Software Engineering ESE

Module 1

(1) Software development process
- (i) Introduction
- (ii) Principles
- (iii) Need
- (iv) Characteristic of good software

(2) SDLC (Software development life cycle)
- (i) Introduction
- (ii) Need
- (iii) Models :
  1. Waterfall : Classic and iterative
  2. Incremental (aka successive version model)
  3. Prototype
  4. Evolgutionary model
  5. Spiral model (aka metamodel
  6. Concurrent model

(3) Component based software development

(4) Agile software development
- (i) Introduction
- (ii) Agile manifesto
- (iii) Agile principles
- (iv) Scrum

Module 2 : Modelling practices & Requirement engineering

(1) Seven Core principles
(2) Principles that guide process
(3) Principles that guide practice
(4) (i) Communication principles
(ii) Planning principles

(5) Creating an SRS Document
(6) Creating Data flow diagram
(7) Risk management ??
(8) Requirement elicitation
(9) Requirement Engineering
(10) Data modelling

Module 3 : Design engineering and testing strategies.

(1) Software testing and maintenance (i) Whitebox (ii) Blackbox
(2) Software design
(3) Design fundamentals

* Module 1 : Notes

→ What is software engineering?
Software engineering is an engineering branch associated with the development of software product using well defined scientific principles, methods and procedures.

IEEE definition
The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software.

→ Principles of software engineering
Two techniques are used to reduce complexity:
① Abstraction ② Decomposition
① Abstraction: This implies that the problem can be simplified by omitting irrelevant details. Abstraction is a powerful way of reducing the complexity of a problem.
② Decomposition: A complex problem is divided into smaller problems and then those problems are solved one by one

→ Characteristics of a good software:
The software must satisfy on the following grounds:
① Operational
② Translational
③ Maintenance
Operational: This aspect tells us how software works in operations. It can be measured on budget, efficiency, correctness and security and safety

Translational : It should be portable and adaptable to changing environment.

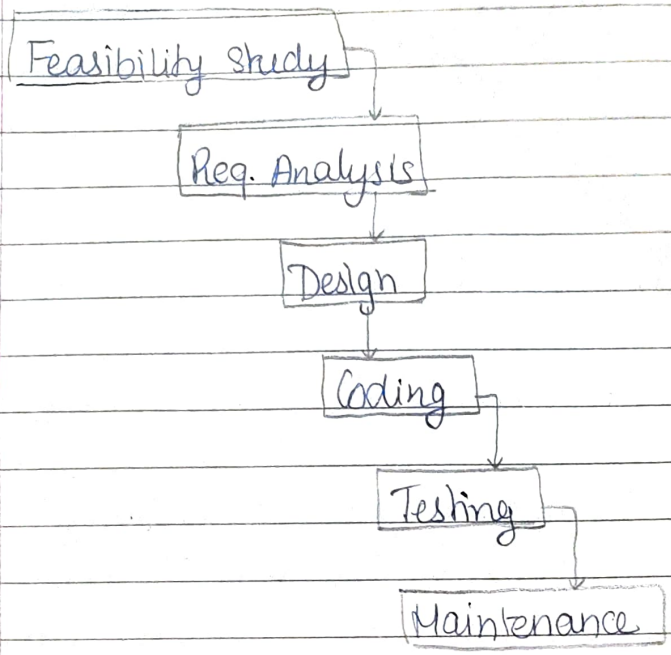
Maintenance : It should be flexible, durable and scalable

* Software development life cycle
- SDLC is a descriptive and diagramatic representation of the software lifecycle.
- It maps the different activities in software development from inception to retirement.
- Basic activities are included in all models though the activities may be carried out in different orders in different models

Need :
- For systematic and discipline manner
- Clear understanding between team members.
- Entry and exit criteria for a phase.
- Monitor the progress of software

Different models

① Classical waterfall model



- Also known as linear sequential flow model.
- Projects must have defined set of requirements.
- Completion of previous phase is required for the next phase to begin
- Does not support going back to the previous phases.

① Feasibility study: The main aim of feasibity study is to determine whether it would be technically and financially feasible to develop the product.

② Requirement analysis: Aim of this phase is to understand exact requirements and document them properly.

(i) Requirement gathering and analysis: Collect all the relevant information from the customer regarding the product to be developed.

(ii) Requirement Specification: User requirement are systematically organized into software requirement specification (SRS) document. The important components of this document are function, non-functional requirements and the goals of implementation.

③ Design: The goal of the design phase is to transform the requirements specified in SRS to a structure that is suitable for implementation in some programming language. In technical terms, in design phase the software architecture is derived from the SRS document.

④ Coding/Implementation Phase: The purposed of the coding phase of software development is to translate the software design into source code. Each component of the design is implement as a program module. The end product of this phase is a set of program modules that have been individually tested.

⑤ Integration and system testing: Integration of different modules is undertaken once they have been coded and unit tested. During this phase, the modules are integrated in a planned manner. The goal of the system testing is to ensure that the developed system conforms to the requirements laid out in the SRS document.

