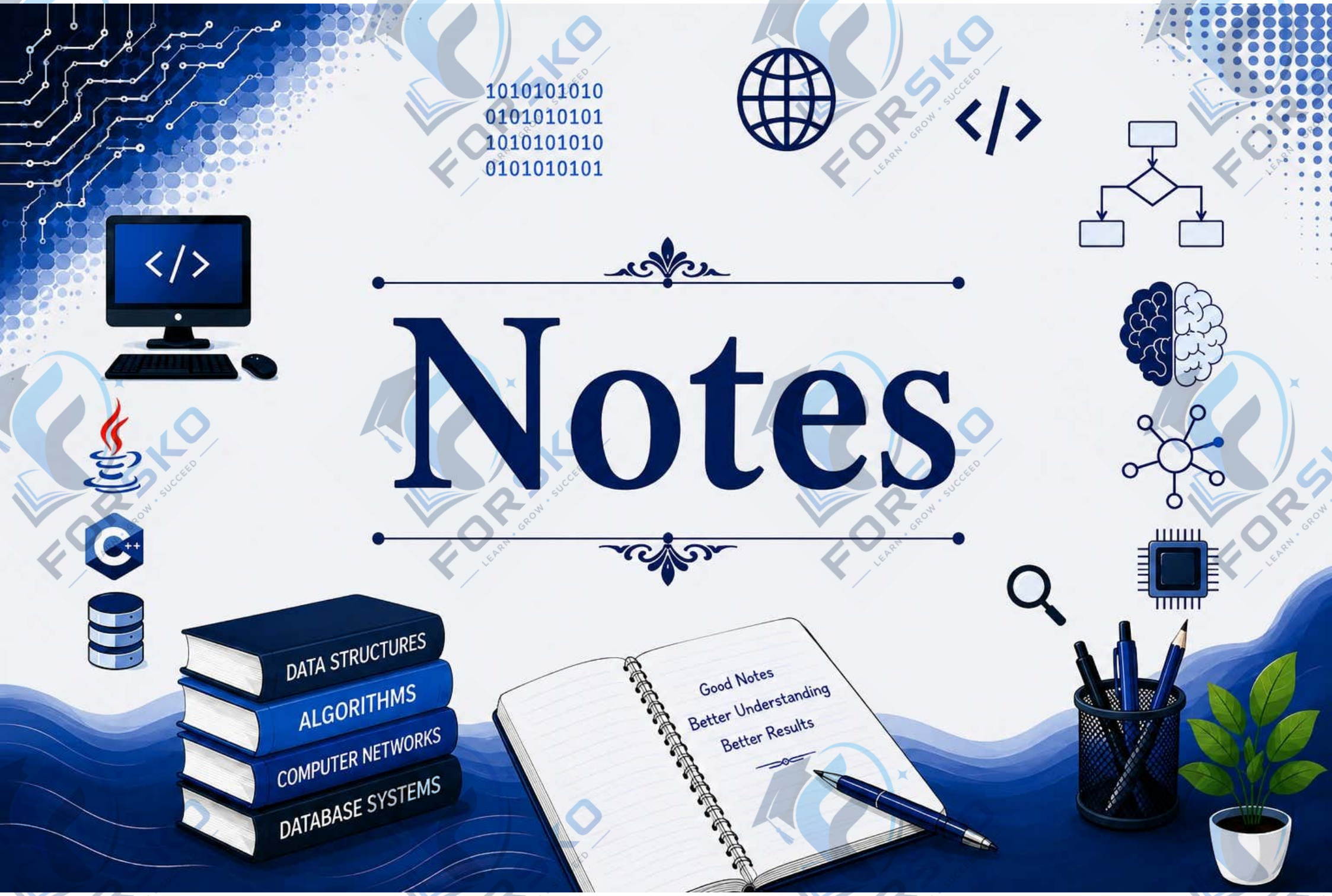

1010101010
0101010101
1010101010
0101010101
Notes
DATA STRUCTURES
ALGORITHMS
COMPUTER NETWORKS
DATABASE SYSTEMS
Good Notes
Better Understanding
Better Results


1010101010
0101010101
1010101010
0101010101
Unit-I
10110
01001
10101
DATA STRUCTURES
ALGORITHMS
COMPUTER NETWORKS
DATABASE SYSTEMS
#include <iostream>
using namespace std;
int main()
{
cout << "Hello, World!";
return 0;
}

Introduction to Software Engineering

1. Introduction:

Software Engineering is an engineering discipline that is concerned with all aspects of software production. It includes systematic approach to development, operation and maintenance of software.

2. Definition:

According to IEEE, "Software Engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software."

3. Characteristics of Software Engineering:

→ Systematic approach
→ Development of Quality software
→ Cost effective
→ Reliability and Efficiency
→ Maintainability
→ Uses engineering principles

4. Objectives:

→ To produce high quality software.
→ To satisfy user requirements.
→ To complete the project within time and budget.
→ To ensure maintainability and reliability.

5. Importance of Software Engineering:

→ Helps in managing large and complex projects.
→ Improves quality of software.
→ Reduces development time and cost.
→ Provides a structured approach.
→ Increases customer satisfaction.

6. Software Engineering Activities:

(i) Requirement Analysis
(ii) Design
(iii) Coding
(iv) Testing
(v) Deployment
(vi) Maintenance

These activities are performed in Software Development Life Cycle (SDLC).

Characteristics of Software

Software has some unique characteristics which distinguish it from hardware and other engineering products. The important characteristics of software are as follows :

1. **Intangible :** Software is not a physical entity. It is a collection of programs, data and documentation. We cannot touch or feel the software.
2. **Developed or Engineered, not Manufactured :** Software is developed by software engineers using programming and design skills. It is not manufactured in the classical sense.
3. **Does not Wear Out :** Software does not wear out like hardware. It does not get physically damaged. However, it may become obsolete due to changes in requirements.
4. **Custom Built :** Most of the software is custom built, i.e., it is developed to meet the specific requirements of a particular user or organization.
5. **Complexity :** Software systems are usually complex because they have to handle large amount of data, perform various operations and interact with many users and systems.
6. **Changeability :** Software must be flexible and changeable. It has to evolve with changing user needs, technology and environment.
7. **Invisibility :** Software is invisible. We cannot see the software while it is executing. We only see its effect or output.
8. **Easy Replication :** Once the software is developed, it can be copied and distributed easily at very low cost.
9. **Maintenance is Important :** A large part of the software life cycle is spent on maintenance. It includes correcting errors, adapting to changes and improving performance.

Types of Software

Software can be broadly classified into the following types :

1. **System Software** : System software is designed to manage and control the computer system hardware and to provide a platform for application software.
   Examples : Operating System, Language Translators (Compiler, Interpreter), Utility Programs, Device Drivers etc.
2. **Application Software** : Application software is designed to help end users to perform specific tasks and solve particular problems.
   Examples : MS Word, MS Excel, Accounting Packages, Payroll Software, Browser, Media Player, Mobile Apps etc.
3. **Programming Software** : Programming software is used by programmers to develop, test and maintain other software.
   Examples : Editors, IDEs (Integrated Development Environment), Debuggers, Compilers, Libraries etc.
4. **Embedded Software** : Embedded software is designed to perform specific tasks within embedded systems and devices.
   Examples : Software in Washing Machine, Microwave, Cars, ATM, Mobile Phones, Routers etc.

Need of Software Engineering

Software Engineering is needed because of the following reasons :

1. **Complexity of Software** : Modern software systems are large, complex and involve many components. Software Engineering provides systematic methods to handle this complexity.
2. **Increasing Size of Software** : Software size is growing rapidly. Traditional methods are not sufficient to develop and maintain such large software.
3. **Changing Requirements** : User requirements keep changing. Software Engineering helps in managing changes effectively through proper planning and documentation.
4. **Quality and Reliability** : It ensures high quality, reliable, secure and efficient software that meets user needs and performs correctly.
5. **On-time Delivery** : It helps in planning, estimating and monitoring the project so that software is delivered on time.
6. **Cost Effectiveness** : It reduces development and maintenance cost by using proper tools, techniques and standards.
7. **Maintainability** : It makes the software easier to maintain, test and update in the future.
8. **Better Management** : It provides good project management, team coordination and communication.
9. **Reusability** : It promotes code reusability which saves time and effort in future projects.
10. **Customer Satisfaction** : It ensures that the final software meets user requirements and gives better satisfaction.

Software Development Life Cycle (SDLC)

SDLC (Software Development Life Cycle) is a systematic process followed by software engineers to develop high quality software. It defines the various stages involved in building a software product from requirement analysis to maintenance.

Phases of SDLC

1. Requirement Analysis

- In this phase, the problem is studied in detail.
- Requirements are collected, analyzed and documented.

2. System Design

- The requirements are converted into a design.
- It includes database design, interface design, architecture design, etc.

3. Implementation (Coding)

- In this phase, the actual coding is done using suitable programming languages.
- The design is converted into a working software.

4. Testing

- The software is tested to find defects and errors.
- Testing ensures that the software works as per requirements.

5. Deployment

- After successful testing, the software is released to the user.
- It includes installation, data conversion, and user training.

6. Maintenance

- After deployment, the software is maintained.
- Errors are fixed, performance is improved and the software is updated according to changes in requirements.

Key Points :

- SDLC helps in producing a quality product.
- It saves time and cost.
- It improves efficiency and customer satisfaction.

Thus, SDLC is a continuous process and each phase is important for the successful development of a software product.

Phases of SDLC

The Software Development Life Cycle (SDLC) consists of the following phases which are carried out in a systematic manner to develop a quality software.

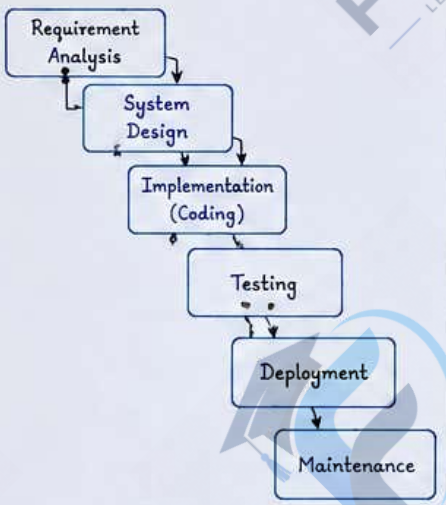
1. **Requirement Analysis** → In this phase, the problem is studied in detail. Requirements are collected from the user and documented.
2. **System Design** → In this phase, the requirements are converted into a design. It includes database design, interface design, architecture design, etc.
3. **Implementation (Coding)** → In this phase, the actual coding is done using suitable programming languages. The design is converted into a working software.
4. **Testing** → In this phase, the software is tested to find defects and errors. Testing ensures that the software works as per the requirements.
5. **Deployment** → In this phase, after successful testing, the software is released to the user. It includes installation, data conversion and user training.
6. **Maintenance** → In this phase, after deployment, the software is maintained. Errors are fixed, performance is improved and the software is updated as per change in requirements.

Software Development Models

A Software Development Model is an abstract representation of a software process. It provides a systematic approach to develop software.
The commonly used Software Development Models are :

1. Waterfall Model

- This is a linear sequential model.
- Each phase is completed before the next phase begins.

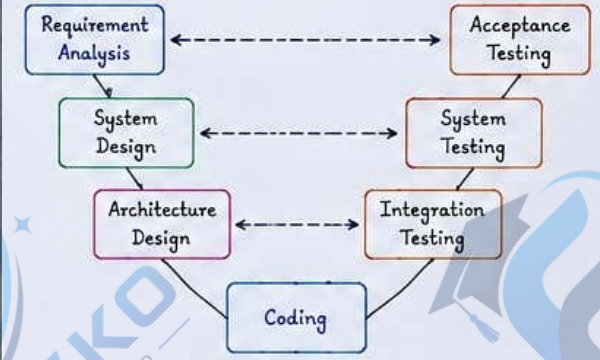


Advantages :
- Simple to understand
- Easy to manage

Disadvantages :
- Inflexible
- Difficult to accommodate changes

2. V-Model (Verification & Validation Model)

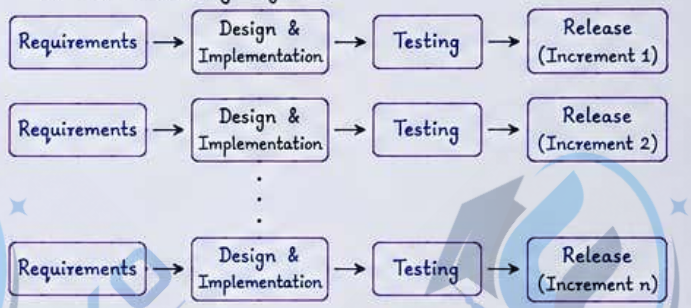
- It is an extension of Waterfall model.
- Each development phase has a corresponding testing phase.



Left side → Verification (Building the product right)
Right side → **Validation** (Building the right product)

3. Incremental Model

- The software is developed in small increments.
- Each increment adds some functionality to the existing system.

