SUDDHANANDA SCHOOL OF MANAGEMENT & COMPUTER SCIENCE

LRECTURE NOTES ON

SOFTWARE ENGINEERING(MCPC1006)

UNIT – I

INTRODUCTION TO SOFTWARE ENGINEERING

Software Engineering is the systematic, disciplined, and organized approach to the development, operation, maintenance, and testing of software.

The term Software Engineering was introduced to overcome the software crisis and to ensure that software products are developed with high quality, within budget, and on time.

According to IEEE:

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

WHAT IS SOFTWARE?

Software is a collection of programs, procedures, documentation, and associated data designed to perform specific tasks on a computer system.

Examples:

- Windows Operating System
- Microsoft Word
- Google Chrome
- WhatsApp
- Banking Software

CHARACTERISTICS OF SOFTWARE

1. Software is developed, not manufactured.
2. Software does not wear out like hardware.
3. Software can be modified and upgraded.
4. Software is intangible.
5. Software maintenance is costly.

NEED FOR SOFTWARE ENGINEERING

As software systems became larger and more complex, traditional programming methods became insufficient.

Software Engineering is needed because:

1. To reduce development cost.
2. To improve software quality.
3. To manage large projects.

4. To ensure timely delivery.
5. To increase customer satisfaction.

SOFTWARE CRISIS

During the 1960s and 1970s, software projects faced serious problems such as:

- Cost overruns
- Delayed delivery
- Poor quality software
- Maintenance difficulties

These issues collectively became known as the **Software Crisis**.

CAUSES OF SOFTWARE CRISIS

1. Increasing software complexity.
2. Poor project management.
3. Lack of development standards.
4. Inadequate testing.
5. Unclear requirements.

SOLUTIONS TO SOFTWARE CRISIS

1. Use Software Engineering principles.
2. Follow software development methodologies.
3. Proper planning and documentation.
4. Quality assurance and testing.
5. Effective project management.

SOFTWARE ENGINEERING LAYERS

Software Engineering is often represented as a layered technology.

Quality Focus

↑

Process

↑

Methods

↑

Tools

1. Quality Focus

Ensures software quality throughout development.

2. Process

Defines framework activities for software development.

3. Methods

Provide technical approaches for software construction.

4. Tools

Automated tools that support development activities.

Example:

- Eclipse
- NetBeans
- Visual Studio

SOFTWARE PROCESS

A software process is a structured set of activities required to develop software.

Main Activities:

1. Requirement Analysis
2. Design
3. Coding
4. Testing
5. Maintenance

SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

SDLC is a framework that describes the stages involved in software development.

PHASES OF SDLC

1. Requirement Analysis

Requirements are collected from customers.

Activities:

- Understanding user needs
- Feasibility study
- Requirement gathering

Output:

- SRS Document

2. System Design

Design of software architecture is prepared.

Activities:

- Database design
- Interface design
- Module design

3. Coding

Programmers convert design into source code.

Languages:

- Java
- Python
- C++
- PHP

4. Testing

Software is tested to identify defects.

Objectives:

- Detect errors
- Verify functionality
- Improve quality

5. Deployment

Software is delivered to users.

6. Maintenance

Software is modified after deployment.

Examples:

- Bug fixing
- Performance improvement
- Feature enhancement

SDLC DIAGRAM

Requirement Analysis

↓

System Design

↓

Coding

↓

Testing

↓

Deployment

↓

Maintenance

SOFTWARE PROCESS MODELS

A Software Process Model is a strategy used for software development.

Popular Models:

1. Waterfall Model
2. Prototype Model
3. Spiral Model
4. Incremental Model
5. Agile Model

WATERFALL MODEL

The Waterfall Model is the oldest software development model.

Development proceeds sequentially from one phase to another.

WATERFALL DIAGRAM

Requirements

↓

Design

↓

Coding

↓

Testing

↓

Maintenance

ADVANTAGES OF WATERFALL MODEL

1. Easy to understand.
2. Simple implementation.
3. Good documentation.
4. Suitable for small projects.

DISADVANTAGES OF WATERFALL MODEL

1. Difficult to accommodate changes.
2. Testing occurs late.
3. Not suitable for large projects.

PROTOTYPE MODEL

A prototype is an early version of software developed to understand requirements.

