Page No:
Date:

- Software Engineering - A systematic collection of good program development practices and techniques
  - An engineering approach to develop software
- Software product - Computer programs and associated documentation such as requirements, specifications, design models and user manuals
  - May be developed under/for a specific customer or may be developed for general market.
  - Types
    (i) Generic - Developed to be sold to a range of different customers
    (ii) Custom - Developed for a single customer according to their specification

Without using software engineering principles, it would be difficult to develop large programs. In industry, it is usually needed to develop large programs to accomodate multiple functions. A problem with developing such large commercial programs is that the complexity and the difficult levels of the programs increase exponentially with their size.

* Software Engineering principles use two important techniques to reduce problem complexity : abstraction & decomposition

Abstraction principle implies that problems can be simplified by omitting irrelevant details. In other words, main purpose of abstraction is to consider only those aspects of the problem that are relevant for certain purpose and suppress other aspects that are not relevant for the given purpose

→ Powerful way of reducing complexity of the problem.

- Decomposition - A complex program is divided into several smaller problems and then smaller problems are divided one by one.

Characteristics of a good software

(i) Operational - Budget, Usability, Efficiency

(ii) Translational - Portability, Interoperability, Reusability

(iii) Maintainance - Modularity, Flexibility, Scalability

Need of software engineering

Because of higher rate of change in user requirements and environment on which the software is working.

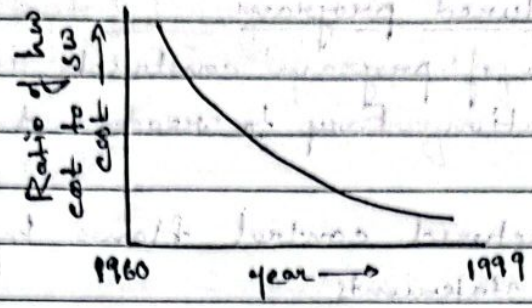
- Large Software - As the size of software become large, engineering has to step to give it a scientific process

- Scalability - If the software process were not based on scientific and engineering concepts, it would be easier to recreate new software than to scale existing one.

- Dynamic Nature - The always growing and adapting nature of software hugely depends upon the environment where user works.

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

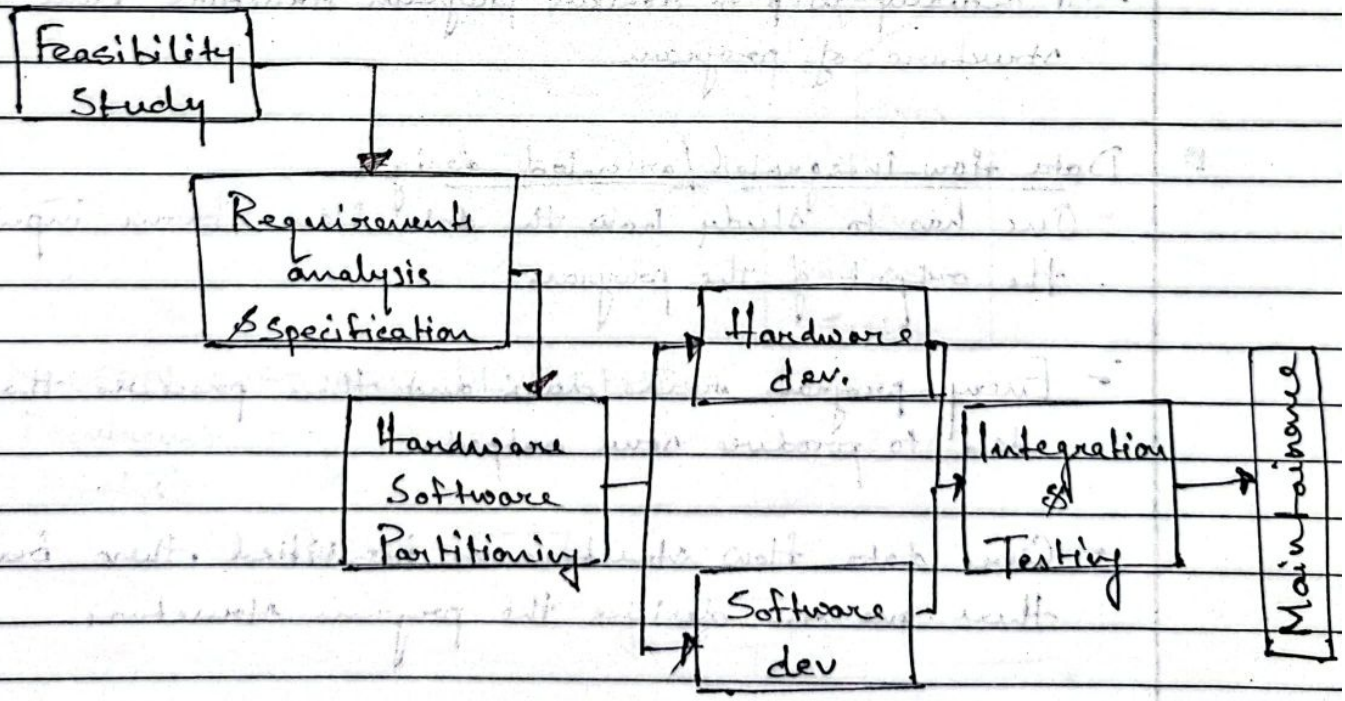


Relative Cost of hardware and softwares

- Factors contributing to software crisis
  - Larger problems
  - Lack of adequate training in s/w
  - Increasing skill shortage
  - Low productivity improvements

- Programs v/s software products

Programs	Software products
- Small in size	- Large
- Ad-hoc development	- Systematic development
- Single developer	- Team of developers
- Lacks proper documentation	- Well documented and user manual prepared



Project Management

Page No:
Date:

Features of a structured program

- It uses three types of program constructs i.e. selection, sequence and iteration. Easy to read and understand.
- They avoid unstructured control flows by restricting the use of GOTO statements
- It consists of well partitioned set of modules
- Uses single entry, single exit program constructs such as if-then-else, do while etc.

STRUCTURED PROGRAMMING PRINCIPLE EMPHASIZES DESIGNING NEAT CONTROL STRUCTURES FOR PROGRAMS

Data structure oriented design

- It is important to pay more attention to design of data structure of program rather than the design of its control structure.
- It actually help to derive program structure from data structure of program.

Data flow integrated/oriented design

- One has to study how the data flows from input to the output of the program
- Every program reads data and then processes that data to produce some output.
- Once data flow structure is identified, then from there one can derive the program structure.

Page No:
Date:

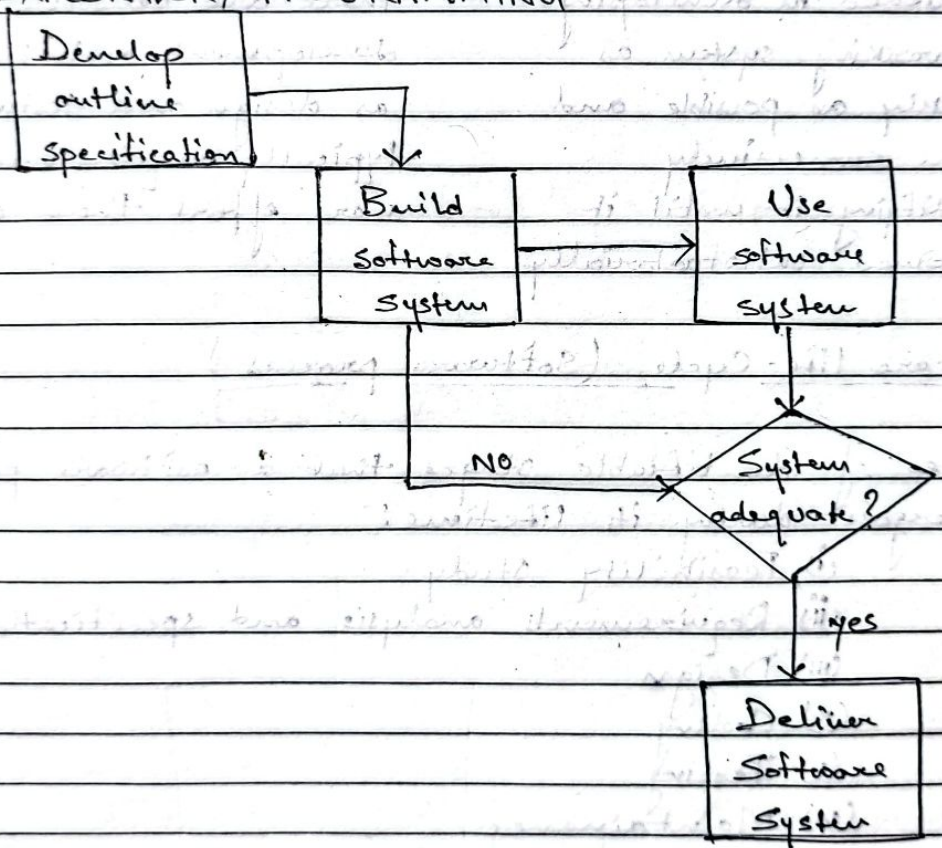
Object Oriented Design

- It has intuitively appealing design approach in which natural objects occuring in the problem are first identified

- Relationships among objects are determined

- Each object essentially acts as a data hiding entity

EXPLORATORY PROGRAMMING



MODERN SOFTWARE DEVELOPMENT PROCESS

Requirement Analysis → Design → Coding → Testing → Maintainance

Exploratory Style	Modern Style
• Emphasis on error correction.	• Emphasis on error prevention
• Coding was considered synonymous with software development.	• Coding is regarded as only a small part of the overall software development activities
• Believed in developing a working system as quickly as possible and then successively modifying it until it performed satisfactorially	• There are several development activities such as design and testing which typically require much more effort than coding

Software life Cycle (Software process)

Series of identifiable stages that a software product undergoes during its lifetime:

(i) Feasibility Study
(ii) Requirements analysis and specification
(iii) Design
(iv) Coding
(v) Testing
(vi) Maintainance

Life Cycle Model

Description or diagrammatic model of software life cycle.