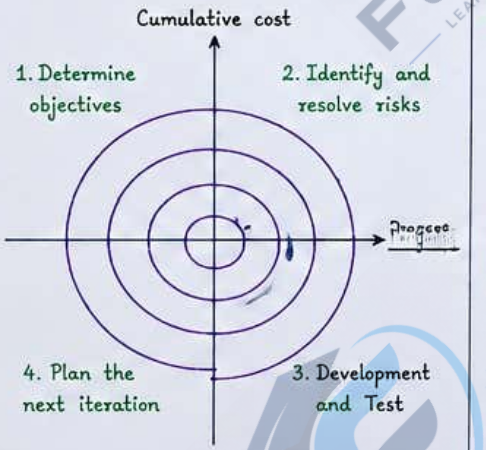


Advantages :
- Early delivery of working software
- Accommodates changes easily

Disadvantages :
- Requires more planning
- System architecture must be strong

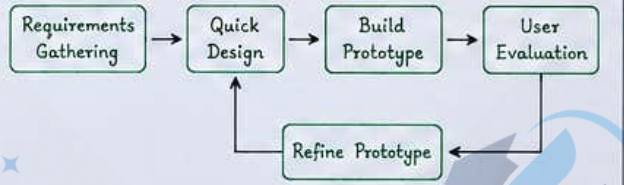
4. Spiral Model

- It is a risk-driven model that combines iterative development with systematic risk analysis.
- Best for large, complex and high-risk projects.



5. Prototype Model

- A prototype (working model) is built, tested and refined based on user feedback.
- Useful when requirements are not clear.

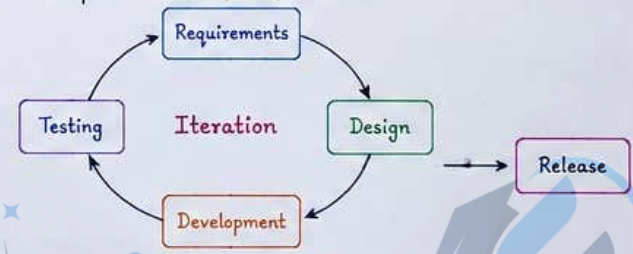


Advantages :
- Clarifies requirements
- User involvement

Disadvantages :
- May lead to scope creep
- Not suitable for large systems

6. Agile Model

- It is an iterative and incremental model.
- Focuses on customer collaboration, flexibility and quick delivery.
- Example : **Scrum, XP, Kanban**



Advantages :
- Flexible and adaptive
- High customer satisfaction

Disadvantages :
- Needs active customer involvement
- Not suitable for fixed budget projects

Introduction to Process Models

A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective. It shows the flow of activities and their relationships. Process models are used to plan, organize and control the software development process.

1. What is a Process Model?

A Process Model is a simplified representation of a software process. It describes the sequence of activities involved in developing a software system and the order in which these activities are performed.

2. Purpose / Need of Process Models

- Provides a framework for planning and managing projects.
- Helps in estimating cost, time and resources.
- Improves quality of software.
- Facilitates communication among team members.
- Helps in monitoring and controlling the development process.
- Serves as a guide for executing the project.

3. Characteristics of Good Process Model

- Simple and easy to understand.
- Flexible to accommodate changes.
- Represents activities in a logical sequence.
- Defines start and end points clearly.
- Facilitates risk management.
- Improves quality and productivity.

4. Elements of a Process Model

A process model consists of the following elements :

- **Activities** – The work tasks performed.
- **Actions** – The specific operations within an activity.
- **Milestones** – Important points or events in the process.
- **Deliverables** – The work products produced.
- **Workflow** – The sequence and flow of activities.
- **Roles** – The people or teams involved.
- **Entry / Exit Criteria** – Conditions to start or complete an activity.

5. Types of Process Models

Process models can be broadly classified as :

1. Prescriptive Process Models – Define a specific process.
2. Descriptive Process Models – Describe how a process is actually performed.
3. Empirical Process Models – Based on observation and measurement.

6. Importance of Process Models

Process models play a vital role in successful software development. They help in better planning, tracking progress, managing risks and ensuring that the final product meets quality standards and customer requirements.

Conclusion : Process models provide a structured approach to software development. They act as a blueprint for development teams and improve quality, efficiency and predictability of the software process.

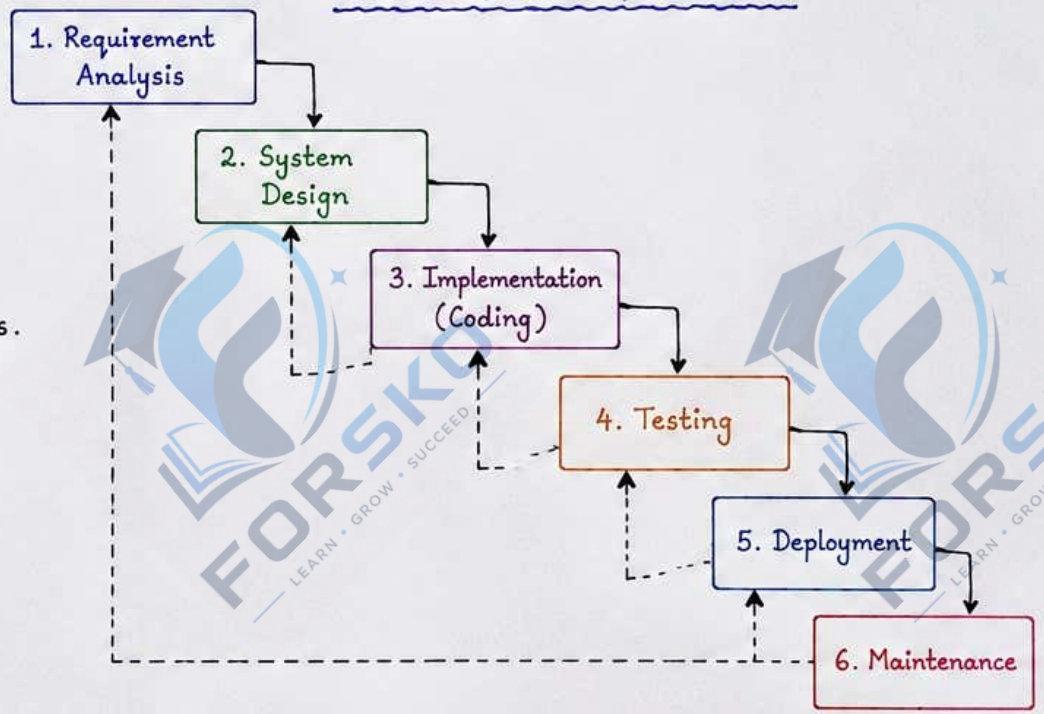
WATERFALL MODEL

Definition : The Waterfall Model is a linear sequential model used in software engineering. In this model, the development process is divided into distinct phases and the output of one phase becomes the input for the next phase. Each phase must be completed before moving to the next one. It is also called a Classic Life Cycle.

Key Points :

- It is simple and easy to understand.
- Phases are processed and completed one at a time.
- Suitable for small projects with well-defined requirements.
- Once a phase is completed, it is not easy to go back to the previous phase.

Phases of Waterfall Model



Explanation of Phases :

1. **Requirement Analysis :** Collect and analyze all requirements from the client.
2. **System Design :** Design the architecture, modules, database and interfaces.
3. **Implementation (Coding) :** Actual coding of the system according to the design.
4. **Testing :** Test the system to find errors and ensure it meets requirements.
5. **Deployment :** Deliver the software to the user and install it.
6. **Maintenance :** Modify and update the software after deployment to fix issues and improve performance.

Advantages :

- → Simple and easy to understand.
- → Well structured and easy to manage.
- → Each phase has specific deliverables.
- → Works well for small projects.
- → Easy to plan and control.

Suitable for :

Waterfall model is suitable for projects with clear and fixed requirements, low risk, and where changes are not expected frequently.

Disadvantages :

- → Inflexible to changes.
- → Working software is not produced until late in the life cycle.
- → Errors found in later phases are costly to fix.
- → Not suitable for large and complex projects.

PROTOTYPE MODEL

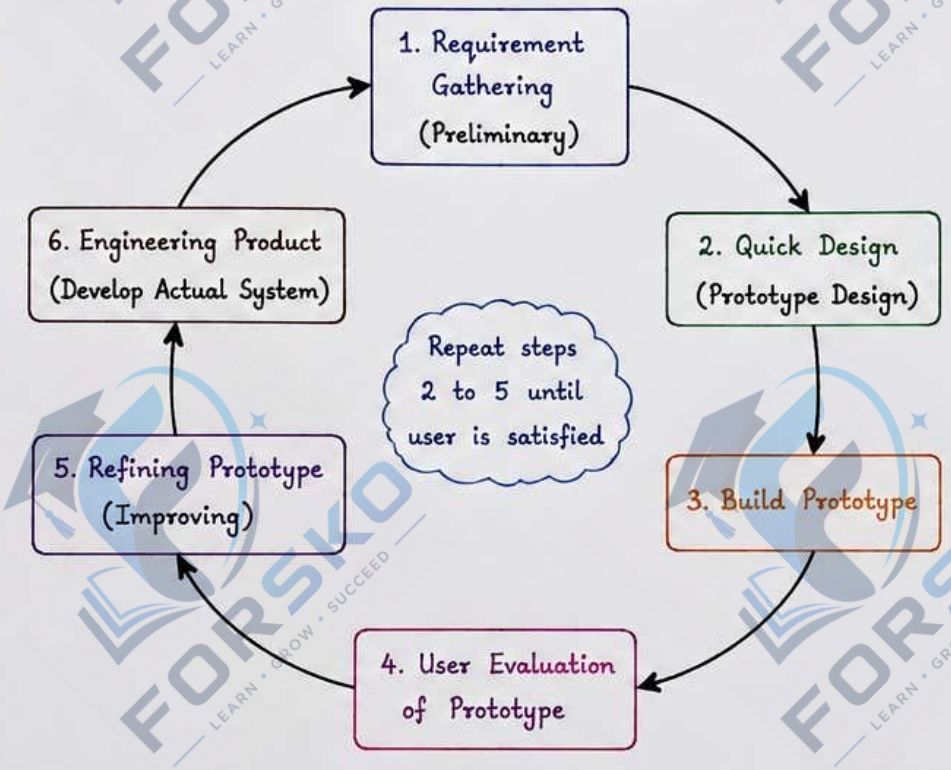
1. Introduction :

Prototype Model is an iterative model in which a working prototype of the system is developed, tested and refined as per the customer's requirements.

2. Key Points :

- It is also called "Evolutionary Model"
- Useful when requirements are not clear.
- User is involved throughout the development.
- A prototype is built, tested, refined and then used to develop the actual system.

3. Phases of Prototype Model :



4. Explanation of Phases :

1. **Requirement Gathering :** Collect preliminary requirements from the customer.
2. **Quick Design :** Create a quick design of the system.
3. **Build Prototype :** Develop a working prototype based on the quick design.
4. **User Evaluation of Prototype :** User tests the prototype and provides feedback.
5. **Refining Prototype :** Modify and improve the prototype as per user feedback.
6. **Engineering Product :** Once the prototype is accepted, the actual system is developed, tested and delivered.

5. Advantages :

- Helps in better understanding of requirements.
- Early user involvement and feedback.
- Reduces risk of requirement changes.
- Improves the final product quality.
- Suitable for complex and large systems.

6. Disadvantages :

- → Time consuming if many iterations are required.
- → May lead to unclear or unstable requirements.
- → Not suitable for very small projects.
- → Too much dependence on user availability and feedback.

7. Suitable For :

Prototype Model is suitable for projects where requirements are not well defined or likely to change frequently. It is commonly used in new product development, R&D projects and user interface design.

Conclusion : Prototype Model helps in building a working model early in the development process, allows user to interact and provide feedback, and ensures the final system meets user needs effectively.

SPIRAL MODEL

1. Introduction :

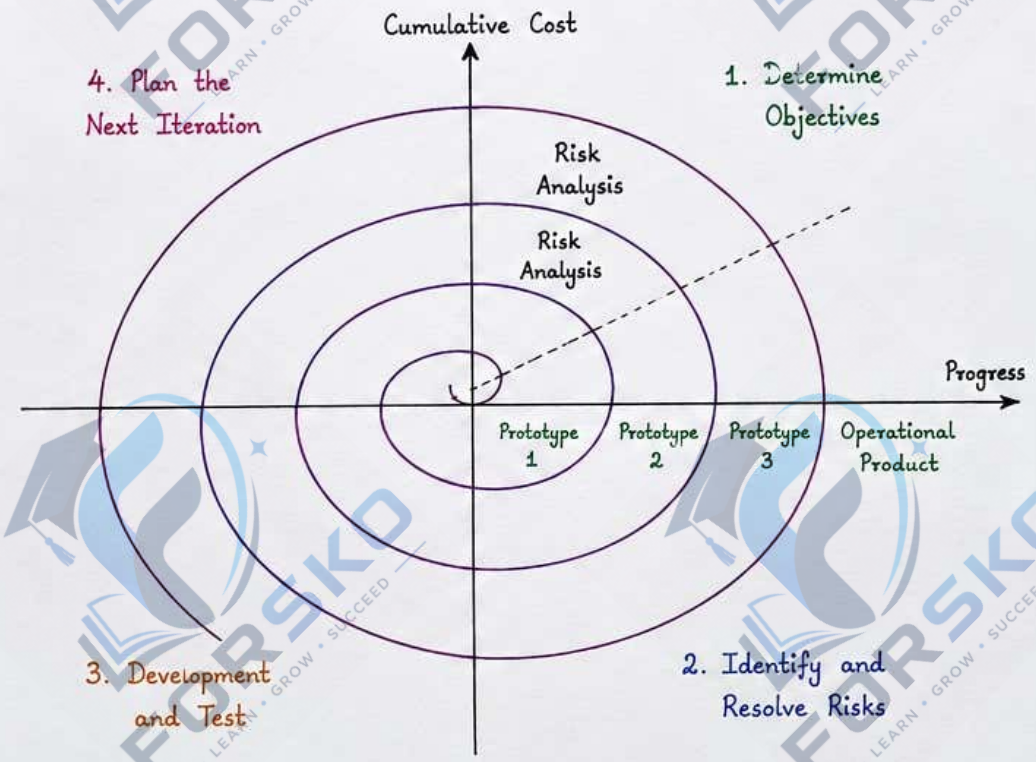
The Spiral Model is a risk-driven software development model. It combines the iterative nature of prototyping with the systematic and controlled aspects of the waterfall model.

