Date:

Topic:

Def: It is a systematic, disciplined, cost-effective, techniques for software development

→ Engineering approach to develop a software

Software Development life cycle: (SDLC)

A structured process followed for developing software applications, ensuring high quality and alignment with customer expectations.

(a) Requirement analysis: (Product owner / Project manager)

gathering, analysing, validating and documenting what customer needs from software product

→ Types of requirements:

1. Functional - what the system should do
2. Non-functional - how the system should behave
3. Domain - constraints or features from the application domain

(b) Design: (System Architect / UI/UX designer)

Translating the SRS into a blueprint for building the software

→ Types of system design:

High level design (Architectural Design) - identifies the system architecture and major modules.

Low level design - detailed design of modules, including algorithms, data structures and interfaces.

→ Elements of system design:

1. Architecture design
2. Data design
3. Interface design
4. Component design
5. UI design

Date:

Topic:

(c) Development : (Frontend / Backend dev.)
where the actual coding of the software takes place based on finalised system design.
most resource-intensive and time consuming part of SDLC.
→ Objectives :
- convert design specifications into source code
- ensure the code meets functional & non-functional req.
- follow coding standards and best practices.

(d) Testing : (Solutions Architect / DevOps / Tester)
where the developed software is evaluated for quality, functionality and performance to ensure it meets the requirements and is free from defects.
→ Types of testing :
1. Unit testing : Test individual fns or modules
2. Integration : check how modules work together
3. System testing : Test the complete system as a whole.
4. Acceptance : Validate software against user req.
5. Regression : Ensure new changes haven't broken existing functionality
6. Performance / Load testing : access software speed, scalability and resource usage.

(e) Deployment : (Data Administrator / DevOps)
where final software products is delivered to the end users or client environment. This phase ensures the product is installed, ~~and~~ configured and ready for use.

Date:

Topic:

(f) Maintenance: (Users / Testers / support managers)
Starts after software is deployed and used by end users. If focuses on keeping the software reliable, secure and up to date by fixing issues and updating features as needed.

→ Process framework activities:
a. Communication
b. Planning
c. modeling
d. construction
e. deployment

→ Umbrella activities:
a. Project tracking and control
b. risk management
c. quality assurance
d. configuration management
e. technical reviews

Process flow:
Describes how the framework activities and the actions and tasks that occur in each framework activity are organised with respect to sequence and time.
↳ Linear
↳ Iterative
↳ Evolutionary
↳ Parallel

Date:

Topic:

Software Methodologies:

Structured approaches to planning, designing, developing, testing and maintaining software systems.
They define how software projects are managed and executed.

(a) Waterfall model:

Requirements → Design → Development → Testing → Deployment → Maintenance

↳ Sequential
↳ No overlap

→ Features:
↳ Sequential approach - each phase of the project is completed before moving to the next one.
↳ Document driven - depends on documentation to ensure that the project is well-defined and the team is working towards a clear set of goals.
↳ Quality control - places high emphasis on quality control and testing to ensure that the final product meets the requirements and expectations of stakeholders.
↳ Rigorous planning - project scope, timelines and deliverables are carefully defined and monitored (rigid)

