working aspect of Software fall into this Category.

* Non-functional requirement are expected characteristics of target Software [Security, Storage, Configuration, Performance, Cost, Interoperability, Flexibility, Disaster Recovery, Accessibility]

→ Software Requirement Specification (SRS)

- SRS is a description of a Software System to be developed.
- It lays on functional and non-functional requirements of the Software to be developed.
- It is a set of use cases that describes user interaction that the Software must provide to the user for perfect interaction.

1. Introduction :-
   - Purpose
   - Intended Audience
   - Scope
   - Definitions
   - References

2. Overall Descriptions :-
   - User interface
   - System interface
   - Constraints, assumption and dependencies
   - User characteristics.

3. System feature and Requirement :-
- Functional requirement
- Use Cases
- External interface requirement
- Logical database requirement
- Non functional requirement

4. Deliver for Approval :-

→ User Requirement :-
- Easy and simple to operate.
- Quick response
- effectively handling operational errors
- Customer Support.

→ User Requirement Specification :-
- The user requirement document (URD) or user requirement Specification (URS) is a document usually used in Software engineering that Specifies what the user expects the Software to be able to do.
- It is a ~~large~~ legal Contractual agreement.

→ DFD :- (Bubble chart) { data flow diagram }
- A graphical tool, useful for Communicating with users, managers and other personnel.
- Useful for analog analyzing existing as well as proposed System.
- Focus on the movement of data b/w external entities and process and b/w processes and data Stores.
- A relatively simple technique to learn and use

Q Why DFD?

→ Dfd provides an overview of
- what data a system processes
- what transformation are performed.
- what data are stored.
- what results are produced.
- Graphical nature makes it good communication tool b/w -
  a) User and analyst.
  b) Analyst and system designer.

→ DFD element :-
- Source / Sinks (External entities) ☐ ← Sign
- Data flow ⟶ Sign
- Processes ○ → Sign
- Data Stores ═ → Sign

→ External Entities :-
Source - Entity that supplies data to be stored.
Sink - Entity that receives data from the system.

External Entities

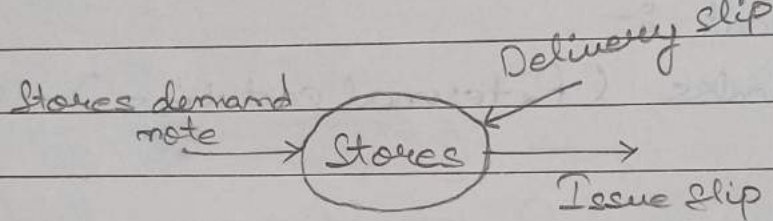
- A rectangle represents an external entity.
- They either supply or receive data
- They do not processes data.

→ Data flow :-

- marks movement of data through the system.
- a pipeline to carry a data.
- Connects the process, external entity and data stores.
- General unidirectional, if same data flows in both directions ~~dou~~ double-headed arrows can be used

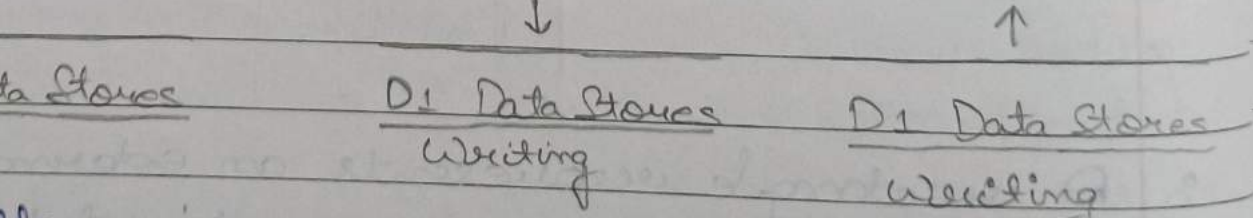
Data Flow :-

Processes



- A Circle represents a processes
- Straight line with incoming arrows are input data flows.
- Straight lines with outgoing arrows are input output data flow.
- Labels are assigned to data flow.

→ Data Stores :-



Data Stores is a repository of data.
Data can be written into the data store. This is depicted by an incoming arrow.
Data can be read from a data store. This depicted by an outgoing arrow.

- External entity cannot 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 cannot flow from :-
  - → External entity to external entity.
  - → External entity to store
  - → Store to external entity.
  - → Store to store.

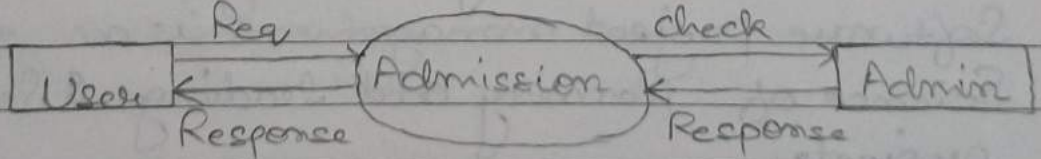
→ Levels of DFD :-

- DFD level 0 also called Context diagram.
- DFD level 1.
- DFD level 2.