- Identifies all the activities required for product development
- establishes a precedence ordering among the different activities
- Divide life cycle into phases

Page No:
Date:

Why life cycle model?

- A written description
  - forms a common understanding of activities among software developers
  - helps in identifying inconsistencies, redundancies and omissions in the development process
  - Helps in tailoring a process model for specific projects

- Processes are tailored for special projects
- Development team must must identify suitable life cycle model and must adhere to it which helps in the development of software in systematic and disciplined manner.

- When program developed by single programmer, he has the freedom to decide his exact steps.

- Life cycle model defines entry and exit criteria for every phase. Phase is considered complete only when exit criteria are satisfied.

Page No:
Date:

Agile Methodology

It is a practice that promotes continuous iteration of development and testing throughout the software development lifecycle of the project. Both development and testing activities are concurrent unlike the waterfall mode.

- Agile software development emphasizes on four core values:
  (i) Individual and team interactions over processes and tools
  (ii) Working software over comprehensive documentation
  (iii) Customer collaboration over contract negotiation
  (iv) Responding to change over following a plan

- Five reasons
  (i) Early time to market need
  (ii) Ever changing requirement - difficult to adopt
  (iii) Basic systems in place (than waterfall)
  (iv) Need of ~~customer~~ continuous interaction with customer
  (v) Waterfall was heavy on documentation

Agile Testing Methodology

- Scrum
- Crystal Methodologies
- DSDM (Dynamic Software Development Method)
- Feature driven development (FDD)
- Learn Software Development
- Extreme programming (XP)

Page No:
Date:

Agile Model	Waterfall Model
• It proposes incremental and iterative approach to s/w design.	• Development of s/w flows sequentially from start point to end point.
• Customer has early and frequent opportunities to look at the product and make decision and changes to the project.	• The customer can only see the product at the end of the project.
• Agile model is considered unstructured compared to the waterfall model.	• They are more secure because they are so plan oriented.
• Development process is iterative, and the process is executed in short weeks iterations.	• The development process is phased, and the phase is much bigger than iteration
• Testers and developers work together.	• Testers work separately from developers

Scrum Methodology

- Scrum is a framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value.
- The scrum framework consists of scrum teams and their associated roles, events, artifacts and rules. All events are time boxed events such that every event has a maximum duration.

Sprint

The heart of scrum is sprint, a time box of two weeks or one month during which a potentially releasable product

increment is created. A new sprint starts immediately after the conclusion of the previous sprint.

Sprints consists of the -
(i) Sprint planning
(ii) Daily scrum meeting
(iii) The development work
(iv) The sprint review
(v) The sprint retrospective

- **Roles in Scrum**
  (i) Scrum Master : Master is responsible for setting up the team, sprint meeting and removes obstacles to progress

  (ii) Product Owners : The product owner creates product backlog, prioritizes the backlog and is responsible for the delivery of the functionality at each iteration.

  (iii) Scrum Team : Team manages its own work and organizes the work to complete the sprint or cycle.

- **Artifacts in scrum**
  (i) Product backlog
  (ii) Sprint backlog
  (iii) Sprint/release burn down chart
  (iv) Increment

Page No:
Date:

eXtreme Programming (XP)

eXtreme Programming is based on the five values

- Communication
- Simplicity
- Feedback
- Courage
- Respect

Extreme Programming is a systematic approach with a set of values, rules and practices for rapidly developing high quality software that provides the highest value for customers.

It is very useful when there is constantly changing demands or requirements from the customers or when they are not sure about the functionality of the system.

It advocates frequent releases of the product in short development cycles which inherently improves productivity of the system and also introduces a checkpoint where any customer requirements can be easily implemented

The XP develops software keeping customer in the target.

- **Basic principles of extreme programming**

- Business requirements are gathered in terms of stories. User stories are simple and informal statements of the customer about the functionalities needed. All these stories are stored in a place called Parking Lot.

- Now based on user stories, project team proposes metaphors. Metaphors are a common vision of how the system would work (design)

- The development team may decide to build a spike (like prototype) for some features.

- In this type of methodology, releases are based on the shorter cycles called iterations with span of 14 days time period. Each iteration includes phases like coding, unit testing and system testing where at each phase some minor or major functionality will be built in the application.

- Activities

(i) Coding – XP argues that code is a crucial part of any development system process since without code it is not possible to have a working system.

(ii) Testing – XP places high importance on testing and considers it be the primary means for developing a fault free software

(iii) Listening – The developers need to carefully listen to the customers if they have to develop a good quality software. Programmers may not need to have an in-depth knowledge of the domain of the system under development.

(iv) Designing – Without a proper design, the system implementation becomes very complex and the dependencies within the system becomes more numerous and becomes difficult to comprehend the solution.

(v) Feedback – It recognises the importance of user feedback in understanding the exact customer requirements. The time that elapses b/w the development of a version and collection of feedback on it is critical to learning and making changes

(vi) Simplicity – Attention should be focussed on specific features that are immediately needed and making them work rather than devoting time and energy on speculations about future requirements.

Classical Waterfall Model

Classical Waterfall Model divides lifecycle into phases –
- Feasibility Study
- Requirement analysis and specification
- Design
- Coding and unit testing
- Integration and system testing
- Maintainance

* Phases b/w feasibility study and testing requires the maximum efforts and is known as the development phase.
* Maintainance phase requires max. efforts among all life cycle phases
* Among development phases, testing phase requires or consumes max. efforts.

- Most organisations usually define
  - Standards on the outputs (deliverables) produced at the output of every phase
  - Entry and exit criteria for every phase

- Feasibility Study – The main idea of feasibility study is to determine the product is financially viable and technically feasible. First gather customers requirements by analysing input and output data. Understand the necessary processes and system constraints. Then explore possible solutions evaluating each based on required resources, development cost and time.

- Requirement analysis and specification: The aim of this phase is to understand and document the customers exact requirements through two activities
  - Requirement gathering and analysis
  - Requirement specification.

- Goals of requirement analysis
  - Collect data from the customers
  - Analyse the data to clarify needs
  - Identify and resolve inconsistencies or incomplete requirements

- Requirements gathering methods
  - One-on-one interviews : With users, management and stakeholders
  - Group interview : Highlights agreements and differing issues
  - Questionnaires / Surveys - Used for graphically dispersed stakeholders
  - User Observations - Observe user interaction at different times
  - Document analysis - Review current systems and processes.

Initial data may contain contradictions & ambiguities which are resolved through discussions. The finalized requirements are documented in Software Requirement Specifications (SRS). Analysts handle this phase.

- Design

It transforms the requirement specification into a form suitable for implementation. It involves deriving the software architecture from the SRS document using two approaches:
Traditional and Object Oriented

Traditional / Procedural design approaches - Based on data flow modelling. It consists of two parts:
- Structured Analysis
  - Identify functions and data flow
  - Decompose functions to sub functions
  - Use data flow diagrams (DFDs)
- Structured design
  - Decompose the ~~modules~~ system into modules and represent model interactions
  - Design algorithms, data structures and objects, refining relationships between or among objects.

Object Oriented Design - Various problems that occur in the problem domain and the solution domain are first identified and the different relationships that exist among them are identified. The object structure is further refined to have a detailed approach.