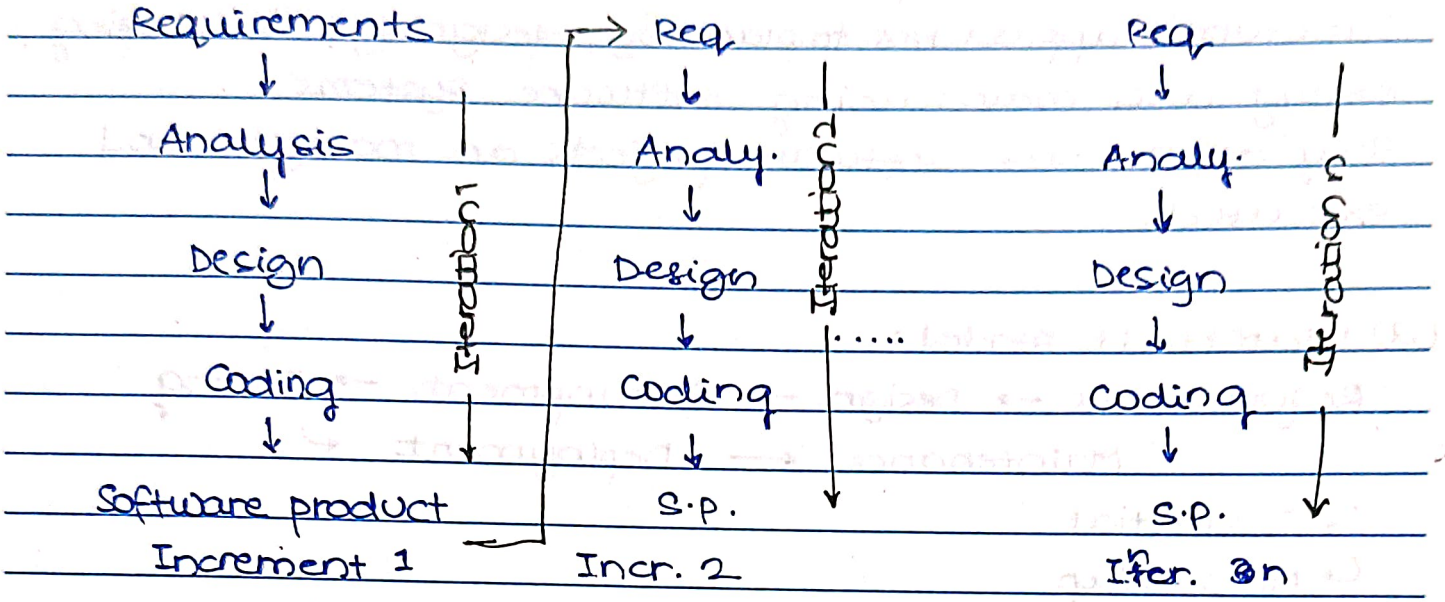
→ Disadvantages:
↳ No feedback
↳ No experiment
↳ No parallelism
↳ High risk
↳ 60% effort maintenance.

→ Eg: Boeing, IBM

Date:

Topic:

(b) Incremental Model:



Def Method of software dev. where the system is built step by step. Instead of delivering the whole system at once, it is developed and delivered in small parts called increments. Each increment builds upon the previous one by adding new ~~feature~~ functionalities.

→ Characteristics:

- Partial system delivery
- Early functionality
- Customer feedback loop
- Flexible to changes
- combination of linear and iterative approaches [combines structured approach of waterfall with flexibility - supports both planning and ongoing improvem]

Date:

Topic:

Types :

→ Staged delivery model.

Requirements Definition → Architecture Design → Subsystem 1 Construction ↘

Validation Test ← Verification Test ← Substystem n construction

→ Parallel development model.