It was proposed by Barry Boehm.

2. Key Points :

- It is a spiral of activities with cycles called "loops".
- Each loop represents a phase of the process.
- It emphasizes risk assessment and reduction.
- Suitable for large, complex and high-risk projects.
- Allows for changes and customer feedback at any stage.

3. Phases of Spiral Model :



4. Explanation of Phases :

1. Determine Objectives :

- Define the objectives, alternatives and constraints of the project.

2. Identify and Resolve Risks :

- Identify the risks associated with the project.
- Analyze the risks and develop strategies to reduce or eliminate them.

3. Development and Test :

- Develop the product for the current iteration.
- Test and validate the developed product.

4. Plan the Next Iteration :

- Review the results.
- Plan the next iteration based on the feedback and results.

5. Advantages :

- Effective risk management.
- Accommodates changes at any time.
- More suitable for large and complex projects.
- Early customer feedback.
- Better control over cost and schedule.
- Higher chance of project success.

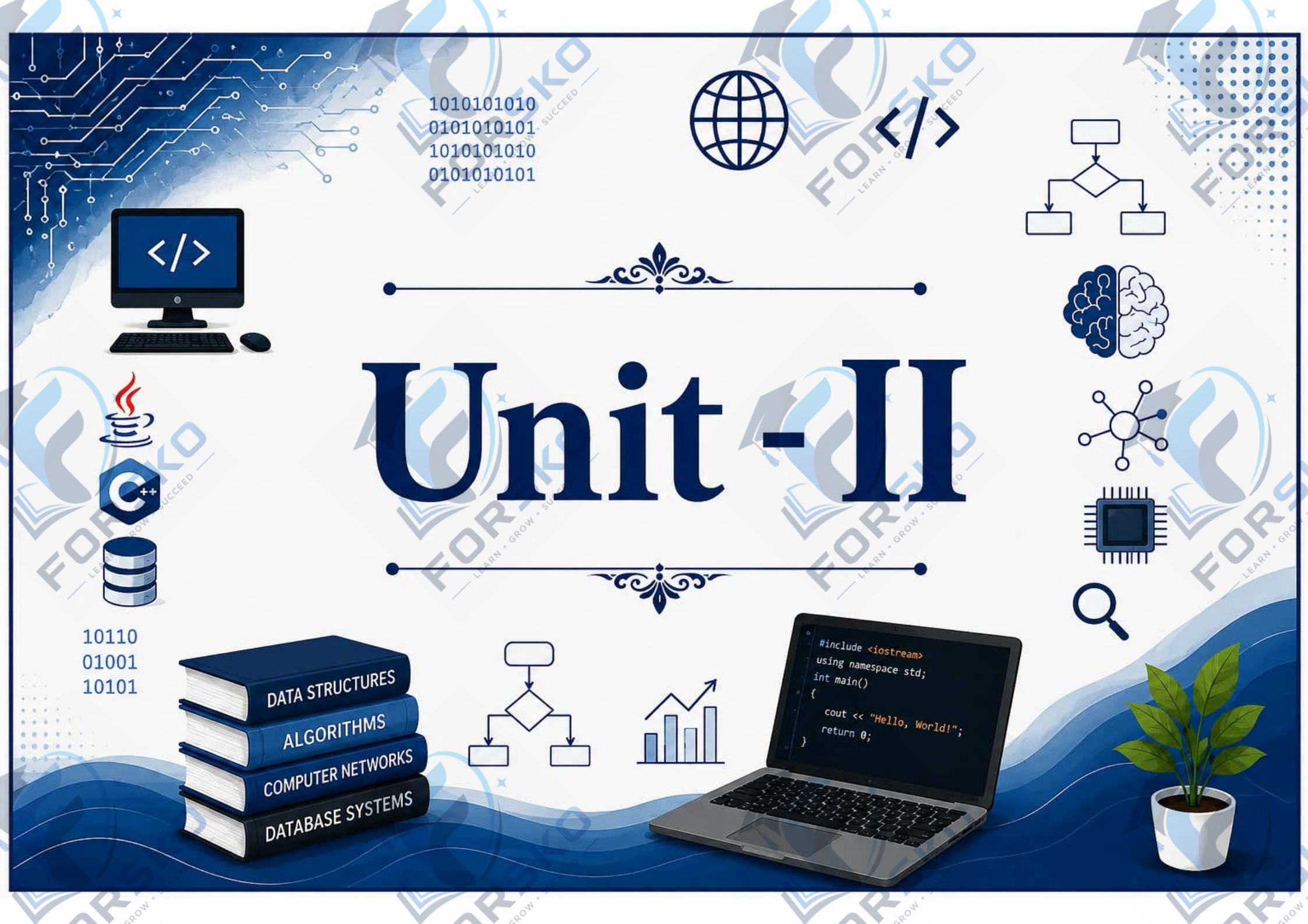
6. Disadvantages :

- Complex and difficult to understand.
- Requires risk assessment expertise.
- Costly and time consuming.
- Not suitable for small projects.
- Need for continuous customer involvement.

7. Suitable For :

- Large scale, complex and high-risk projects.
- Projects with incomplete or unclear requirements.
- New product development.
- Projects where changes are expected.
- R&D and innovative projects.

Conclusion : The Spiral Model provides a systematic and iterative approach to software development with strong emphasis on risk analysis. It is highly flexible and is suitable for large, complex and high-risk projects.


1010101010
0101010101
1010101010
0101010101
Unit -II
10110
01001
10101
DATA STRUCTURES
ALGORITHMS
COMPUTER NETWORKS
DATABASE SYSTEMS
#include <iostream>
using namespace std;
int main()
{
cout << "Hello, World!";
return 0;
}

SOFTWARE REQUIREMENTS

Software Requirements are the capabilities, features, functions, constraints and qualities that a system must possess to satisfy the needs of stakeholders and solve a problem.

1. Introduction

- Requirements are the foundation of any software system.
- They describe what the system should do and the constraints under which it must operate.
- Clear and complete requirements are essential for successful software development.

2. Characteristics of Good Requirements (SMART)

- Specific – Well defined and unambiguous.
- Measurable – Can be measured or tested.
- Achievable – Realistic and attainable.
- Relevant – Important to stakeholders.
- Time-bound – Defined within a time frame.

3. Types of Requirements

A. Functional Requirements

- Define the functions or services the system should provide.
- Describe what the system should do.
- Example: User login, Calculate total amount, Generate reports.

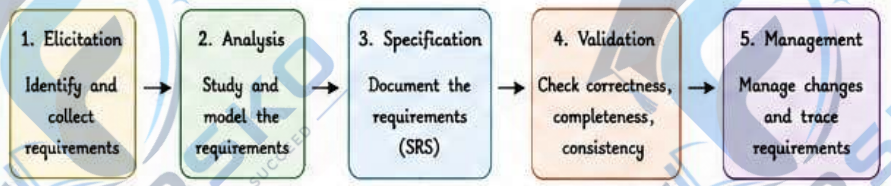
B. Non-Functional Requirements

- Define the quality attributes and constraints of the system.
- Describe how well the system should perform.
- Examples:
  - Performance (e.g., response time < 2 sec)
  - Reliability (e.g., 99.9% uptime)
  - Usability (e.g., easy to use)
  - Security (e.g., data encryption)
  - Maintainability (e.g., easy to modify)

4. Sources of Requirements

- Customers / End Users
- Stakeholders
- Domain experts
- Existing systems and documents
- Laws, regulations, standards
- Market research

5. Requirements Engineering Process



6. Software Requirements Specification (SRS)

- SRS is a formal document that describes all requirements in detail.
- It acts as a contract between the customer and the development team.
- It is the basis for design, development, testing and maintenance.
- Typical SRS Contents:
  1. Introduction
  2. Overall Description
  3. Functional Requirements
  4. Non-Functional Requirements
  5. External Interface Requirements
  6. System Features
  7. Constraints
  8. Appendices

7. Importance of Good Requirements

- Provide a clear understanding among all stakeholders.
- Help in accurate estimation of cost and time.
- Reduce changes and rework.
- Improve quality of the software.
- Ensure customer satisfaction.

8. Problems due to Poor Requirements

- Misunderstandings and confusion.
- Scope creep (uncontrolled changes).
- Cost and schedule overrun.
- Low quality and customer dissatisfaction.
- Project failure.

9. Requirements Documentation

Requirements can be documented using:

- Natural Language
- Use Cases
- User Stories
- Data Flow Diagrams (DFD)
- Entity Relationship Diagrams (ERD)
- Prototypes
- Models

Summary : Software requirements define what the system should do and the constraints under which it must operate. Proper elicitation, analysis, specification and management of requirements are critical for successful software development.

FUNCTIONAL REQUIREMENTS

Functional requirements define what the system should do. They describe the functions or services that the system must provide, the inputs it should accept, the outputs it should produce and how it should respond in specific situations.

1. Introduction

- Functional requirements specify the behavior of the system.
- They describe the services or functions the system must perform.
- They are written from the point of view of the user.
- Each requirement should express a single function or service.

2. Characteristics of Good Functional Requirements

- **Correct** – Should reflect the real need.
- **Unambiguous** – Should be clear and easy to understand.
- **Complete** – All required functions should be included.
- **Consistent** – No conflicting requirements.
- **Verifiable** – Can be tested or demonstrated.
- **Modifiable** – Easy to change if required.
- **Traceable** – Each requirement should be traceable to a source.

3. Sources

- Users / Customers
- Stakeholders
- Domain experts
- Existing systems
- Business rules and policies
- Regulations and standards

4. How to Write Functional Requirements

- Use clear and simple language.
- Use active voice.
- Each requirement should start with a verb.
- Specify input, output and processing.
- Use tables, use cases or scenarios where necessary.

Template :

The system shall [perform a function] when [certain condition] by [system response].

5. Examples of Functional Requirements

No.	Requirement Description	Type
1	The system shall allow a registered user to login using valid username and password.	User Authentication
2	The system shall allow the user to search for a product by name or category.	Search
3	The system shall allow the user to add selected products to the cart.	Shopping Cart
4	The system shall calculate the total amount including taxes and discounts.	Calculation
5	The system shall generate an order confirmation after successful payment.	Order Processing
6	The system shall allow the administrator to manage (add, update, delete) products.	Admin Function

6. Types of Functional Requirements

- **Input Requirements** – Data or information to be provided to the system.
- **Output Requirements** – Information that the system should produce.
- **Processing Requirements** – Data processing or business logic to be performed.
- **Control Requirements** – Controlling the flow of activities.

7. Functional Requirements vs Non-Functional Requirements

Functional Requirements	Non-Functional Requirements
Define what the system does.	Define how well the system performs.
Describe functions, services and behavior.	Describe quality attributes and constraints.
Examples: Login, Search Product, Generate Report.	Examples: Performance, Security, Usability.

8. Importance

- Provide a clear understanding of system functions.
- Basis for design, development and testing.
- Help in estimation of cost and time.
- Help in validating the system by users.

Conclusion : Functional requirements describe the specific functions the system must perform. Well-defined functional requirements are essential for building a system that meets user needs and expectations.

NON-FUNCTIONAL REQUIREMENTS (NFRs)

Non-functional requirements (NFRs) define **how well** the system should perform certain functions. They specify the **quality attributes**, **constraints** and **characteristics** of the system rather than specific behaviors.

1. Introduction

- Non-functional requirements describe the quality attributes of a system.
- They define the constraints under which the system operates.
- They affect system design, implementation, testing and maintenance.

2. Characteristics of Good Non-Functional Requirements

- **Measurable** – Can be measured or tested.
- **Specific** – Clearly defined and unambiguous.
- **Achievable** – Realistic and attainable.
- **Relevant** – Important to stakeholders.
- **Time-bound** – Defined for a specific period or condition.
- **Verifiable** – Can be verified or validated.
- **Consistent** – No conflict with other requirements.