Advantages
- Lower development effort and time
- Better maintainability

- Implementation Translates software design to source code

During implementation phase,
- each module of design is coded
- each module is unit tested
  - tested independently as stand alone unit and debugged
  - each module is documented

Purpose - Test if individual modules work correctly

- Integration and system testing
  - Integrate modules incrementally, testing the system after each step
  - System testing ensures that the system meets requirements from the SRS

- Maintainance

Requires more effort than development (typically 40:60 ratio)

- Corrective - Fix errors missed during development
- Perfective - Improve and enhance system functionality
- Adaptive - Port software to new requirements

Iterative Waterfall Model

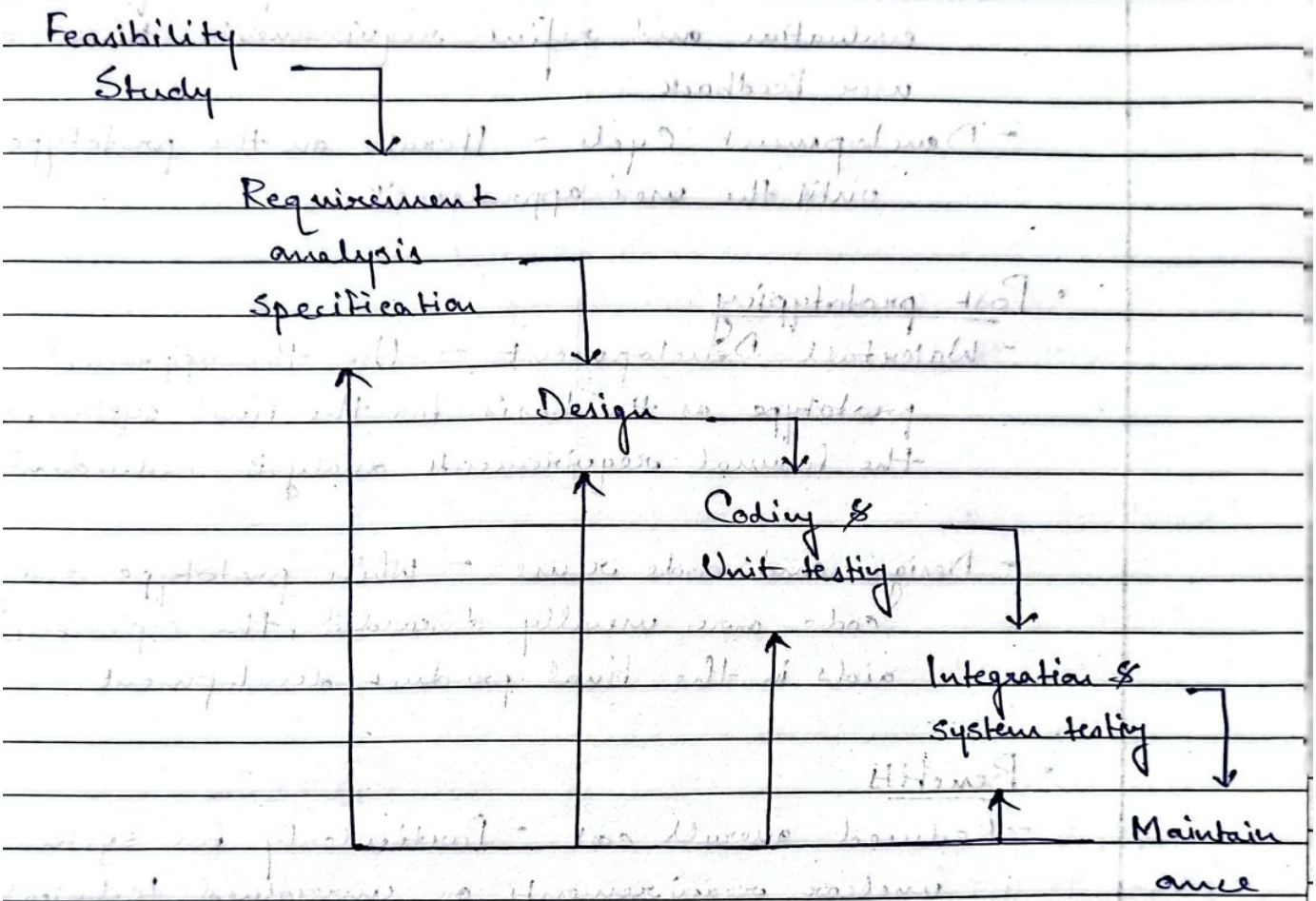
The classical waterfall model is idealistic, assuming no defects are introduced during any phase. In practice, defects often arise and detected later in the life cycle, necessitating rework of previous phases.

(i) Defect Detection - Defects often go unnoticed until later phase

(ii) Feedback paths - To address defects, feedback paths are necessary to visit and revise earlier phases.

(iii) Error detection - Errors should ideally be detected in the same phase they are introduced to simplify corrections.

(iv) Phase containment of errors - This principle involves detecting and addressing errors as close to their introduction point as possible to minimize rework



Prototyping Model

Before actual development, it is essential to build a working prototype of the system. A prototype is a simplified, often non functional version of the system with limited capabilities

- Reasons for prototyping
  - Demonstrate input data formats, messages, reports or interactive dialogs to the customers
  - Address technical issues and major design decisions such as hardware response times or algorithm efficiency
  - The prototype helps refine requirements and improve design.

- Prototyping process
  - Start with approximate requirements - Create a quick design using shortcuts and dummy functions
  - Customer feedback - Submit the prototype for evaluation and refine requirements based on user feedback
  - Development Cycle - Iterate on the prototype until the user approves it.

- Post prototyping
  - Waterfall Development - Use the approved prototype as the basis for the final system, rendering the formal requirements analysis redundant
  - Design and code reuse - While prototype design and code are usually discarded, the experience aids in the final product development

- Benefits
  - Reduced overall cost - Particularly for systems with unclear requirements or unresolved technical issues, prototyping can prevent costly redesigns and change requests later in the development process.

Evolutionary Model

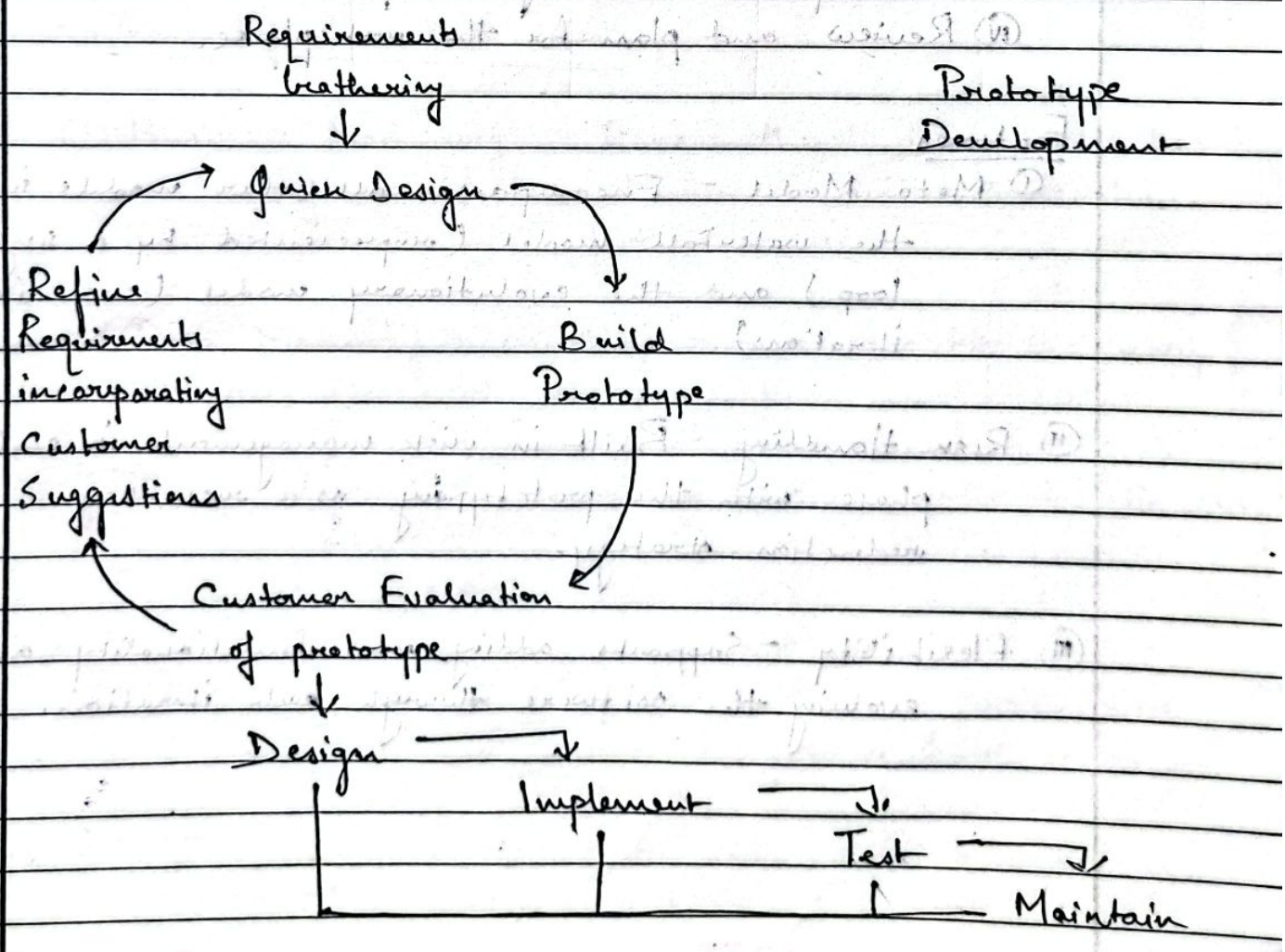
The evolutionary model allows users to experiment with a partially developed system early on, helping refine user requirements before the full system is built.