Requirements Definition → Architecture design ↘

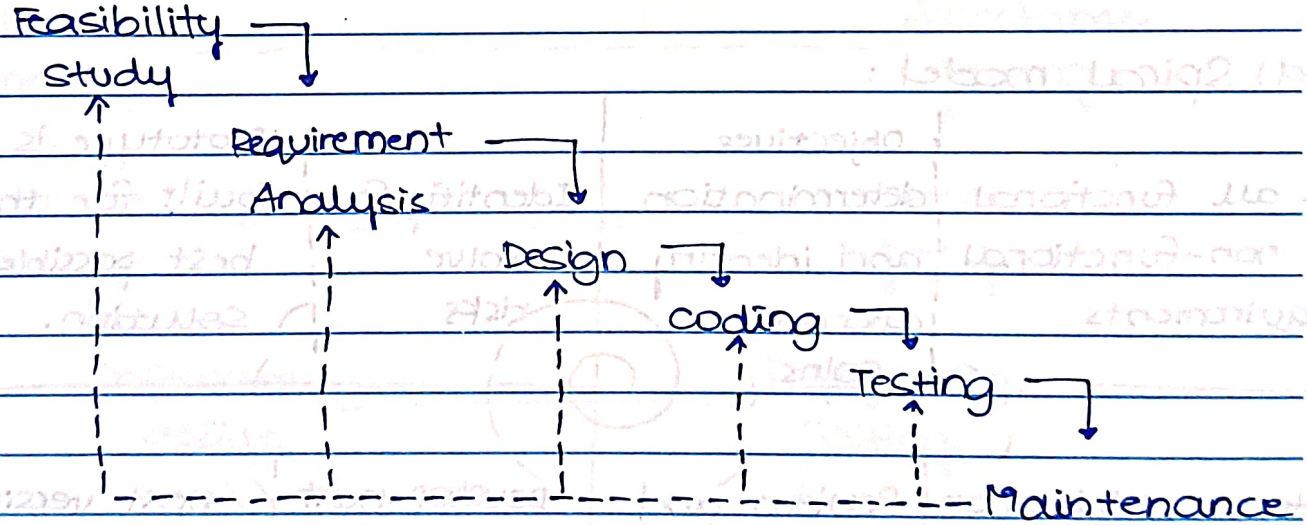
Team 1 : Subsystem 1 constr.

Team 2 : Subsystem 2 constr.

⋮

Validation Test. ← Verification Test ↵

(c) Iterative model / Iterative waterfall model



* (Diff between iterative & incremental) - ppt. pg 30

Date:

Topic:

→ Applications :

↳ Essential needs are established, but later finer points may become relevant

↳ Programmers have learning curve to climb when they use new technology.

↳ Resources needed to complete large project are constrained, hence on smaller ~~side~~ scale, the automation is more temporary.

→ Drawbacks :

↳ Difficult to incorporate change requests.

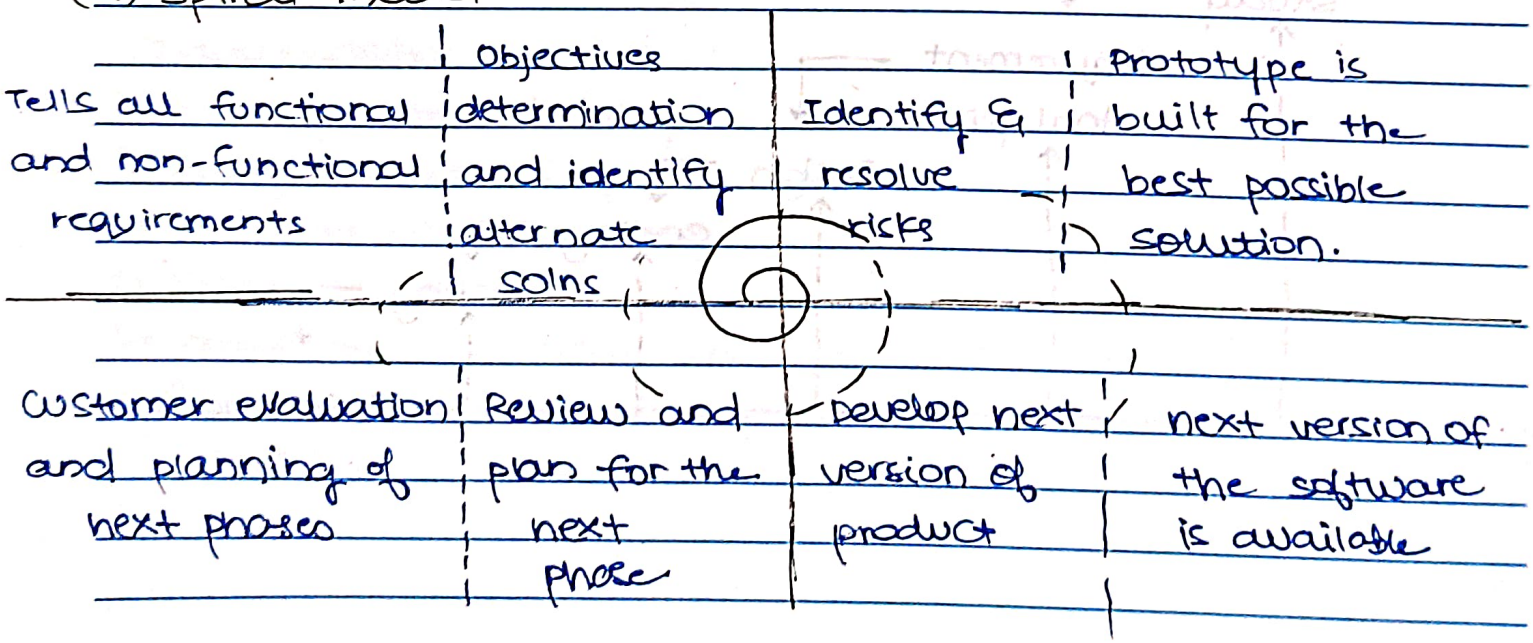
↳ Incremental delivery not supported.

