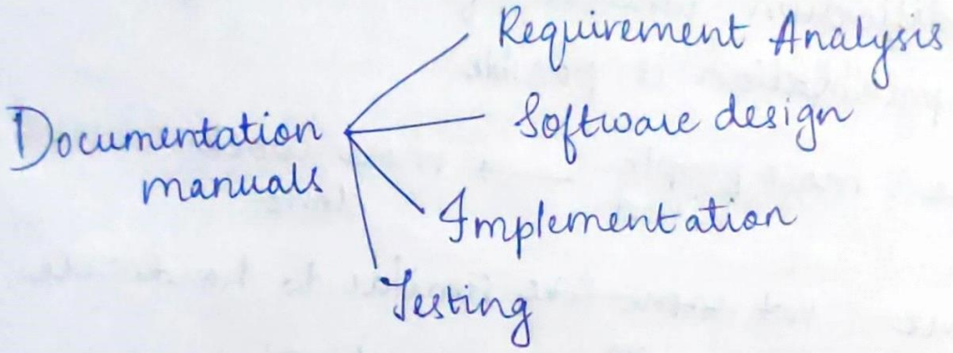
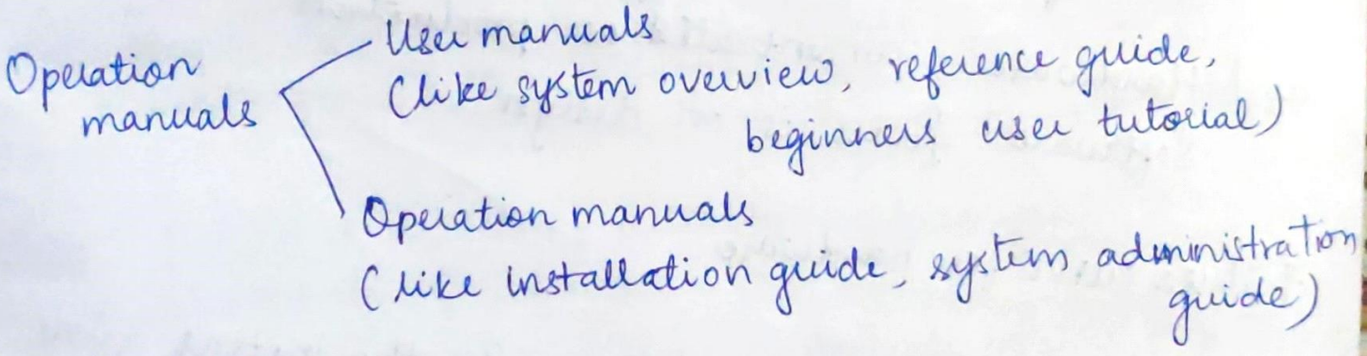
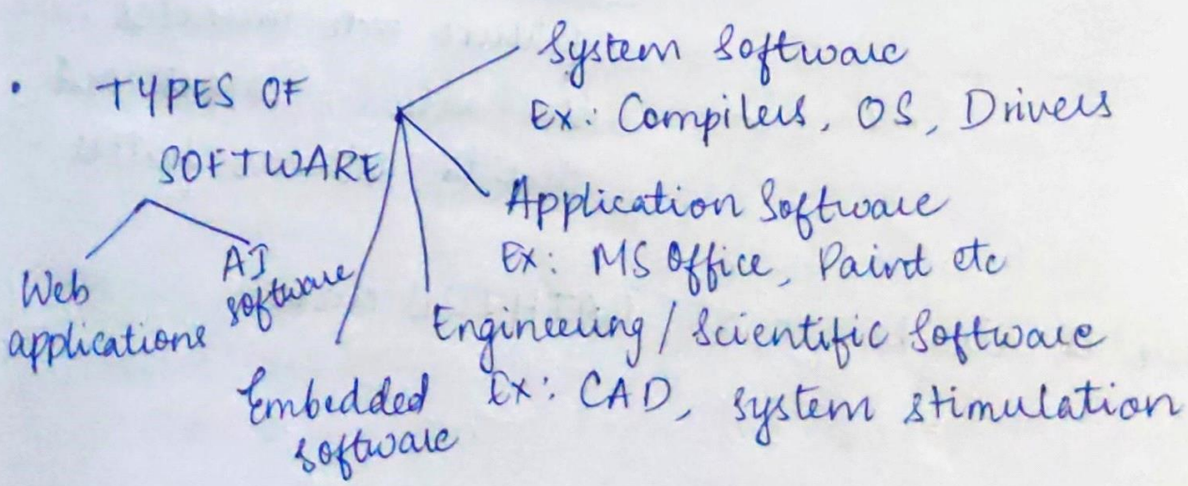
UNIT-①
Software Process & Agile Development

- Program: Set of instructions
- SOFTWARE: It can be defined as the combination of
  i) Set of programs when executed provide desired features, function and peformance
  ii) Documentation
  iii) Operation manuals





- Software Product: something that is delivered to the customer (like source code, manuals)
- Software Process: It is the way in which a software is produced



- Hardware vs. Software:

Hardware : manufactured, wears out, built using compo
relatively simple

Software : developed/engineered, deteriorates, custom bu
complex

- Manufacturing vs. Development:

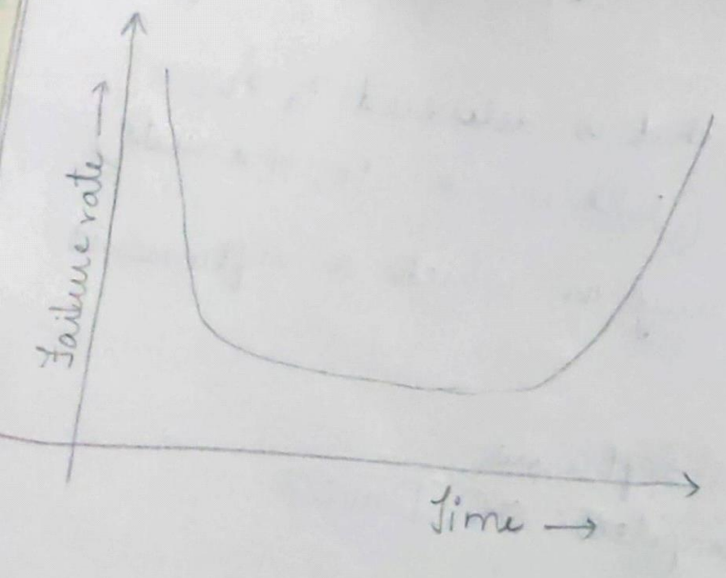
i) After the hardware is manufactured, it is difficult to make modification, whereas after the software is developed upgradation is possible.