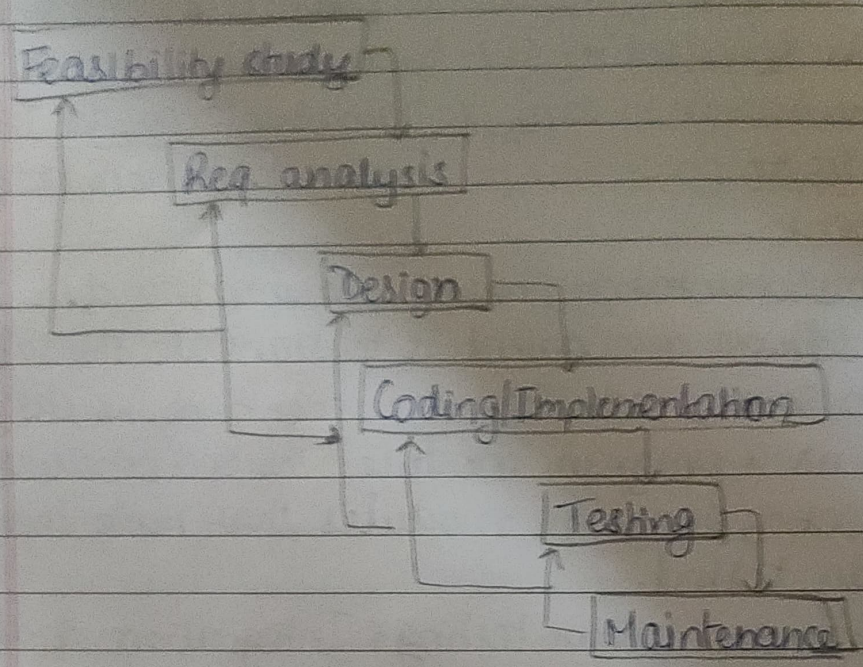
⑥ Maintenance: Maintenance involves any of the following activities:

① Corrective maintenance: Correcting errors that were not discovered in the development phase.

② Perfective maintenance: Improving the system and enhancing the functionality of the system.

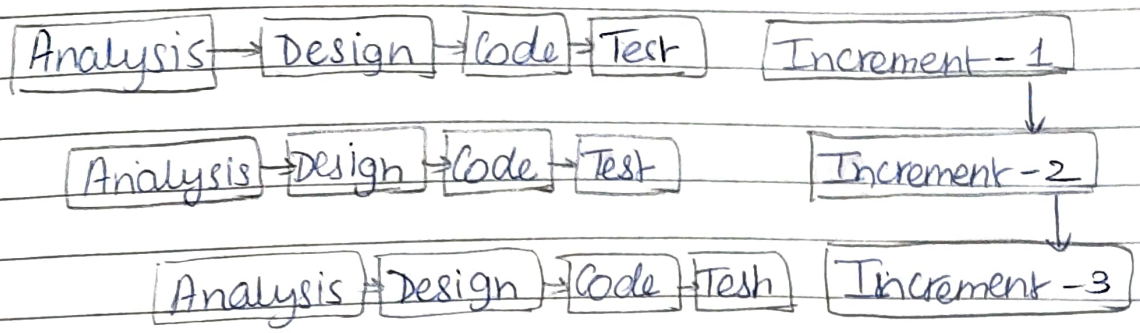
③ Adaptive maintenance: Porting the software to work in a new environment.

② Iterative waterfall model:



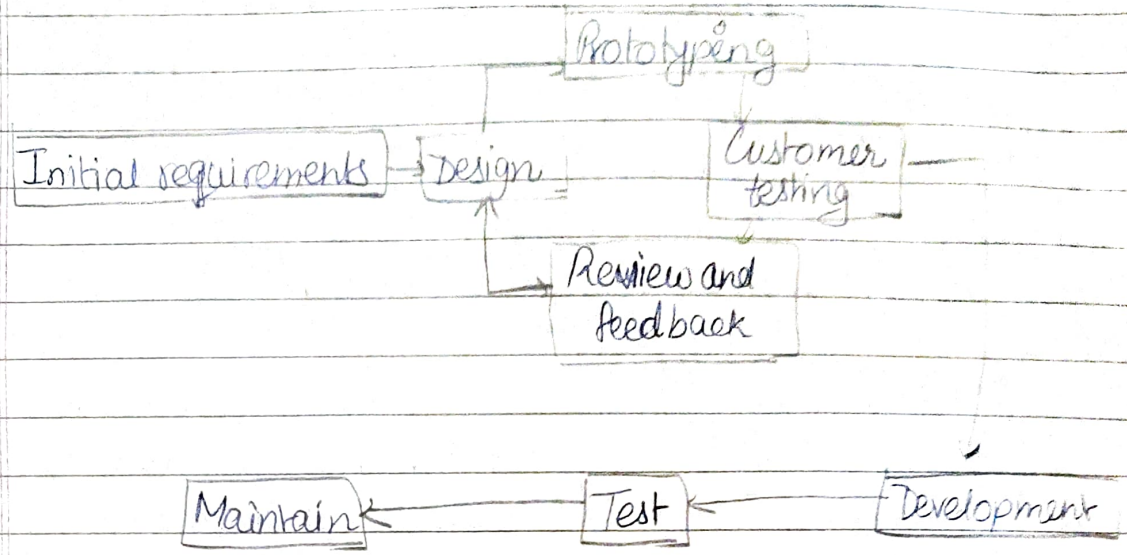
- Overcomes the shortcomings of the classical waterfall model.
- Feedback path is provided for error correction if when detected later in a phase.
- Operation are same as the classical waterfall model except there are made some changes made to increase the efficiency of the software development.