↳ overlapping of phases not supported

↳ risk handling not supported

↳ limited customer interactions

(d) Spiral model :



Date:

Topic:

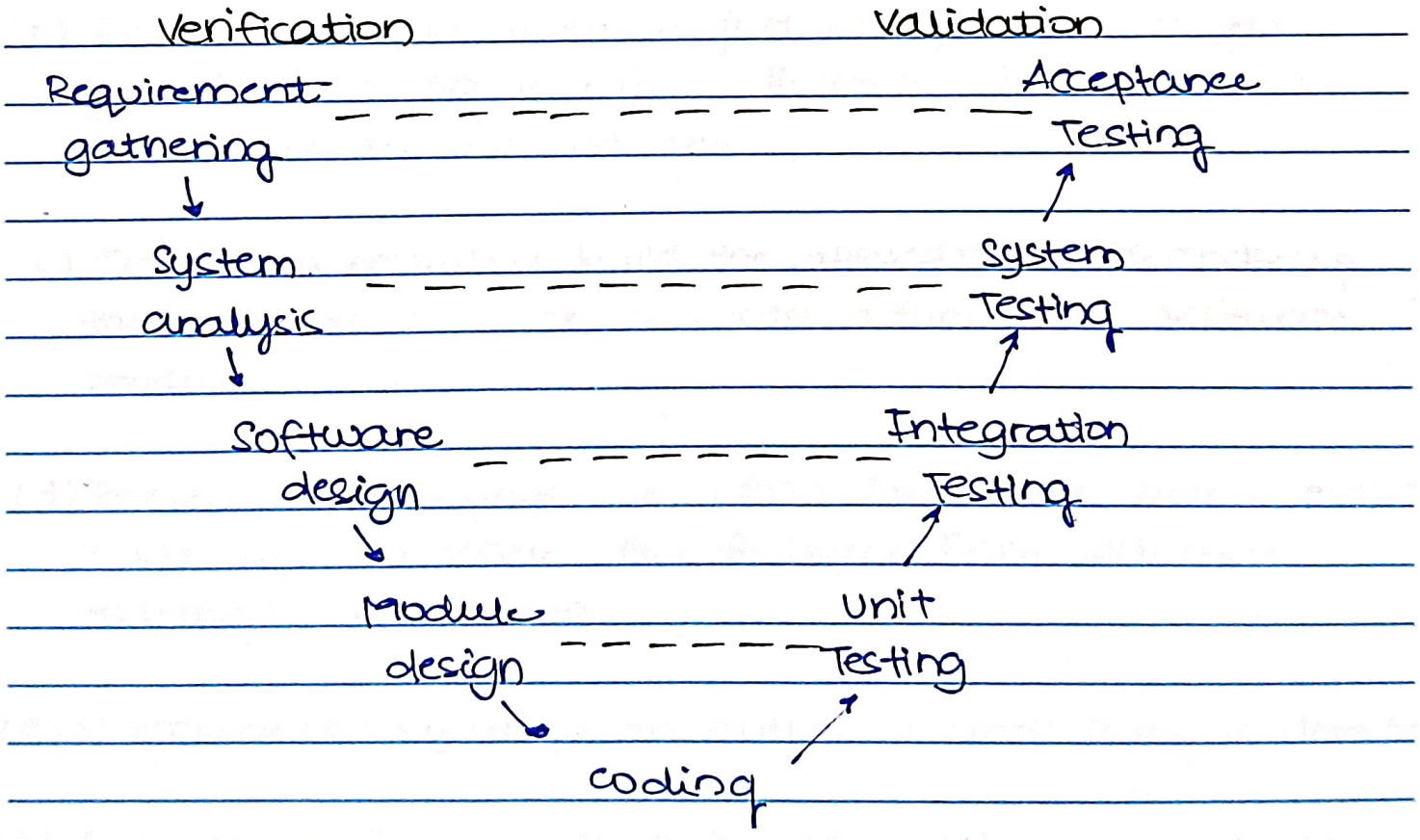
→ risk handling
→ radius of spiral = cost
→ angular dimension = progress
→ meta model.