ii) In hardware : more people hired ⟶ more work done

In software : not ~~same like~~ similar to hardware

iii) Hardware : concentrated on production
Software : focus is on design

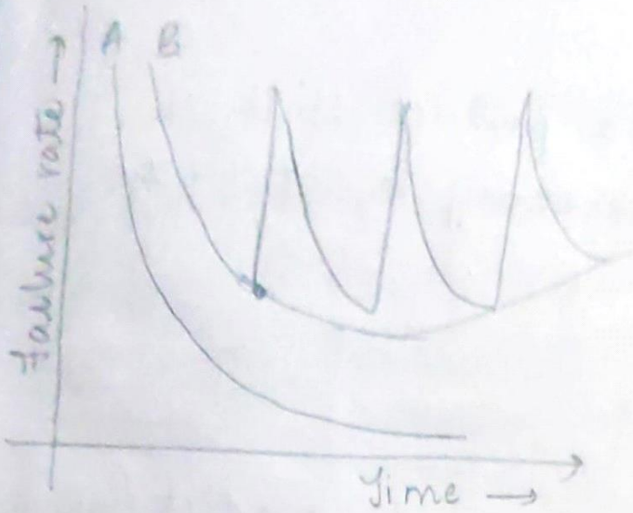
Failure curve for hardware:



In the initial time period, the ~~error~~ failure rate is high. Then gradually the failure rate reduces and then almost becomes constant. Then again the failure rate increases since the hardware wears out due to external factors.

This curve is also known as BATHTUB curve.

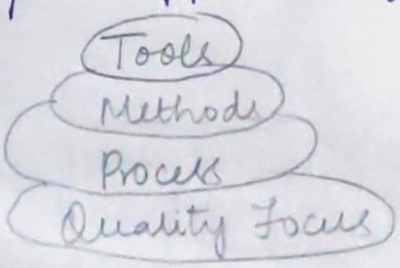
Failure curve for software:



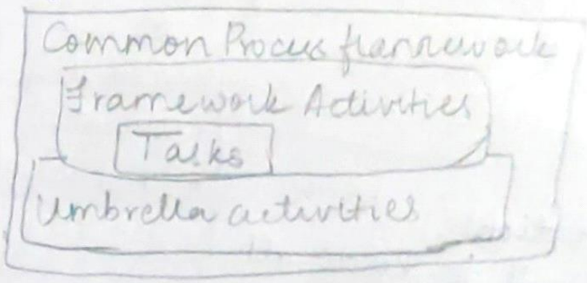
A → ideal curve
B → actual curve

Initially failure rate is high. Gradually it is reduced. Then when the software gets modified, the failure rate gets increased again and reduces gradually after some time period.

- Software Engineering: Systematic, disciplined, quantifiable approach to the development, operation and maintenance of software
- Layred approach for SE:



- Software Process:



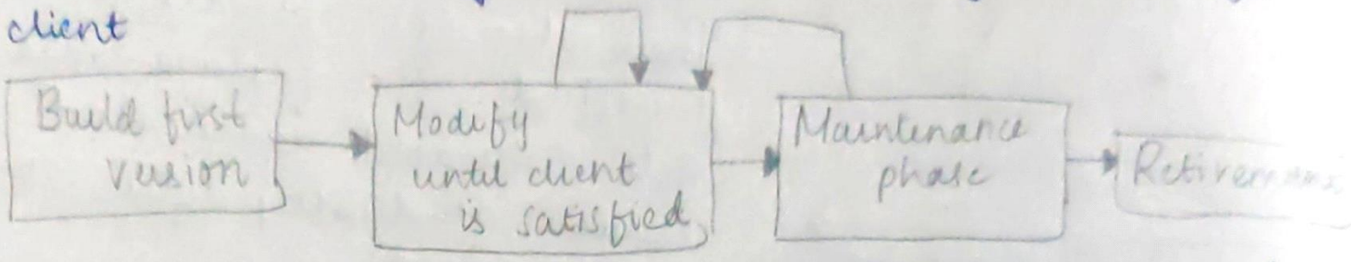
- Software process is a set of activities whose goal is the development of software:
- Activities in software processes: COMMUNICATION, PLANNING, MODELING, CONSTRUCTION, TESTING, DEPLOYMENT
- Attributes of a good software: Maintainability, Usability, Dependability, Efficiency
- SOFTWARE PROCESS FLOW
  - Linear flow
  - Iterative flow
  - Evolutionary flow
  - Parallel flow

Software process model: It prescribes a distinct set of activities, actions, tasks, miles tones and work products required to engineer high quality software.