3. Examples of Non-Functional Requirements

Quality Attribute	Example
Performance	The system shall respond to user requests within 2 seconds.
Reliability	The system shall be available 99.9% of the time.
Usability	The system shall be easy to use for all categories of users.
Security	The system shall protect data from unauthorized access.
Maintainability	The system shall allow easy modifications with minimal effort.
Portability	The system shall run on Windows, Linux and Mac OS.

4. Common Types of Non-Functional Requirements

Type	Description
Performance	• Response time, throughput, processing time • Example: The system shall generate reports within 5 seconds.
Reliability	• Availability, fault tolerance, recoverability • Example: The system shall be available 24/7 with 99.9% uptime.
Usability	• Ease of use, learnability, user interface • Example: The system shall be easy to learn and use.
Security	• Confidentiality, integrity, authentication, authorization • Example: The system shall ensure data encryption during transmission.
Maintainability	• Ease of maintenance, modularity, testability • Example: The system shall be designed in modules for easy maintenance.
Scalability	• Ability to handle increased load • Example: The system shall support up to 10,000 concurrent users.
Portability	• Ability to run on different platforms • Example: The system shall be platform independent.
Compatibility	• Works with other systems or software • Example: The system shall be compatible with Chrome, Firefox and Edge browsers.

5. How to Write Non-Functional Requirements

Template :

The system shall be [quality attribute] and shall [specific measurable criteria] under [specific conditions].

Examples :

- The system shall response to user requests within 2 seconds under normal load.
- The system shall be available 99.9% of the time, measured monthly.
- The system shall support up to 5000 concurrent users.

6. Importance of Non-Functional Requirements

- Improve overall quality and user satisfaction.
- Help in making design and architectural decisions.
- Define constraints and standards for development.
- Help in testing and acceptance of the system.
- Reduce maintenance cost and increase reliability.

7. Difference: Functional vs Non-Functional

Functional Requirements	Non-Functional Requirements
Define what the system does.	Define how well the system does it.
Describe specific functions or services.	Describe quality attributes or constraints.
Example: User login, Add product.	Example: Response time, Security.
Visible to end users.	Usually invisible to end users.
Change less frequently.	May change with environment or standards.

Conclusion : Non-functional requirements ensure that the system is not only functionally correct but also reliable, usable, secure, fast and maintainable. They play a vital role in building a high-quality software system.

USER REQUIREMENTS

User requirements are high-level statements of what the users (customers) need from a system. They describe the services, features and goals of the system from the user's point of view, without focusing on how the system will be built.

1. Introduction

- User requirements describe what the users expect from the system.
- They focus on the users' tasks, goals and needs.
- They are written in natural language and are easy for all stakeholders to understand.
- They do not describe the technical or implementation details.

2. Characteristics of Good User Requirements

- Clear and easy to understand
- Express user needs and goals
- Independent of technology
- Feasible and realistic
- Verifiable and testable
- Unambiguous and complete

3. Sources of User Requirements

- End users
- Customers
- Stakeholders
- Interviews and discussions
- Questionnaires and surveys
- Observation
- Existing documents and systems

4. User Requirements vs System Requirements

User Requirements	System Requirements
Describe what users need.	Describe what the system must do and how it will do.
Written in natural language.	Can be in natural or formal language.
High-level and general.	Detailed and specific.
Focus on user goals and tasks.	Focus on system functions and constraints.
Do not include technical details.	Include technical and implementation details.

5. Examples of User Requirements

No.	User Requirement Description
1.	The system shall allow users to register and create an account.
2.	The system shall allow users to login securely.
3.	The system shall allow users to search for products by name or category.
4.	The system shall allow users to add selected products to the cart.
5.	The system shall allow users to place an order and receive confirmation.
6.	The system shall provide users with order history and tracking.
7.	The system shall allow users to update their profile information.
8.	The system shall provide help and support to users.

6. How to Write User Requirements

- Use simple and non-technical language.
- Focus on the user and their tasks.
- Use active voice and positive statements.
- Each requirement should express a single need.
- Avoid solution or design details.
- Use "shall" or "should" to state requirements.

7. Format / Template

The system shall [provide a service or feature] so that [user can achieve a goal or perform a task].

(This template ensures clarity and testability.)

8. Importance of User Requirements

- Help in understanding user needs and goals.
- Provide a basis for deriving system requirements.
- Improve communication among stakeholders.
- Reduce misunderstanding and rework.
- Ensure the final system meets user expectations and satisfaction.

Conclusion : User requirements are the foundation of any software development project. They capture the users' needs and expectations in simple terms and help in building a system that solves the right problem for the right users.

SYSTEM REQUIREMENTS

System requirements are detailed descriptions of what the system must do and the constraints under which it must operate. They convert user requirements into a precise, technical specification that guides design, implementation, testing and maintenance.

1. Introduction

- System requirements describe the behavior, features, interfaces, data and constraints of the system in technical terms.
- They are the basis for system design, development, testing and validation.
- They include both functional and non-functional requirements.

2. Characteristics of Good System Requirements

- **Correct** – Should reflect the real need.
- **Complete** – All necessary requirements should be included.
- **Consistent** – No conflicting requirements.
- **Unambiguous** – Clear and single interpretation.
- **Verifiable** – Can be tested and validated.
- **Feasible** – Achievable with available technology and resources.
- **Necessary** – Include only essential requirements.
- **Traceable** – Traceable to user requirements.

3. Sources of System Requirements

- User requirements
- Stakeholders
- System analysis and modeling
- Domain experts
- Existing systems and documents
- Laws, regulations and standards
- Business rules and policies
- Market and technical constraints

4. Types of System Requirements

- **Functional Requirements** : Describe what the system shall do.
- **Non-Functional Requirements** : Describe how well the system shall do it and the constraints on its operations.

5. Components of System Requirements

Component	Description
Functional Requirements	Functions, services and behaviors the system must provide.
Non-Functional Requirements	Quality attributes and constraints such as performance, security, reliability, usability etc.
Data Requirements	Data to be stored, processed, validated and maintained by the system.
Interface Requirements	Interfaces with users, other systems and hardware devices.
Operational Requirements	Requirements for installation, operation, maintenance, backup and recovery.
Design Constraints	Constraints on standards, platforms, tools, budget, time, laws, etc.

6. Examples of System Requirements

No.	System Requirement Description
1.	The system shall allow users to register and create an account.
2.	The system shall authenticate users using username and password.
3.	The system shall allow users to search for products by name or category.
4.	The system shall generate invoices for confirmed orders.
5.	The system shall respond to user requests within 2 seconds.
6.	The system shall ensure 99.9% availability during normal operations.
7.	The system shall protect user data using encryption.

7. How to Write System Requirements

- ✓ Use precise and technical language.
- ✓ Each requirement should be clear, complete and testable.
- ✓ Use active voice.
- ✓ Use "shall" or "should" for requirement statements.
- ✓ Include necessary details such as input, output, processing, data, interfaces and constraints.
- ✓ Ensure each requirement can be verified.

8. System Requirements Specification (SRS)

- System requirements are documented in the SRS.
- SRS acts as a contract between customer and development team.
- It is used for design, development, testing, maintenance and user training.
- Typical SRS Contents :

1. Introduction
2. Overall Description
3. Functional Requirements
4. Non-Functional Requirements
5. Interface Requirements
6. Data Requirements
7. Design Constraints
8. Operational Requirements
9. Assumptions and Dependencies
10. Appendices

9. Importance of System Requirements

- Provide a clear and precise understanding of the system.
- Serve as a basis for design and development.
- Help in estimating cost, time and resources.
- Help in testing, validation and quality assurance.
- Reduce risks of misunderstandings and rework.
- Ensure the final system meets user needs and business goals.

Conclusion : System requirements translate user needs into a detailed and testable specification. They guide the entire software development process and ensure that the system is correct, reliable, efficient and meets all constraints.

REQUIREMENTS ENGINEERING PROCESS

Requirements engineering is the process of discovering, analyzing, documenting and managing the requirements of a software system. It ensures that the right system is built in the right way.

1. Introduction

- Requirements engineering is the first and most important activity in software development.
- It deals with understanding the needs of stakeholders and converting them into a complete, consistent and testable requirements specification.
- It continues throughout the life cycle of the system.

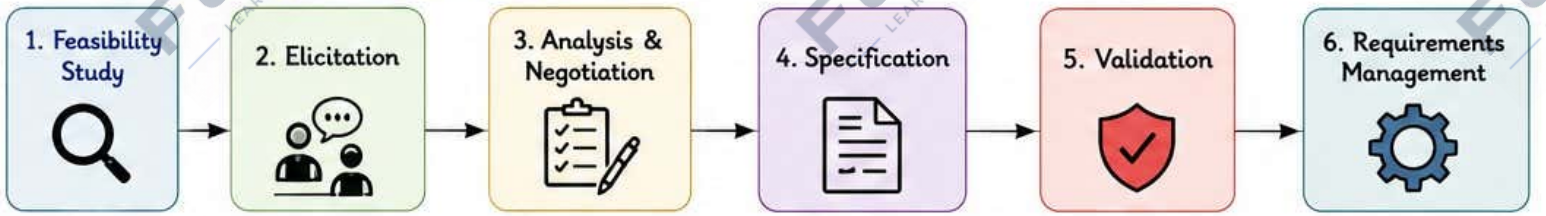
2. Objectives

- To understand the needs of stakeholders.
- To produce high quality requirements.
- To form a baseline for design, development, testing and validation.
- To manage changing requirements.

3. Key Activities

- Elicitation of requirements
- Analysis and negotiation
- Specification and documentation
- Validation
- Management of requirements

4. Requirements Engineering Process (Phases)



1. Evaluate the feasibility and scope of the proposed system.
2. Discover and collect requirements from stakeholders.
3. Analyze requirements for correctness, consistency and feasibility. Resolve conflicts and prioritize.
4. Document requirements in a clear, complete and unambiguous manner (SRS).
5. Validate requirements to ensure they are correct, consistent, complete and testable.
6. Manage changes to requirements and maintain traceability throughout the project.

5. Details of Each Phase

Phase	Purpose	Key Activities	Output / Deliverables
1. Feasibility Study	Determine whether the project is worth doing.	• Technical feasibility • Economic feasibility • Operational feasibility	Feasibility Report
2. Elicitation	Discover and collect requirements.	• Interviews, questionnaires • Observation • Brainstorming, workshops • Study existing documents	Raw requirements (Preliminary list)
3. Analysis & Negotiation	Understand, model and prioritize requirements.	• Classify and organize • Modeling (use cases, DFD, ERD) • Identify conflicts • Prioritize requirements	Analyzed and prioritized requirements
4. Specification	Document requirements in standard form.	• Write SRS • Use standard templates • Define functional and non-functional requirements	Software Requirements Specification (SRS)
5. Validation	Ensure requirements are correct and complete.	• Reviews and walkthroughs • Prototyping • Test case derivation	Validated requirements
6. Requirements Management	Handle changes and maintain traceability.	• Change control • Version control • Traceability matrix	Requirements baseline, Change reports, Traceability matrix

6. Principles of Requirements Engineering

- The customer and stakeholder are involved.
- Requirements are understandable.
- Requirements are complete, consistent and feasible.
- Requirements evolve over time.
- Requirements are traceable.
- High quality requirements are essential for success.

7. Benefits

- Leads to better understanding of user needs.
- Reduces rework and project risks.
- Improves communication among stakeholders.
- Provides a clear baseline for design and testing.
- Helps in delivering a system that adds real value.

8. Summary

The requirements engineering process ensures that we build the right system and build the system right. It is an iterative activity and continues throughout the software development life cycle.

Note : Good requirements = Clear + Complete + Consistent + Feasible + Verifiable + Traceable + Modifiable

REQUIREMENTS ELICITATION AND ANALYSIS

1. Requirements Elicitation

Requirements elicitation is the process of discovering, gathering and understanding the needs and expectations of stakeholders for a proposed system.

1.1 Objectives

- To discover all possible requirements.
- To understand the real needs of stakeholders.
- To identify constraints and priorities.
- To establish a basis for further analysis and specification.

1.2 Techniques for Elicitation

→ Interviews – One to one discussion with stakeholders.
→ Questionnaires – Set of questions to collect information.
→ Surveys – Collect data from large number of people.
→ Observation – Observe the existing system and users.
→ Workshops / JAD – Group discussion with stakeholders.
→ Brainstorming – Generate as many ideas as possible.
→ Document Analysis – Study existing documents, reports, manuals etc.
→ Prototyping – Build a quick prototype to understand user needs.
→ Use Cases – Identify actors and their goals.

Stakeholders : Users, Customers, Managers, Developers, Domain Experts, End Users, etc.

1.3 Elicitation Process

2. Requirements Analysis

Requirements analysis is the process of studying elicited requirements to understand, organize, model and validate them.

2.1 Objectives

- To classify and organize requirements.
- To check consistency, completeness and feasibility.
- To resolve conflicts and remove ambiguity.
- To prioritize requirements.
- To create a clear and structured model of requirements.