③ Incremental model (aka successive version model)



- Simple working system is built and delivered to the customer
- Successive models are built after the clients review.
- Each module goes through analysis, design, code, test phase.
- Module functionality is increased with each new relcase

Prototype model :



Prototypes :

- ~~Exhibits~~ Represents toy implementation of the system
- Exhibits limited functionality capabilities, low reliability, and inefficient performance compared to the actual model.
- It is essential as costs are controlled and system stakeholders can experiment in the early stage of the software development

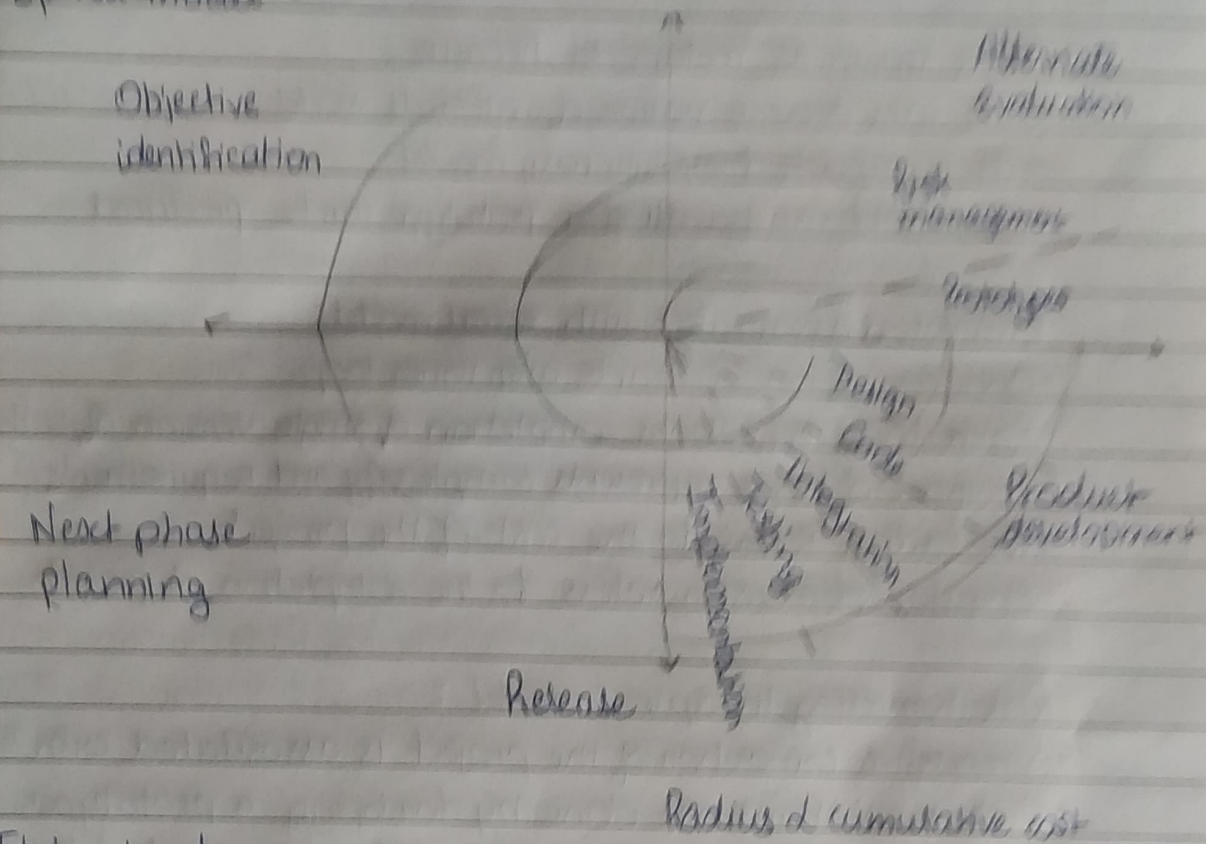
Need :

(1) User requirements are incomplete and continuously changing
(2) Technical issues are not clear.
(3) When it's impossible to get the desired outcome on the first attempt.

Type of a prototype model :

(1) Rapid throwaway model: The initial prototype is not the part of the ultimate prototype.
(2) Evolutionary prototyping : It prototype developed early on is incrementally refined
(3) Incremental prototyping : The final product is broken down and each module is developed individually and finally merged together to make the final product.

Spiral model :



- It is also known as meta model.
- It is the most important model which supports risk handling

① First quadrant (Objective identification) :
- During the first quadrant it is needed to identify objectives of the phase.
- Examine the risks associated with these objectives.