Types of Software Process Models:

Build and fix model:

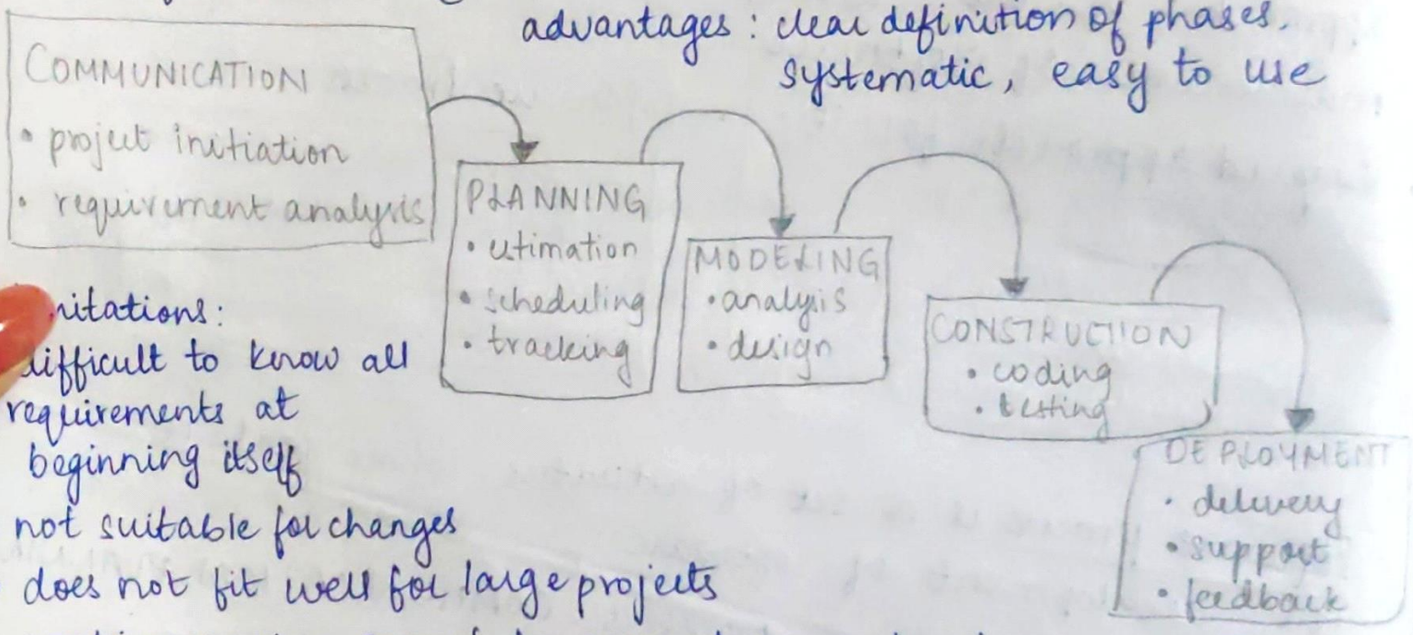
- Approach: developers simply build a product that is reworked as many times as necessary to satisfy the client



- limitations: high maintenance, impossible to predict/mana

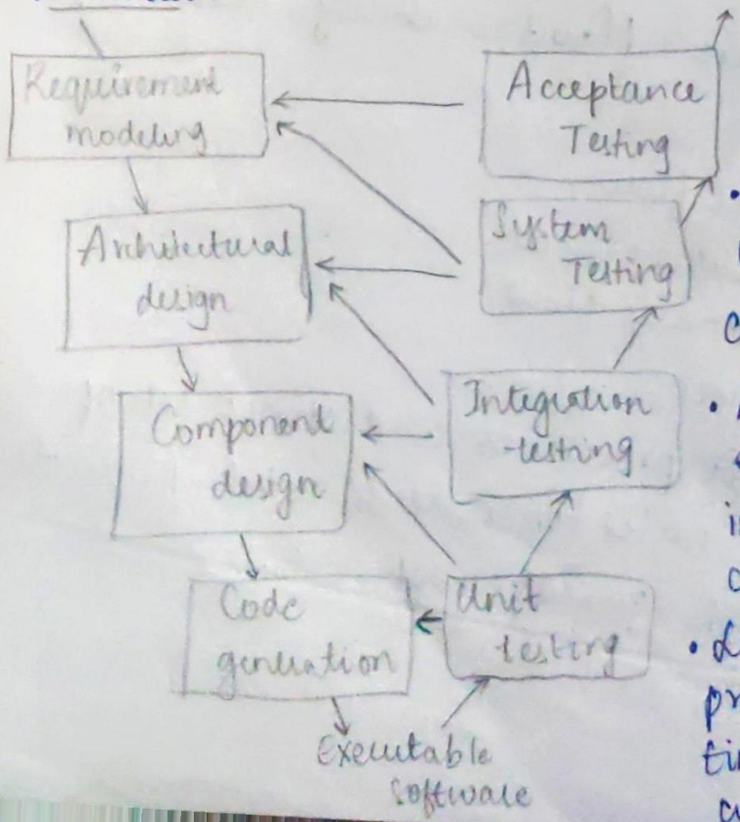
Waterfall model: (Classical model / Linear model)

advantages: clear definition of phases, systematic, easy to use



limitations:
difficult to know all requirements at beginning itself
- not suitable for changes
- does not fit well for large projects
- working version cannot be seen at an early stage.

V-model:



left side: VERIFICATION
right side: VALIDATION

- Approach: association of testing phase for each corresponding development stage
- Advantages: good for small projects, easy to use, emphasis on testing, improved traceability, better communication
- Limitations: not good for complex projects, cannot adapt to changes, time-consuming, high risk & uncertainity

Spiral model: (It is an EVOLUTIONARY model)

- Approach: Iterative nature of PROTOTYPING + Controlled & Systematic aspect LINEAR SEQUENTIAL MO

- realistic for large scale systems
- high amount of risk analysis
- good for large & mission-critical projects
- software is produced early in the software life cycle

} Advantage

- not widely used
- expensive
- highly specific expertise is needed
- success of project relies on risk analysis phase
- difficult to convince customer that evolutionary projects are controllable

} Limitation

- Key idea is spiral with many loops and no. of loops varies from project to project
- each loop of the spiral represents a phase in the development process.
- Phases involved : PLANNING, RISK ANALYSIS, ENGINEERING, EVALUATION and the next spiral begins again with new planning phase.