It can be more of level 2 means 3, but it is very uncommon.

DFD for University Management System

level-0



It represents the entire software requirement as a single process with input and output data denoted by incoming and outgoing arrows.

If we add some more feature then it becomes the DFD level-1 and few more are added and they give more description then it becomes level-2.

→ logical and Physical DFD :-

- DFDs considered so far are called logical DFD
- A physical DFD is similar to a document flow diagram
- It specifies who does the operations specified by logical DFD.
- Physical DFD may depict physical movements of the goods
- Physical DFDs can be drawn ~~flat~~ fact gathering phase of life cycle.

→ Function Oriented vs Object Oriented Design :-

F.O Design	O.O Design
• System is designed from a functional viewpoint.	• System is viewed as a collection of objects.
• Top-down decomposition.	• Bottom-up approach
• Divide and Conquer approach	• UML is used
• DFD is used	

→ SPM (Software Project Management)

Software project management is an art and 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 project or team.
- Various factors make this job very complex.

→ Job responsibilities of Project Manager :-

- Planning
- Organizing
- Staffing
- Directing
- Monitoring
- Controlling
- Innovating
- Representing.

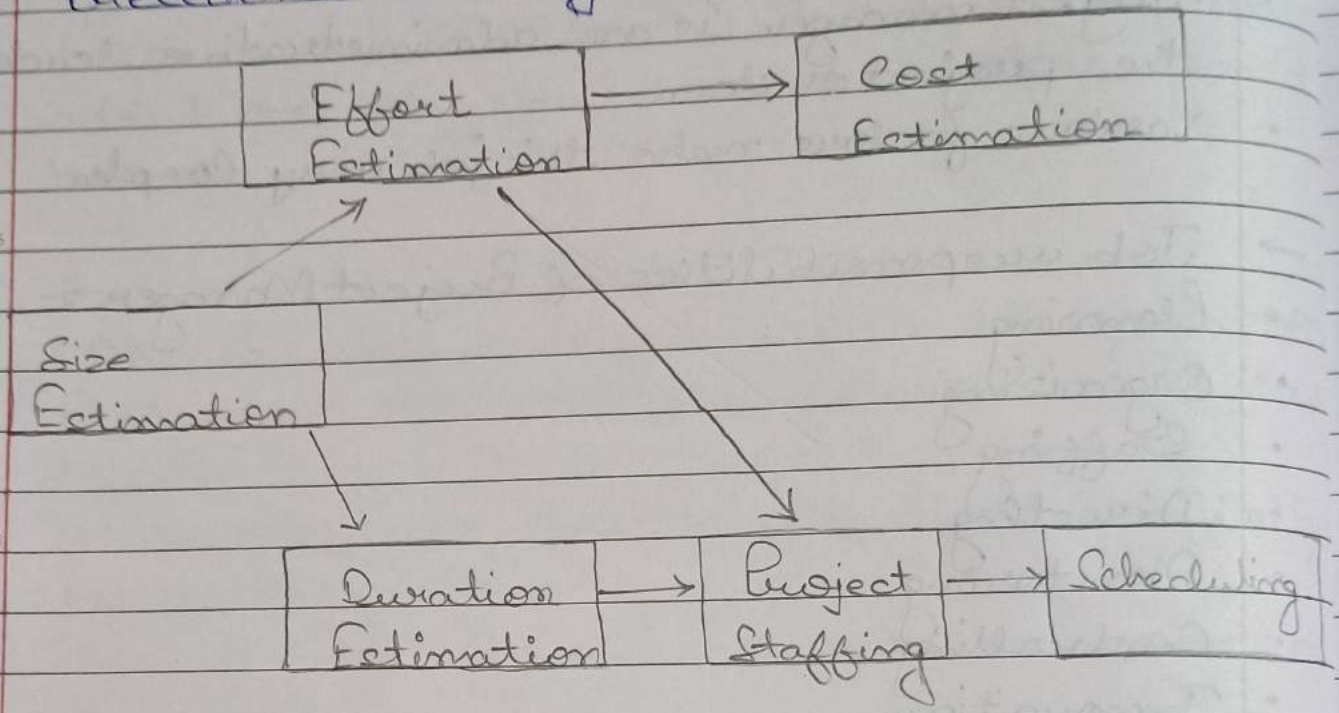
→ Skills required for Project Manager :-

- Managerial Skills
- Technical Skills
- Problem Solving Skills
- Conceptual Skills
- leadership Skills
- Communication Skills

→ Project Planning :-

- Estimation → Costing.
- Staffing → Staff plans
- Scheduling manpower and other resources.
- Risk management
- Miscellaneous Plans.

→ Precedence Ordering :-



→ Risk Management :-

- A risk is an uncertain event or condition that, if it occurs, has a positive or negative effect on a project's objectives.
- A risk management plan is a document that a project manager prepares to foresee risks, estimate impacts and define response to risks.