② Second quadrant (Risk assessment and reduction)
- A detailed analysis is carried out for each identified risk.
- Steps are taken to reduce the risk. (ie create a prototype model.)

③ Third quadrant (Development and validation)
- Develop and validate the next level of the product after resolving the identifying risks

④ Fourth quadrant (Review and planning)
- Review the results achieved so far with the customer and plan the next iteration around the spiral.
- Progressively complete more versions of the software gets built with each iteration around the spiral.

- It is known as metamodel because:
  - It uses iterative nature of waterfall model
  - It is modified evolutionary model
  - To avoid risk handling a prototype can be produced

Parameters assosciated with spiral model:
- Operation of model starts with inner loop.
- Each loop represents completion of single version of software
- Number of loops represents complexity and requirements.
- The radius refers to the cost of the project
- Angular dimension refers to the completion of project.

Risk handling in spiral model:
- Successful completion of the project is assosciated with the risk
- Risk resolutions are done by developing a prototype.
- Risks must be analyzed properly before the start of the project.

* Component based software development

- In CBSE, we develop software using already available components.
- In CBSE we don't build software from scratch
- Components communicate with each other using well defined interfaces.

Component:
- A component is an independent software unit that can be composed with other units to create a software system.

Component characteristics / properties:
(1) Independent: A component should be independent.
(2) Composable: A component should be composable which means that all external interactions must take place through publicly defined interfaces.
(3) Deployable: A component has to be self contained and must be able to operate as a standalone entity.

Benefit in CBSE:
(1) Management of complexity
(2) Reduce development time
(3) Increased productivity
(4) Improved quality
(5) Maintenance costs are reduced

Lifecycle in CBSE
- Requirement analysis
- Software architecture selection construction, analysis and evaluation.
- Component identification and customization.
- System integration
- System testing
- Maintenance.

* Agile Software Development

Agile software development is a group of software development methods based on iterative and incremental development, where requirement and solutions evolve through collaboration between self-organizing and crossfunctional teams.

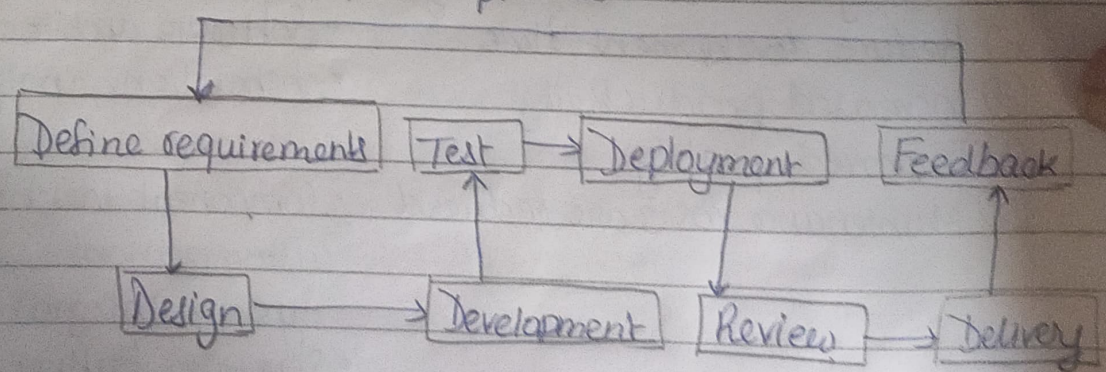
It promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible responses to change.

- Software developed during one unit of time referred to as an iteration, which may last from one to four weeks.
- Agile methods also emphasize working software as the primary measure of the progress.

Time-boxed approach:

- In time management, a time box allots a fixed period of time for an activity.
- Time boxing plans activity by allocating time boxes.

Agile software development process



* Agile manifesto :
Fears regarding software development led to pioneers and practioners to develop agile manifesto based on firm values and principles.
Values/Manifesto :
(1) Individuals and interaction over process and tools
(2) Working software over comprehensive documentation
(3) Customer collaboration over contract negotiation
(4) Responding to change over following a plan.

Principles for agile :
(1) Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
(2) Welcome changing requirements even late in development.
(3) Deliver working software frequently
(4) Bussiness people and software developer must work together daily throughout the project
(5) Build projects around motivated individuals
(6) The most efficient and effective way of conveying information within a development team is through face-to-face communication
(7) Working software is the primary measure of progress.
(8) Agile development promotes a sustainable development the sponsors, the developers, and users should be able to maintain a constant pace indefinitely.
(9) Continuous attention to technical excellence and good design enhances agility.

* Scrum

Scrum is an agile framework for completing complex projects. Scrum originally was formalized for software development projects but it works well for any complex, innovative scope of work. The possibilities are endless. The scrum framework is deceptively simple.

Principles of scrum :

1. Welcome changing requirements even late in development
2. Deliver working software frequently
3. Early visibility of self organizing teams to business
4. Product owner must work together with the developers daily throughout the project.
5. Inspect and adapt.

