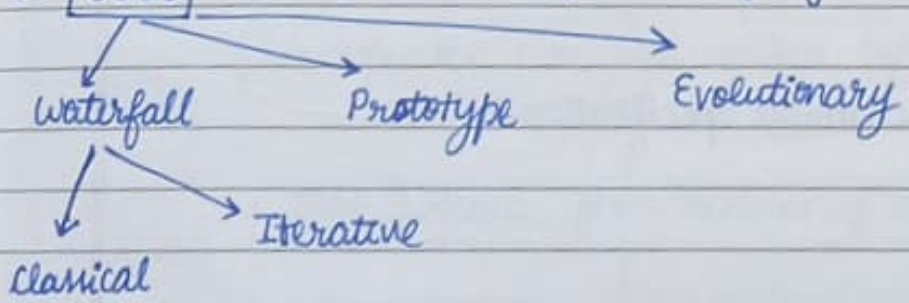
02/02/23 | Bloom's Taxonomy

> Software Engineering
> Software Crisis
> Factors contributing to Software Crisis
> Programs v/s software Products
> Object-oriented Design
> Evolution of other Software Engineering Techniques
> Differences b/w exploratory style & modern software development practice.
> CASE tools
> Life Cycle Model
> Waterfall Method [Classical]
↳ Requirement Gathering & Analysis (SRS)
↳ System Design
↳ Implementation
↳ Integration & Testing
↳ Deployment of system
↳ Maintenance (maximum efforts) — almost 60%.
• Advantages
• Disadvantages
↳ unidirectional
↳ minimal involvement of the client

> SDLC → Software Development Life Cycle



PAGE :
DATE : / /

03/02/23

→ [Toy implementation of a system]

> Prototyping Model [Dummy model to present it to the client]

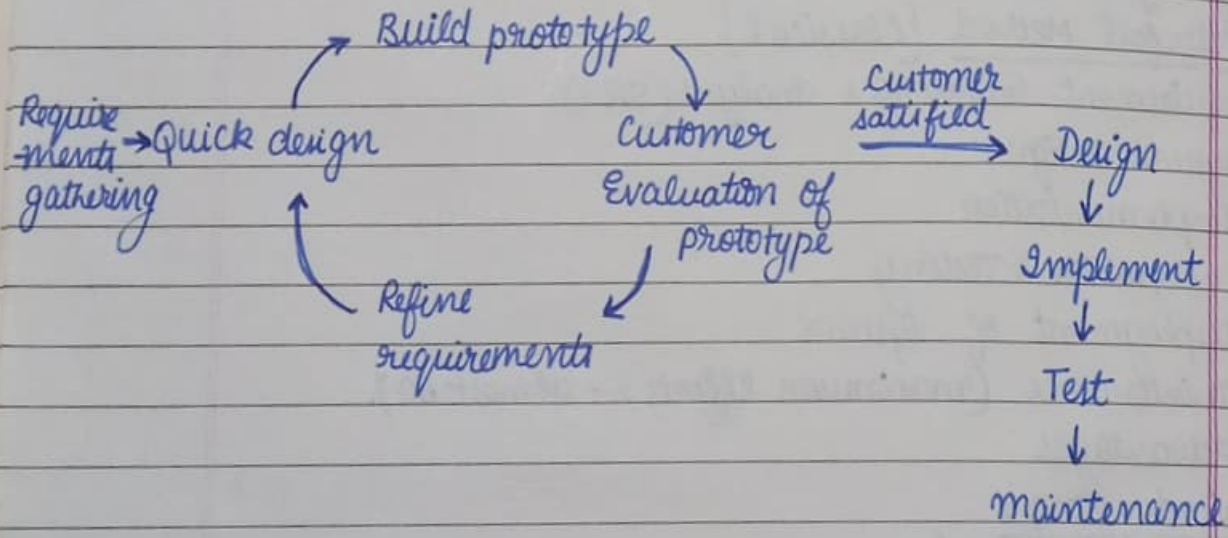
- limited functional capabilities
- low reliability
- inefficient performance

} Drawbacks of a prototype

> The actual system is developed using waterfall model approach.

Imp.