→ Reactive vs Proactive Management Strategies :-

- Reactive risk management tries to reduce the damage of potential threats and speed an organizations recovery from them but answer that those threats will happen eventually.
- Proactive risk management identifies threats and aim to prevent those events from ever happening in the first place.

→ Types of risks :-

- Business Risk :- Building a product that no one wants or loose budgetary commitments.
- Technical Risk :- Concerned with quality, design, implementation, interface, maintenance problems.
- Project Risk :- Concerned with Schedule, Cost, resources, Customer related issue.

→ Risk Assessment :-

- It is a process to rank the risk in terms of their damage.
- Determine the average probability of occurrence value for each risk.
- Determine the impact for each component based on impact assessment matrix.
- Risk assessment value on determined by multiplying scores for the probability and security value together.

$$RA = P \times S$$

↓

Risk Assessment

→ Risk Control vs Risk Mitigation :-

- Risk Control is basically the specific actions to reduce a risk events probability of happening. For example, Correct inspection and maintenance of the car reduce the likelihood of mechanical failures.
- ~~wher~~ whereas Risk Mitigation is the set of actions to reduce the impact of Risk event. For example, Airbags are used to reduce the impact of an accident on the passenger and driver.

→ Risk Mitigation and Control Strategies :-

- Risk Avoidance
- Risk Transfer
- Risk Reduction
- Risk Monitoring.

→ Basic COCOMO and intermediate with Numerical :-

– Project estimation Techniques :-

Estimation of various project parameters is an important project planning activity. The different parameters of a project that needs to be estimated include -

- Project Size
- Effort required to Complete the project.
- Project Duration and
- Cost.

- Accurate estimation of the parameters is important for resource planning and Scheduling. Estimation Techniques can be classified as :-
  - Emphirical Estimation Techniques.
  - Heuristics Estimation Techniques.
  - Analysiscal Estimation Techniques.

→ Emphirical Estimation Technique :-
- Emphirical estimation techniques are based on making an ~~edu~~ educated guess of the project parameters and common sense.
- This technique is based on prior experience of development of similar product and projects.
- An educated guess based on past experience.
- Two popular emphirical estimation technique are :-
  i) Expert Judgement Technique.
  ii) Delpi cost Estimation.

i) Expert Judgement Technique :-
- In this an expert makes an educated guess of the problem size after analyzing the ~~prob~~ problem ~~thro~~ thoroughly.
- The expert estimates the cost of the different component of the system :- eg - GUI.
- Combines them to arrive at the overall estimate.

ii) Delpi Cost Estimation :-
- Is carried out by a team of a group of experts and a coordinator.
- The Coordinator provides each estimator with

a copy of the SRS document and a form of recording list cost estimate.

- Estimator Complete their individual estimates anonymously and submit to the coordinator.

→ Heuristic Techniques :-

- In Heuristic Techniques the relationship that exist among the different project parameters can be modeled using suitable mathematical expression.
- Once parameter ide independent are known the dependent parameters can be easily determined by substituting the values of the independent parameters in the corresponding mathematical expression.
- assume that the characteristics to be estimated can be expressed in terms of some mathematical expression.
- Can be classified as Single variable and multivariable models.

→ Single variable models :-

$$\text{Estimated Parameter} = c_1^{2*} e^{d*_1}$$

- e is the characteristics of the Software which has already been estimated (independent variable)
- estimated parameter is the dependent parameter to be estimated.
- The dependent parameter to be estimated could be effort project duration, staff size etc.
- c1 and d1 are constant and are usually determined using data collection from historical data.
- The basic COCOMO model is an example of Single var

Variable cost estimation model.

→ A multivariable cost estimation model takes the following form

$$\text{Estimated Resource} = c_1 * e_1^{d_1} + c_2 * e_2^{d_2} + \dots$$

- where $e_1, e_2 \dots$ are the basic characteristics of the Software already estimated.
- $c_1, c_2, c_3, d_1, d_2 \dots$ are Constant.
- The intermidiate COCOMO model can be conciderred to be a exmple of a multivariable estimated model.

→ Analytical Estimation Techniques :-

- Analytical estimation technique derive the required result starting with certain basic assumption regarding a project.
- This technique have certain Scientific basis
- Halstead's Software Science is an example of this technique.

→ Halstead's Software Science :-

- An analytical Technique Halstead's Software Science is an analytical technique to measure size, development effort and development cost of Software products, Halstead used a few primitive program parameters to develop the expression for over all program length potential minimum value, actual Volume, effort
- Basically useful for estimating Software maintenance efforts.

→ COCOMO (Constructive Cost model)

- Was first proposed by DR. Barry Bohem in 1981
- Is a heuristic estimation technique which assumes that relationship among different parameters can be ~~mod~~ modeled using some mathematical expression.
- This approach implies that size is primary factor of cost other factors have lesser effect.
- "Constructive" implies that the complexity.
- COCOMO prescribes a three Stages process for project estimation.
- An ~~fi~~ initial estimate is obtained and over ~~next~~ next two Stages the initial estimate is refined to arrive at a more accurate estimate.