RADIUS OF SPIRAL ⟶ Cost of the project

ANGULAR DIMENSION ⟶ progress of the project

Concurrent Development model (It is an EVOLUTIONARY model)

- it is a series of framework activies such that all the activities exist concurrently but in different states
- Different states includes : i) None state
  ii) Awaiting state
  iii) Under development state
- Principle: defining events that will trigger transitions from one state to other state for each activity involved in SE.
- In this model, various SE activities are performed parallelly
- Advantages:
  i) applicable to all types Software development processes
  ii) easy to use & easy to understand
  iii) immediate feedback from testing
  iv) at any point of time, the ~~state~~ current state of the project is known
- Disadvantages:
  i) better communication b/w team members is necessary
  ii) status of various activities needs to be remembered

Prototyping model: (It is an EVOLUTIONARY model)

- prototype is a small working model

Steps: ① Communication ② Quick plan ③ Modeling Quick design ④ Construction of prototype ⑤ Deployment Delivery & feedback

- good approach when : the customer's objectives are general/vague

- can be used as a standalone process model
- Problems: i) While building prototype, the working is given more priority and efficiency is not taken care of.
  ii) Customer cries foul and demand that few fixes can be applied to the prototype.

Incremental process model:

- In this model, all the requirements are prioritized and according to that each requirement is fulfillied in each piece.
- Key idea: Software in "USABLE PIECES"
  each piece is relivant on previous pieces that are already delivered.

each piece has:

C - Communication
P - Planning
M - Modeling
C - Construction
D - Deployment

- Delivery is done in different increments and in each increment, required functionality is added.

As Increments gradually get delivered one after another, the functionalities of the software also get increased.

During the construction phase of one increment, the communication phase of the next increment is stated.

Highest priority requirements are included in early increments.

lower risk of overall project failure.

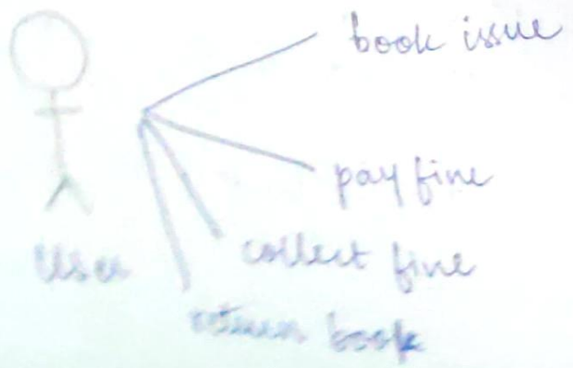
- Highest priority system services tend to receive the most testing.
- Advantages: Sequential approach, helpful when low no of staff are present, flexible, easier to test & debug, easier to manage risk.
- Limitations: Time-consuming, each increment is rigid & do not overlap each other, problem may occur with system architecture since all requirements are not gathered together.

Component-based development: (has many characteristics of spiral model)

- creating applications from prepacked software components
- Used when our objective is reusability of the code
- Whenever two projects have similar components, the component of one project can be used in the other project as well. This results to code reusability and since the code is reused, project cost also reduces. (like libraries/generic functions, templates, inheritance etc)

Unified process:

- It is use-case driven, architecture-centric, iterative & incremental software process closely aligned with UML.
- use-case driven: when different functionalities are to be done by various users.



all functions are done by users.

Component based development:

- Construction activities begin with identification of candidate components
- Classes used in previous projects are stored in library
- After identification of candidate classes, they are searched in the class library.
- If existing class library → use,
  if not existing → build class, store it, use it
- Advantages: software reusability, reduction in development cycle time, reduction in project cost, high productivity

Unified Process Model:

- Key aspects: use-case driven, architecture-centric, iterative & incremental
- Architecture-centric: Architecture is defined using UML
- In this model, the software architecture describes the system in different views/aspects.
- The different architectural views include
  i) Design view / logical view: (Object oriented decomposition)
  - it shows how are functionalities designed inside the system. (focusses on static structure & dynamic behaviour)
  - It also consists of vocabulary of the problem space and solution space
  - In this view, static aspects are represented using class & object diagrams and dynamic aspects

are represented using interaction diagrams, state chart diagrams and activity diagrams

ii) Implementation view: (Subsystem decomposition)

- it represents organization of core components and files
- It focuses on configuration management and the organization of different actual software modules in the development environment.
- this view represents physical-level artifacts

iii) Process view: (Process decomposition)

- it takes into picture, the non-functional aspects such as performance, scalability & throughput
- It tells about the process architecture
- Each process is a group of tasks that form an executable unit and each task is a thread of control that executes with association among different structural elements

iv) Deployment view / Physical view:

- It involves the mapping of hardware to software
- It represents the deployment of system in terms of physical architecture
- It addresses about the nodes that form the topology of system's hardware.
- Nodes are mapped to processes, tasks & objects
- It comprises of distribution, delivery & installation.

Agility

M Abhigna AP21110010343

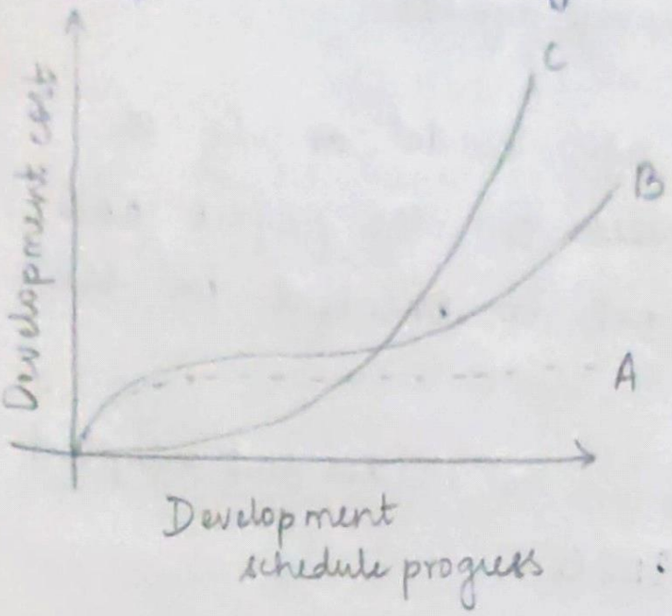
- Agility means effective, rapid and adaptive response to change and hence effective communication among the stakeholders
- In agile software engineering, we emphasize more on delivery rather than design & analysis
- In agile software engineering, an agile team (which means that the team is self-organizing in nature) develops the software using any of the agile process models.
- When the team is agile in nature, there is ~~efficient~~ effective communication and collaboration.
- Agile process: This is a process which is adaptable to changes in requirements, which has incrementality and works on unpredictability.
- Principles of an agile team include:
  i) Customer satisfaction
  ii) Accept changes in requirements
  iii) Deliver value frequently
  iv) Collaboration or working together
  v) Self-organized teams
  vi) Building projects with right enviroment
  vii) Encouraging face-to-face interaction