2.2 Activities in Requirements Analysis

→ Classification – Group similar requirements.
→ Organization – Structure the requirements.
→ Modeling – Use models like Use Case Diagrams, Data Flow Diagrams, ER Diagrams.
→ Prioritization – Identify must have, should have, could have requirements.
→ Validation – Check requirements with stakeholders.
→ Conflict Resolution – Resolve contradictory requirements.
→ Feasibility Study – Technical, economic and operational feasibility.

2.3 Qualities of Good Requirements

1. Correct	6. Verifiable / Testable
2. Complete	7. Necessary
3. Consistent	8. Prioritized
4. Unambiguous	9. Traceable
5. Feasible	10. Modifiable

2.4 Elicitation vs Analysis

Elicitation	Analysis
Discovers and gathers requirements.	Studies and understands requirements.
Focus on collecting information.	Focus on organizing and refining information.
Output: Raw requirements	Output: Analyzed and structured requirements.

REQUIREMENTS VALIDATION

1. Introduction

Requirements validation is the process of checking the requirements to ensure that they are correct, complete, consistent, feasible and testable before system development begins.

2. Objectives

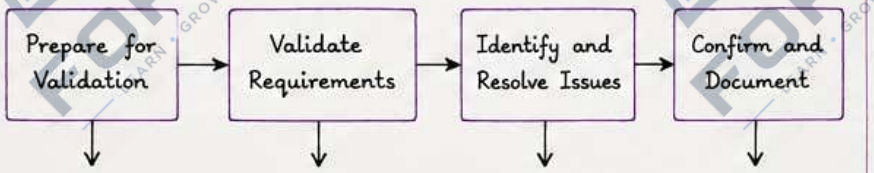
- To ensure that requirements reflect the actual needs of stakeholders.
- To detect errors, omissions, inconsistencies and ambiguities.
- To ensure that requirements are realistic, feasible and testable.
- To reduce the cost of changes by finding problems early.

3. Importance

→ Improves quality of requirements.
→ Reduces risk and rework.
→ Ensures customer satisfaction.
→ Provides a good baseline for design, development and testing.
→ Saves time and cost.

Note : Finding errors in requirements later (during design or coding) is much more expensive. Therefore, validate early and validate thoroughly.

4. Validation Process



- Plan the validation process, identify stakeholders and gather requirements documents.
- Check requirements using various validation techniques and criteria.
- Log all issues found and resolve them with the help of stakeholders.
- Confirm that all issues are resolved and document the validation results.

5. Validation Techniques

→ **Reviews** – Systematic examination of requirements by reviewers.
→ **Walkthroughs** – Author explains requirements to stakeholders.
→ **Inspections** – Formal process with defined roles and checklists.
→ **Prototyping** – Build a prototype to validate requirements.
→ **Test Case Validation** – Check if requirements can be tested.
→ **Consistency Analysis** – Check for conflicts and inconsistencies.
→ **Model Validation** – Validate models like use case, DFD, ERD etc.
→ **Requirement Traceability** – Trace requirements to source and downstream artifacts.

6. Validation Criteria (Qualities)

1. Correct
2. Complete
3. Consistent
4. Unambiguous
5. Feasible
6. Verifiable / Testable
7. Necessary
8. Prioritized
9. Traceable
10. Understandable

7. Activities in Validation

✓ Validate each requirement with stakeholders.
✓ Check for completeness and correctness.
✓ Ensure requirements are unambiguous.
✓ Check consistency among requirements.
✓ Verify feasibility with available technology, budget and time.
✓ Ensure requirements are testable.
✓ Obtain sign-off from stakeholders.

8. Validation vs Verification

Validation	Verification
Are we building the right product?	Are we building the product right?
Focus on the correct requirements.	Focus on the correctness of work.
Done with stakeholders (customers, users).	Done by the development team.
Occurs before development.	Occurs during and after development.

9. Deliverables

→ Validated Requirements Document
→ Validation Report
→ List of Issues and Resolutions
→ Signed-off Requirements Baseline

Conclusion : Requirements validation ensures that the requirements are accurate, complete and acceptable to all stakeholders. It is a critical step to build the right system and avoid costly changes later.

REQUIREMENTS MANAGEMENT

1. Introduction

Requirements management is the process of identifying, organizing, tracking and controlling requirements and changes to requirements throughout the project.

2. Objectives

- To manage changes to requirements.
- To maintain consistency and traceability of requirements.
- To ensure that requirements are complete, correct and up-to-date.
- To communicate requirements effectively among stakeholders.
- To control the impact of changes on cost, time and quality.

3. Importance

→ Accommodates changing needs of stakeholders.
→ Reduces the impact of requirement changes.
→ Maintains requirement integrity.
→ Improves communication and coordination.
→ Ensures better planning, estimation and tracking.
→ Helps in delivering the right product.

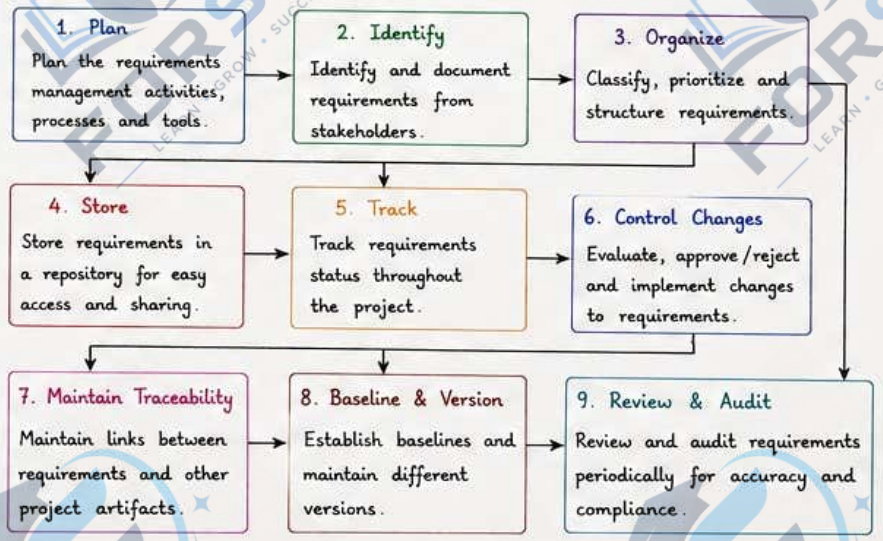
Note : Good requirements management ensures that the right requirements are built, changes are controlled and the final product meets stakeholder needs.

Requirements Management is the process of managing changes to requirements throughout the software development life cycle.

4. Key Activities in Requirements Management

- ☐ **Requirements Identification** – Identify and document requirements.
- ☐ **Requirements Organization** – Structure and organize requirements.
- ☐ **Requirements Storage** – Store requirements in a repository.
- ☐ **Requirements Traceability** – Maintain links among requirements and other work products.
- ☐ **Change Management** – Handle changes to requirements.
- ☐ **Version Control** – Maintain versions of requirements.
- ☐ **Status Tracking** – Track the status of requirements.
- ☐ **Requirements Baseline** – Establish and maintain baselines.
- ☐ **Impact Analysis** – Analyze the impact of changes.
- ☐ **Requirements Audit** – Review requirements for correctness and compliance.

5. Requirements Management Process



6. Change Management Process

1. Change Request	→ A change is requested by a stakeholder.
2. Impact Analysis	→ Analyze impact on scope, cost, time and quality.
3. Change Evaluation	→ Evaluate the change request (Approve / Reject / Defer).
4. Change Implementation	→ Implement the approved changes.
5. Update & Communicate	→ Update documents, baseline and inform stakeholders.
6. Review	→ Review the change and its effects.

7. Tools for Requirements Management

→ DOORS (IBM Engineering Requirements Management DOORS)
→ JIRA
→ Polarion
→ ReqIF
→ Microsoft Excel / Word (for small projects)
→ Caliber RM

8. Deliverables

- Requirements Management Plan
- Requirements Repository
- Requirements Traceability Matrix
- Change Request Log
- Requirements Status Report
- Requirements Baseline

Conclusion : Requirements management ensures that requirement changes are identified, evaluated, controlled and tracked properly. It helps in delivering a high quality product that meets stakeholder needs within time and budget.

Unit -III


1010101010
0101010101
1010101010
0101010101
</>
</>
10110
01001
10101
DATA STRUCTURES
ALGORITHMS
COMPUTER NETWORKS
DATABASE SYSTEMS
#include <iostream>
using namespace std;
int main()
{
cout << "Hello, World!";
return 0;
}

RISK MANAGEMENT

1. Introduction

Risk management is the process of identifying, analyzing and controlling potential events (risks) that may affect the success of a project or the system.

2. Objectives

- To identify potential risks early.
- To analyze the impact and probability of risks.
- To plan and implement strategies to reduce or eliminate risks.
- To monitor risks throughout the project.

3. Types of Risks

Type of Risk	Description
Project Risks	Risks related to schedule, cost, resources, and planning.
Technical Risks	Risks related to technology, tools, or technical issues.
Business Risks	Risks that affect the business objectives or stakeholders.
External Risks	Risks from outside the project (market, legal, environment).
Operational Risks	Risks related to day-to-day operations and processes.

4. Risk Management Process

1. **Risk Identification**
   Identify all possible risks that could affect the project.
2. **Risk Analysis**
   Analyze each risk in terms of probability and impact.
3. **Risk Evaluation**
   Prioritize risks based on their severity (High / Medium / Low).
4. **Risk Mitigation (Response)**
   Plan and implement strategies to reduce or eliminate risks.
5. **Risk Monitoring**
   Track identified risks and monitor new risks throughout the project.
6. **Risk Review & Control**
   Review risk status and control changes when required.

5. Risk Analysis

- Probability (P) : Likelihood of risk occurring.
- Impact (I) : Effect of risk on project if it occurs.
- Risk Exposure (RE) = Probability (P) × Impact (I)

Probability (P)	Impact (I)	Risk Level
High (0.7 – 1.0)	High	High
Medium (0.3 – 0.6)	Medium	Medium
Low (0 – 0.2)	Low	Low

6. Risk Mitigation Strategies

→ Avoid : Change the plan to eliminate the risk.
→ Reduce : Take action to reduce probability or impact.
→ Transfer : Shift the risk to a third party.
→ Accept : Acknowledge the risk and prepare a contingency plan.

7. Risk Monitoring

- Regularly review risk status.
- Track new risks and changes.
- Update risk management plan.
- Ensure mitigation actions are effective.

8. Risk Management Plan – Key Contents

* Risk Identification
* Risk Analysis
* Risk Evaluation
* Mitigation Strategies
* Risk Monitoring Plan
* Roles and Responsibilities
* Reporting Mechanism
* Review and Update Plan

9. Benefits of Risk Management

✓ Improves project success rate.
✓ Reduces unexpected problems and costs.
✓ Helps in better planning and decision making.
✓ Increases stakeholder confidence.

Conclusion : Risk management helps in identifying potential problems early and taking appropriate actions to ensure project success within time, cost and quality.

RISK STRATEGIES

1. Introduction :

Risk strategies are the planned actions taken to manage the risks that may affect the success of a project. The main aim is to reduce the probability or impact of risks or to handle them effectively.

2. Objectives :

- To reduce the negative impact of risks.
- To take advantage of positive risks (opportunities).
- To improve the chances of project success.
- To ensure better planning and decision making.

3. Types of Risks :

→ Project Risks – related to cost, time, scope, resources.
→ Technical Risks – related to technology, tools, technical issues.
→ Business Risks – related to market, customers, competition.
→ External Risks – related to legal, political, environmental factors.
→ Operational Risks – related to day-to-day operations, processes.

4. Risk Strategies (Responees) :

The main risk strategies are :

Strategy	Description	When to Use	Example
1. Avoid	Change the plan to eliminate the risk completely.	When the risk is too high and cannot be tolterated.	If a technology is too risky, choose an alternative technology.
2. Mitigate	Take actions to reduce the probability or impact of the risk.	When the risk cannot be eliminated but can be reduced.	Add more testing to reduce the chance of software failure.
3. Transfer	Shift the risk to a third party who is better able to manage it.	When another party is more capable of handling the risk.	Buy insurance to transfer financial risk.
4. Accept	Acknowledge the risk and do nothing unless it occurs.	When the risk is low or the cost of dealing with it is more than the benefit.	Accept minor delays that do not affect the overall project.
5. Exploit (for Opportunities)	Take steps to ensure that an opportunity (positive risk) occurs.	When the opportunity can give a significant benefit.	Use a new tool to improve productivity and quality.
6. Enhance (for Opportunities)	Increase the probability and impact of an opportunity.	When the opportunity can give more advantage.	Provide training to enhance team skills and performance.
7. Share (for Opportunities)	Share the opportunity with others who can help maximize it.	When collaboration can increase the benefit of the opportunity.	Partner with another company for better market reach.

5. Key Points :

* No strategy is perfect for all risks.
* Choose the best strategy based on risk type, impact, cost and resources.
* Risk strategies should be reviewed and updated regularly.

6. Conclusion :

Effective risk strategies help in reducing threats, taking advantage of opportunities and ensuring the success of the project.

SOFTWARE RISKS

1. Introduction :

Software risks are potential problems that may occur during software development and affect the project in terms of cost, time, quality or scope.