> ★ Prototyping Model :-



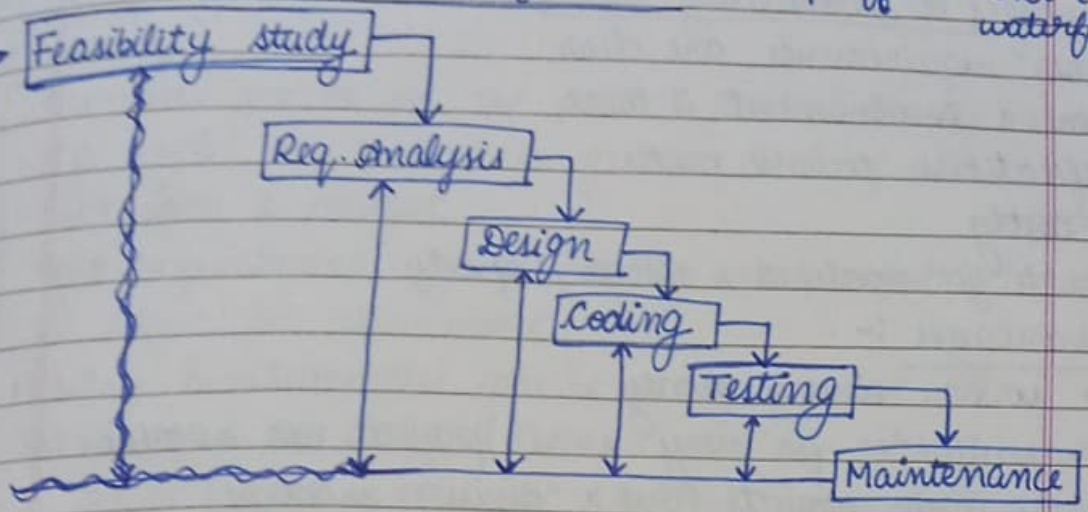
> Advantages of Prototype
> Disadvantages of prototype
> Refer to ppt.

★ SRS → Software Requirements Specifications

DATE : __/__/__
PAGE :

04/02/23

Iterative Waterfall Model :- * preffered over classical waterfall model



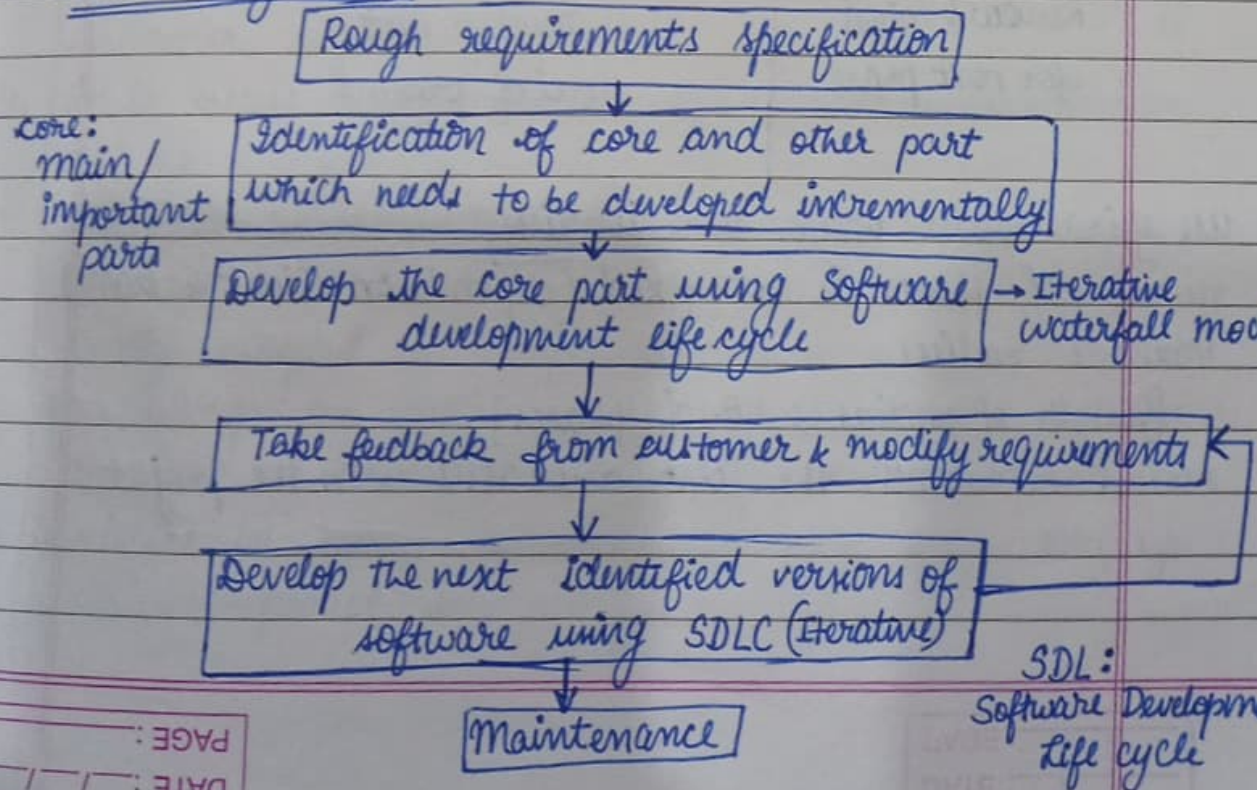
> Feasibility study ↴

capability to carry out the project, merits, demerits & requirements of the project.

* once a project is taken up, it can't be disgarded.