- Key features
  - The system is divided into modules that are incrementally implemented and delivered.

- Core functionalities are developed and tested thoroughly reducing errors in the final product.
- Each new version adds new functionalities or enhances existing ones, allowing the system to gradually evolve into the final product.

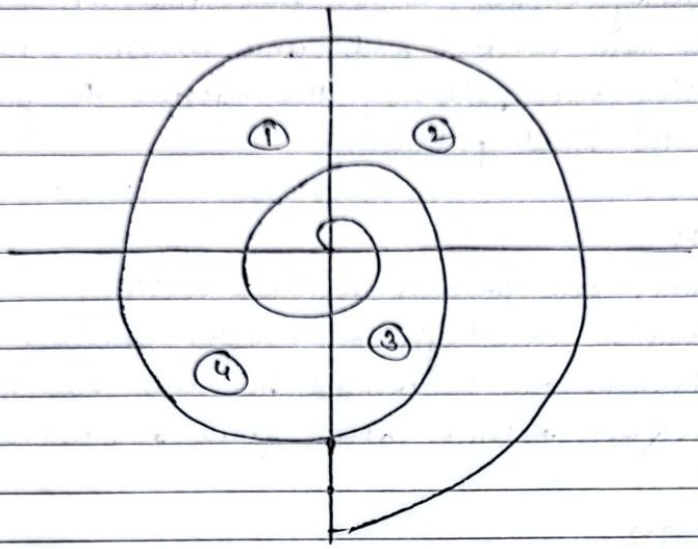
- **Benefits**
  - Users can provide feedback earlier
  - Training can begin on earlier versions
  - Markets can be created for new functionalities
  - Frequent releases allow quick resolution of issues.

- **Challenges**
  - Difficult to divide complex problems into functional units for incremental delivery.
  - Best suited for large projects where models can be developed incrementally



Page No:
Date:

Spiral Model



(I) Determine objectives and identify alternative solutions
(II) Identify and resolve risks
(III) Develop next level of the product
(IV) Review and plan for the next phase.

Features

(I) Meta Model - Encompasses all other models including the waterfall model (represented by a single loop) and the evolutionary model (succeeding iterations)

(II) Risk Handling - Built in risk management in each phase with the prototyping as a risk reduction strategy.

(III) Flexibility - Supports adding new functionality and evolving the software through each iteration.

Page No:
Date:

- Advantages
  - Enhanced risk management
  - Suitable for large, mission-critical projects
  - Allows early software delivery
  - Strong documentation and approval control

- Disadvantages
  - Can be costly
  - Requires specialized expertise on risk analysis
  - Success depends heavily on risk assessment
  - Not ideal for smaller projects.

Rapid Application Development (RAD)

It is an incremental model where components or functions are developed in parallel like mini-projects. These developments are time boxed, delivered and then assembled into a working prototype, allowing early customer feedback.

- Phases of the RAD Model

(i) Business Modelling - Design the business model based on information from business activities to capture a complete view of the process.

(ii) Data Modelling - Identify and analyse the necessary data based on the business model

(iii) Process Modelling - Plan the processing and functionality or modification of data ~~based on business model~~ to achieve business functionality

(iv) Application Modelling - Develop the application with coding and implement using automation tools

(v) Testing and turnover - Perform testing activities on the developed applicatio

- Advantages
  - Fast development and delivery
  - Minimal testing needed
  - Progress is easy and easy to track
  - Cost effective and resource efficient

- Disadvantages
  - Required highly skilled resources
  - Constant client feedback is necessary
  - Automated code generation can be expensive
  - Hard to manage and not ideal for large-long term projects.

Agile Model

It combines iterative and incremental processes with a focus on adaptability and customer satisfaction through rapid delivery of working software. Agile breaks the product into small incremental builds, delivered in iterations typically lasting 1-3 weeks. Each iteration involves teams working on ~~teams~~ planning, requirements, design, coding, testing and showcasing the product to stakeholders

- Advantages
  - Realistic approach to development
  - Promotes teamwork and cross training
  - Requires minimal ~~training~~ resources and delivers early working solutions.

- Disadvantages
  - Not ideal for complex dependencies
  - Requires overall plan, agile leadership & PM practices
  - Strict delivery management required to meet deadlines.

Page No:
Date:

Requirement Engineering

- Requirement Gathering – Also popularly known as the requirement elicitation. The primary objective of the requirements gathering task is to collect the requirements from the stakeholders

Methods:
(i) Studying existing documentations
(ii) Interviews
(iii) Task, scenario

- Requirement Analysis – The main purpose of the requirement analysis activity is to analyse the gathered requirements to remove all ambiguities, incompleteness, and inconsistencies from the gathered customer requirements and to obtain a clear understanding of the software to be developed.

- During requirement analysis, the analyst needs to identify and resolve three main types of problems in the requirements:
  - Anomaly – Anomaly is an ambiguity in the requirement. When a requirement is anomalous, several interpretations of that requirement are posible. Any anomaly in any of the requirements can lead to the development of an incorrect system.
  - Inconsistency – Two requirements are said to be inconsistent, if one of the requirements contradicts the other.
  - Incompleteness – An incomplete set of requirements is the one in which some requirements have been overlooked

Page No:
Date:

Software Requirement Specification (SRS)

It is a formal document capturing all user requirements and serves multiple audiences like developers, testers and project managers.

- Key characteristics of a good SRS
  - (i) Concise, unambiguous and complete
  - (ii) Implementation - independent
  - (iii) Traceable - Requirements linked to design elements and vice versa
  - (iv) Modifiable and well structured
  - (v) Verifiable - All requirements must be tested
  - (vi) Identification of response to undesired events - Describe ~~events~~ system behaviour under conditions

- Important Components of an SRS Document
  - (1) Functional requirements
    Clearly describes each function that the system will perform along with input/output data.
  - (2) Non functional requirements
    - External interfaces
    - Performance, security, maintainability, portability and usability
    - Design and implementation constraints
  - (3) Goals of implementation
    General future proofing ideas like easy support for new device or reusability considerations
  - (4) Design and implementation constraints
    Specific limitations like regulatory policies, technology stack, hardware interfaces etc.

- Functional requirements for organisation of SRS document

This section can classify the functionalities either based on the specific functionalities invoked by different users, or the functionalities that are available in different modes etc, what may be appropriate.

1. User class 1
   (a) Functional requirement 1.1
   (b) Functional requirement 1.2
2. User class 2
   (a) Functional requirement 2.1
   (b) Functional requirement 2.2

(Ex) Withdraw cash from ATM

R1 : Withdraw Cash

Description: The withdraw cash function first determines the type of account that the user has and the account number from which the user wishes to withdraw cash. It checks the balance to determine whether the requested amount is available in the account. If enough balance is available, it outputs the required cash, otherwise it generates an error message.

R.1.1 : Select withdraw amount option
Input : "Withdraw amount" option selected
Output: User prompted to enter account type.

R.1.2 : Select account type
Input : User selects options from any one of the following
- savings / checking / deposit
Output : Prompt to enter amount.

R.1.3 : Enter required amount
Input : Amount to be withdrawn in integer values greater than 100 and less than 10000 in multiples of 100
Output : The requested cash and printed transaction statement.

• External Interface Requirements

(i) User Interface - Standards for GUI, navigation, error handling etc.

(ii) Hardware Interface - Systems interaction with the hardware components

(iii) Software Interfaces - Interaction with databases, OS and other applications

(iv) Communication Interfaces - Protocols, network server interactions etc.

• Other Non functional requirements

(i) Performance - Ex: Transactions per second

(ii) Safety - Avoid loss/damage due to software errors

(iii) Security - Identity authentication, data privacy etc.