2. Characteristics of Software Risks :

→ They are uncertain events.
→ They can have negative or positive impact.
→ They should be identified early.
→ They can change over time.
→ They can affect project success.

3. Sources of Software Risks :

- Requirements
- Technology
- People
- Process
- Tools
- Environment
- Project Management

4. Types of Software Risks :

Risk Category	Description	Examples	Impact	Related To
1. Requirement Risks	Risks related to problems in understanding, defining or managing requirements.	• Unclear requirements • Changing requirements • Missing requirements	Scope changes, rework, delays, cost overrun	Requirements
2. Technical Risks	Risks related to technology, tools, hardware or technical complexity.	• New or unproven technology • Incompatible systems • Performance issues	Integration problems, technical failures, performance degradation	Technology
3. People Risks	Risks related to people involved in the project.	• Lack of skilled resources • High staff turnover • Poor communication	Low productivity, delays, quality issues	People
4. Process Risks	Risks related to the software process and development methods.	• Inadequate process • Not following standards • Poor estimation	Rework, defects, schedule slippage	Process
5. Tool Risks	Risks related to tools and software development environments.	• Tool failures • Tool not suitable • Lack of tool support	Reduced productivity, data loss, delays	Tools
6. Project Management Risks	Risks related to planning, scheduling, monitoring and controlling the project.	• Unrealistic schedules • Poor risk management • Inadequate resources	Cost overrun, delays, project failure	Project Management
7. Environmental Risks	Risks related to external environment factors.	• Changes in market • Legal or regulatory changes • Natural disasters	Project disruption, increased cost, delays	Environment
8. Quality Risks	Risks related to software quality and reliability.	• High defect rate • Poor testing • Maintainability issues	Low quality product, customer dissatisfaction	Quality

5. Software Risk Management Steps :

→ Risk Identification
→ Risk Analysis
→ Risk Prioritization
→ Risk Response Planning
→ Risk Monitoring
→ Risk Control

6. Importance :

- Helps in early identification of potential problems.
- Reduces uncertainty and improves decision making.
- Helps in completing the project on time, within budget and with quality.
- Improves chances of project success.

Note :
Not all risks can be avoided, but they can be managed.

Risk Identification

1. Introduction :

Risk Identification is the process of finding, recognizing and describing potential risks that may affect the success of a project.

2. Objectives :

- ✓ To identify potential risks early.
- ✓ To understand the causes and sources of risks.
- ✓ To document risks for further analysis and action.
- ✓ To provide input for risk analysis, planning and management.

3. Importance :

- → Helps in early detection of problems.
- → Reduces uncertainty and surprises.
- → Improves planning and decision making.
- → Saves time, cost and resources.
- → Increases the chances of project success.

4. Risk Identification Process :

1. Plan Risk Identification
Decide how, when and by whom risk identification will be done.

↓

2. Identify Risks
Find and list all possible risks that may affect the project.

↓

3. Document Risks
Record identified risks with details such as causes, impact and category.

↓

4. Categorize Risks
Group risks into categories for better understanding and analysis.

↓

5. Review and Update
Review the list regularly and add new risks as project changes.

5. Techniques for Risk Identification :

- * Brainstorming – Group discussion to generate list of risks.
- * Checklists – Using predefined checklists of common risks.
- * Interviews – Asking experts and stakeholders for their views.
- * Delphi Technique – Getting opinions from a panel of experts.
- * SWOT Analysis – Identifying risks from Weaknesses and Threats.
- * Assumption Analysis – Identifying risks from project assumptions.
- * Historical Data – Using data from past projects and records.

6. Risk Identification Template (Example) :

Risk ID	Risk Description	Category	Possible Cause	Potential Impact	Identified By	Date
R-01	Delay in requirement gathering	Requirement	Unclear or changing needs	Project delay, Cost overrun	Project Team	20/05/24
R-02	Technology not compatible	Technical	New or untested technology	System failure, Rework	Technical Lead	20/05/24
R-03	Resource unavailability	Resource	Lack of skilled resources	Schedule delay, Low productivity	Project Manager	20/05/24
R-04	Budget overrun	Cost	Incorrect estimation	Financial loss, Scope reduction	Finance Team	20/05/24

7. Sources of Risks :

- Project scope and requirements
- Technology and tools
- People and resources
- Processes and methods
- External environment (legal, market, economic)
- Stakeholders

8. Good Practices :

- → Involve the right people in identification.
- → Use multiple techniques for better results.
- → Document all identified risks properly.
- → Keep the risk list updated throughout the project.

Note :
Not all risks can be identified at once. New risks may appear as the project progresses. Continuous identification is important.

Conclusion : Risk identification is the first and most important step in risk management. A complete and early identification of risks helps in proper planning, risk analysis and taking effective actions.

RMMM PLAN

RMMM Plan stands for Risk Mitigation, Monitoring and Management Plan.

1. Introduction :

RMMM Plan is a documented plan that describes how risks will be mitigated, monitored and managed throughout the project.

2. Objectives :

- To define strategies to mitigate identified risks.
- To monitor risks continuously.
- To manage risks effectively to minimize negative impact.
- To ensure project objectives are achieved.

3. Components of RMMM Plan :

1. Risk Mitigation Plan
2. Risk Monitoring Plan
3. Risk Management Plan
4. Reporting Plan

4. RMMM Plan Process :

1. Identify Risks
↓
2. Analyze & Prioritize Risks
↓
3. Plan Mitigation Strategies
↓
4. Monitor, Manage & Review

5. RMMM Plan Details :

Plan Component	Description	Activities	Tools / Techniques	Responsible	Frequency	Outputs
1. Risk Mitigation Plan	Defines actions to reduce the probability or impact of risks.	• Identify mitigation options • Evaluate and select actions • Implement mitigation actions	• Risk mitigation strategies • Cost-benefit analysis • Expert judgment	Risk Owner / Project Manager	As per risk priority	Mitigation action plan, updated risk register
2. Risk Monitoring Plan	Defines how risks will be tracked and monitored.	• Monitor risk indicators • Track risk status • Detect new risks	• Risk dashboards • Key risk indicators (KRI) • Review meetings	Risk Monitor / Project Team	Regularly (Weekly / Monthly)	Risk status reports, issue logs
3. Risk Management Plan	Defines actions to manage risks when they occur.	• Execute response plan • Escalate if required • Reassess and adapt	• Risk response strategies (Avoid, Mitigate, Transfer, Accept) • Contingency plans	Project Manager / Risk Owner	When risk is triggered	Action taken records, updated risk status
4. Reporting Plan	Defines how risk information will be communicated.	• Prepare risk reports • Communicate to stakeholders • Document lessons learned	• Risk reports • Meetings & presentations • Documentation	Project Manager / Risk Monitor	Periodic (Monthly / Milestone)	Risk reports, presentations, lessons learned

6. RMMM Plan Template (Example) :

Risk ID	Risk Description	Category	Probability (L/M/H)	Impact (L/M/H)	Risk Level (L/M/H)	Mitigation Strategy	Monitoring Method	Contingency Plan	Owner	Review Frequency	Status
R-01	Delay in requirement finalization	Schedule	High	High	High	Early stakeholder meetings, clear communication	Weekly progress review	Allocate extra buffer time	Project Manager	Weekly	Open
R-02	Resource unavailability	Resource	Medium	High	High	Resource planning, cross-training	Resource utilization report	Use alternate resources	Resource Manager	Monthly	Open

7. Benefits of RMMM Plan :

- Provides a structured approach to handle risks.
- Reduces uncertainty and surprises.
- Improves decision making and project control.
- Enhances communication with stakeholders.
- Increases the chances of project success.

Conclusion : RMMM Plan helps in proactively managing risks by mitigating them, monitoring their status and managing them effectively. It ensures that potential threats are controlled and project objectives are achieved successfully.

Note : RMMM Plan should be a living document and updated regularly as the project progresses.

Design Engineering :

Design process and Design quality.

1. Design Process :

Design process is a systematic approach to convert requirements into a solution. It provides a blueprint for construction and implementation.

Steps in Design Process :

1. Requirement Analysis → Understand and analyze the requirements of the system. Identify the objectives, constraints and functional needs.
2. Design Specification → Define the overall architecture, data, interfaces and components. Prepare design specifications.
3. Conceptual Design → Develop high-level design concepts and models. Focus on the overall structure of the system.
4. Detailed Design → Refine the design in detail. Specify algorithms, data structures, modules, interfaces and user interface.
5. Design Evaluation → Review and evaluate the design for correctness, completeness and quality. Identify defects or improvements.
6. Design Documentation → Prepare design documents, diagrams, models and specifications for future reference and communication.
7. Design Review → Conduct formal reviews to ensure the design meets requirements and quality standards before implementation.

2. Design Quality :

Design quality refers to how well the design meets the requirements and satisfies the needs of users and stakeholders.

Key Attributes of Good Design Quality :

1. Functionality : The design should fulfill all specified requirements and perform intended functions.
2. Reliability : The design should be reliable and available under expected conditions.
3. Usability : The design should be easy to understand, learn and use by the users.
4. Efficiency : The design should utilize resources (time, memory, CPU, etc.) efficiently.
5. Maintainability : The design should be easy to modify, correct and enhance.
6. Scalability : The design should support future growth and changes in requirements.
7. Security : The design should protect data and resources from unauthorized access and threats.
8. Testability : The design should facilitate easy testing and defect identification.
9. Reusability : The design should promote reuse of components and reduce duplication.
10. Portability : The design should be adaptable to different platforms and environments.

Note : A good design ensures high quality software, reduces rework and cost, improves customer satisfaction and leads to project success.

Introduction to Software Architecture

1. Introduction :

Software Architecture is the fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution. It provides a blueprint for the system and helps to manage complexity.

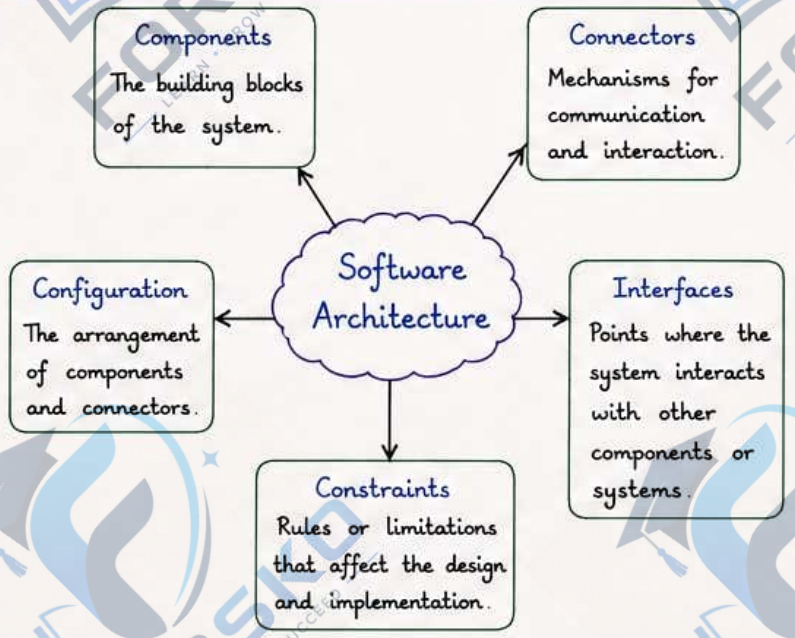
2. Definition :

Software Architecture is the structure or structures of the system, which comprises software elements, the externally visible properties of those elements, and the relationships among them.

3. Purpose / Importance :

- → Provides overall system structure.
- → Helps in managing complexity.
- → Facilitates communication among stakeholders.
- → Supports early analysis of quality attributes.
- → Acts as a basis for design, implementation and testing.
- → Improves maintainability, reusability and scalability.

4. Key Elements of Software Architecture :



5. Types of Software Architecture (Examples) :

- Layered Architecture : System is divided into layers, each layer uses services of the layer below.
- Client-Server Architecture : Clients request services from server.
- Microservices Architecture : System is composed of small, independent services.
- Pipe and Filter Architecture : Data is processed through a series of filters (components).
- Event-Driven Architecture : Components communicate by producing and consuming events.

6. Characteristics of Good Architecture :

- ✓ Modularity : Divides system into manageable parts.
- ✓ Abstraction : Hides unnecessary details.
- ✓ Cohesion : High cohesion within components.
- ✓ Coupling : Low coupling between components.
- ✓ Scalability : Handles growth and change easily.
- ✓ Maintainability : Easy to modify and enhance.
- ✓ Reusability : Components can be reused.
- ✓ Reliability : Works correctly and consistently.

7. Architecture Vs Design :

Aspect	Architecture	Design
Focus	Overall structure and high-level organization.	Detailed solution and implementation.
Level	High-level (Big picture)	Low-level (Details)
Decisions	Major design decisions, frameworks, patterns.	Algorithms, data structures, classes, etc.
Changes	Hard and costly to change.	Relatively easier to change.
Purpose	Defines the skeleton of the system.	Defines the working of each part.