Module 2: Modelling practices and requirement engineering

→ Seven core principles for software engineering: because of which
① Remember the reason ~~that~~ the software exists
② Keep the software ~~simple~~ design and implementation simple
③ Maintain a clear vision of the project
④ Always specify, design & implement without any ambiguity such that later ~~if someone tries~~ it is easy to understand and modify your code.
⑤ Build software that can be easily changed and adapted
⑥ Plan ahead for software reuse
⑦ Have clear and complete thoughts which will produce better results

* Principles that guide process
① Be agile
- Basic principles of agile SD should govern your approach irrespective of the model you chose
- Keep technical approach as simple as possible
② Focus on quality at every step.
- For every process activity to enter into next phase focus on quality of the work product that has been produced
③ Be ready to adapt
- Be open to changes, challenges and complaints.
④ Build an effective team
- Initiate a self organizing team if permitted and possible
- A self organizing team has mutual trust and respect
- Communicate the responses to the team as a whole.

⑤ Establish mechanisms for communication and coordination
- Initiate face-to-face commication between developers, stakeholders and clients.

⑥ Assess risk
- ~~In~~ Don't be afraid of risks
- In case of possible risk find the best possible strategies for risk handling and serve the final product.

⑦ Manage change
- Welcome changing requirements even late in development

⑧ Create work products that provide value for others.
- Create only those work products that provide value for other process activities, action or tasks.

Date
Page

* Principles that guide practice
1. Divide and conquer
2. Understand the use of abstraction
3. Strive for consistency
4. Focus on transfer of information
5. Build software that exhibits effective modularity
6. Look for patterns
7. When possible, represent the problem and its solution from a number of different perspectives
8. Remember that someone will maintain the software
9.

① Divide and conquer
- Analysis and design should always emphasize separation of concerns.
- A large problem is easier to solve if it is subdivided into a collection of elements.

② Understand the use of abstraction
The abstraction is the process of ignoring certain details in order to simplify the problem and facilitate the specification, design & implementation
Implementation abstraction
Design abstraction

③ Strive for consistency
- The principle of consistency suggests that a familiar context make the software easier to use

④ Focus on transfer of Information.
- Software is about information transfer - from a database to an end user, from a legacy system to a webapp, from an end user to graphical user interface

- In every case information flow across an interface
- The certainty of error, or omission or ambiguity increases
- Such parameters should be considered during analysis design, construction and testing of interfaces.

⑤ Build software that exhibits effective modularity.
- Modularity should be effective
- Each module should focus on one well constrained app of the system.

⑥ Look for patterns
- The goal of patterns within the software community is to create a body of literature to help resolve recurring problems encountered throughout all of software development.

⑦ When possible, represent the problem and its solution from a number of different perspectives.

⑧ Remember that someone will maintain the software.
- Over the long term, software will be corrected as defects are uncovered, adapted as its environment changes, and enhanced as stakeholders request more capabilities
- These maintenance activities can be facilitated if software engineering practice is applied throughout the software process.

SRS Document
Format by IEEE

I] Introduction
1.1 Purpose
1.2 Document convention
1.3 Intended audience and reading suggestions
1.4 Product scope
1.5 References

II] Overall description
2.1 Product perspective
2.2 Product functions
2.3 User class and characteristics
2.4 Operating environment
2.5 Design and Implementation constraints
2.6 User documentation
2.7 Assumptions and dependencies

3] External interface requirements.
3.1 User interface
3.2 Hardware interface
3.3 Software interface
3.4 Communication interface

4] System features
4.1 System feature I
4.1.1 Description and priority
4.1.2 Stimulus | Response sequence
4.1.3 Functional requirements

5] Other non-functional requirements
5.1 Performance Requirement
5.2 Safety Requirement
5.3 Security Requirement
5.4 Software quality attributes
5.5 Business rules

6] Other Requirements

* Definition of SRS document:
A software requirements specification (SRS) is a document that describes what the software will do and how it will be expected to perform. It also describes the functionality the product needs to fullfill all stakeholders (business, users) needs.

Importance of SRS document.
- An SRS minimizes the time and effort required by developers to achieve desired goals and also minimized the development cost.
- A good SRS defines how an application will interact with system hardware, other programs and human users in real world situations.
- An SRS forms the basis of organization's project. It sets out the framework that all the development team will follow. It provides critical information to all the teams including development, operations, quality assurance and maintenance, ensuring the team are in agreement

→ Characteristics of a good SRS document

1. Correctness
2. Unambiguity
3. Completeness
4. Consistency
5. Ranking
6. Verifiability
7. Modifiability
8. Traceability

① Correctness :

- An SRS is correct if and only if every requirement stated therein is one that the software shall meet.
- SRS can be compared to a superior document (System requirement Specification) during a review process to filter out all the errors and contradictions.

② Unambiguity :

- An SRS is unambiguous if and only if every requirement stated therein has only one interpretation.
- Each characteristic of the final product must be defined only using a single term.

③ Completeness :

An SRS is complete if and only if

- (i) All significant requirement such as functionality, performance, design, etc are acknowledged and treated.
  (ii) Specify the responses to both valid and invalid inputs.
  (iii) Full labels and references to all figures, tables in the SRS and the definition of all terms and units of measure.
- Completeness is hard to achieve, especially in business system where requirements continuously change. Completeness is something agile methodologies can not interpret.