Techniques for analysing complex logic

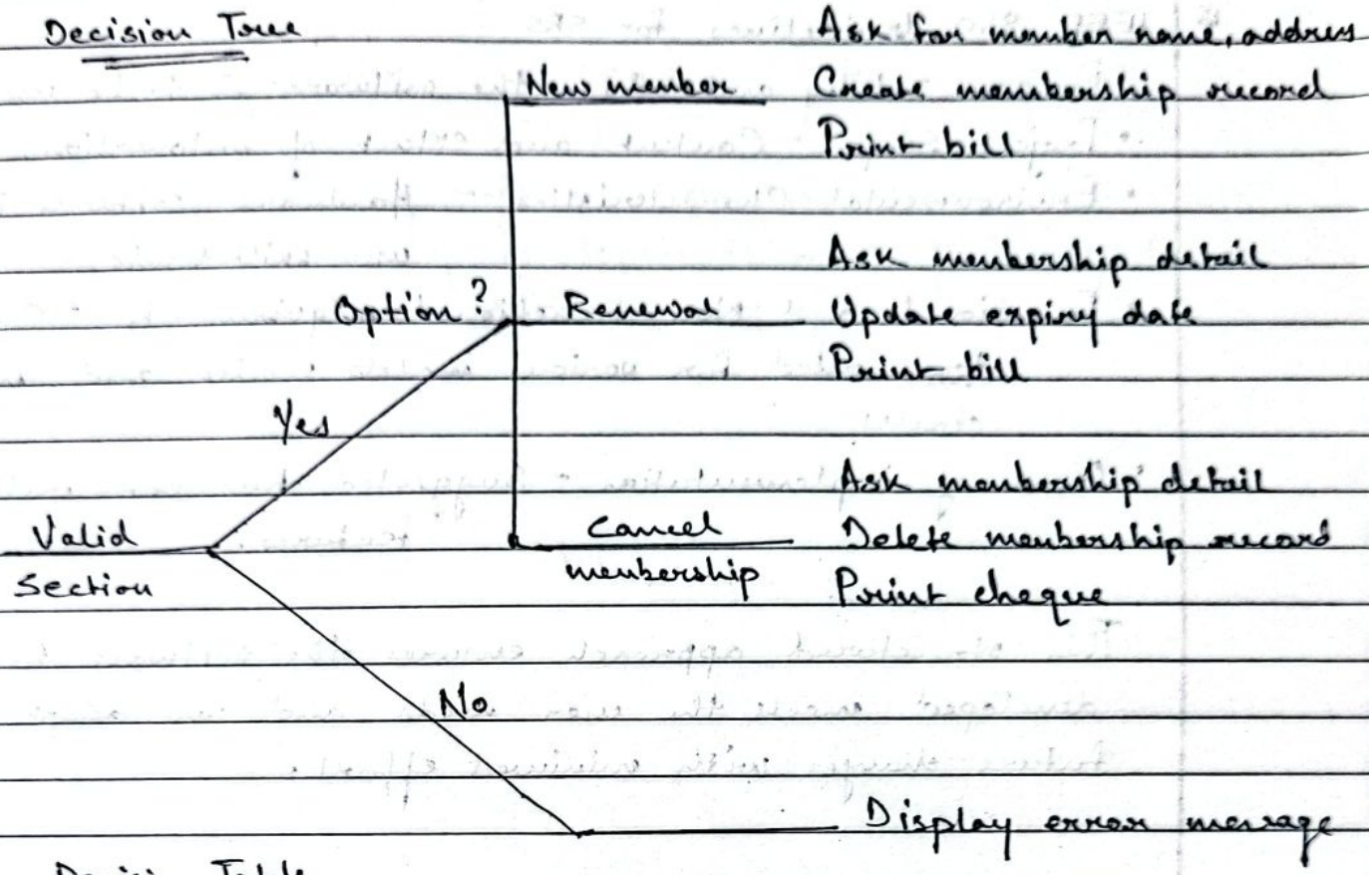
(i) Decision Tree

A visual flowchart for decision making logic and corresponding actions. Decision tables specify which variables are to be tested, and based on this what actions are to be taken depending upon the outcome of the decision making logic and the order in which decision making is performed.

Library Management System (LMS)

Page No:
Date:

Decision Tree



Decision Table

Conditions				
Valid Selection	NO	YES	YES	YES
New member	-	YES	NO	NO
Renewal	-	NO	YES	NO
Cancellation	-	NO	NO	YES
Actions				
Display error message	X			
Ask members name etc		X		
Build customer record		X		
Generate bill		X	X	
Ask membership detail			X	X
Update expiry date			X	
Print cheque				X
Delete record				X

Page No:
Date:

IEEE 830 Guidelines for SRS

- Purpose - Why and where the software is to be used.
- Project Scope: - Content and extent of automation.
- Environmental Characteristics - Hardware, software interaction user skill levels
- Functional and Non functional requirements - Clearly documented for various ~~models~~ modes and user classes
- Goals of implementation - Suggested but not critical features.

This structured approach ensures the software being developed meets the user needs and can adapt to future changes with minimal effort.

Page No:
Date:

Software Project Management

The main goal of software project management is to enable a group of developers to work efficiently / effectively towards the completion of a successful project.

Complexities

① Invisibility - Progress is hard to gauge until the software is operational.

② Changability - Software is easily modified, leading to the frequent changes due to business needs or user demands

③ Complexity - Even small software projects involve numerous interconnected parts (functions, state transitions, dependencies)

④ Uniqueness - Each software project have different challenges making it difficult to apply one-size-fits-all solutions.

⑤ Exactness of solutions - Software requires exact matches in parameters making reuse across projects difficult.

⑥ Team oriented and intellect intensive work - Software development is collaborative and requires high levels of intellectual effort, adding to its complexity

Responsibilities of a software project manager:

① Project Planning
- Estimating size, cost, effort and duration
- Scheduling resources and manpower
- Staffing and organising the project team.

- Risk management (identifying, analysing & mitigating risks)
- Quality assurance and config. management

(2) Project management and control
- Ensuring that project stays on track and is completed as planned.
- Managing risks and changes throughout project lifecycle.

Skills required for project managers

(i) Knowledge of project management techniques
(ii) Strong decision making abilities
(iii) Previous experience managing similar projects
(iv) Communication, team building and customer interaction skills
(v) Cost estimation, risk management and configuration management expertise

Project Planning Techniques

(i) Size estimation - Estimating the size of the software is critical as it directly affects cost, duration and effort calculations.

(ii) Scheduling - Once size, cost and effort are estimated, the schedule for manpower and other resource is created.

(iii) Risk Management - Identifying and planning for potential risks to avoid project delay.

(iv) Miscellaneous plans - Quality assurance, validation, verification and configuration management planning

Page No:
Date:

Sliding window planning

In large projects, its challenging to make accurate plans from the start, so sliding window planning involves planning the project in stages, allowing for flexibility and adjustment as the project progresses.

Software Project Management Plan (SPMP) Document

① Introduction - Objectives, functions, performance issue, management & technical constraints

② Project Estimates - Historical data used
- Estimation techniques used
- Effort, resource, cost and project duration estimates

③ Schedule - Work breakdown structure
- Task Network Representation
- Gantt Chart Representation
- PERT Chart Representation

④ Project Resources - People
- Hardware and Software
- Special resources

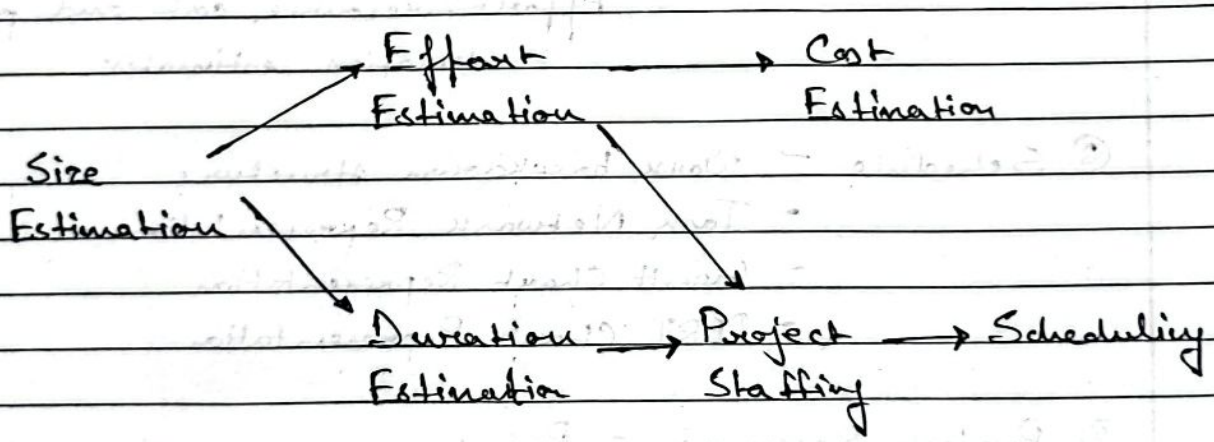
⑤ Staff Organisation - Team Structure
- Management Reporting

⑥ Risk Management Plan - Risk Analysis
- Risk Identification
- Risk Estimation
- Risk Abatement Procedures

⑦ Project tracking and control plan - Metrics to be tracked
- Tracking plan
- Control plan

⑧ Miscellaneous plans - Process tailoring
- Quality assurance plan
- Config. management plan
- Validation & Verification
- System Testing Plan
- Delivery, Installation and the maintainance plan

■ Precedence Ordering among planning activities



■ Metrics - project size estimation

① LOC (Lines of code) - Measures the size of a software project by counting the source instruction in the developed program. It excludes comment and header lines.