> Phase Containment of Errors :- finding errors in the same phase we are working with.

> Evolutionary model :-



PAGE : _____
DATE : __/__/__

> Advantages of Evolutionary model :-

- customer requirements are clear
- customer's involvement is there
- modifications can be made
- ~~less costly~~
- core parts get developed & tested properly

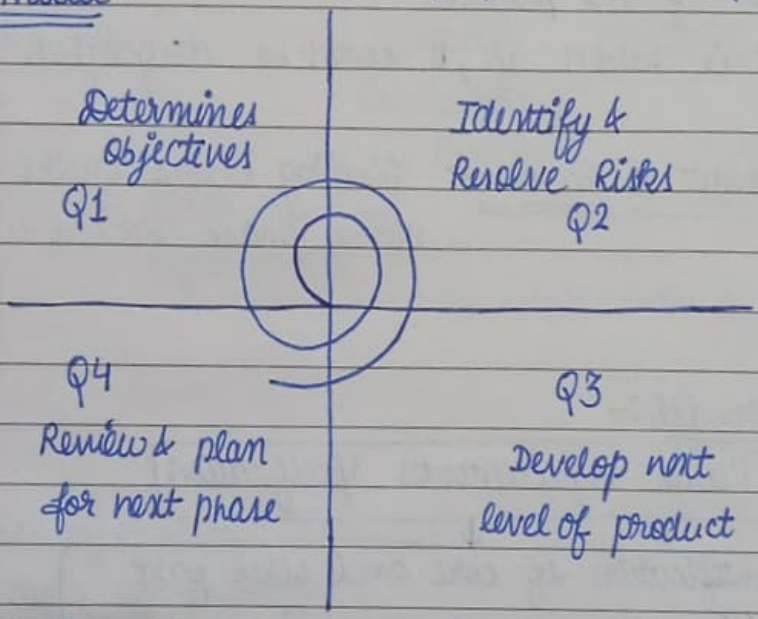
> Disadvantages :-

- costly w.r.t. time & money
- neither suitable for very small projects nor suitable for very large projects (due to division problem)

> Spiral Model :-

Q :- Quadrants

★ Proposed by Boehm 1988.



★ Use spiral model when there is risk.
★ number of phases is not fixed (depends on client & team)
★ variable radius
[length of radius ∝ no. of phases]
★ radius represents the cost associated with the project

PAGE :
DATE : / /

07/02/23

> Prototyping Model

#Advantages

1. Customers get to see the partial product early in the life cycle. This ensures a greater level of customer satisfaction & comfort.
2. New requirement can be easily accommodated as there is scope for refinement.
3. Missing functionalities can be easily figured out.
4. Errors can be detected much earlier thereby saving a lot of effort and cost, besides enhancing the quality of the software.
5. The developed prototype can be reused by the developer for more complicated projects in the future.
6. Flexibility in design.

#Disadvantages

1. Costly w.r.t time as well as money
2. There may be too much variation in requirements each time the prototype is evaluated by the customer.
3. Poor Documentation due to continuously changing customer requirements.
4. It is very difficult for developers to accomodate all the changes demanded by the customer.
5. There is uncertainity in determing the number of iterations that would be required before the prototype is finally accepted by the customer.
6. After seeing an early prototype, the customers sometimes demand the actual product to be delievered soon.
7. Developers in hurry to build prototypes may end up with sub-optimal solutions.

DATE : __/__/__
PAGE :

07/02/23

Revision for test

Software Engineering

- systematic collection of past experience:
  - techniques
  - methodologies
  - guidelines

Software Crisis

- software products:
  - fail to meet user requirements
  - frequently crash
  - expensive
  - difficult to alter, debug and enhance
  - often delivered late
  - use resources non-optimally

factors contributing to the software Crisis

- Larger problems
- Lack of adequate training in software engineering
- Increasing skill shortage
- Low productivity improvements

Programs V/s Software Products

1. Usually small in size	1. large
2. author himself is sole user.	2. large number of users
3. simple developer	3. team of developers
4. lacks proper user interface	4. well-designed interface
5. lacks proper documentation	5. well-documented & user-manual prepared
6. Ad hoc development.	6. systematic development.

PAGE
DATE : / /

Object-Oriented Design (80s)

Evolution of Software Engineering Techniques

- life cycle models
- specification techniques
- project management techniques
- testing techniques
- debugging techniques
- quality assurance techniques
- software measurement techniques
- CASE tools, etc.