→ Project used in this model have following attributes :-

1. ranging in size from 2000 to 100000 lines of codes.
2. programming language ranging from assembly to PL/1.
3. These projects were based on the waterfall model of Software development.

→ Bohem stated that any Software development project can be classified into three categories :-

Organic :-

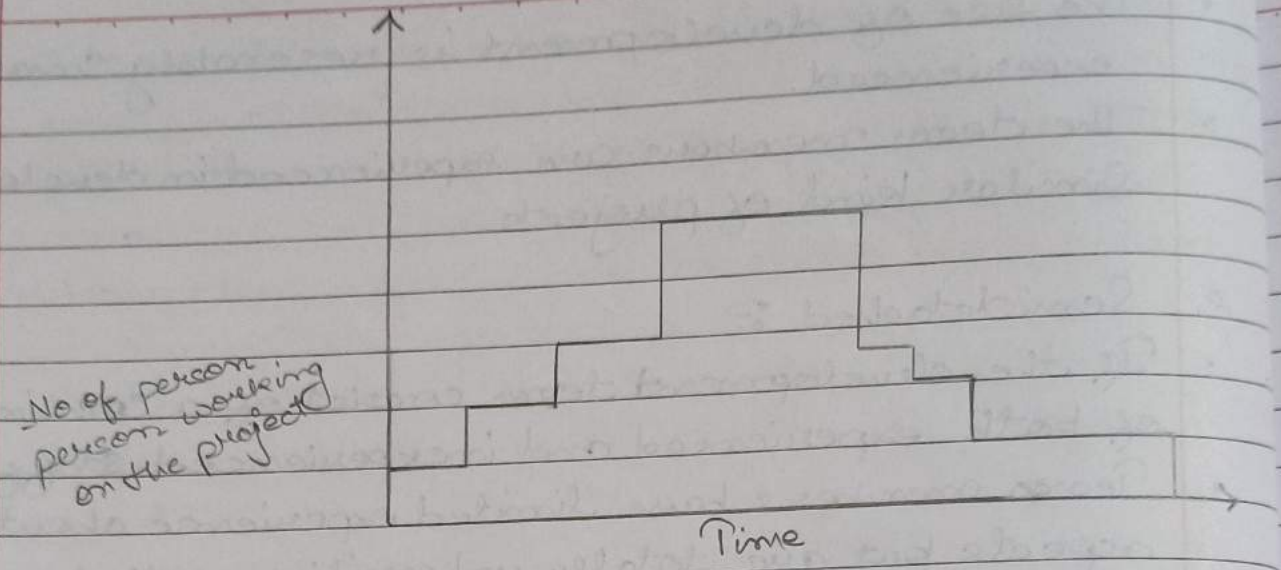
- If the project deal with developing a well understood application program.

- The size of development is reasonably small and experienced.
- The team members are experienced in developing similar kind of projects.

2. Semidetached :-
- If the development team consist of a combination of both experienced and inexperienced staff.
- Team members have limited experience about some aspects but are totally unfamiliar with some aspect of the system being developed.
- Mixed Experience.

3. Embedded :-
- If the software being developed is strongly coupled to complex hardware.
- Software project that must be developed within a set of tight software, hardware and operational constraints.

→ Person Month (PM) :-
- The effort estimation is expressed in units of person month.
- An effort of 100 PM does not imply that 100 persons should work for 1 month nor does it imply that 1 person should be employed for 100 months, but it denotes the area under the person month curves.
- It is the area under the person-month plot.



→ First Version : COCOMO 81

- This model parameters are derived from fitting a regression formula using data from 61 historical project.
- According to Boehm, Software Cost estimation should be done through three Stages :
  - i) Basic COCOMO
  - ii) Intermediate COCOMO
  - iii) Complete COCOMO

i) Basic COCOMO :-

This model Computes Software development effort, time and cost as a function of ~~pro~~ program size. Program size is expressed in estimated thousand of source line of codes (SLOC, KLOC)

$$\text{Effort} = a_1 \times (\text{KLOC})^{a_2} \text{ PM}$$
$$T_{dev} = b_1 \times (\text{EFFORT})^{b_2} \text{ Month}$$

Merits :-

Basics COCOMO is good for quick, rough and ~~easy~~ early estimate of Software Cost.

Demerits :-

- It does not account for difference in hardware constraints, personnel quality and experience, use of modern tools and techniques and so on.
- The ~~over~~ accuracy is limited of this model.

→ Intermediate cocomo model :-

- Intermediate cocomo computes software development effort as function of program size and a set of "cost driver" that includes subjective assessment of product, hardware, personnel and project attributes.
- This model refines the initial estimate obtained from the basic cocomo by scaling the estimate ~~up~~ up or down based on attributes of software development.
- This model uses a set of 15 cost drivers, these cost drivers are multiplied with the initial cost and effort estimate to scale estimate up and down.
- This extension considered a set of "cost driver" each with a number of subsidiary attributes.

i) Product attribute

- Required software reliability.
- Size of application database
- Complexity of the product.

ii) Hardware attributes

- Run time performance constraints
- Memory constraints
- Volatility of the virtual machine environment
- Required turn about time

Date / /

iii) Personal attributes :-
- Analyst Capability
- Software engineering capability
- Application experience
- Programming language experience.

iv) Project attributes :-
- Use of Software tools
- Application of SE methods
- Required development Schedule.