- Agile models: These models are primarily designed to he a project adapt quickly to change requests
- To finish the project using an agile model, agility is achieved by fitting the process to the project and removing activities that may not be essential for the project
- In agile process models, changes can be made and adapted at all stages of the SDLC
- Iterative development is done in agile models.
- Also, the entire project is broken down to smaller parts to help in reducing the project risk
- Phases of an agile model are:
  i) Requirements gathering
  ii) Design the requirements
  iii) Construction / Iteration
  iv) Testing / Quality assurance
  v) ~~Dely~~ Deployment
  vi) Feedback

Characteristics of a software engineer in an Agile development process:

i) Competence
ii) Common focus
iii) Collaboration
iv) Self-organization
v) Mutual trust and respect
vi) Decision-making ability
vii) Problem-solving ability

- Agility and cost of change:



Curve A

It represents the idealized cost of change using agile process

Curve B:

It represents the actual cost of change using agile process

Curve C:

It represents the cost of change using ~~agile processes~~ the conventional software processes

→ When we use conventional software development, the cost of change increases non-linearly as the project progresses

→ So when changes are made in conventional SD, the costs escalate quickly.

→ So the need is that even though changes are made, cost must not get escalated.

→ So the solution is AGILITY.

→ A well-designed agile process "flattens" the ~~co~~ change of cost curve.

Abhi
AP21

Extreme Programming

- Extreme programming is a software development model th part of what's collectively known as agile methodo
- XP is based on values, principles and practices
- XP aims to generate high-quality software with sma to mid-sized teams, ensuring that adapting to changing requirements is possible
- XP Process:
  - uses an object-oriented approach
  - In the XP process, we have 4 framework activities. They are:

i) Planning: Customer meets with development team and presents his/her requirements. Then the team develops a release plan which is further broken into Iterations to complete the needed functionality piece by piece

ii) Designing: XP follows KIS (keep it simple principle). So, a simple design is preferred over complex representation. The design acts an implementation guide for the project. XP encourages CRC cards that identify and organize object-oriented classes that are relevant to the current software increment. XP also encourages refactoring.

pike
lution:
ation of
erational
totype that
art of the
gn, whenever
cult design
encountered

iii) Coding: After completion of planning & designing, we do not move to coding directly. Instead a series of unit tests are developed. Then, coding is started and during coding, a key concept called "pair programming" is used.

XP recommends that 2 people work together at 1 workstation to create a code for a story. This provides a mechanism for real-time problem solving (2 heads are better than one) and real-time quality assurance (the code is reviewed as it is created

After pair programmers complete their work, the code they developed is integrated with work of others.

iv) Testing:

The unit tests that are created before coding are now tested on the code developed. In XP the individual unit tests are organized into "universal testing suite". This provides validation testing of the system on a daily basis. This also provides continual indication of progress and also raises warning-flags early.

XP has a defined set of 5 values that establish it's foundation. The 5 values are as below:

i) Communication (b/w software engineers & stakeholders)

ii) Simplicity (restriction to design only immediate needs)

iii) Feedback (derived from software itself, customers and software team members)

iv) Courage (discipline to design for today, recognizing that future requirements may change)

v) Respect (agile team inculcates respect among team members and stakeholders)