Differences b/w the exploratory style & modern software development practices

- use of life cycle models
- stages of software development :-
  - requirements analysis
  - specification
  - design
  - coding
  - testing
- Emphasis has shifted
  - from error correction to error prevention
- detection of errors as close to their point of introduction as possible.
- during all stages of development process:
  - periodic reviews are being carried out
- Software testing has become systematic :
  - standard testing techniques are available
- projects are being thoroughly planned:
  - estimation
  - scheduling
  - monitoring mechanisms
- use of CASE tools

PAGE :
DATE : / /

Life cycle model

- a descriptive and diagrammatic model of software life cycle
  - identify all activities required for the product development
  - establishes a precedence ordering among different activities
  - Divides life cycle into phases
  - defines entry and exit criteria for every phase.
  - phase exit criteria for software requirements specification phase:
    SRS document is complete, reviewed & approved by the customer
  - It becomes easier for software project managers:
    to monitor the progress of the project.

Life cycle models

↳ Classical waterfall model
↳ Iterative waterfall model
↳ Evolutionary model
↳ Prototyping model
↳ Spiral model

DATE : / /
PAGE :

08/02/23

> Spiral model is a meta model. Why? (H.W)
> Comparison of Different life cycle models

> Feasibility Study → It is a measure of the software product in terms of how much beneficial product development will be for the organization in a practical point of view.

↳ Factors
↳ Types
- Technical [hardware & software requirements]
- operational [how easy product will be to operate & maintain]
- Economic [cost and benefit of the project]
- Legal [legal implementation of project] → license, copyright, data protection act
- Schedule [timelines / deadlines analysis]

09/02/23

> Requirements Gathering (Requirements Elicitation)

↳ collect requirements from stakeholders

1. Studying existing documentation [SOP document]
2. Interview [identify diff. categories of users] → Statement of Purpose
3. Task analysis [tasks being done by the software]
4. Scenario analysis [different situations that software might face]
5. Form analysis [forms filled by stakeholders]
   ↳ analysis of forms to determine data input & data output

> functional and non-functional Requirements

- functional: given by the client/user, *mandatory
- non-functional: we add to software ourselves
  - Security
  - Scalability
  - Performance (speed & responsiveness of the system)
  - usability

PAGE : / /
DATE : / /

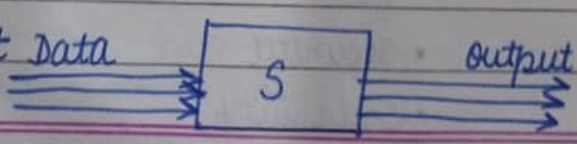
> Difference b/w functional & non-functional requirements

Functional Requirements	Non-functional Requirements
1. A system or its components.	1. the quality attribute of a software system.
2. It specifies "what should the software system do?"	2. It places constraints on "how should the software system fulfill the functional requirem
3. specified by user	3. specified by technical people
4. mandatory	4. not mandatory
5. Helps you verify the functionality of the software.	5. helps you verify the performance of the software
6. Usually easy to define.	6. Usually more difficult to defin
7. Example,	7. Example,
(i) Authentication of user	(i) Emails should be sent with a latency of no greater than 12 hrs from such an activity
(ii) System shutdown in case of a cyber attack.	(ii) The site should load in 3 seconds when no. of simultaneous users are >1000
(iii) Verification email is sent to user whenever she registers for the first time	

> Black-box Specification :-

↳ the system is considered as a black box whose internal details are not known.

↳ only its visible external (input/output) behaviour is documented.



PAGE :
DATE : / /

10/02/23

> Analysing all the requirements

{verify, understand & check} ⇒ Reasons for analysis :-

- Anomalies (unclear)
- Inconsistencies (contradictions)
- Incomplete Data (insufficient data)

⇒ Goals of requirement analysis and specification phase :-

- fully understand the user requirements
- remove inconsistencies, anomalies etc. from requirements
- document requirements properly in an SRS document

↳ the person who undertakes requirements analysis & specification known as systems analyst.

↳ final output = SRS document.
Software Requirements Specification

> Specifications ★ 'what to' part

SRS document is useful in:

↳ statement of user needs
↳ contract document
↳ reduces future rework
↳ provides basis for estimating cost & schedules

> SRS document is known as black-box specification.

PAGE :
DATE : / /

> Properties of a good SRS document

1. It should be concise
2. It should specify what the system must do
3. Easy to change / modify
4. Should be consistent
5. complete
6. verifiable
7. traceable

> SRS document contains three important parts

1. functional requirements (input data, output data, procum
2. non functional requirements (maintainability, portability, usabil
3. constraints on the system (restrictions)
   ↳ things that the system should or shouldn't do.

> Organization of SRS document

1. Introduction (purpose, scope, environment)
   ↳ hardware & software requirements & performance
2. functional requirements
3. non functional requirements (External interface
4. constraints

> Worst type of SRS document

1. Difficult to change
2. Difficult to precise
3. Difficult to be unambiguous
4. scope for contradictions
5. Anomalies
6. Overspecification
7. Noise (unwanted)
8. Wishful thinking

> Forward References → references to aspects of problem
↳ defined only later in the text.