- Each of the 15 attributes receives a rating on a six point scale that ranges from "very low" to "extra high".
- Formula

$$E = a_1 (KLOC)^{(b1)} \times (EAF)$$

Merits :-
- This model can be applied to entire Software product for easy and rough cost estimation during early stage.
- It can be applied at the Software product component level for obtaining more accurate cost estimation.

Demerits :-
- The effort multipliers are not dependent on phases
- A product with many component is difficult to estimate.

→ Complete COCOMO model :-

- Incorporates all characteristics of the intermediate version with an assessment of the cost drivers impact on each step of the Software engineering process.
- The Complete COCOMO model considers the differences in characteristics of all the Subsystem and estimate the effort and development time as Sum of the estimates for the individual System
- Uses different effort multipliers for each Cost driver attribute. These phase Sensitive effort multipliers are each to determine the amount of effort require to complete each phase.
- The effort is Calculated as function of program Size and a set of Cost-drivers given according to each phase of Software life Cycle.
- Cost of each Sub-System is estimated Separately.
- Costs of Sub-Systems are added to obtain total Cost.
- Reduces the margin of error in the final estimate.

Example :-

- A management information System for an organisation having offices at Several places across the country.

a) Database part (Semi detached)
b) Graphical User Interface (GUI) part (Organic)
c) Communication part (embedded)

→ Verification vs Validation :-

Verification	Validation
1. Are you building it Right?	1. Have you build the Right thing?
2. check wheather an artifact conform its previous artifact.	2. Check the final product against Specification
3. Done by developer	3. Done by tester.
4. Concerned with phase Containment of error.	4. Aim is ~~no~~ to make final product.
5. methods involve : Review, Inspection, Unit Testing, Integration testing Static and Dynamic activities.	5. Involve System Testing. 6. Only Dynamic.

→ Testing :- levels of Testing

Software Testing.

Unit Testing [Jab wah dev phase mein hai, tab hota hai yeh]

Integration Testing
- Bang Bang
- Topdown
- Bottom up
- Mixed

Jab 2 unit ek sath integrate ho tab hota hai yeh testing jisse pta chle ki woh apas mein integrate sahi se hua hai ya nahi.

System Testing
Based on who is doing Testing
- alpha { Kudh }
- Beta { Some people }
- Acceptance { Customer }

Regression-testing
Performance / non functional.
- Volume
- Load
- Stress
- Configuration
- Compatibility
- Recovery
- Install

Full / poore Software ko testing karna tab woh hota System testing.

→ Cohesion and Coupling :-

Intro :-

A module consist of

i) Several function

ii) associated data structures

→ Modularity :-

- Modularity is a fundamental attributes of any good design.

Decomposition of a problem clearly into modules :-

1. modules are almost independent of each other
2. divide and conquer principle.

In technical terms, modules should display :-

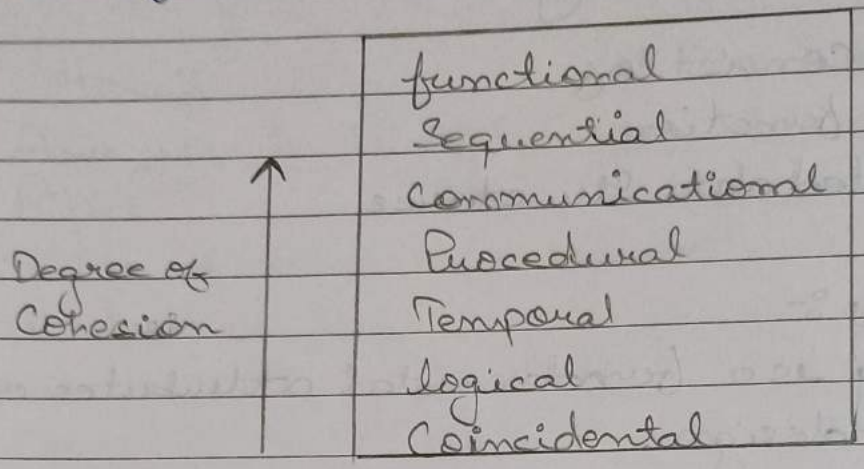
1. high Cohesion → within module
2. low Coupling → different module.

→ Cohesion is a measure of functional strength of a module. A Cohesive module perform a single task or function.

→ Coupling b/w two modules : a measure of the degree of interdependence or interaction b/w two modules.

→ A module having high Cohesion and low Coupling. functionally independent of other modules : A functionally independent module has minimal interaction with other modules.