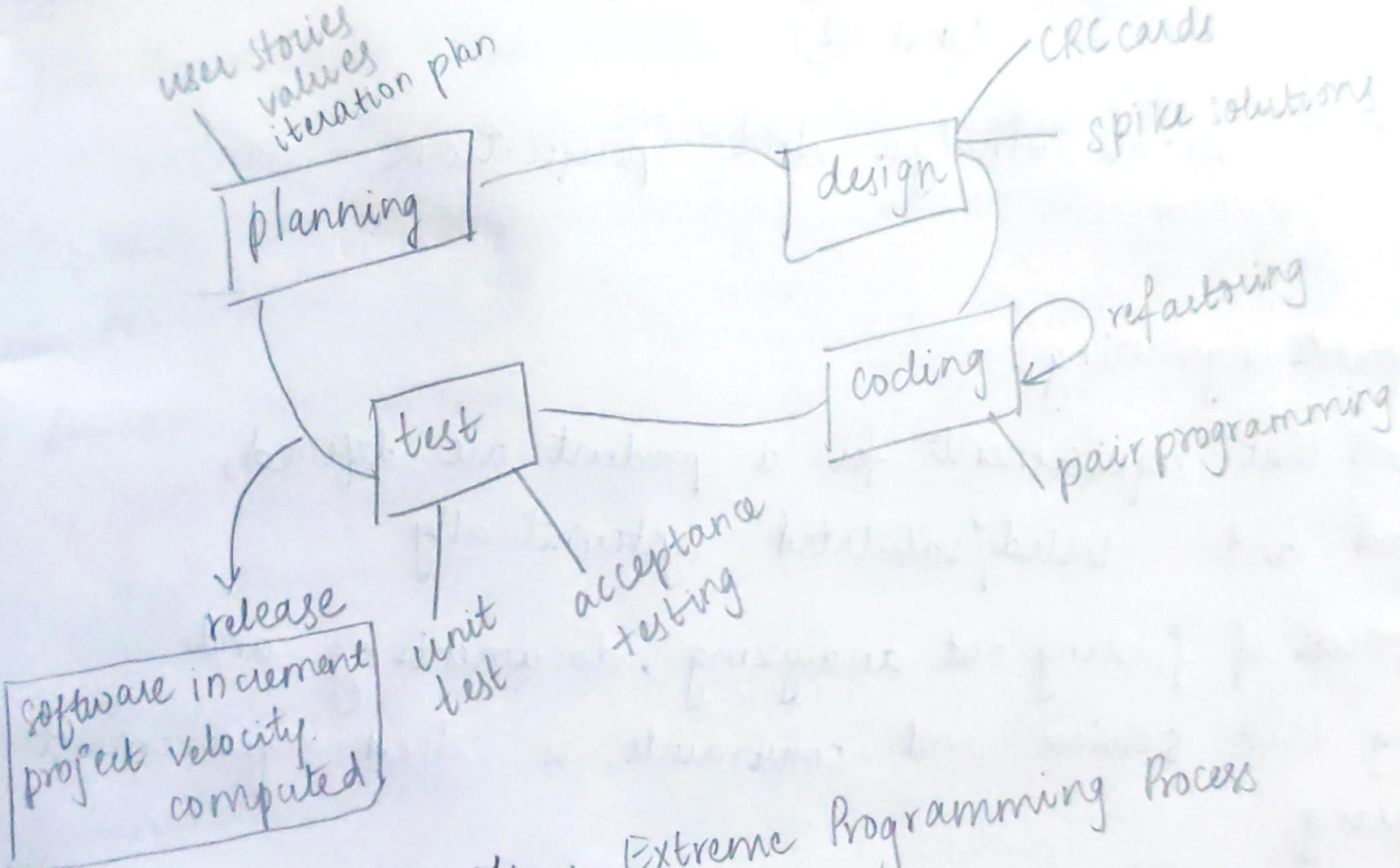


fig: Extreme Programming Process

Unit - (2)

Requirements Analysis And Specification

Requirements engineering:

- It means that requirements for a product are defined, managed and tested/validated systematically
- The process of finding out, analyzing, documenting and checking these services and constraints is called requirement engineering.
- RE determines the goals and constraints of the software
- RE generates a document that contains a description of the system and it's various functionalities.

Requirements:

- It is an abstract statement about the service/constraint that the system should provide.
- Types:

Requirements
- User requirements : tells about the intended tasks/functionalities of the software
- System requirements : minimum environment required to execute the software

Functional Requirements:

→ about services system should provide
→ about how system must react to particular inputs
→ about how system should behave in particular situations
→ depends on type of software being developed

- Non Functional Requirements
  → constraints on services or functions offered by the system such as timing constraints, constraints on the development process, standards etc
- Domain requirements:
  → these are requirements taken from the domain of the application rather than the specific needs of users

Requirements imprecision:

- Whenever requirements are designed with ambiguity, the system fails.
- When requirements are not stated clearly, the way of interpretation by different developers and users gets effected.

* Requirements should be complete and consistent:

Consistent: no conflicts or contradictions about the system facilities

Specify all the needs completely

should not give different outputs for same input at different times

Requirements characteristics: complete, consistent, feasible, modifiable, unambiguous, testable

Non-functional requirements:

these define system properties and constraints

- Reliability
- response time
- I/O device capability

there may also be process requirements like a specified IDE, programming language or development method

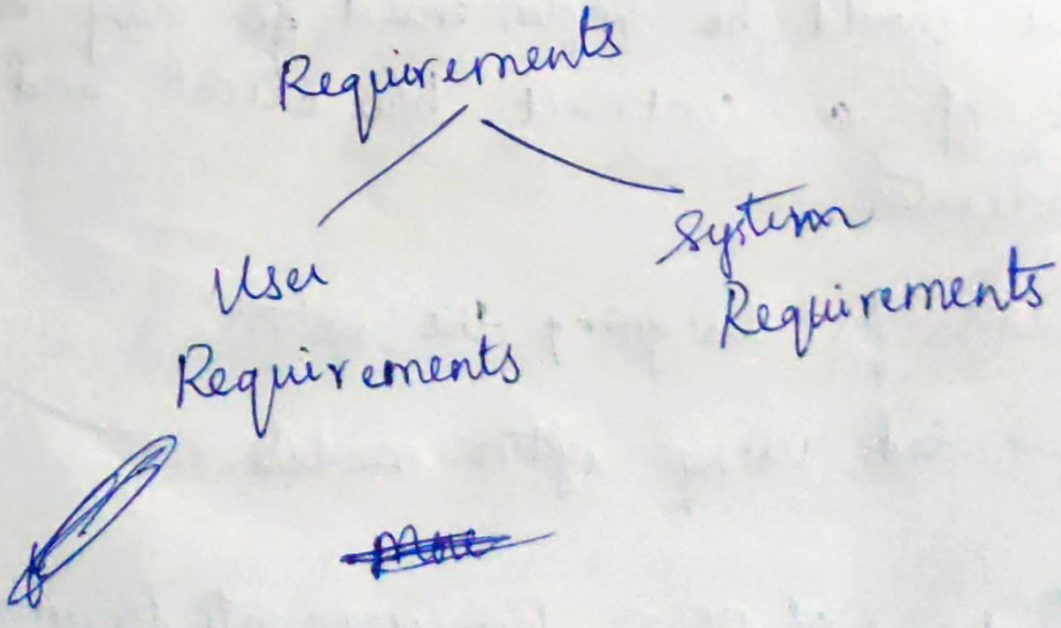
Software engineering:

Software requirements:
services that it provides
constraints on its operation } given by the system/software

RE: process of finding out, analyzing, documenting and checking these services and constraints.

A requirement is simply a high-level abstract statement about the service that a system should provide.

It can also be detailed, formal definition of a system function.