Shortcomings -

- Focus on coding alone - LOC assumes that coding effort determines the entire software development effort, which neglects important aspects like designing and testing.

- Subjectivity in Coding Style - LOC counts can vary due to different coding styles leading to inconsistency.
- LOC metric penalises modern programming techniques like code reusability and higher level languages, as fewer lines of code are required to achieve the same functionality.
- LOC measures lexical complexities but fails to address logical and structural complexities

(2) Function Point (FP) It was proposed by Albrecht in 1983 as an alternative to LOC. It measures software size based on functionality, making it easier to compute from problem specification.

FP Computation

Step 1: Compute the Unadjusted Function Point (UFP) based on the five characteristics: Inputs, Outputs, Enquiries, Files and Interfaces

UFP = (Inputs * 4) + (Outputs * 5) + (Inquiries * 4) + (Files * 10) + (Interfaces * 10)

Step 2: Refine UFP based on complexity (simple, average, complex) of each parameter.

Step 3: Adjust UFP for project specific complexities using Technical Complexity Factor (TCF)
Compute TCF as TCF: 0.65 + 0.01 * DI (where DI is the sum of 14 influence factors)

Final FP = UFP * TCP

Page No:
Date:

Shortcomings of FP

- FP does not account for algorithmic complexity. only the number of functions leading to possible misjudgment of effort required for complex ~~structo~~ features.
- The feature point metric was developed to address this by incorporating algorithm complexity

Empirical Cost Estimation Techniques

① Expert Judgement

- Estimations is based on the experience of the expert who divides the projects into units and provides an estimation for each.
- It is prone to human errors and individual bias and may overlook important factors

② Delphi Estimation

- Involves a team of experts providing independent estimates which are refined over multiple iterations without direct discussions to avoid influence.
- A coordinator facilitates by summarizing the results and rationale after which experts reestimate.
- Delphi avoids direct confrontation and encourages unbiased collective jugement.

COCOMO (Constructive Cost Estimation Model)

COCOMO is a widely used software cost estimation model. It helps to estimate the effort and time required to develop software based on project size, expressed in Kilo Lines of code (KLOC).

• COCOMO Project Categories

① **Organic** – Small teams working on well understood applications (Ex. data processing programs)

$$\text{Effort} = 2.4 \times (\text{KLOC})^{1.05} \text{ PM}$$
$$\text{Time } (T_{dev}) = 2.5 \times (\text{Effort})^{0.38} \text{ month}$$

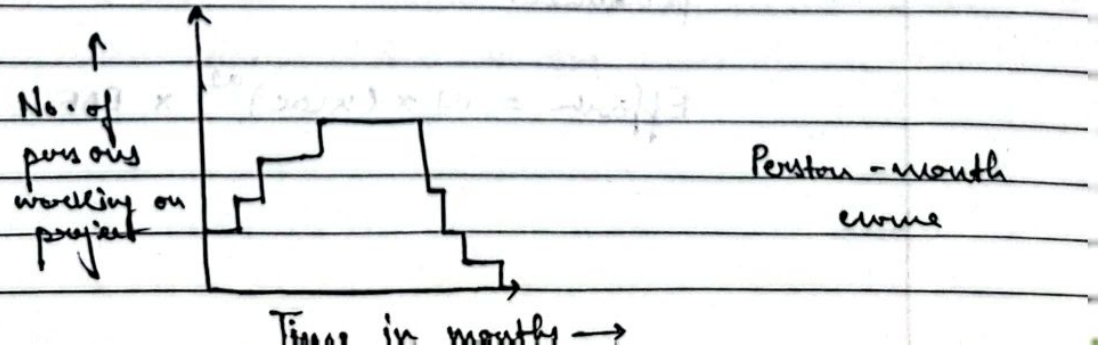
② **Semi detached** – Teams of mixed experience working on moderately familiar systems (Ex. Compiler, linker)

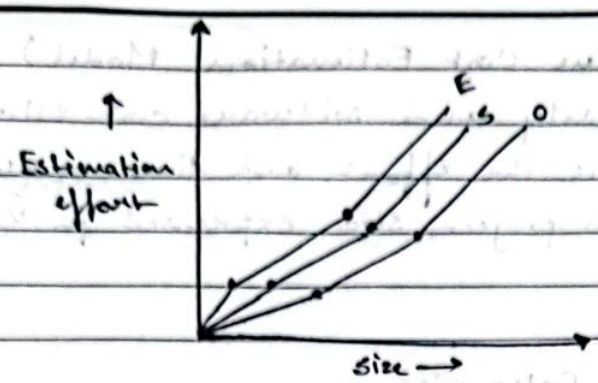
$$\text{Effort} = 3.0 \times (\text{KLOC})^{1.12} \text{ PM}$$
$$\text{Time } (T_{dev}) = 2.5 \times (\text{Effort})^{0.35} \text{ month}$$

③ **Embedded** – Complex, real time systems with tight hardware software integration (Ex. OS)

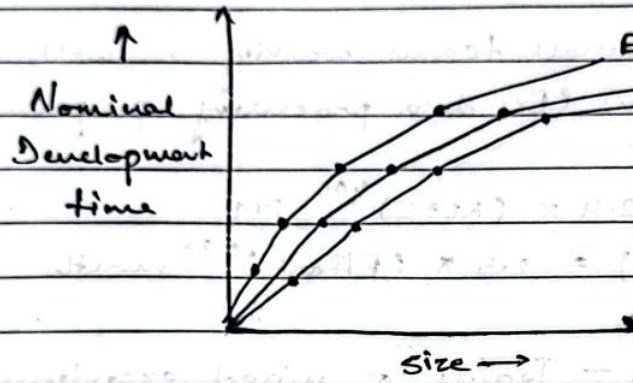
$$\text{Effort} = 3.6 \times (\text{KLOC})^{1.20} \text{ PM}$$
$$\text{Time } (T_{dev}) = 2.5 \times (\text{Effort})^{0.32} \text{ month}$$

• Person Month (PM) – One person month is the effort an individual can typically put in a month. The person month estimate implicitely takes into account the productivity





Effort v/s product size



Development time versus size

- Three stages of COCOMO

(i) Basic COCOMO – The single variable estimation model using project size to approximate effort and time.
- Predicts efforts (in person months) and time (in months) based on product size

$$\text{Effort} = a_1 \times (\text{KLOC})^{a2}$$
$$\text{Time} = b1 \times (\text{Effort})^{b2}$$

(ii) Intermediate COCOMO – Refines basic COCOMO by accounting for 15 cost drivers, such as reliability requirements and experience of personnel.

$$\text{Effort} = a1 \times (\text{KLOC})^{a2} \times \text{EAF}$$

(iii) Complete COCOMO – Extends the intermediate models by dividing the project into sub systems
- Each sub system is estimated separately (Ex - Organic, semi detached, embedded)
- Final cost is the sum of all sub systems cost.

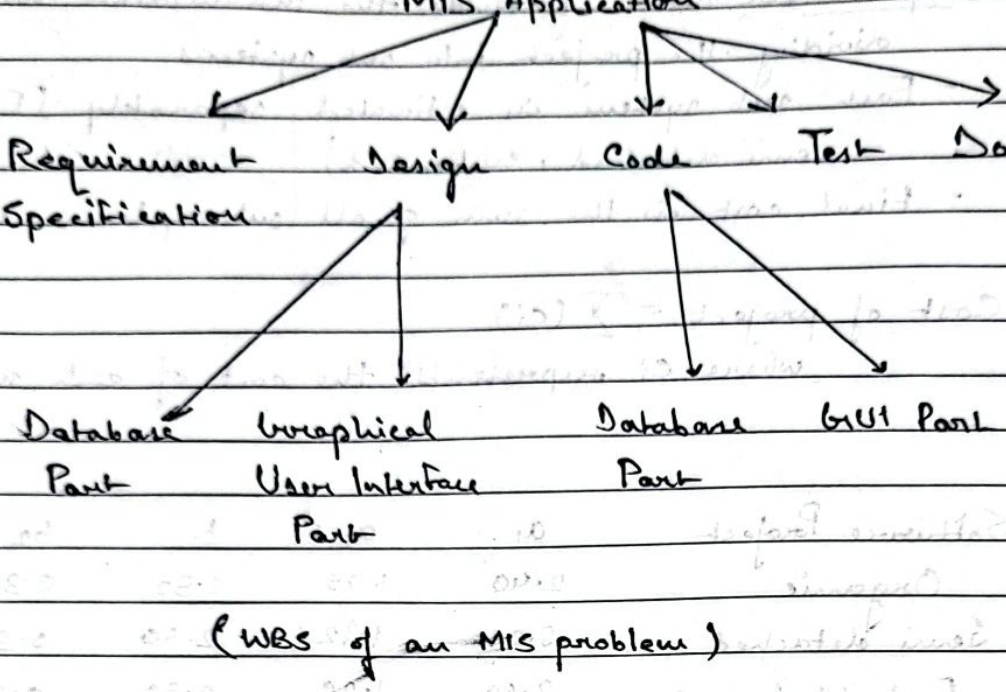
Cost of project = $\sum (c_i)$
where c_i represents the cost of each sub system.