→ Classification of Cohesiveness :-



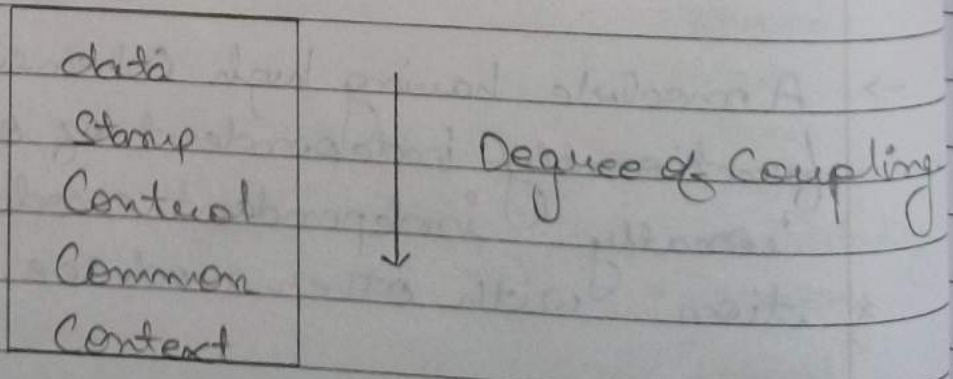
→ logical Cohesion :-

- All element of the module perform similar operations : eg- error handling, data input, data Output etc.
- when logically categorised elements are put together into a module, it is called logical cohesion.

→ Coupling :-

- Coupling indicates how closely two modules interact or how interdependent they are.
- The degree of coupling b/w two modules depends on their interface complexity.

Classes of Coupling



→ Characteristics of module Structure :-

Fan-in

- It indicates how many modules directly invoke a given module.
- High fan-in represents code reuse and is in general encouraged.

Depth

- no. of levels of Control

Width

- Overall Span of Control.

Fan-out

- a measure of number of modules directly Controlled by given module.

→ Unit Testing :-

- It is the process of checking small pieces of code to ensure that individual parts of a program work popularly on their own.
- Unit tests are used individual block of functionality.
- It is done by developers.

→ Integration Testing :-

- Integration Testing is Conducted to evaluate the the Compliance of a System or Component with Specified functional requirement.
- It occurs after unit testing and before System testing.

- Types :-
  - Bing Bang
  - Mixed
  - Top down
  - Bottom-up

→ System Testing :-

- System Testing is a level of Testing that validates the complete and fully integrated Software product.
- The purpose of System test is to evaluate the end-to-end System Specification.
- System testing is a black box testing.
- System testing is categories based on : who is doing the testing?
- System testing is categories based on : Functional | Non functional Requirement.
- It is performed by Testing team.

→ White Box vs Black Box Testing :-

W.B Testing	B.B Testing
• The developers can perform.	• The testing team perform
• what the Software is supposed to do also aware of how it does it.	• what the Software is supposed to do but is not aware of how it does it.
• To perform WBT, we should have an understanding of programming lang.	• no need to have understanding programming lang.
• we will look into the Source code and test the logic of code.	• we will verify the functionality of application based on the requirement Specification.

• developers should know internal design of the code • Test design technique: Control flow testing, Data flow testing, Branch testing etc. • Can be applied in unit testing.	• no need to know about the internal design of code. • Test design technique: Division table testing, All pairs testing, Cause & effect graph. • Can be applied virtually to every level of Software testing.

→ White box testing :

- Also known as clear box, glass box, transparent box and structural testing.
- Top white box testing tools - Numit, Cpp unit.

White Box test design technique :-

i) Control flow testing
ii) Data flow testing
iii) Branch testing
iv) Statement coverage
v) Decision Coverage
vi) Path testing

i) Statement Coverage technique :-

- Statement Coverage technique is used to design white box test cases.
- This technique involves execution of all statement of the source code at least once.
- It is used to calculate total no of created statement in the source code out of total statement present in the source code.
- This technique covers dead code, unused code & branches.

→ Data flow Testing :-

- It is one of the white box testing technique
- Data flow Testing focuses on two points -
  - In which statement the variable on defined
  - In which statement the variable are used.
- It design the test cases that cover control flow path around variable definitions and their uses in the modules.

Ex :-

```
1. read a, b, c ;
2. if ( a > b)
3.    x = a + 1
4.    print x
5. else
6.    x = b - 1
7.    print z
```

Variable	Define	Use
a	1	2,3
b	1	2,5
c	1	NA
x	3,5	4
z	NA	6

Advantage :-

- A variable that is declared but never use.
- A variable that is use but never declare
- A variable that is defined multiple times before it is use.
- Deallocating a variable before it is used.

→ Boundary Value Testing :-

- The black box testing techniques are helpful for detecting any errors or threats that happened at the boundary values of valid or invalid partition rather on focusing on the centre of the input data.