④ Consistency:

Subset of Individual

An SRS is internally consistent if and only if the ~~internal~~ requirements ~~into~~ described in it do not conflict.

⑤ Ranking:

An SRS is ranked for importance and/or stability if each requirement in it has an identifier to indicate either importance or stability of that requirement. Typically all the requirements in an SRS are not equally important.

⑥ Verifiability:

An SRS is verifiable if and only if every requirement stated therein is verifiable. A requirement is verifiable if and only if there exists some finite cost effective process with which a person or a machine can check that the software meets the requirement.

⑦ Modifiability:

An SRS is modifiable if and only if its structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining the structure and style.

⑧ Traceability:

An SRS is traceable if the origin of each of its requirements is clear and if it facilitates the reference of each requirement in future development or enhancement documentation.

Backward traceability

Forward traceability

→ Creating data flow diagram

Definition

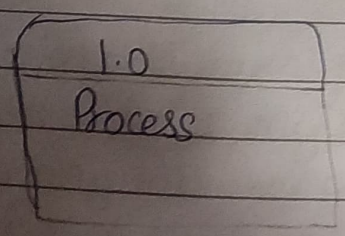
- DFDs model events and processes ie activities which transform data within a system.
- DFD is a pictorial representation of data that flows from one process to another process inside a system (generally information system) along with mentioning how data flow into the system and out of the system

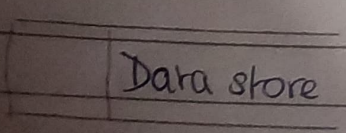
Importance/Usefulness:

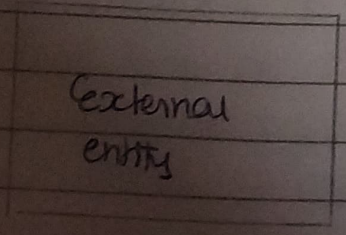
- DFD shows abstract or functional view of the system to be developed
- A graphical representation easily overcomes any gap between 'user and system analysts' and 'analyst and system designer' in understanding a system.

Symbols used in a DFD

Gane and Sarson

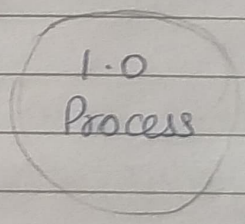


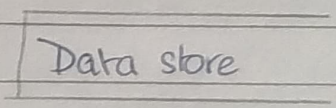


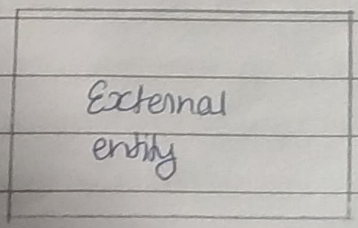


data flow →

Yourdon and Coad







data flow →

* Requirements elicitation:

Requirements elicitation is the process of collecting the requirements of a system from users customers and other stakeholders. It is also known as requirements gathering.

→ Techniques for requirement elicitation:

1. Stakeholder analysis.
2. Brainstorming.
3. One on one interview
4. Group interview
5. Document analysis
6. Focus group
7. Interface analysis
8. Observation social analysis
9. Prototyping.
10. Questionaire
11. Survey
12. Use cases and scenarios.
13.

1. Stakeholder analysis

Stakeholder analysis identifies all the users and stakeholders who may influence or be impacted by the system. This makes sure that the need of all those involved are taken into account.

2. Brainstorming

It is utilized in requirement elicitation to gather good number of ideas from people. Usually brainstorming is used in identifying all the possible solutions to problems and simplifies the details of opportunities. It casts a broad net, determining various discreet possibilities.

③ One on one interview
The most common technique is to sit down with the clients and ask what they need. The discussion should be planned ahead of time based on the requirements you're looking for.

④ Group interview
If there are more than one person during interview usually 2 or 4 of these people must be one some level less time required.

⑤ Document analysis
Document analysis is an important gathering technique. Evaluating the documentation of a present system can assist when making as-is process documents and also when driving the gap analysis for scoping of the migration projects.

* Requirements engineering

Requirements : • It is the description of features and functionalities of the largest system.
• It is the description of what the system should do.

• Requirements engineering: It is the process of defining, documenting and maintaining requirements in the engineering design process.