PAGE : _____
DATE : __/__/____

15/02/23

CA-1 SRS document Deadline – 1st April

↳ 15–20 pages
↳ pdf & printouts
↳ Topic : 360 security – Internet Security

Software Design

> Designing Documents ★ 'how to' part

⇒ Outcomes :-

1. module structure (dividing the work in different functi
2. control relationship among the modules
3. interface among different modules (data items being shared)
4. data structures of individual modules [arranging data]
5. algorithms for individual modules [Processing of data]
   ↓
   searching & sorting of data

16/02/23

> Classification of Design activities

1. High-level Design (Preliminary) {1-3 outcomes}
2. Detailed Design {outcome 4 & 5}

> Good Software Design

1. Correctness
2. Understandable
3. Efficient [cost, time & resource optimization]
4. Maintainable (because it is not a one time project)

★ An important characteristic of a good design solution is layering of the modules, to achieve abstraction & easy to understand & debug.

Feasibility Study
↓
Requirements Gathering
↓
Requirement Analysis
↓
SRS document
↓
Designing
↓
Coding

PAGE :
DATE : / /

> Modularity → Decomposition of a problem (modules)
↳ Cohesion (togetherness) → high cohesion modules intra dependency
↳ Coupling (interdependency) → low coupling.

> Quick Recap :-

Feasibility Study
↓
Requirements Gathering
↓
Requirement analysis
↓
SRS document
↓
Designing
↓
Coding

⇒ Feasibility Study is carried out based on many purposes to analyze whether software product will be right in terms of development, implantation, contribution of project to the organization.

> Summary :-

(I) The aims of Requirements Analysis :-

1. Gather all user requirements
2. Clearly understand exact user requirements
3. Remove inconsistencies and incompleteness
4.

(II) The aims of Specifications :-

1. systematically organize requirements
2. document the requirements in an SRS documents

DATE : __/__/__
PAGE : _____

Unit-II Issues in Software Design

> Basic issues in software design
> Modularity
> cohesion [codependency] ⩵ High
> coupling and layering ⩵ Low * Advantages of low coupling.
> function oriented software design
> Data flow diagram & structure chart

17/02/23

★ C.T. MCQ based Wednesday

> Functional Independence

↳ coupling (Low) & cohesion (High)

> Advantages of functionally Independent module :-

1. Easy maintenence
2. Error isolation [if there is error in one module it won't affect the other module.]
3. ~~Under~~ less complexity
4. Reuse
5. Efficient

> Cohesion [cooperation within the module]

Types :-

↳ { Lowest cohesion }

1. Co-incidental cohesion (accidentally) [either no or very less cooperation]
2. Logical cohesion (similar functions being performed)
3. Temporal cohesion (time) → [start & shutdown processors]
   ↳ functions with same time-span
4. Procedural cohesion (step by step functions implementation)
   [login, check availability, get the book issued, log out]
   ↳ some order is followed

DATE : / /
PAGE :

Stack → LIFO → Last In First Out Queue → FIFO → First In First Out

18/02/23 ★ Data Structures :- ways of storing & organising data
↳ arrays, queue, stack, LL

5. Communicational cohesion (all the functions in module are either referring to the same data structures or updating it)
6. Sequential cohesion (a sequence is followed) → [input for next function will be dependent on output of previous]
7. Functional cohesion (many functions are cooperating to perform a single task)
Ex, employee module
[calculate working hours, calculate overtime, calculate deduction] → salary calculation {Highest cohesion}

{1. → 7. Cohesion Increases}

> Coupling [The degree of interdependence between two modules]

Types :-
1. Data (if 2 modules communicate using an elementary data item that is passed as a parameter b/t the two)
2. stamp (communicate using a composite data item)
3. control (if data from one module is used to direct the order of instruction execution in another)
4. common (if 2 modules share some global data items)
5. content (if two modules share code) → jump from one module into the code of the another module.
[worst form of coupling]

{1 → 5. Low → High}

↳ stamp eg.

```
structure stu
{
int R_no
char Name
float CGPA
}
```

↳ control eg, a flag (boolean set) set in one module and tested in another module.

PAGE : / /
DATE : / /

> Organisation of the SRS Document

1. Introduction
   (i) Purpose
   (ii) Project scope
   (iii) Environmental Characteristics

2. Product Perpective
3. Product features
4. User classes
5. Operating environment
6. Design and implementation constraints
7. User documentation