Software Project	a_1	a_2	b_1	b_2
Organic	2.40	1.05	2.50	0.38
Semi detached	3.00	1.12	2.50	0.35
Embedded	3.60	1.20	2.50	0.32

Scheduling – The scheduling problem in essence, consists of deciding which tasks would be taken ~~by~~ up when and by whom.

A software manager must
(i) Identify major activities
(ii) Break activities into tasks
(iii) Determine task dependencies
(iv) Estimate time durations
(v) Represent tasks in activity network
(vi) Set tasks start/end dates
(vii) Identify the critical path (longest sequence of tasks)
(viii) Allocate resources to tasks

A work breakdown structure (WBS) decomposes activities into smaller tasks which are then scheduled. The task decomposition continues until the task duration is manageable (<2 weeks), hidden complexities are revealed or the scene of components is identified.



(WBS of an MIS problem)

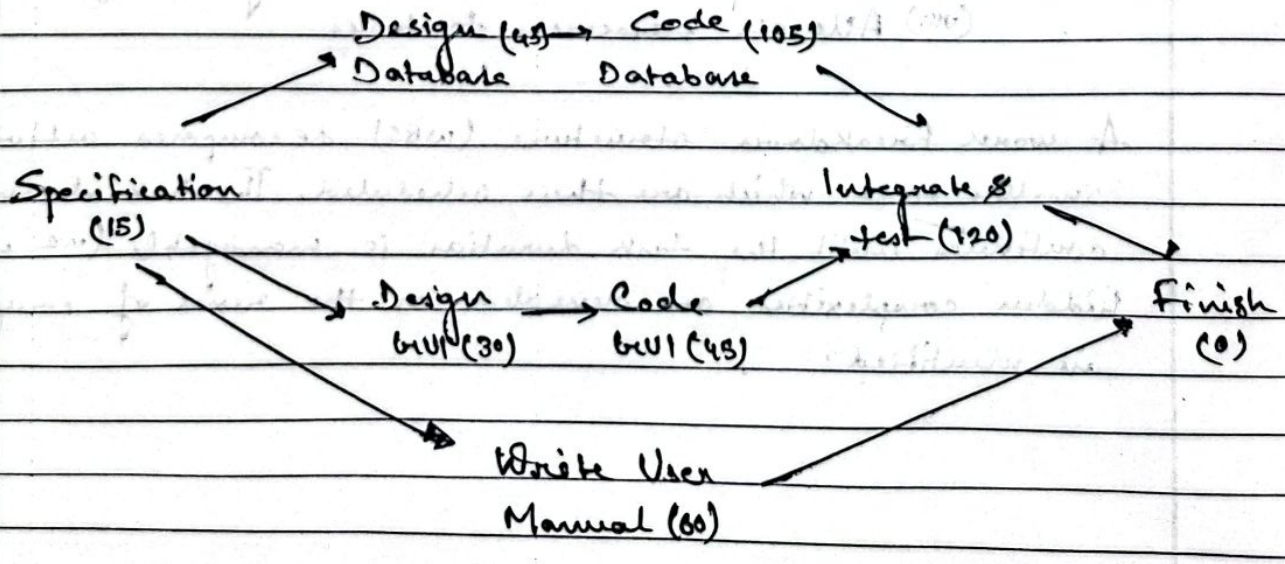
- **Activity Networks (AN)**

An activity network shows the different activities making up a project, their estimated durations and their interdependencies.

Two types of activity networks exist -

① Activity on Node (AoN) - Tasks on nodes
Dependencies on edges

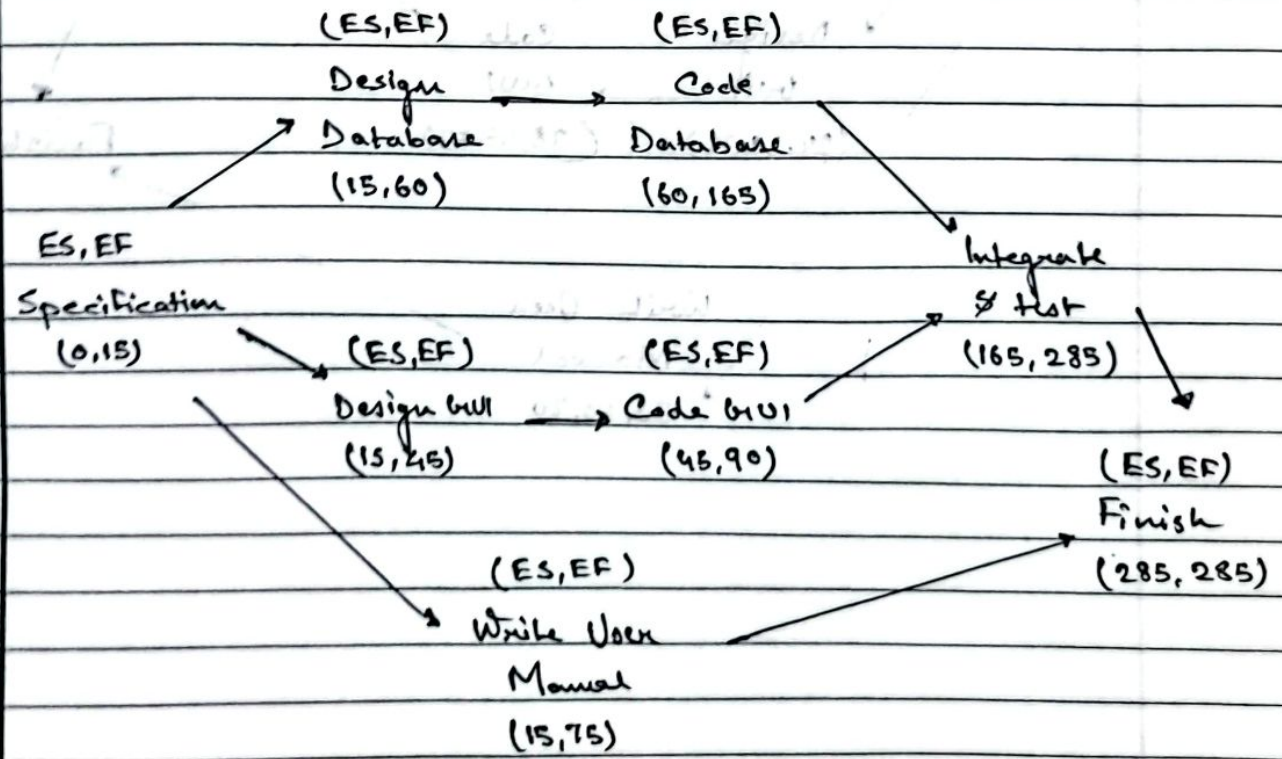
② Activity on Edge (AoE) - Task on edges
Nodes are project milestones



Page No:
Date:

Critical Path Method (CPM) It is an algorithmic approach to determine the critical paths and slack time for tasks not on the critical path involves calculating the following quantities:

- **Minimum time (MT)** - Min time to complete a project determined by the longest path in activity network.
- **Earliest Start Time (ES)** - The earliest the task can start, based on the completion of preceeding tasks
- **Latest start time (LS)** = MT - (max (path from this task to the end))
- **Earliest finish time (EF)** = ES + Duration of task
- **Latest finish time (LF)** = MT - Max. path duration from this task to the end
- **Slack time (ST)** = LS - ES or LF - EF