PHASES OF PROTOTYPE MODEL

1. Gather requirements
2. Build prototype
3. Customer evaluation
4. Refinement

5. Final product

ADVANTAGES

1. Better requirement understanding.
2. Improved customer interaction.
3. Reduced development risk.

DISADVANTAGES

1. Increased development cost.
2. Time-consuming modifications.

SPIRAL MODEL

Developed by Barry Boehm.

Combines:

- Waterfall Model
- Prototyping

Focuses on risk analysis.

PHASES OF SPIRAL MODEL

1. Planning
2. Risk Analysis
3. Engineering
4. Customer Evaluation

ADVANTAGES

1. Risk management.
2. Suitable for large projects.
3. Flexible development.

DISADVANTAGES

1. Expensive.
2. Complex management.

INCREMENTAL MODEL

Software is developed in small increments.

Each increment adds new functionality.

ADVANTAGES

1. Early delivery.
2. Easy testing.
3. Better customer feedback.

AGILE MODEL

Agile is a modern software development methodology.

Development occurs in small iterations.

AGILE PRINCIPLES

1. Customer collaboration.
2. Continuous delivery.
3. Frequent feedback.
4. Adaptability to change.

ADVANTAGES

1. Fast development.
2. Customer involvement.
3. High flexibility.

FEASIBILITY STUDY

A feasibility study determines whether a project is practical and economically viable.

TYPES OF FEASIBILITY

1. Technical Feasibility

Checks technological requirements.

Questions:

- Is technology available?
- Are resources sufficient?

2. Economic Feasibility

Checks financial viability.

Questions:

- Is project profitable?
- Is cost justified?

3. Operational Feasibility

Checks user acceptance.

Questions:

- Will users use the system?
- Is training required?

4. Legal Feasibility

Checks legal constraints.

Questions:

- Does project comply with laws?

5. Schedule Feasibility

Checks project completion timeline.

SOFTWARE QUALITY

Software quality refers to the degree to which software satisfies specified requirements.

QUALITY FACTORS

1. Reliability
2. Efficiency
3. Maintainability
4. Portability
5. Usability
6. Security

SOFTWARE MAINTENANCE

Maintenance refers to modifications made after software deployment.

TYPES OF MAINTENANCE

1. Corrective Maintenance

Fixes software errors.

Example:

- Bug removal

2. Adaptive Maintenance

Adjusts software to new environments.

Example:

- OS upgrades

3. Perfective Maintenance

Enhances functionality.

Example:

- New feature addition

4. Preventive Maintenance

Improves future maintainability.

Example:

- Code optimization

SOFTWARE ENGINEERING ETHICS

Software engineers should:

1. Maintain honesty.
2. Protect user privacy.
3. Deliver quality software.
4. Respect intellectual property.

5. Follow professional standards.

APPLICATIONS OF SOFTWARE ENGINEERING

1. Banking Systems
2. Hospital Management Systems
3. E-Commerce Applications
4. Mobile Applications
5. Airline Reservation Systems
6. Government Portals

ADVANTAGES OF SOFTWARE ENGINEERING

1. Better software quality.
2. Reduced development cost.
3. Improved project management.
4. Easier maintenance.
5. Customer satisfaction.

DISADVANTAGES

1. Initial development cost may be high.
2. Requires skilled professionals.
3. Extensive documentation needed.

UNIT – II

SOFTWARE REQUIREMENT ENGINEERING (SRE):

INTRODUCTION TO REQUIREMENT ENGINEERING

Requirement Engineering (RE) is the process of identifying, analyzing, documenting, validating, and managing the requirements of a software system.

It is one of the most important phases of Software Development Life Cycle (SDLC) because the success of a software project depends largely on understanding customer requirements correctly.

A requirement specifies what the system should do and the constraints under which it should operate.

DEFINITION OF REQUIREMENT ENGINEERING

Requirement Engineering is a systematic approach used to discover, document, analyze, validate, and maintain software requirements.

Its primary objective is to ensure that developers and customers have a clear understanding of the system to be developed.

IMPORTANCE OF REQUIREMENT ENGINEERING

1. Helps understand customer needs.
2. Reduces software development cost.
3. Minimizes project risks.
4. Improves software quality.
5. Reduces future modifications.