→ Advantages :
  ↳ risk handling
  ↳ large projects
  ↳ flexible
  ↳ customer satisfaction

→ Disadvantages :
  ↳ complex
  ↳ expensive
  ↳ too much risk analysis
  ↳ time

(e) V-model (Verification & validation model)

→ extension of waterfall model
→ testing is associated with every phase of lifecycle.
~~→ verification~~ (.



Date:

Topic:

→ Advantages:

↳ Time saving

↳ Good understanding of project in the beginning

↳ Every component must be testable.

↳ Progress can be tracked easily

↳ Proactive defect tracking

→ Disadvantages:

↳ No feedback so less scope of changes

↳ Risk analysis not done

↳ Not good for big or object oriented project

Agile Software Engineering

large projects ⇉ small chunks → release → feedback

Date: _____ re-release ← enhance ↵

Topic: _____

Agile software engineering:

Move quickly, project divide into small phases, face to face communication, changes time.

concept → Inception → Iteration → Release
Retirement ← Maintenance ↵

(a) Concept: Defining the scope of the project and product. required to write the minimum documentation because the focus is on working software
↳ Project scope - work that needs to be done in order to deliver a product or service
↳ Product scope - defined as functions or features that characterise a product or service.

(b) Inception: Goal is to do just enough work to get your team going in right direction. Start building and be ready for changes

(c) Iteration: Actually build the application. The mock-up that was previously created into a functional software product

(d) Release: release your application. Production environment is set up and stable. Get feedback from customers to prepare next version.

(e) Maintenance: system is on autopilot until a bug is detected.

(f) Retirement. The product or feature is either obsolete or being replaced by something else.

Date:

Topic:

→ Principles of Agile Methodology:

1. Customer satisfaction
2. changing requirement
3. Frequent delivery
4. Promoting collaboration
5. Motivated individuals
6. Face to face communication
7. Maintain a constant pace
8. Measure progress.
9. Technical excellance
10. Simplicity
11. Self organised Teams
12. Continuous Improvements.

→ Advantages:
- frequent delivery
- ~~for~~ phase to phase communication with client
- more chainges
- less time

→ Disadvantages:
- less documentation

↳ Agile methodology - most popular

→ SCRUM: Diagram in ppt

Management framework that teams use, to self-organise tasks and work towards a common goal.

It divides the whole process ~~to~~ in small sprints.

→ Advantages:
- fast moving and money efficient
- scrum framework works by dividing the large product into small sub-products. (divide and conquer)
- customer satisfaction is very important
- adaptive in nature because of short sprints.
- it ~~rely~~ relies on constant feedback. therefore the quality of product increases in less amount of time.

→ Disadvantages:
- ~~no~~ more efficient in small team size.
- no changes in sprint.

Date:

Topic:

→ Extreme Programming (XP):

one of the most important software development frameworks of Agile models. It is used to improve software quality and responsiveness to customer requirements.