> External Interface Requirements

1. User interface
2. Hardware interfaces
3. Software interfaces
4. Communications interfaces

> Non-functional Requirements

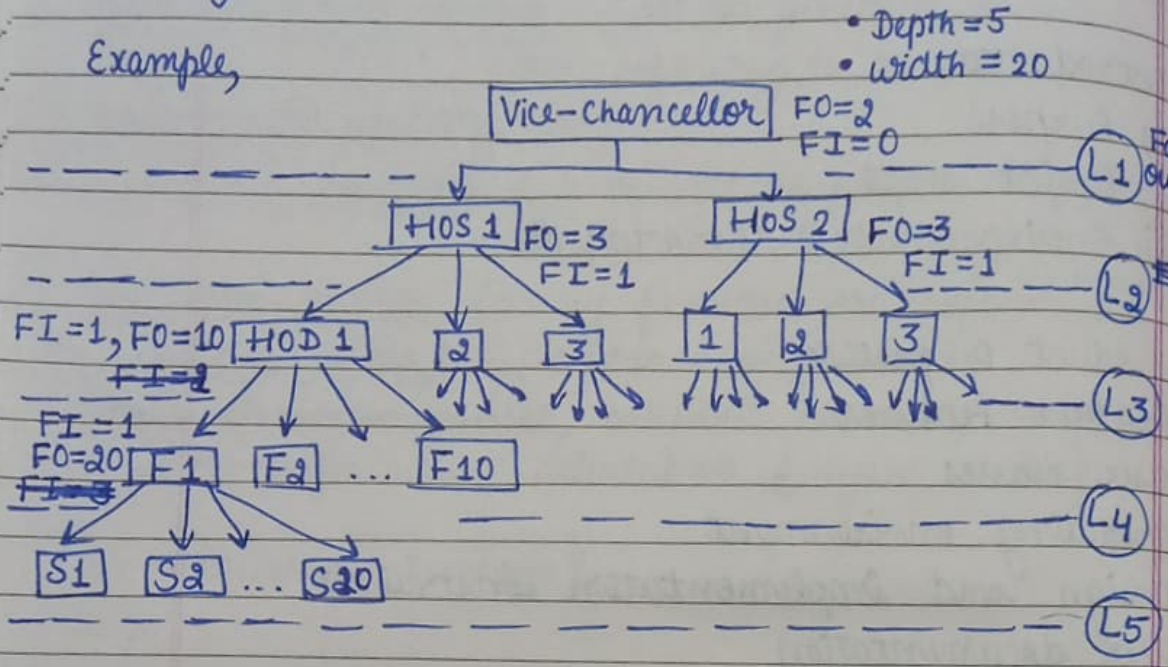
1. Performance
2. safety
3. security
4. Database
5. Platform
6. web-support

PAGE :
DATE : / /

22/02/23

> Layering (Control Hierarchy)

Example,



↳ modules arranged into several layers based on their call relationships is called layering.

↳ A module is allowed to call only the modules at the lower layer.

↳ Modules at highest level = Manager Modules

↳ Modules at lowest level = Worker Modules (do not invoke services of any module & perform carry out their own responsibil

↳ Terminologies :-

1. superordinate & subordinate modules
2. Visibility [lower level modules are visible to higher level]
3. Control Abstraction [~~high~~ hiding of data] { data of higher level is abstra from lower lev
4. Depth and width → [maximum number]
5. Fan out [number of modules that higher layer is directly controlling.]
6. Fan in [no. of modules that are controlling the lower level.]

> Manager modules ↴

(consider only direct link

Invokes services of lower levels to discharge responsibilities

DATE : / /
PAGE :

> MCQs

1. What is Software Engineering?

↳ Application of engineering principles to develop a software

2.

> High-level design

1. modules
2. control relationships among modules
3. interface among modules

> Good software design

1. Correctness
2. understandability

DATE : / /

PAGE :

23/02/23

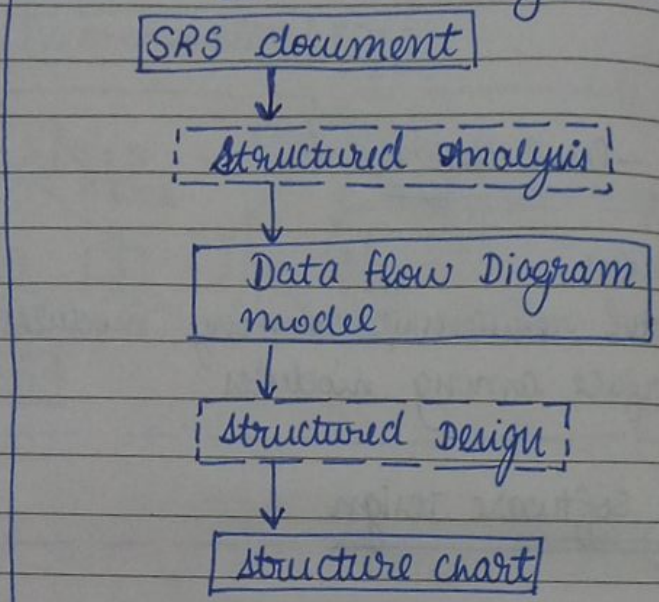
CA2 - Next week - MCQs → OAS

CA 1 - SRS Document - 1st April

> MCQs → (Test)

1. c) ✓
2. a) ✓
3. c) ✓
4. a) ✓
5. c) ✓
6. c) ✓
7. d) ✓
8. a) ✓
9. a) ✓
10. ~~b) ✓~~ c)
11. c)
12. d)
13. c)
14. c)
15. a)
16. b)
17. d)
18. c)
19. ~~a)~~ d)
20. d)

> SA/SD Methodology



DATE : __/__/__

PAGE : _____

24/02/23

> SA/SD Methodology → Structured analysis
→ Structured Design

> Design Approaches

> Difference b/w SA & SD.