6. Ensures customer satisfaction.

REQUIREMENT ENGINEERING PROCESS

The Requirement Engineering Process consists of the following activities:

Requirement Gathering

↓

Requirement Analysis

↓

Requirement Specification

↓

Requirement Validation

↓

Requirement Management

REQUIREMENT GATHERING

Requirement gathering is the first activity in Requirement Engineering.

During this phase, information is collected from:

- Customers
- End Users
- Stakeholders
- Domain Experts

The goal is to understand what the customer expects from the software.

STAKEHOLDERS

Stakeholders are individuals or organizations affected by the software system.

Examples:

1. Customers
2. Users
3. Developers
4. Managers
5. Government Agencies

TYPES OF REQUIREMENTS

Software requirements are mainly classified into:

1. Functional Requirements
2. Non-Functional Requirements

FUNCTIONAL REQUIREMENTS

Functional requirements describe what the system should do.

They specify services and functionalities provided by the system.

Examples

For Online Banking System:

- User Login
- Money Transfer
- Balance Inquiry
- Account Creation
- Mini Statement

CHARACTERISTICS OF FUNCTIONAL REQUIREMENTS

1. Specific
2. Measurable
3. Testable
4. User-Oriented

NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements specify quality attributes and system constraints.

They describe how the system should perform.

Examples

1. Security
2. Reliability
3. Performance
4. Portability
5. Scalability

Example

A banking application should process transactions within 2 seconds.

This is a non-functional requirement because it specifies performance.

DIFFERENCE BETWEEN FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

Functional Requirement	**Non-Functional Requirement**
Describes what system does	Describes how system works

Functional Requirement	Non-Functional Requirement
Feature-oriented	Quality-oriented
Easy to identify	Difficult to measure
Example: Login System	Example: Security

REQUIREMENT ELICITATION

Requirement elicitation is the process of collecting requirements from stakeholders.

It is one of the most important activities because incorrect requirements may lead to project failure.

REQUIREMENT ELICITATION TECHNIQUES

1. Interview
2. Questionnaire
3. Observation
4. Brainstorming
5. Prototyping
6. JAD (Joint Application Development)

1. INTERVIEW

Direct interaction between analyst and customer.

Advantages

1. Detailed information
2. Clarifies doubts
3. Better communication

Disadvantages

1. Time-consuming
2. Requires skilled interviewer

2. QUESTIONNAIRE

A set of written questions distributed to users.

Advantages

1. Cost effective
2. Large audience coverage

Disadvantages

1. Limited interaction
2. Incomplete responses

3. OBSERVATION

Analyst observes users while performing tasks.

Advantages

1. Real-world understanding
2. Accurate information

Disadvantages

1. Time consuming
2. Observer bias

4. BRAINSTORMING

Group discussion to generate ideas.

Advantages

1. Creative solutions
2. Multiple viewpoints

5. PROTOTYPING

A preliminary version of software is developed.

Users provide feedback.

Advantages

1. Better understanding
2. Reduced misunderstandings

REQUIREMENT ANALYSIS

Requirement Analysis is the process of examining and organizing collected requirements.

The analyst studies requirements to:

- Identify conflicts
- Remove ambiguities
- Check completeness
- Verify feasibility

OBJECTIVES OF REQUIREMENT ANALYSIS

1. Understand user needs.
2. Detect inconsistencies.
3. Resolve conflicts.
4. Prioritize requirements.

ACTIVITIES OF REQUIREMENT ANALYSIS

1. Requirement Classification
2. Requirement Prioritization
3. Requirement Modeling
4. Conflict Resolution

REQUIREMENT MODELING

Requirement Modeling provides a graphical representation of software requirements.

It helps developers understand the system clearly.

TYPES OF REQUIREMENT MODELS

1. Data Flow Diagram (DFD)
2. Entity Relationship Diagram (ERD)
3. Use Case Diagram
4. State Transition Diagram

DATA FLOW DIAGRAM (DFD)

DFD represents the flow of information through a system.

Components:

1. Process
2. Data Flow
3. Data Store
4. External Entity

DFD SYMBOLS

Symbol	**Meaning**
Circle	Process
Arrow	Data Flow
Rectangle	External Entity
Open Rectangle	Data Store

EXAMPLE OF DFD

Student Registration System

Student

↓

Registration Process

↓

Database

ENTITY RELATIONSHIP DIAGRAM (ERD)

ERD represents relationships among data entities.