↳ customer satisfaction

↳ adaptability

↳ high quality code

→ continuous delivery of small functional parts of the project and take feedbacks

→ 5 core values:

a. communication (open & constant comm.)

b. simplicity (simple design & decisions)

c. feedback (continuous feedback)

d. courage (bold decisions)

e. respect (collaborative & supportive environment).

SCRUM:

→ lightweight, iterative, incremental framework

→ the development time for each sprint is maximised and dedicated, thereby managing only one sprint at a time.

→ scrum team has scrum master, product owner with constant communication on daily basis.

Keywords:

Backlog – place where basic design, requirements are stored.

Daily scrums – a regular meet with p.m. and stakeholder. (10-15min)

Scrum master –

Product owner –

Date:

Topic:

→ Agile principles:

- working software is the key measure of progress in a project. hence, software should be developed and delivered rapidly in small increments.
- Face to face communication is preferred over documentation
- continuous feedback and involvement of customers are necessary for developing good-quality software
- a simple design that involves and improves with time is better than doing an elaborate design up front

→ RAD Model (Rapid Application Development).

- a type of iterative and incremental model in which there is a concise development cycle.
- used when the requirements are fully understood and the component-based construction approach is adopted.
- multiple teams work parallelly.

→ Key features:

- Emphasis on fast development cycles
- involves user participation throughout
- uses component based construction.
- requires skill developers and modular approach.

→ Phases of RAD:

- Business modelling
- Data modelling
- Process modelling
- Application Generation
- Testing and turnover.

Diff betⁿ all models!!

Date:

Topic:

MODULE 2!!:

→ Advantages:

- Faster delivery
- Active user involvement
- Reusable components.
- Better risk management for changing req.

→ Limitations:

- needs skilled workforce.
- not suitable for large scale complex systems.
- heavy dependence on modelling tools.
- difficult with poor modularization.

Date:

Topic:

MODULE 2.1 : Requirement Engineering:

systematic process of defining, documenting and maintaining requirements for software system.
It ensures that the software being developed meets customer needs, is feasible, and can be implemented within constraints (time, budget, technology).

→ Objectives:
- ↳ understand customer needs clearly
- ↳ translate those needs into precise requirements.
- ↳ avoid ambiguity, inconsistencies and incompleteness
- ↳ provide a foundation for design, dev. and testing.

→ Difficulties of Requirement Engineering:
- ↳ Req. are difficult to uncover
- ↳ req. changes
- ↳ tight project schedule
- ↳ communication barrier
- ↳ lack of resources

→ Types of Requirements:
- ↳ Functional req. - define what the system should do.
- ↳ Non-functional req. - the system should allow users to log in using email and password.
- ↳ Domain req - constraints from application domain

→ Importance of RE:
- ↳ prevents scope creep and misunderstanding
- ↳ reduces development cost and rework
- ↳ provides clarity for developers and testers
- ↳ increases customer satisfaction by delivering what is actually needed.

→ Quality function deployment (QFD) - rate the requirement based on the most important to least.

Date:

Topic:

→ Use case approach - structured desc of user req. graphical repr. of how system works.

→ RE process:

(a) Elicitation (Gathering)

collect requirements from stakeholders

Technique: interviews, questionnaire, use cases, etc

(b) Analysis:

check req. for conflicts, feasibility and clarity

categorise into functional and non-functional.

(c) Specification (SRS)

acts as a contract between stakeholders and developers

(d) validation

ensures req. are correct, consistent and testable

Techniques: reviews, prototyping, model validation.

(e) management:

handle changes in req. during project lifecycle.

includes versioning, traceability and impact analysis.

(a) Gathering:

Process of collecting needs, expectations, and constraints from stakeholders

- identify stakeholders: customers, end-users, managers, regulatory bodies.
- elicitation technique: interviews (brain storming), questionnaire, surveys, workshops, ethnography, document analysis, prototyping
- task analysis
- scenario analysis
- form analysis

→ Delphi technique => write the req on a piece of paper, then these req are exchanged and others revise the req.