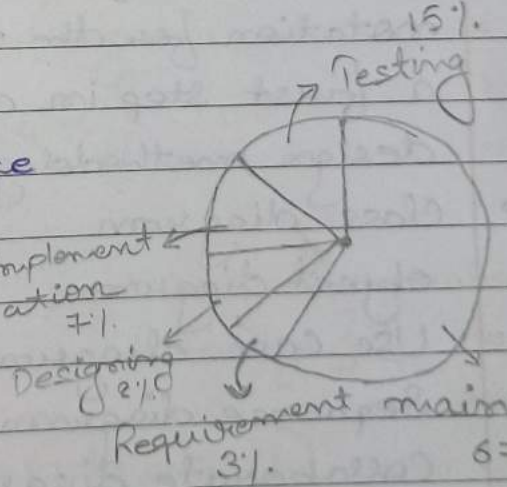
Test case no. 1 :-
Let us assume a test case that takes the speed of a car from 40 to 80 to get the best fuel efficiency.

Boundary Value Test Case

Invalid Test Case (min Value - 1)	Invalid Test Case (min, min+, max-, max)	Invalid Test Case Max Value + 1
39	40, 41, 79, 80	81

→ Software Maintenance :-
- Its primary goal is to modify and update software applications after delivery to correct errors and improve performance.
- Software maintenance is an inclusive activity that includes :-
  i) Error Correction
  ii) Deletion of obsolete Capabilities
  iii) Enhancement of Capabilities
  iv) Optimization

Types :-
i) Corrective maintenance
ii) Adaptive maintenance
iii) Preventive maintenance
iv) Perfective maintenance



→ Maintenance Metrics :-
1) Mean Time Between Failure :-
It is the average time available for a System or Component to perform its normal operations.
→ MTBF = (Sum of operational time / total no. of failures)

2) Mean time To Repair :-
It is the average time required to repair a failed Component.
→ MTTR = (Sum of downtime periods / total no. of failures)

Availability = (MTBF / MTBF + MTTR)

→ CASF example :-
- Planning and Requirement phase
- Design
- Coding
- Testing
- Web Development tools
- Quality Assurance Tools
- Maintenance

→ UML Diagram :-
UML (Unified modelling language) is a standard notation for the modeling a real world objects as a first step in developing an object-oriented design methodology.
- Class diagram
- object diagram
- Use case diagram
- Sequence diagram
- Collaborate diagram
- activity diagram
- Component diagram
- Deployment diagram.

→ Use case diagram :-

 - actor → roleplay
if real person

◯ - use case → Capability

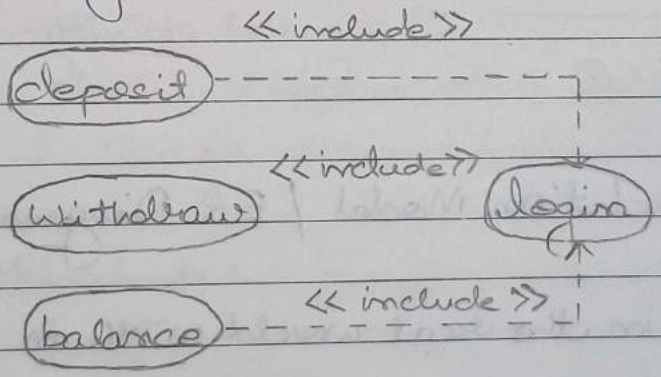
—— - connector → (interaction)

——→ - generalisation

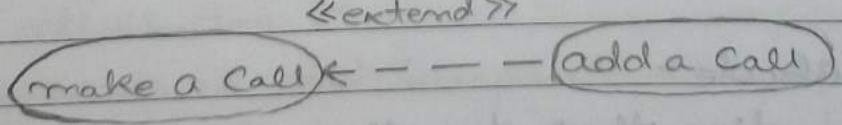
- - - - - → - Stereotype (relationship)

Things to be added in diagram

i) << include >> - Implicit function { compulsory jo hota uske liye }
eg: banking in online



ii) << extend >> - Explicit fn { jo compulsory nahi hota uske liye }



→ 5 must things for use case diagram :-

i)  actor
{person}

ii) << include >>
<< exclude >>

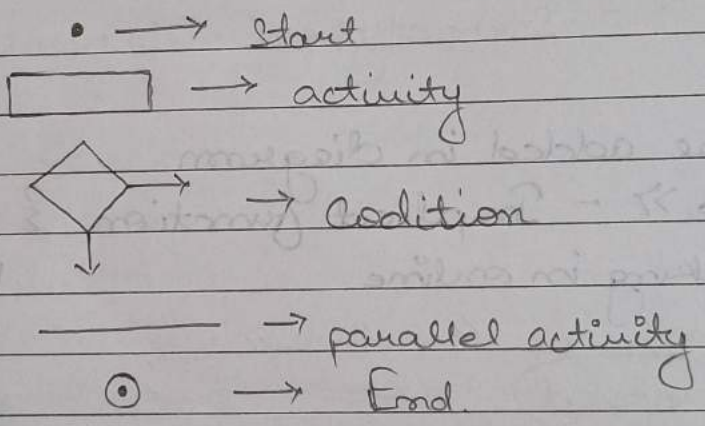
iii) Use Case ⬭

iv) Actors

v) System boundaries

→ Activity diagram :-

- It describes the dynamic aspects of the system.
- It is basically a flowchart to represent the flow from one activity to another activity.