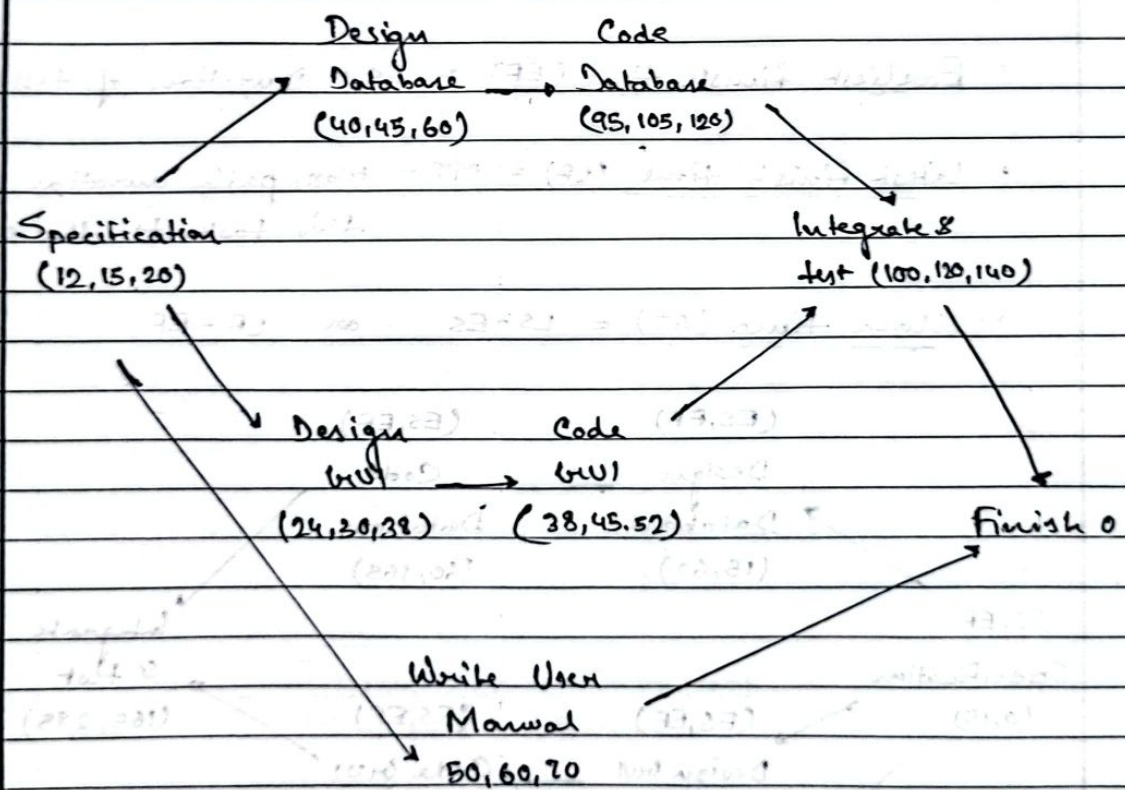
Page No:
Date:

PERT Chart - Project Evaluation and Review Technique
- consists of network of boxes and arrows

Boxes → Activities Arrow → Task dependencies

PERT chart represent statistical variations in the project estimates assuming normal distribution

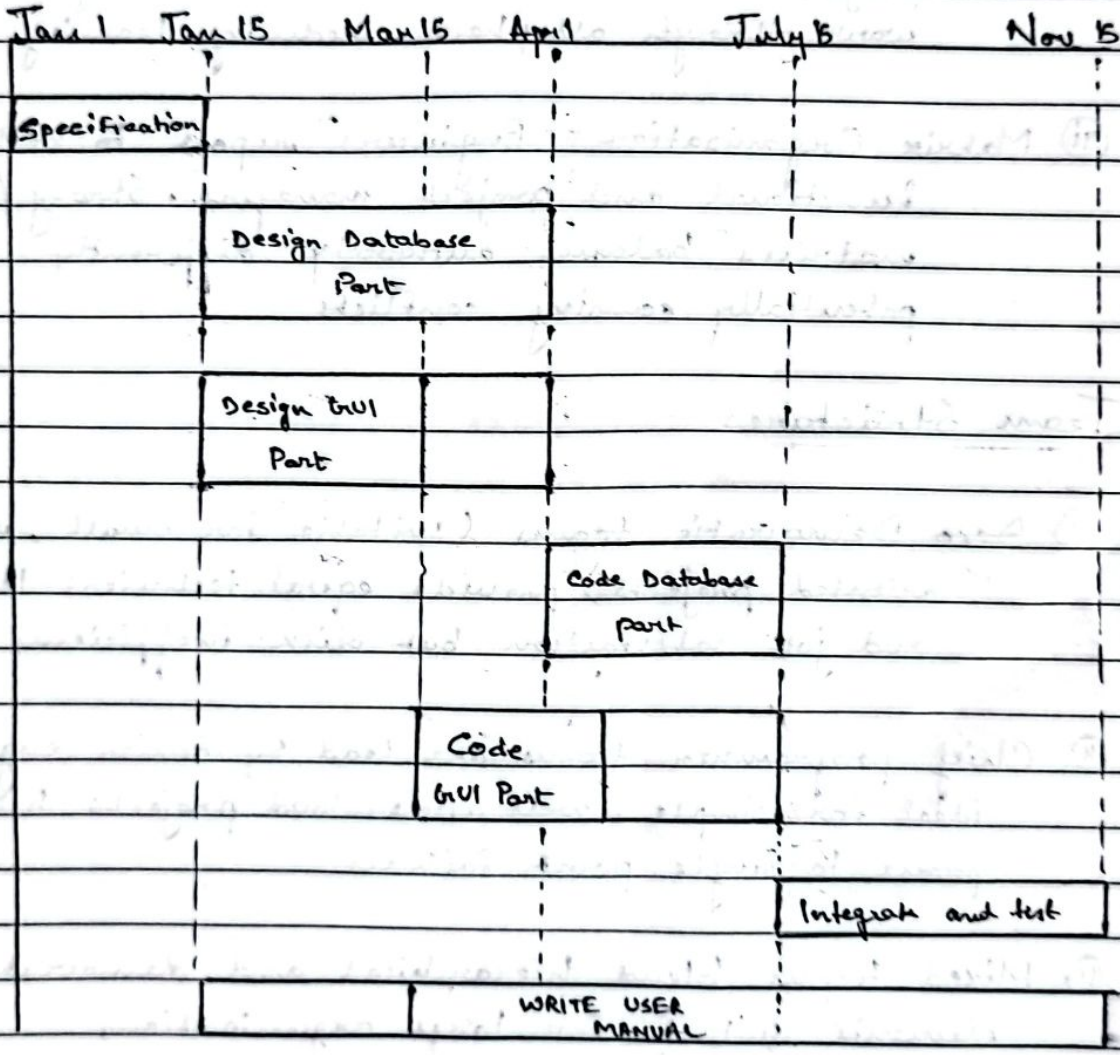
- Instead of making single estimate for each task,
  - (I) Pessimistic, worst case (W)
  - (II) Most likely estimates (M)
  - (III) Optimistic estimates (O) are made



Page No:
Date:

Gantt Chart

- Mainly used to allocate resources to activities
- The resources allocated to activities include staff, hardware and software, a special type of bar chart where each bar represents an activity.
- Bars are drawn along a time line. Each bar is an activity.
- Length of each bar is proportional to the duration of time planned for corresponding activity.



Page No:
Date:

Organisational Structures

(i) Functional Organisation - Engineers work in functional groups (Ex: Coding, testing) with projects borrowing engineers. Advantages include specialization ease of staffing and good documentation.

(ii) Project Organisation - Engineers remain with the project for its entire duration, allowing them to work through all phases, reducing learning time.

(iii) Matrix Organisation - Engineers report to both functional and project managers. Strong/weak matrices balance authority differently potentially causing conflicts

Team Structures

(i) ~~Dem~~ Democratic teams (suitable for small, research oriented projects) provide equal technical leadership and job satisfaction but risk inefficiency

(ii) Chief programmer teams are lead by senior engineer, ideal for simple, well understood projects but prone to single point failures

(iii) Mixed teams blend hierarchical and democratic elements suitable for large organisations

Page No:
Date:

Risk Management

① Project risks – Risks concern varies form of
- Budgetary
- Schedule
- Personnel
- Resource and
- Customer related problems

⑪ Technical risks – Risk concern
- Potential design
- Implementation
- Interfacing
- Testing and
- Maintainance problems

⑪ Business risks
- Building an excellent product that no body wants
- Losing budgetary or personnel commitments.

Risk Assessment

$$p = r * s$$

where p = priority
r = probability
s = severity of damage

Likelihood \ IMPACT	High	Low
High	High impact and likely to occur	Low impact and likely to occur
Low	High impact and unlikely to occur	Low impact and unlikely to occur

Page No:
Date:

Risk Containment

(i) Avoid the risk – discussing with the customer to change the requirements to reduce the scope of the work, giving incentives to the engineers to reduce the risk of manpower turnover etc.

(ii) Transfer the risk – getting the risky component developed by a third party, buying insurance cover etc.

(iii) Risk reduction – It involves planning ways to contain the damage due to a risk

Risk leverage

$$= \frac{\text{Risk exposure before reduction} - \text{Risk exposure after reduction}}{\text{Cost of reduction}}$$