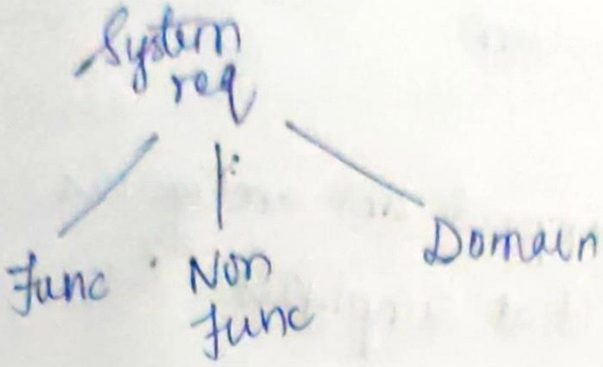
User req: (general)

- in natural language
- may have diagrams / tables
- describes about services & operational constraints of the system
- should describe functional / non-functional requirements
- should be understandable by the user who dont have proficiency in technical knowledge

System req: (more specific)

- detailed descriptions of the system's functions, services and operational constraints
- what should be implemented so may be part of a contract b/w client and contractor
- basis for designing the system
- illustrated using system models

Outcome of RE : Requirements document
↓ may be part of system development contract



Func req:

→ what system should do / not do
→ services of system
→ reaction to particular inputs
→ behaviour in particular situations

depends on: type of software
expected users
approach of the company

may vary from general to specific

↓

reflects local way of working or company's existing projects

Imprecision in requirements —cause→ problem

- easy for developer to interpret an ambiguous requirement in a way that simplifies it's implementation
- so customer need is not fulfilled and hence delays system delivery and cost ↑es

Func. reqments must be:

COMPLETE	&	CONSISTENT
all requirements needed/desired must be specified		requirements should not be contradictory.

but this is impossible for large/complex systems bcoz:

- easy to make mistakes/omissions
- many stakeholders - so different opinions & inconsistent needs
- inconsistencies may be evident only after deeper analysis / after delivery to customers.

Non functional req's:

- these are about the quality constraints that the system must satisfy according to the project contract.
- priority of these differs from project to project
- non-behavioral requirements
- Specify constraints of system as a whole
- Ex: Portability, Scalability, Reliability, Performance, Security etc.
- more critical than functional requirements because:
  failing to meet an non-functional requirement can mean that whole system is unusable.
- difficult to relate functional requirements to non-functional requirements. becoz:
  - non func affect overall architecture than individual components
  - (1) single non-func requirements generates lot of related func reqments
  - may generate requirements that restrict existing requirements.

non-functional reqments arise through needs because of:

- budget constraints
- product requirements
- organizational requirements
- need for interoperability
- ~~due to~~ safety regulations / privacy legislation

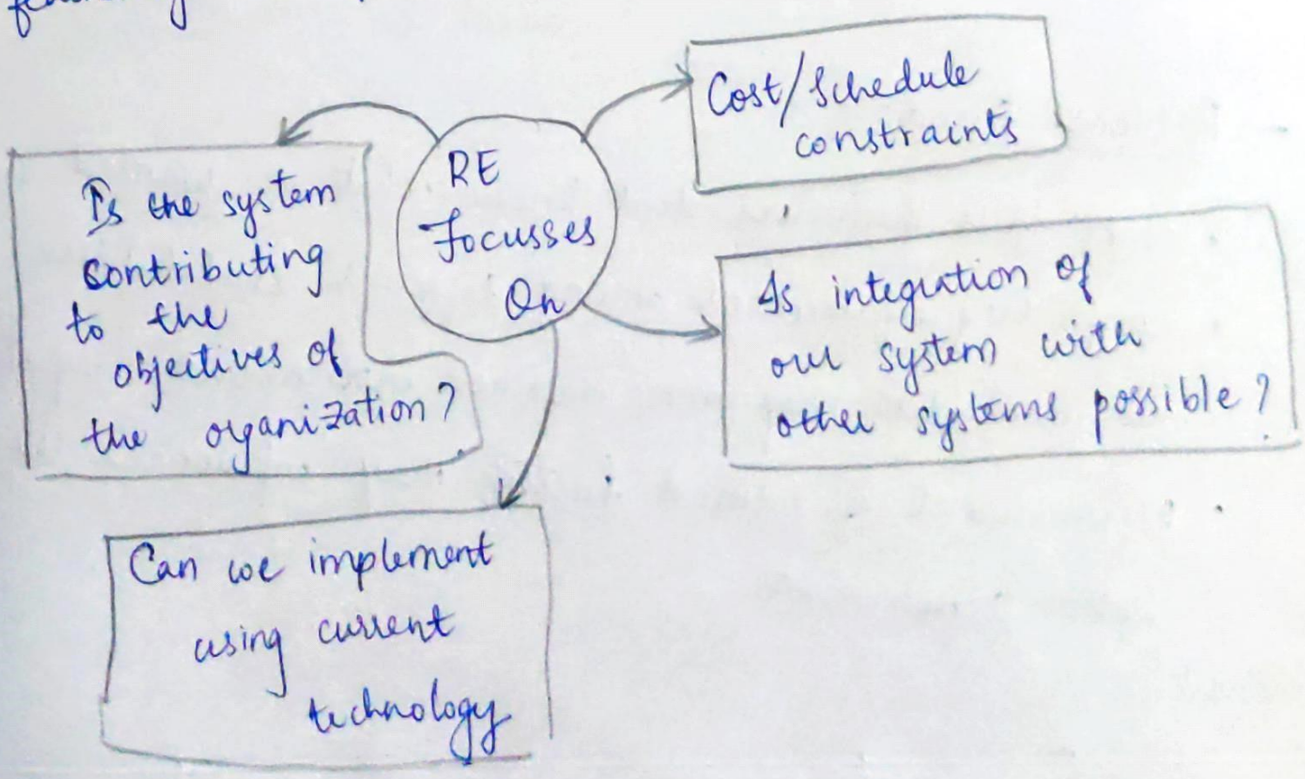
- Processes used for RE differ based on the application domain, people involved and organization
- Some activities that are generic to all RE processes are
  i) Requirement elicitation
  ii) Requirement analysis
  iii) Requirement validation
  iv) Requirement ~~specification~~ management

All these activities occur in an iterative way
↓
represented using the spiral view of RE process

- Feasibility Study:

Input: Set of preliminary business requirements
Output: Report (it should recommends whether it is worth to carry on with RE)

* New RE processes must always start with a feasibility study



Requirements elicitation:

- Collecting/gathering requirements for the system from users, customers & other stakeholders → end-users, managers, domain experts, trade unions
- requirements are gathered by: asking questions, writing answers, asking other questions etc
- While collecting the requirements, it is important to remember that developers & users have different mindset, expertise and vocabularies.
- While gathering requirements, there are different stages involved:
  - Requirements discovery (interacting with stakeholders to discover their requirements)
  - Requirements classification & organization (group similar requirements into clusters)
  - Requirements prioritization & negotiation (Prioritise the requirements and resolving conflicts)
  - Requirements specification. (frame the SRS document & send it as input to next round of spiral)

- Problems faced:
  - stakeholders themselves dont know what is wanted
  - conflicting requirements among different stakeholders
  - new stakeholders may come / bussiness environment may change
  - organisational & political factors may influence the system requirements

- Requirement elicitation techniques:
document analysis, observation, interview, prototyping, brainstorming, workshop, JAD, FAST, ~~Focus~~ Reverse engineering, Surveys / Questionnaire

* Reverse engineering: working model of system is present, but SRS no more exists. So we study the legacy system and develop an SRS and make changes to the existing system

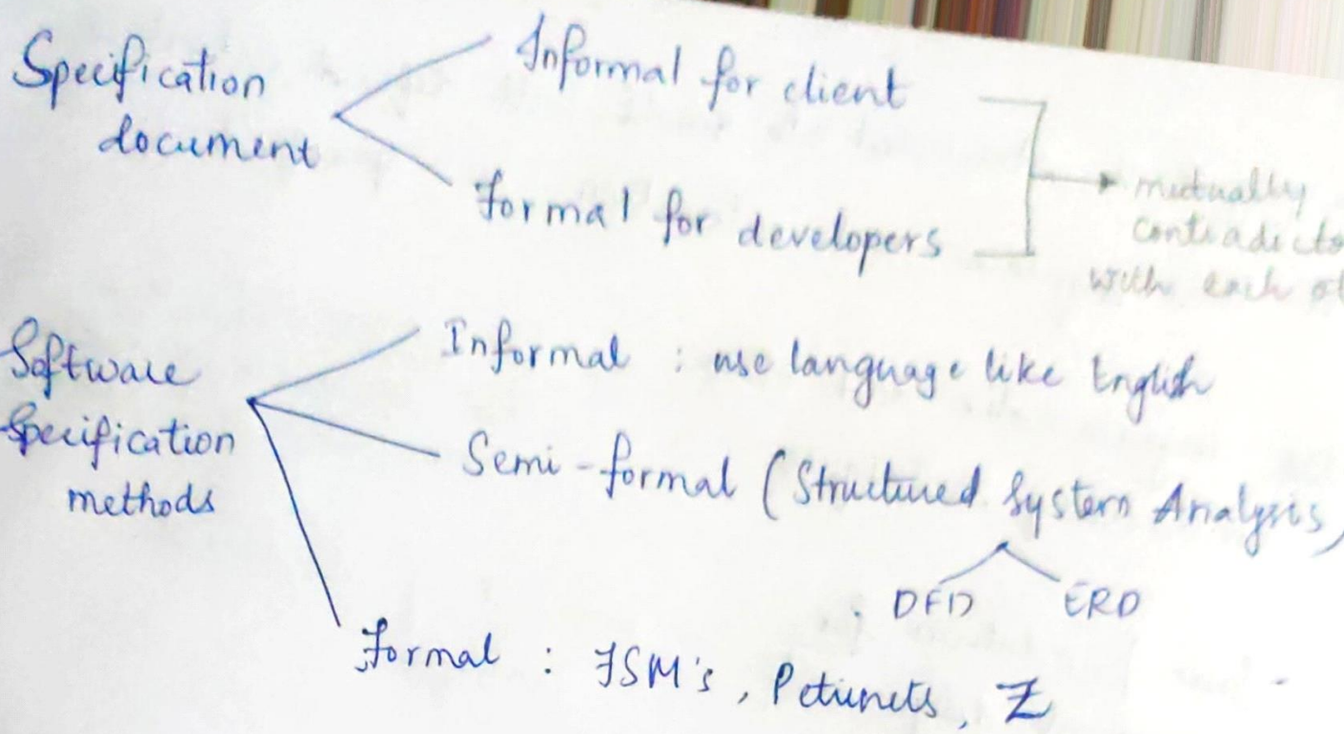
* Interview (formal/informal) — Closed interviews: based on pre-defined list of questions
— Open interviews: various issues are explored with stake holders

Effective interviewing: open minded, avoid pre-conceived ideas, don't be conservative

interviews — good → for getting understanding of what stakeholders want and how they may interact with system
— bad → for knowing domain requirements

* Questionnaires: gathering requirements with lesser cost and within less time
limitations: depends on resource availability
anonymity of the respondent

* Brainstorming: invent new way of doing things / when much is unknown → no criticism/debate
2 main activities: i) The Storm: generating as many ideas as possible (quantity ✓, quality ×)
ii) The Calm: filtering out of ideas to keep the best ones
Roles: Scribe (writes down all ideas), moderator (provoke), participants



Structured System Analysis:

- It is a development method that allows the analyst to understand the system and it's activities in a logical way.
- Systematic approach, that uses graphical tools that analyze and refine the objectives of an existing system and develop a new system specification which can be easily understandable by user.

- ATTRIBUTES:
  - graphic → specifies the presentation of application
  - divides processes so that it gives a clear picture of system flow
  - logical rather than physical (elements of system do not depend on hardware)
  - approach that works from high-level overviews to lower-level details

- The project is structured into small, well-defined activities and their sequence and interaction is specified

Use diagrammatic and other modelling techniques : for precise definition that is understandable by both users and developers

- Firm foundation for subsequent design and implementation : provided by structured analysis

Asking right questions : system analysts must ask questions that it is often difficult for a technical person

Structured System Analysis
- Modelling System function (DFD, System flowchart)
- Modelling stored data (ER, data dictionary)

Petrinets:

Petrinet is a mathematical modeling tool that is defined using the following tuples:

1. Set of places, P, represented using ◯
2. Set of transitions, T, represented using ▭
3. Set of I/p functions, $\underline{I}$
4. Set of O/p functions, O