→ Entity relation Model / ER Diagram :-

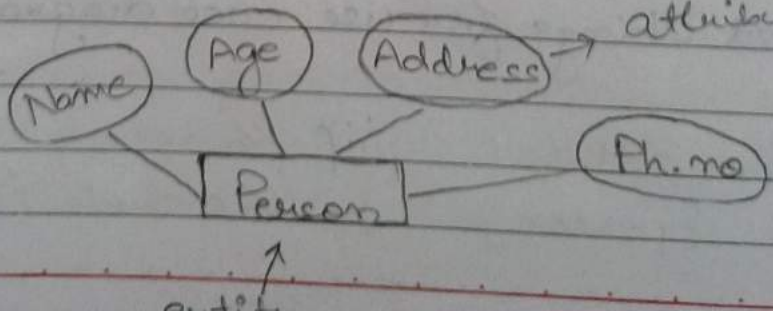
- Entity -

→ A thing in the real world with an independent existence.

→ Maybe an object with physical existence or with a conceptual existence

- Attribute -

→ Properties that describes the entities.



→ Composite attributes :-
- Can be divided into further part.
  eg - name

→ Simple attributes :-
- Cannot be divided further
  eg - weight

→ Single Valued attributes
- Have a ~~high~~ Single value ~~of~~ for a particular entity.
  eg - age

→ Multi Valued attributes
- Can have a set of values for a particular entity.
  eg - College degree, language known.

→ Derived attributes
- Can be be derived from other attributes.
  eg - age { Can be derived from D.O.B }

→ Stored attributes
- From which the value of others attributes are derived
  eg - Birthdate of person

* Complex attributes :-
- Has multivalued and Composite Component in it.
- Multivalued attributes → represented within '{ }'
- Composite attributes - represent within '( )'.
- Ex : { College Degree (College, year, Degree, Field)}

* Null Values :-
- Null is something which is not applicable a unknown.

ER diagram can express the overall logical structure of a database graphically.

ER diagram consist of following major components
- Rectangle
- Eclipse
- Diamonds
- Lines
- Double Ellipse
- Dashed Ellipse
- Double lines
- Double Rectangle

→ Software Quality Assurance :- SQA
- It is an umbrella activity for checking the quality of the software whether the software we are making is quality or not.
- SQA is not checked at any particular phases, it run along with software development. After every development process, software quality assurance is checked.
- Generally the quality the software is verified by the third party organisation like ISO, CMM to ensure that the software must contain quality or not.
- A software quality product has several quality :-

i) Portability
ii) Reliability
iii) Functionability

→ Benefits :-

- By using SOA, the quality of software is high.
- If the software is light quality then it will save our time and cost.
- Ease of use.
- Security.

→ Software Configuration System :-

- It is defined as a process of systematically manage, organise and control the changes in the document, code etc during the software development life cycle.

It defines the no. of tasks like -

- Software Configuration
- Version Control
- Change Control
- Configuration Audit
- Status report

→ Software Reliability :-

- also called operational Reliability.
- described as ability of system/component to perform its required function under static condition for a static period.
- Can be measured directly and estimated using historical data.

* measures /

$$MTBF = MTTF + MTTR$$

$$\text{Availability} = \frac{MTTF}{MTTF + MTTR} \times 100$$

→ Performance Testing :-

- It can determine how a system performs in terms of responsiveness and stability under a particular workload.
- A performance test measures stability, Speed, Scalability and throughput of your product.

i) Load Testing :- It evaluates the product's ability to perform to perform under increasing load.

ii) Stress Testing :- It is normally used to understand the upper limits of capacity within the system.

iii) Volume Testing :- It is basically testing the ability of a database management system to handle a large amount of data.

iv) Endurance Testing :- This type of testing evaluates the system performance over a long period of time, usually under normal or expected load conditions, to identify any memory leaks or performance degradation over time.

v) Response time testing :- This type of testing evaluates the time it takes for the system to respond to a request, with goal of determining the acceptable response time for end-user.

vi) Spike Testing :- tests the product reaction to sudden increase and decrease in the load.

→ Regression Testing :-
Regression testing is a type of Software testing that verifies that changes made to the System, such as bug fixes or new features, do not impact previously working functionality.

Types of Regression testing :-

- Full regression testing :- Testing the entire application from start to finish after changes have been made.
- Partial Regression testing :- Testing only those part of the application that were affected by the changes.

Logical DFD :- i) Depicts how the business works operates
ii) Represents business activities and functions
iii) Data is collected irrespective of how it is stored
iv) Represents the business control.

Physical DFD :- i) Depicts how the system will be implemented
ii) Represents manual processes, programs and program modules.
iii) Databases along with physical and manual files
iv) System control is depicted for validation of input data. to sap