Function-oriented

↳ view system as a black-box that provides a set of services to the user of a software.

↳ services are called high-level functions.

↳ TOP-DOWN DECOMPOSITION.

↳ This module structure poses all characteristics of a good software design.

★ Top-down decomposition
successive decomposition of a set of high level function into detailed functions.

Object-oriented

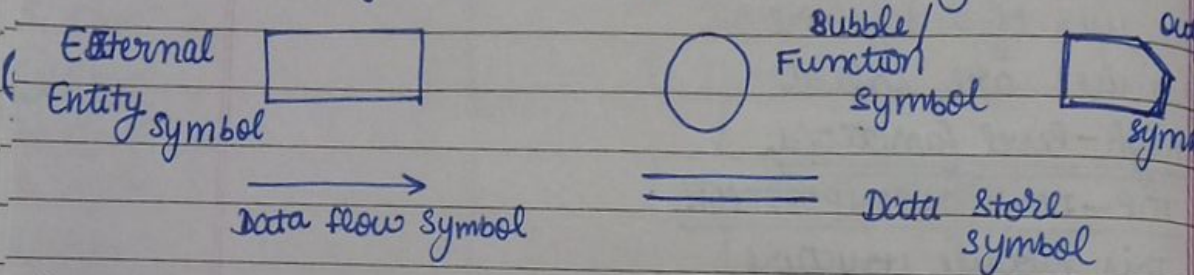
> Structured Analysis → context diagram
↳ entire system is represented as a single bubble.

↳ Principles :-

[Data flow Diagram]

#Impt.
★ 10 marks Que in ETE

> DFDs (bubble charts)
↳ simple to understand and use
↳ simple graphical formalism used to represent system in terms of input data, various processes procured.
↳ hierarchical model.
↳ Primitive symbols used for constructing DFDs :-



> External Entity Symbol (terminator, source or sink)
↳ input data to the system (source) or consume data from the system (sink)

> Function Symbol (process, transform or bubble)
↳ should be verbs

> Data Flow Symbol

> Data Store Symbol
A logical file can be
- a data structure
- a physical file on disk

> Output Symbol
↳ hardcopy output produced by the system

PAGE :
DATE : / /

25/02/23

Operations

> Synchronous Operation
↳ two bubbles connected via data flow arrow.
↳ synchronous
↳ Level 1

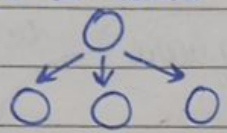
> Asynchronous Operation
↳ two bubbles connected via data store.
↳ not synchronous

> Structured Analysis

Level 0 → DFD

Context diagram → called so because it is used as a context (rough idea) for further levels
↳ process is represented by single bubble.
↳ no verb in bubble
↳ multiple external entities can be used.
↳ we cannot have a data store at level 0.

> Decomposition



↳ decomposition of a bubble also called factoring or exploding.
↳ each bubble is decomposed to b/w 3 to 7 bubbles.
↳ Too few bubbles make decomposition superfluous
↳ useless
↳ too many bubbles make it complex to understand.

> Number of bubbles
↳ to help uniquely identify any bubble in the DFD from its bubble number.
↳ when a bubble (x) is decomposed its children bubbles are numbered x.1, x.2, ...

★ Level-1 DFD doesn't have external entities.

DATE : / /
PAGE :

01/03/23 (Structured Analysis) Input, output

SA

SRS → high level functions → sub, sub → DFD → Structured charts

SD (Structured Design)

★ CA 2 → OAS → 30 marks : 4th March [NO-negative marking]