→ FAST (facilitated application specification technique) joint team of customer & developer. make small teams focussing on a particular subtopic of requirements.

Ecommerce website:

→ gathering => interviews with sellers, buyers & logistics team

Date: → analysis => Functional - users should track their order

Topic: => Non-functional - system should handle 10000 users simultaneously

(b) Analysis:

The process of studying, refining, modeling, and validating the gathered requirements to ensure they are clear, consistent and feasible

- classification of req.
  - ↳ Functional vs non-functional
  - ↳ user vs system
  - ↳ domain vs interface
- prioritization: Moscow (Must have, Should have, Could have, Would have)
- Feasibility study: technical, operational, economic, legal feasibility.
- modeling - use case diagrams, context diagrams, DFDs.
- validation: reqs are checked for clarity, consistency, completeness and correctness

→ challenges in Req. analysis and gathering:

- communication gap bet^n customers and developers.
- stakeholders may not know what they want
- requirements keep changing (volatility)
- conflicting reqs from different stakeholder.
- ambiguity in natural language descriptions.
- incomplete set of reqs.

→ Deliverables:

- ↳ SRS doc.
- ↳ Requirement models (Use cases, UML)
- ↳ Requirement Traceability Matrix (RTM) for validation.

Date:

Topic:

(c) → Software Requirement Specification (SRS)

formal document that describes what a software system should do and how it should perform. It is prepared in the req. analysis phase.

- It acts as a contract betn stakeholders & dev. team.

- Why spend time and resource on SRS:
  - ↳ forms an agreement betn customers & developers
  - ↳ reduces future works
  - ↳ provides a basis for estimating costs and schedules.
  - ↳ provides a baseline for validation & verification.
  - ↳ facilitates future extensions.

- Good SRS:
  - ↳ concise
  - ↳ implementation indep.
  - ↳ traceable
  - ↳ modifiable
  - ↳ verifiable
  - ↳ identification of response to undesired events.

- Bad SRS:
  - ↳ over-specification
  - ↳ forward references
  - ↳ wishful thinking
  - ↳ noise

- Components of SRS:

(a) Introduction:
  - ↳ Purpose of the software
  - ↳ scope of the system
  - ↳ intended audience
  - ↳ definitions, acronyms, abbreviations

Date:

Topic:

(b) Overall description
- ↳ product perspective (relation with other systems)
- ↳ product functions (summary of major features)
- ↳ user characteristics
- ↳ assumptions and dependencies.

(c) Functional requirements:
what the system should do (features, fns, behaviours).

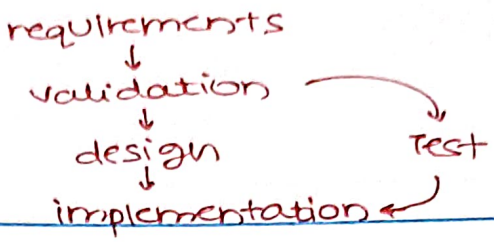
(d) Non-functional requirements:
non-negotiable obligations that must be supported by the software
quality attributes (performance, security, usability, scalability, reliability)

(e) System models (optional but recomm.)
- ↳ use case diagrams
- ↳ data flow diagrams
- ↳ entity-relationship diagrams.

(f) constraints
Technical limitations, regulatory standards, hardware/software req.

Date:

Topic:



(d) → **Use cases:**

use case describes a sequence of interaction between a user and the system to achieve a specific goal.

- Elements:
  - ↳ actors
  - ↳ precondition
  - ↳ main flow
  - ↳ alternate flows
  - ↳ postconditions.

(e) → **Requirement validation**

process of ensuring that the document req truly represent the user's needs and are correct, consistent and testable.