Components:

1. Entity
2. Attribute
3. Relationship

EXAMPLE

Student

|

Enrolls

|

Course

USE CASE DIAGRAM

Use Case Diagram describes interactions between users and system.

Main Elements:

1. Actor
2. Use Case
3. Relationship

Example

ATM System

Actor:

- Customer

Use Cases:

- Withdraw Cash
- Deposit Money
- Check Balance

SOFTWARE REQUIREMENT SPECIFICATION (SRS)

SRS stands for:

Software Requirement Specification

It is a formal document that describes all software requirements.

It serves as an agreement between customer and developer.

OBJECTIVES OF SRS

1. Define software requirements.
2. Provide project foundation.
3. Facilitate communication.
4. Reduce misunderstandings.

CONTENTS OF SRS DOCUMENT

1. Introduction
2. Overall Description
3. Functional Requirements
4. Non-Functional Requirements
5. System Interfaces
6. Constraints

CHARACTERISTICS OF GOOD SRS

A good SRS should be:

1. Correct
2. Complete
3. Consistent
4. Verifiable
5. Modifiable
6. Traceable

BENEFITS OF SRS

1. Better communication.
2. Easy maintenance.
3. Reduced development cost.
4. Improved quality.

REQUIREMENT VALIDATION

Requirement Validation ensures that requirements accurately represent customer needs.

Purpose:

- Detect errors before development starts.

VALIDATION TECHNIQUES

1. Requirement Review
2. Prototyping
3. Test Case Generation
4. Automated Analysis

REQUIREMENT REVIEW

Requirements are examined by stakeholders.

Objectives:

1. Detect errors
2. Identify missing requirements
3. Verify correctness

REQUIREMENT MANAGEMENT

Requirements may change during development.

Requirement Management controls and tracks these changes.

OBJECTIVES OF REQUIREMENT MANAGEMENT

1. Track requirement changes.
2. Maintain consistency.
3. Improve project control.
4. Reduce project risks.

REQUIREMENT CHANGE MANAGEMENT PROCESS

Change Request

↓

Impact Analysis

↓

Approval

↓

Implementation

↓

Documentation Update

CHALLENGES IN REQUIREMENT ENGINEERING

1. Changing requirements
2. Communication gaps
3. Ambiguous requirements
4. Incomplete information

5. Stakeholder conflicts

REAL-LIFE EXAMPLE

Online Shopping System

Functional Requirements:

- User Registration
- Login
- Product Search
- Add to Cart
- Online Payment

Non-Functional Requirements:

- Response time less than 3 seconds
- Secure transactions
- 99% system availability

ADVANTAGES OF REQUIREMENT ENGINEERING

1. Better understanding of project goals.
2. Improved software quality.
3. Reduced development cost.
4. Lower maintenance effort.
5. Increased customer satisfaction.

DISADVANTAGES

1. Time-consuming process.
2. Requires experienced analysts.
3. Frequent changes increase complexity.

UNIT – III

SOFTWARE DESIGN AND DESIGN ENGINEERING:

INTRODUCTION TO SOFTWARE DESIGN

Software Design is the process of transforming software requirements into a blueprint for software construction.

After the Requirement Analysis phase, developers prepare the design of the system before actual coding begins.

Software Design acts as a bridge between:

Requirement Analysis → Coding

It provides a clear structure for developers to implement the software efficiently.

DEFINITION OF SOFTWARE DESIGN

Software Design is the process of defining the architecture, components, interfaces, and data structures of a software system to satisfy specified requirements.

OBJECTIVES OF SOFTWARE DESIGN

1. Convert requirements into implementation.
2. Improve software quality.
3. Reduce development complexity.
4. Enhance maintainability.
5. Improve system performance.
6. Facilitate testing and debugging.

IMPORTANCE OF SOFTWARE DESIGN

A good software design helps in:

- Easy coding
- Easy maintenance
- Better reliability
- Improved scalability
- Reduced development cost

DESIGN PROCESS

The Software Design Process consists of the following activities:

Requirement Analysis

↓

Architectural Design

↓

Interface Design

↓

Component Design

↓

Database Design

↓

Implementation

DESIGN PRINCIPLES

Good software design should follow certain principles.