8. Conclusion :

Software Architecture is the foundation of any successful software system. A well-designed architecture ensures that the system meets functional requirements as well as quality attributes, and can evolve with changing needs.

Note : Good architecture leads to systems that are robust, maintainable, scalable and cost-effective throughout their lifecycle.

Data design and Architectural pattern

1. Data Design

Data design is the process of designing the data elements, data structures, database schema and data access mechanisms for a software system. It ensures that the data used in the system is accurate, consistent, organized and easy to access.

Objectives :

- To organize data to meet the requirements of the system.
- To reduce data redundancy and inconsistency.
- To ensure data integrity and security.
- To improve data access efficiency.
- To support future changes and scalability.

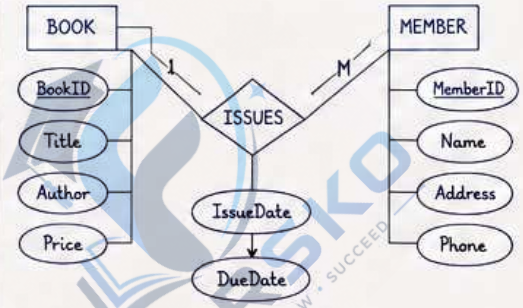
Steps in Data Design :

1. Data Collection → Collect and analyze data requirements from different sources.
2. Data Analysis → Identify data objects, their attributes and relationships.
3. Data Modeling → Create conceptual data model (ER diagram) to represent data and relationships.
4. Data Structuring → Map the conceptual model to logical model (tables, keys, relationships).
5. Database Design → Design schema, constraints, indexes and data access methods.
6. Validation → Check the design for consistency, integrity, performance and security.
7. Implementation → Create database and implement data access mechanisms.

Data Design Activities :

- Data modeling (ER diagram)
- Design of database schema
- Design of data structures
- Design of data dictionary
- Design of data access methods
- Design of database security

ER Diagram Example (Library System) :



Relational Schema Example :

Table Name	Attributes
BOOK	BookID (PK), Title, Author, Price
MEMBER	MemberID (PK), Name, Address, Phone
ISSUES	IssueID (PK), BookID (FK), MemberID (FK), IssueDate, DueDate

2. Architectural Patterns

An architectural pattern is a general, reusable solution to a commonly occurring problem in software architecture. It provides a proven structure for organizing the system.

Need for Architectural Patterns :

→ Provide reusable solutions to common problems.
→ Improve maintainability, scalability and flexibility.
→ Help in effective communication among developers.
→ Reduce design risks and development time.

Common Architectural Patterns :

Pattern	Structure / Diagram	Description	Advantages	Use Cases
1. Layered (n-tier)	Presentation Layer → Business Layer → Data Access Layer → Database Layer	System is divided into layers. Each layer has specific responsibility and communicates with adjacent layers.	• Separation of concerns • Easy maintenance • Modularity • Scalability	• Enterprise applications • Web applications
2. Client-Server	Client ↔ Server, Server → Database	Clients request services from the server. Server processes requests and provides results.	• Centralized data • Easy management • Data security	• Web systems • Distributed applications
3. MVC (Model View Controller)	Controller; View ↔ Model	Separates application into Model (data), View (UI) and Controller (control logic).	• Separation of concerns • Easy testing • Supports parallel development	• Web applications • Desktop applications
4. Microservices	Service, Service, Service (each with database)	System is composed of small, independent services that communicate via APIs.	• Independent deployment • Scalability • Fault isolation • Technology flexibility	• Large scale systems • Cloud based applications
5. Pipe and Filter	Input → Filter 1 → Filter 2 → Filter n → Output	Data is passed through a series of filters. Each filter performs a specific task.	• Reusability • Easy to modify • Suitable for batch processing	• Compilers • Data processing systems

Note : Good data design leads to efficient data storage, fast retrieval, data integrity and easy maintenance of the system.

Conclusion : Data design ensures that data is well-organized and efficient to use, while architectural patterns provide structured and proven solutions for building scalable, maintainable and flexible software systems.

User Interface Design : Golden Rules

User Interface (UI) design is the process of designing the look, feel and interactions of a software application. Good UI makes the system easy to learn, efficient to use and pleasant to experience.

Golden Rules of User Interface Design :

No.	Golden Rule	Explanation	Example
1.	Strive for Consistency	• Similar actions should look and behave in the same way throughout the application.	• Same icons, colors and menu placements used across all pages.
2.	Enable Frequent Users to Use Shortcuts	• Provide shortcuts for frequent users to speeds up interaction.	• Ctrl + S to save, Ctrl + Z to undo.
3.	Offer Informative Feedback	• Always keep users informed about what is happening through appropriate feedback.	• Show progress bars, success messages, or alerts.
4.	Design Dialogs to Yield Closure	• Structure dialogs so that users know when a task is completed successfully.	• After submitting a form, show "Registration Successful" message.
5.	Prevent Errors	• Design the system to prevent problems from occurring in the first place.	• Disable "Submit" button until required fields are filled.
6.	Permit Easy Reversal of Actions	• Allow users to undo or redo actions easily.	• Provide "Undo", "Back" or "Cancel" options.
7.	Support Internal Locus of Control	• Users should feel in control of the system, not the other way around.	• Allow users to navigate freely and make choices.
8.	Reduce Short-Term Memory Load	• Minimize the amount of information users need to remember between interactions.	• Use menus, labels and reminders instead of asking users to remember everything.
9.	Keep it Simple	• Avoid unnecessary complexity. Focus on what is essential.	• Avoid clutter, use clear language and straightforward navigation.
10.	Be Visually Attractive	• Use appropriate colors, fonts, spacing and layouts to create a pleasant and engaging experience.	• Use good color contrast, readable fonts and balanced layouts.

★ **Key Takeaway** : A good user interface is not about making it look beautiful only, but about making it useful, usable and satisfying for the users. Following these golden rules leads to effective and user-friendly software.

Interface Design Steps

1. Identify Users and Goals → Understand who the users are, what their goals are, and what tasks they need to perform.
2. Analyze Tasks → Study the tasks in detail. Identify inputs, outputs, sequences and exceptions.
3. Gather Requirements → Collect all functional and non-functional requirements related to the interface.
4. Develop Design Concepts → Create rough ideas or sketches for the interface. Explore different layouts and navigation options.
5. Design the Interface → Design screen layouts, menus, dialogs, buttons, forms and messages.
6. Evaluate the Design → Review the design with users and experts. Check for usability, consistency and effectiveness.
7. Prototype the Interface → Build a prototype or mockup to simulate the interface and how users will interact with it.
8. Test with Users → Let real users try the interface. Observe their behavior and collect feedback.
9. Refine the Design → Improve the interface based on feedback and test results. Make necessary changes.
10. Implement the Interface → Develop the final interface using appropriate tools and technologies.
11. Review and Maintain → After implementation, review the interface periodically and make updates to meet changing user needs.


1010101010
0101010101
1010101010
0101010101
Unit - IV
10110
01001
10101
DATA STRUCTURES
ALGORITHMS
COMPUTER NETWORKS
DATABASE SYSTEMS
#include <iostream>
using namespace std;
int main()
{
cout << "Hello, World!";
return 0;
}

Overview of System Models

1. Introduction :

A system model is an abstract representation of a system. It helps us understand, analyze, design and communicate the structure, behavior and views of a system.

2. Purpose of System Models :

→ To visualize the system.

→ To specify the structure and behavior.

→ To manage complexity.

→ To help in communication among stakeholders.

→ To guide system design, development, testing and maintenance.

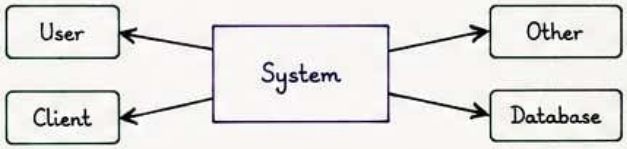
3. Characteristics of Good System Models :

1. Simplicity	:	Easy to understand.
2. Completeness	:	Covers all important aspects.
3. Consistency	:	No conflicting information.
4. Accuracy	:	Correctly represents the system.
5. Abstraction	:	Shows essential details, hides unnecessary ones.
6. Modularity	:	Divides system into manageable parts.

4. Types (Views) of System Models :

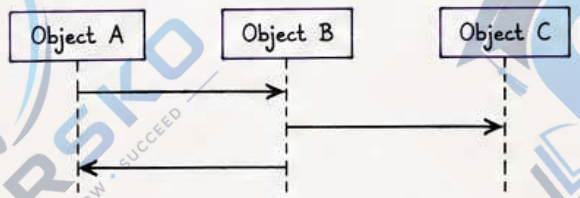
1. Context Model (External View)

→ Shows the system as a whole and its relationship with external entities.



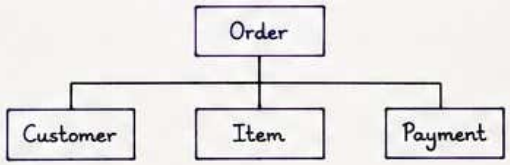
2. Interaction Model

→ Shows how objects or components interact with each other.



3. Structural Model (Static Model)

→ Shows the static structure of the system: classes, objects, data and their relationships.



4. Behavioral Model (Dynamic Model)

→ Shows how the system behaves over time. Includes states, activities and events.

5. Data Model

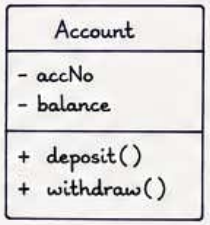
→ Represents data structure, data relationships, constraints and operations on data.

6. Functional Model

→ Shows the functions or transformations performed by the system.

7. Object Model

→ Describes the objects in the system, their attributes, operations and relationships.



5. Conclusion :

System models provide different perspectives of a system. Using these models, we can better understand the system, reduce complexity, improve communication and build high quality systems.

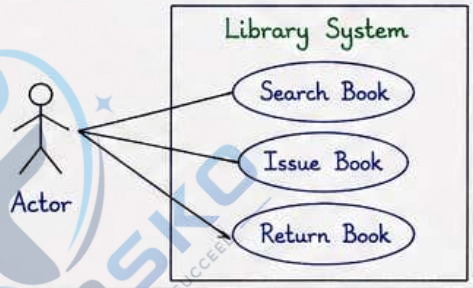
★ **Note :** No single model is complete by itself. A combination of different models gives a better understanding of the system.

UML Diagrams

UML (Unified Modeling Language) is a standard visual language used to design, visualize, construct and document the software systems.

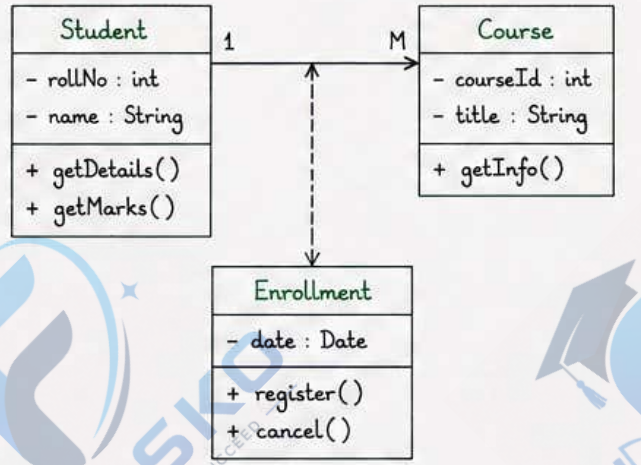
1. Use Case Diagram

- Shows the functional requirements of the system.
- It represents the interaction between actors and use cases.



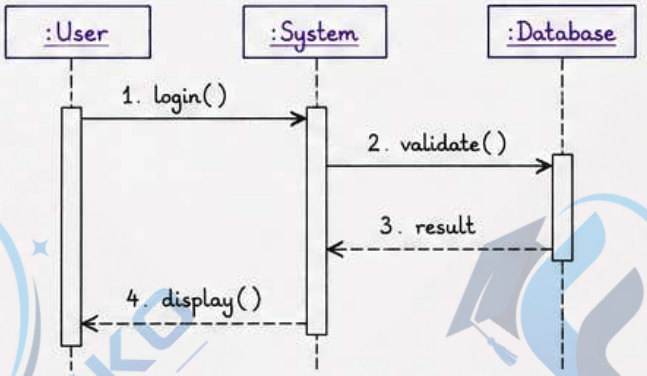
2. Class Diagram

- Shows the static structure of the system.
- It represents classes, their attributes, operations and relationships.



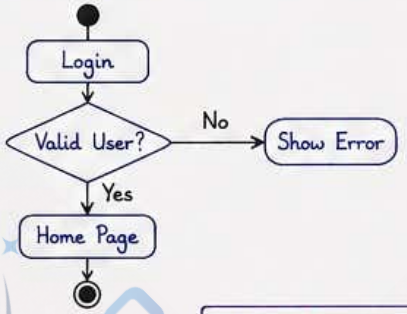
3. Sequence Diagram

- Shows the interaction between objects in a time sequence.
- It represents the order of messages exchanged between objects.