- Objectives:
  - ↳ correctness
  - ↳ completeness
  - ↳ consistency
  - ↳ feasibility
  - ↳ testability
  - ↳ traceability
- Techniques:
  - ↳ reviews and inspections - formal walkthrough with stakeholders, peer review of SRS
  - ↳ Prototyping - building mockups or working models to validate usability and requirements.
  - ↳ model validation - using UML diagrams to cross-check consistency
  - ↳ checklists - using predefined questions
  - ↳ automated validation tools - tools that check for ambiguity, redundancy or inconsistencies in requirements.

Date:

Topic:

→ Common issues during validation:
- ↳ ambiguity in language ("fast", "high")
- ↳ contradictory requirements from diff. stakeholders
- ↳ overlooked requirements due to incomplete elicitation
- ↳ unrealistic req. (zero downtime).

→ ~~Agile~~ Industrial prac:
- ↳ Agile teams often use acceptance criteria in user stories as a validation step.
- ↳ In large enterprises, Joint application development (JAD) sessions and req. workshops are common.
- ↳ many companies mandate formal req. review sign-off before moving to design phase.

- This is a quality assurance activity at the requirements stage, it reduces costly errors later in design and dev.

d)
→ Defining user requirements:
describe what the end users expect from the system in terms of features, performance and usability. usually written in ~~formal~~ natural language, supported with diagrams, so that both technical and non-technical people can understand them.
- ↳ clear and unambiguous
- ↳ complete
- ↳ consistent
- ↳ feasible
- ↳ verifiable

Date:

Topic:

(f)

→ Requirement Traceability

ability to link requirements to other projects artifacts to ensure that every req is implemented, tested and verified.

(a) Forward traceability:

- ensures every req. is implemented and verified.
- linking req → design → code → test cases.

(b) Backward traceability:

- linking design / code / test cases → original req.
- ensures nothing unnecessary is built

(c) Bi-directional

- both forward + backward
- ideal in safety-critical and regulated industries

Importance:

↳ confirms complete coverage of req.
↳ helps in impact analysis when req. change
↳ supports regulatory compliance.
↳ improves quality and maintainability of software

How to achieve requirement traceability:

↳ req identification - assign unique ids to req.
↳ RTM - a table mapping reqs → design → code → test cases.
↳ tools and automation - use CASE tools (Jira, DOORS, Jama)
↳ change management - maintain history of changes and update trace links.
↳ review and validation - regular audits to ensure req. are linked across all phases.

Date:

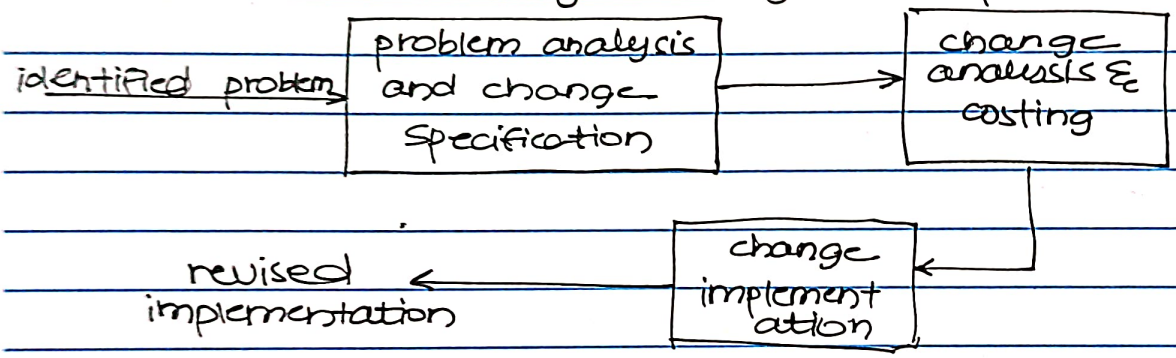
Topic:

Requirement Traceability Matrix:

Requirement id	Requirement Descr.	Design module	Code module	Testcase id

- Traceability policies – to keep a record of the defined relationships between each requirement and the system designs which will help to minimise the risks.
- Tool support – Tools like MS Excel, spreadsheets or a simple database system can be used.

→ Requirement change management process:



→ Working of requirement management process