1. Modularity

The system should be divided into smaller modules.

Advantages:

- Easy maintenance
- Better understanding
- Independent development

2. Abstraction

Abstraction means showing essential features while hiding unnecessary details.

Example:

ATM Machine

Users see:

- Withdraw Money
- Deposit Money

Users do not see internal processing.

3. Information Hiding

Each module hides its internal implementation.

Benefits:

- Security
- Reduced complexity
- Better maintainability

4. Functional Independence

Each module should perform a single task independently.

Functional independence is achieved using:

1. High Cohesion
2. Low Coupling

SOFTWARE DESIGN CONCEPTS

Important design concepts include:

1. Abstraction
2. Refinement

3. Modularity
4. Architecture
5. Information Hiding
6. Functional Independence

REFINEMENT

Refinement means breaking a large problem into smaller manageable components.

Example:

Online Shopping System

Main Function:

- Order Processing

Refined into:

- Product Selection
- Payment Processing
- Order Confirmation
- Delivery Tracking

SOFTWARE ARCHITECTURE

Software Architecture refers to the overall structure of a software system.

It defines:

- Components
- Relationships
- Interactions

Architecture serves as the foundation of software development.

IMPORTANCE OF SOFTWARE ARCHITECTURE

1. Better system organization.
2. Easier maintenance.
3. Improved performance.
4. Scalability.
5. Reusability.

ARCHITECTURAL DESIGN

Architectural Design identifies major system components and their interactions.

TYPES OF SOFTWARE ARCHITECTURE

1. Data-Centered Architecture

Centralized database shared among components.

Example:

Banking System

Users

↓

Database

↑

Applications

Advantages

1. Easy data sharing.
2. Centralized control.

Disadvantages

1. Database failure affects entire system.

2. Data Flow Architecture

Data moves through a series of processing steps.

Example:

Compiler

Input

↓

Lexical Analysis

↓

Syntax Analysis

↓

Code Generation

Advantages

1. Easy testing.
2. Simple implementation.

3. Client-Server Architecture

One or more clients communicate with a server.

Examples:

- Web Applications
- Online Banking
- Email Systems

Structure

Client

↓

Server

↓

Database

Advantages

1. Centralized management.
2. Improved security.

Disadvantages

1. Server failure affects clients.

4. Layered Architecture

Software divided into layers.

Example:

OSI Model

Application Layer

Presentation Layer

Session Layer

Transport Layer

Network Layer

Data Link Layer

Physical Layer

Advantages

1. Easy maintenance.
2. Better modularity.

COHESION

Cohesion refers to the degree to which elements within a module are related.

Higher cohesion is always desirable.

TYPES OF COHESION

1. Coincidental Cohesion

Unrelated functions grouped together.

Worst type of cohesion.

2. Logical Cohesion

Functions logically related.

Example:

- Input functions

3. Temporal Cohesion

Activities executed at the same time.

Example:

- System startup tasks

4. Procedural Cohesion

Functions follow a sequence.

5. Communicational Cohesion

Functions use same data.

6. Sequential Cohesion

Output of one function becomes input of another.

7. Functional Cohesion

All activities contribute to a single function.

Best type of cohesion.

COHESION ORDER

Coincidental

↓

Logical

↓

Temporal

↓

Procedural

↓

Communicational

↓

Sequential

↓

Functional

COUPLING

Coupling refers to the degree of interdependence between modules.

Lower coupling is desirable.

TYPES OF COUPLING

1. Content Coupling

One module directly accesses another module.

Worst coupling.

2. Common Coupling

Modules share global data.

3. External Coupling

Modules share external interfaces.

4. Control Coupling

One module controls another.

5. Stamp Coupling

Whole data structure passed unnecessarily.

6. Data Coupling

Only necessary data passed.

Best coupling.

COUPLING ORDER

Content

↓

Common

↓

External

↓

Control

↓

Stamp

↓

Data

DIFFERENCE BETWEEN COHESION AND COUPLING

Cohesion	**Coupling**
Within module	Between modules
Should be High	Should be Low
Improves maintainability	Reduces dependency
Functional Cohesion is best	Data Coupling is best

STRUCTURED DESIGN

Structured Design is a systematic method of designing software using modules.