10 - Intro model

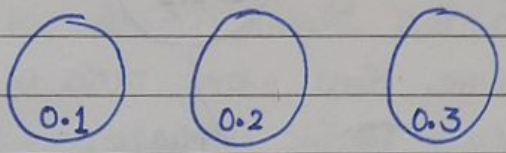
10 - till SRS

10 - till DFDs

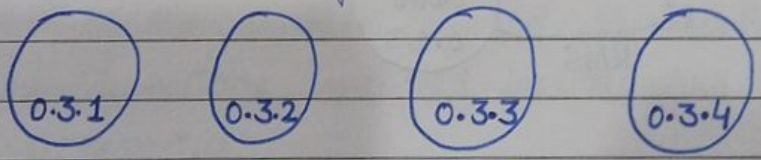
> Numbering of bubbles

Level - 0

Level - 1

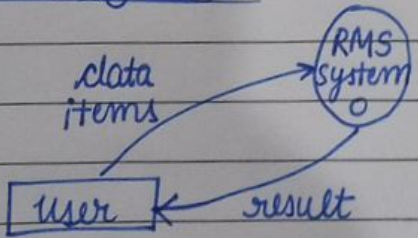


Level - 2



> Example, RMS calculating Software

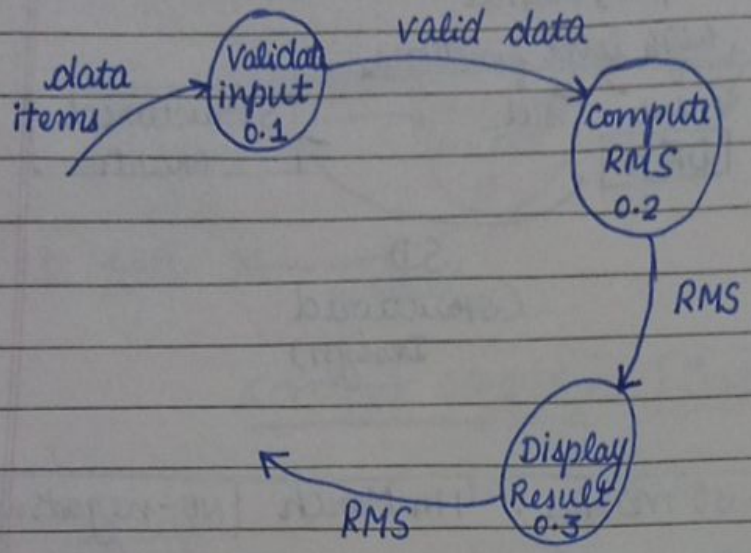
Level-0 Context diagram



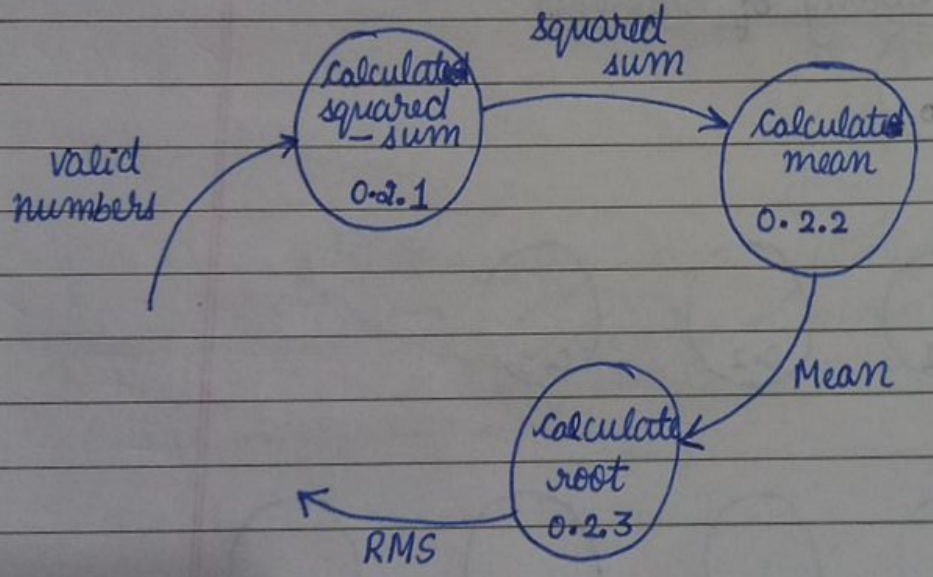
PAGE :

DATE : / /

Level-1 RMS calculating software



Level-2 Exploding the 2nd bubble



> Rules of Data Flow

⇒ Data can flow from	⇒ Data cannot flow from
↳ External entity to process	↳ External entity to External entity
↳ Process to External entity	↳ External entity to store
↳ Process to store & back	↳ store to external entity
↳ Process to Process	↳ store to store

02/03/23

> Common Errors in DFDs

- ↳ More than one bubble in context diagram
- ↳ Use of external entities in upper level
- ↳ Either too few or too many bubbles in DFD.
- ↳ Leaving DFD unbalanced
- ↳ Try to represent control & other information in DFD.
- ↳ A data flow arrow should not connect two data store or even a data store with an external entity.
- ↳ All the functionalities of the system must be captured by the DFD model. [SRS document]
- ↳ no function of the system should be overlooked.
- ↳ only those functions specified in the SRS document should be represented.
- ↳ The data & function names must be intuitive
- ↳ Don't clutter DFDs with too many data flow arrows
- ↳ Incomplete data dictionary & incorrect composition of data items.

DATE : / /
PAGE :