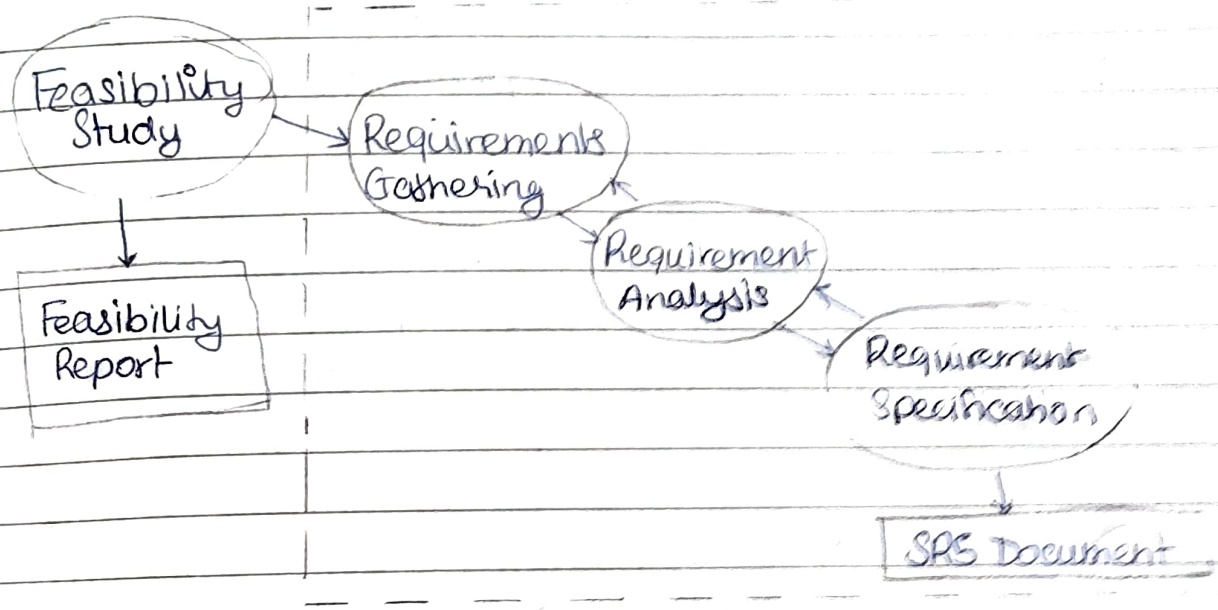
• It is a four step process, which includes -
1. Feasibility study
2. Requirement gathering/elicitation.
3. Software requirement specification.
4. Software requirement validation.

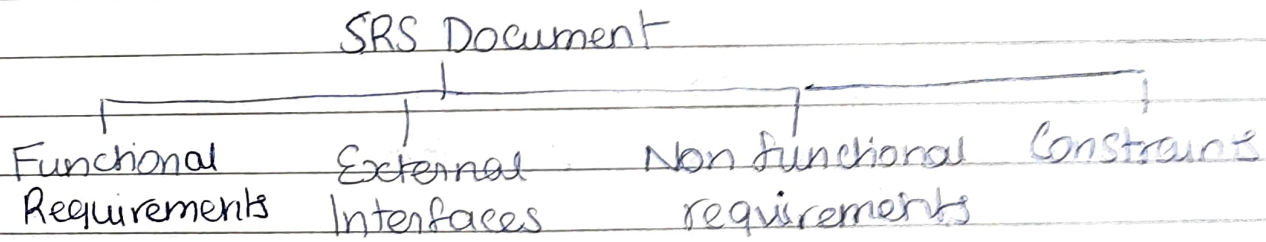
* Functional requirements vs non-functional requirements.

• Requirements which are related to functional/working aspect of software fall into this category.
• Non-functional requirements 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 the software system to be developed.
• It lays out functional and non-functional requirements of the software to be developed.
• It may include a set of use cases that describe user interactions that the software must provide to user for perfect interaction.

Requirements Engineering Process :-





What is requirement ?

- Requirement may range from a high level abstract statement to a detailed mathematical functional application
- The requirements for a system are the descriptions of what the system should do the services that it provides and the constraints on its operation.

Types of requirements :

① User requirement

- Statements in the natural language plus diagram of the service the system provides and its operational constraints
- Written for customers.

② System requirements :

- A structured document setting detailed description of the system's function, services and operational constraints.
- Defines what should be implemented so may be part of a contract between client and contractor.

* Data modeling

Data model is a collection of conceptual tools for describing data, data relationships, data semantics and consistency constraint.

Different data models :

① Conceptual : describes what the system contains

② Logical : describes how the system will be implemented, regardless of the DBMS.

③ Physical : describes how the system will be implemented using a specific DBMS.

Module 3: Design engineering and testing strategies

1. System description
2. Analysis model
3. Design model

→ Analysis model

- Analysis model acts as link between the 'system description' and 'design model'
- In analysis model information, functions, behavior of the system is defined
- These are translated into architecture, interface and component level design in the design modeling.

→ Elements of analysis model:

1. Scenario based element
   - Use case diagram
   - User stories
2. Class based elements
   - Class diagram
   - Collaboration diagram
3. Behavioral elements
   - Sequenced diagram
   - State diagram
4. Flow oriented elements
   - Data flow diagram
   - Control flow diagram

→ Design model
- data design
- architectural design
- interface design
- procedural design

* Data design
- The data design action translates data defined as part of analysis model into data structures
- The data objects, attributes, and relationships depicted in entity relationship diagrams and the information stored in data dictionary provide a base for data design activity

* Architectural design (aka system design)
- Requirements of the software should be transformed into an architecture that describes the software's top-level structure and identifies its components. This is achieved through architectural design.

* UI design
- User interface is the front end application view to which user interacts in order to use the software.