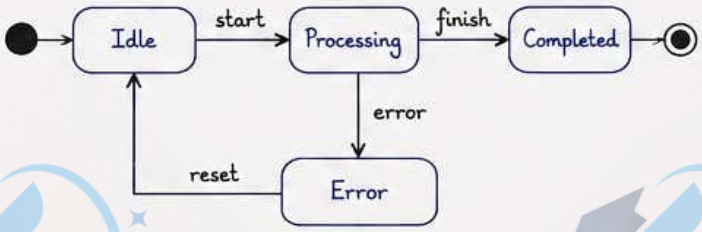
4. Activity Diagram

- Shows the flow of activities or control flow in a process.
- It represents the workflow from one activity to another.



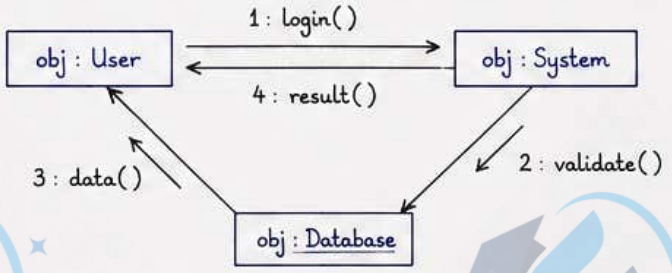
5. State Machine Diagram

- Shows the different states of an object and the transitions between states.
- It represents the behavior of an object in response to events.



6. Collaboration Diagram

- Shows the interaction between objects and their links.
- It emphasizes the organization of objects rather than time sequence.



★ Note : These UML diagrams work together to provide a complete view of the system from different perspectives. They help in better analysis, design and communication.

Testing Strategies : A Strategic Approach to Software Testing

1. Introduction :

Testing is an important activity in the Software Development Life Cycle (SDLC). A testing strategy is a high-level plan that defines the approach, objectives, resources and schedule of testing activities. It helps in ensuring quality software by identifying defects early and reducing risks.

2. Objectives of Testing Strategy :

- To ensure the quality of software.
- To minimize defects and risks.
- To ensure early detection of defects.
- To optimize time, cost and resources.
- To increase customer satisfaction.

3. Key Elements of Testing Strategy :

→ Scope of Testing
→ Test Objectives
→ Test Approach
→ Test Levels
→ Types of Testing
→ Test Environment,
→ Test Tools
→ Entry and Exit Criteria
→ Resources and Responsibilities
→ Schedule and Estimation
→ Risk Management,
→ Metrics and Reporting

4. Levels of Testing :

1. Unit Testing – Tests individual units or components.
2. Integration Testing – Tests the interface between integrated units.
3. System Testing – Tests the complete system as per requirements.
4. Acceptance Testing – Validates the system from user's perspective.

5. Types of Testing :

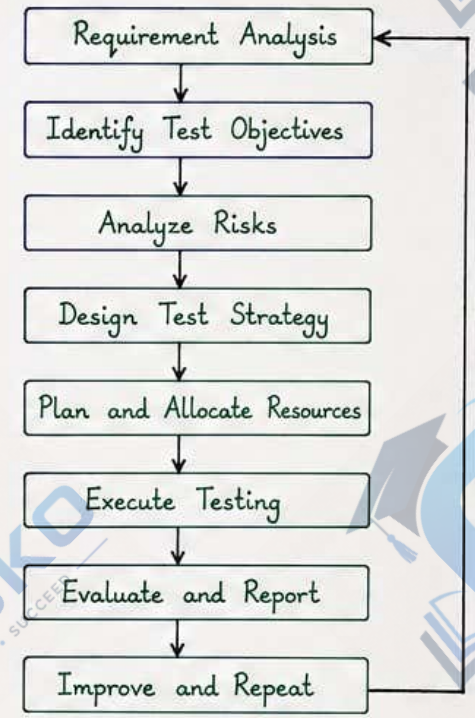
- Functional Testing
- Non-Functional Testing
- Performance Testing
- Security Testing
- Usability Testing
- Compatibility Testing
- Regression Testing
- Smoke Testing
- Sanity Testing
- Recovery Testing

6. Test Approach :

The test approach defines the methods and techniques used for testing. It may include:

1. Black Box Testing – Focuses on functionality.
2. White Box Testing – Focuses on internal code structure.
3. Grey Box Testing – Combination of both.

7. Testing Strategy Process :



8. Benefits of a Good Testing Strategy :

- Provides clear direction to the testing team.
- Ensures effective use of resources.
- Helps in early defect detection.
- Improves product quality and reliability.
- Increases customer confidence and satisfaction.

Note : A good testing strategy is the foundation of successful testing. It helps organizations deliver high quality software within time and budget.

Test Strategies for Conventional Software

1. Introduction :

Conventional software is developed using traditional programming techniques and follows well-defined requirements. Testing strategies for such software aim to ensure quality, reliability, performance and user satisfaction.

2. Objectives of Test Strategies :

- To find defects as early as possible.
- To ensure that the software meets the specified requirements.
- To improve software quality and reliability.
- To reduce development and maintenance costs.
- To deliver a user-friendly and bug-free product.

3. Key Test Strategies :

(1) Requirement-Based Testing : Based on the requirements and specifications.

(2) Risk-Based Testing : Focus on the high-risk areas and critical functionalities.

(3) Defect-Based Testing : Based on past defects to prevent similar defects.

(4) Priority-Based Testing : Focus on important modules or features first.

(5) Change-Based Testing : After any change in requirements or code.

4. Types of Testing Strategies :

1. Unit Testing – Tests individual units or components.
2. Integration Testing – Tests interaction between integrated units.
3. System Testing – Tests the complete system as per requirements.
4. Acceptance Testing – Validates the software from user's point of view.
5. Regression Testing – Ensures that existing functions work after changes.
6. Performance Testing – Checks speed, load, stress and scalability.
7. Security Testing – Ensures protection of data and resources.
8. Usability Testing – Checks ease of use and user interface.

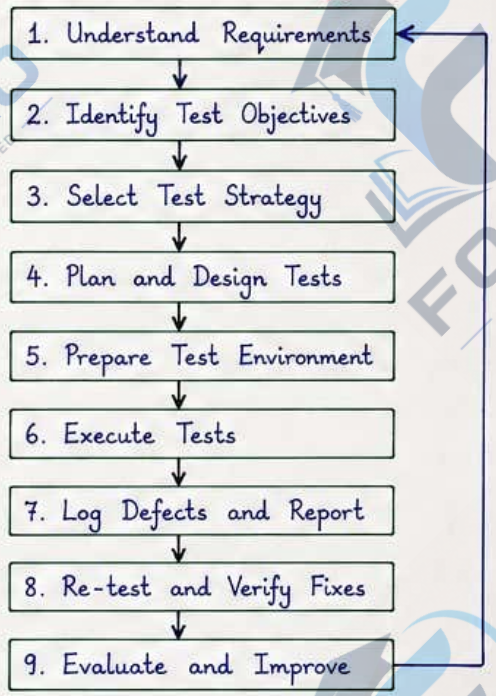
5. Test Levels and Strategies :

Test Level	Test Strategy
Unit Testing	White Box, Code coverage, Boundary value
Integration Testing	Top-Down, Bottom-Up, Sandwich
System Testing	Black Box, Functional, Performance, Security, Usability
Acceptance Testing	Alpha Testing, Beta Testing, User Acceptance Testing (UAT)

6. Test Techniques Used :

- Black Box Testing
- White Box Testing
- Grey Box Testing
- Boundary Value Analysis
- Equivalence Partitioning
- Decision Table Testing
- State Transition Testing
- Cause-Effect Graphing
- Exploratory Testing

7. General Test Strategy Process :



8. Conclusion :

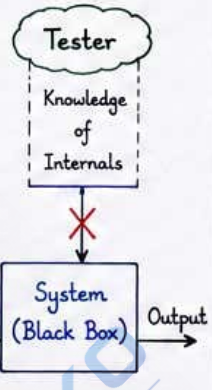
A well-planned test strategy is essential for the success of conventional software projects. It ensures the delivery of high-quality software within time and budget.

Black-Box and White-Box Testing

Two fundamental approaches to software testing.

1. BLACK-BOX TESTING

Black-box testing is a technique of testing the functionality of an application without knowing its internal code or structure. Tester focuses on input and output.



Key Points :

- Tests the external functionality of the system.
- No knowledge of internal code, design or structure is required.
- Focuses on what the system does.
- Also called Behavioral or Functional testing.

Characteristics :

- Based on requirements and specifications.
- Test cases are derived from user perspective.
- Independent of programming language.
- Helps to validate the system against business requirements.

Techniques Used :

Equivalence Partitioning, Boundary Value Analysis, Decision Table Testing, State Transition Testing, Use Case Testing, Exploratory Testing, etc.

Advantages :

- Simple to understand and execute.
- Can be done early in SDLC.
- Tester and developer can work in parallel.
- Validates system from user's perspective.

Disadvantages :

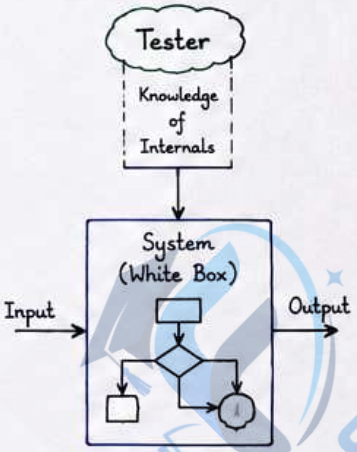
- Limited to functionality.
- Cannot test internal code, structure or hidden paths.
- Less effective in finding internal errors.

Examples :

- Testing login with valid/invalid credentials.
- Testing registration form, search functionality.
- Testing the system as per user requirements.

2. WHITE-BOX TESTING

White-box testing is a technique of testing the internal code, logic and structure of an application. Tester has full knowledge of the code.



Key Points :

- Tests the internal structure and logic of the system.
- Requires knowledge of programming language and code.
- Focuses on how the system does it.
- Also called Structural, Glass-box or Clear-box testing.

Characteristics :

- Based on code structure, logic and design.
- Test cases are derived from the internal code.
- Can test all paths, loops and branches.
- Helps to find hidden errors and improve code quality.

Techniques Used :

Statement Coverage, Branch Coverage, Path Coverage, Condition Coverage, Loop Testing, Data Flow Testing, Control Flow Testing, etc.

Advantages :

- High test coverage and thorough testing.
- Finds hidden errors and security issues.
- Helps to improve code quality.
- Effective for complex logic and calculations.

Disadvantages :

- Requires programming knowledge.
- Time consuming and costly.
- Cannot be done early in SDLC.
- Not feasible for large systems.

Examples :

- Testing internal logic of if-else, loops, conditions.
- Testing individual functions, methods and classes.
- Checking code coverage, paths and branches.

Conclusion : Black-box testing ensures that the system works as per requirements, while White-box testing ensures that the system works correctly from the inside. Both are complementary and essential for quality software.

System Testing

System testing is a level of software testing in which the complete and integrated software is tested as a whole. The main objective is to evaluate the system's compliance with the specified requirements.

1. **Objective :**
   - To test the complete integrated system.
   - To verify whether the system meets the specified requirements.
   - To identify defects in the system.
   - To ensure the system is ready for delivery to the user.
2. **Scope :**
   - It includes testing of the entire system including all modules, interfaces, databases, and external systems.
   - Both functional and non-functional requirements are validated.
3. **Performed by :** Independent Testers / QA Team
4. **Entry Criteria :**
   - All integration testing is completed.
   - The system is stable and ready for system testing.
5. **Types of System Testing :**
   1. Functional Testing
   2. Performance Testing
   3. Security Testing
   4. Usability Testing
   5. Compatibility Testing
   6. Recovery Testing
   7. Installation Testing
   8. Reliability Testing
6. **Outcome :**
   - It provides confidence that the system meets the business requirements and is ready for deployment.
   - It helps in identifying any remaining defects before release.

The Art of Debugging

Debugging is the process of finding, analyzing and removing errors (bugs) from a program so that it works as expected.

* Why Debugging is Important ?

- Even small errors can cause program failure.
- It improves program reliability and performance.
- Helps in delivering quality software.
- Saves time and cost in the long run.

* Common Types of Errors (Bugs) :

①	Syntax Errors	– Violations of programming language rules.
②	Logical Errors	– Faulty logic, program runs but gives wrong output.
③	Runtime Errors	– Occur during program execution (e.g., divide by zero).
④	Semantic Errors	– Meaning of code is incorrect.

* Debugging Process / Steps :

①	Reproduce the Error	– Run the program and make the error occur.
②	Identify the Error	– Locate the line or part causing the problem.
③	Analyze the Error	– Understand why the error is occurring.
④	Fix the Error	– Correct the code.
⑤	Test the Fix	– Run the program again to check.
⑥	Prevent Future Errors	– Improve code and add checks to avoid similar bugs.

Golden Rule :
"Fix the cause, not just the symptom."

* Debugging Techniques / Tools :

- Print Statements (Tracing)
- Debugger (Step by Step Execution)
- Breakpoints
- Code Review / Peer Review
- Logging
- Unit Testing
- Rubber Duck Debugging (Explain the problem to a rubber duck!)

* Conclusion :

Debugging is not just about finding errors, it is about understanding the problem deeply and writing better, error-free code.