* Procedural design
Software's procedural design converts and translates structural elements into procedural explanations.

* Need of ~~data~~ software design

- Design is the place where quality is fostered in software engineering
- Design is the only way that we can accurately translate customers requirements into a finished software product or system.
- Without design we risk building an unstable system.

* Design principles :

- The design should be structured to accomodate change.
- The design should be reviewed to minimize conceptual ~~errors~~ (semantic) errors.
- The design should exhibit ~~to~~ uniformity and integration.
- The design should be traceable to the analysis model.

Design concepts

- Abstraction
- Refinement
- Modularity
- Architecture.
- Control hierarchy
- Structural partioning
- Data structure
- Software procedure
- Hiding

Abstraction

- data abstraction
- procedural abstraction
- control abstraction

Date
Page

* Stepwise Refinement
- Abstraction and refinement are complementary ~~objects~~ concepts
- Abstraction enable a designer to specify procedure & data & yet suppress low level details.
- Refinements helps the designer to reveal low level details as design process.
- Elaboration of detail for all ~~kinds~~ of abstractions.

* Modularity
- Software divided into separately named & addressable components called modules.
- Monolithic software can not be grasped easily by readers
- It is easier to build, easier to change, easier to fix

* Software Architecture:
- Provides the overall structure of the software & and the ways in which that structure provides conceptual integrity for a system.
- Properties of software architecture
  1. Structural properties: components like modules, objects
  2. Extra functional properties: performance, capacity, reliability security, etc.
  3. Families of related system: Reusability.

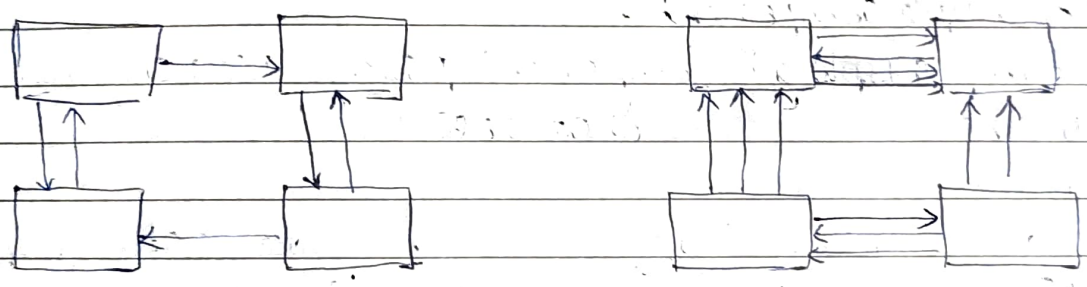
* Control Hierarchy: Represents the organization of program components (modules) and implies a hierarchy of control.

* Coupling and Cohesion :

→ Cohesion : Cohesision is a measure that defines the degree of intra-dependability within the elements of a modules. The greater the cohesion, the better is the program design.

→ Coupling :

- Coupling between modules is the strength of interconnection between modules or a measure of independence among modules
- To solve and modify a module separately, we would like the module to be loosely coupled with other modules.



Loosely coupled some dependencies Highly coupled many dependencies

High coupling makes modifying parts of the system difficult eg modify a component modifies all the components connected to that module.

Note : The best designs have high cohesion (also called strong cohesion) within a module and low coupling (also called weak coupling) between modules.

* Types of coupling:
- Content coupling (also know as pathological coupling)
- Common coupling
- Control coupling
- Stamp coupling
- Data coupling

→ Content coupling: One component refers content of another.
→ Common coupling: Modules are bound together by a global data structure.
→ Control coupling: Component passes control parameters to couple components.
→ Stamp coupling: Component passes data structure to another components that does not have access to the entire structure.
→ Data coupling: Two components are data coupled if there are homogenous data items.

6
~~Ran~~ Cohesion:
Range of cohesion:
① Coincidental: Parts of the component are only related by their position in source code.
② Logical: Elements of the component are related logically not functionally.
③ Temporal: Elements of a component are related by timing ie all elements must be executed in the same span of time.
④ Procedural: Elements of a component are related only to ensure a particular order of execution.
⑤ Communicational: Operate on same input data or produce same output data.
⑥ Sequential: Output from one part serves as input for another part.
⑦ ~~Functional~~ information cohesion: Module performs a number of actions, each with its own entry point, with independent code for each action, all performed on same data.
⑧ Functional cohesion: All elements contribute to the execution of one and only one problem related task.

* Cohesion	Coupling
① Cohesion indicates the relationships within a module.	① Coupling indicates the relationships between modules
② Cohesion shows the modules relative functional strength	② Coupling shows the relative independence among the modules.
③ Cohesion is the degree to which a component focuses on a single thing	③ Coupling is a degree to which a module is connected to other modules.
④ While designing you should strive for high cohesion. ie cohesive component should focus on a single task.	④ While design you should strive for low coupling. ie dependency between modules should be less
⑤ Cohesion is intra-module concept	⑤ Coupling is intermodule concept

* Golden rules for interface design
① Place user in control
② Reduce user's memory load
③ Make the interface consistent.