Main objectives:

1. High cohesion
2. Low coupling
3. Modularity

ADVANTAGES OF STRUCTURED DESIGN

1. Easier testing.
2. Easier debugging.
3. Better maintenance.
4. Improved software quality.

DESIGN DOCUMENTATION

Design documentation describes the software design in detail.

Contents include:

1. Architecture Design
2. Database Design
3. User Interface Design
4. Module Design

USER INTERFACE (UI) DESIGN

User Interface Design focuses on interaction between users and software.

A good interface improves usability and customer satisfaction.

CHARACTERISTICS OF GOOD UI DESIGN

1. Simplicity
2. Consistency
3. User Friendliness
4. Efficiency
5. Reliability

UI DESIGN PRINCIPLES

1. User Familiarity

Interface should be familiar to users.

Example:

- Shopping cart icon in e-commerce websites

2. Consistency

Similar operations should have similar appearance.

3. Minimal Surprise

Users should not face unexpected behavior.

4. Recoverability

Users should recover from errors easily.

Example:

- Undo option

5. User Guidance

Provide help and instructions.

Example:

- Error messages

USER INTERFACE COMPONENTS

1. Menus
2. Buttons
3. Dialog Boxes
4. Forms
5. Navigation Links

USER INTERFACE DESIGN PROCESS

Requirement Analysis

↓

User Analysis

↓

Interface Design

↓

Prototype Creation

↓

Evaluation

DESIGN PATTERNS

A Design Pattern is a reusable solution to commonly occurring software design problems.

ADVANTAGES OF DESIGN PATTERNS

1. Reusability
2. Improved quality
3. Faster development
4. Better maintainability

COMMON DESIGN PATTERNS

1. Singleton Pattern

Only one object created.

Example:

- Database Connection

2. Factory Pattern

Creates objects without exposing creation logic.

3. Observer Pattern

Used for event handling.

Example:

- Notification Systems

SOFTWARE DESIGN QUALITY

A good design should be:

1. Correct
2. Efficient
3. Reusable
4. Maintainable
5. Understandable
6. Scalable

REAL-LIFE EXAMPLE

Online Banking System Design

Modules:

1. User Authentication
2. Account Management
3. Fund Transfer
4. Transaction History
5. Report Generation

Each module should have:

- High Cohesion
- Low Coupling

for better performance and maintainability.

ADVANTAGES OF SOFTWARE DESIGN

1. Reduces coding effort.
2. Improves software quality.
3. Enhances maintainability.
4. Simplifies testing.
5. Supports future expansion.

DISADVANTAGES

1. Time-consuming.
2. Requires experienced designers.
3. Additional documentation effort.

UNIT – IV

SOFTWARE TESTING, VERIFICATION AND VALIDATION:

INTRODUCTION TO SOFTWARE TESTING

Software Testing is the process of evaluating a software system to identify defects, errors, and missing requirements before it is delivered to customers.

The main purpose of testing is to ensure that the software works correctly according to the specified requirements.

Testing helps in:

- Finding defects
- Improving software quality
- Increasing reliability
- Reducing maintenance cost

DEFINITION OF SOFTWARE TESTING

Software Testing is a process of executing a program with the intent of finding errors and verifying that the software meets specified requirements.

OBJECTIVES OF SOFTWARE TESTING

1. Detect defects and errors.
2. Verify software functionality.
3. Validate customer requirements.
4. Improve software quality.
5. Increase user satisfaction.
6. Ensure system reliability.

NEED FOR SOFTWARE TESTING

Without testing:

- Software may contain bugs.
- Security vulnerabilities may exist.
- Customer dissatisfaction may occur.
- Financial losses may result.

Example:

A bug in an online banking system can cause incorrect transactions and huge financial losses.

SOFTWARE TESTING PROCESS

Test Planning

↓

Test Case Design

↓

Test Execution

↓

Defect Reporting

↓

Defect Fixing

↓

Retesting

VERIFICATION AND VALIDATION

Verification and Validation are two important quality assurance activities.

VERIFICATION

Verification ensures that the software is developed correctly according to specifications.

It answers:

"Are we building the product right?"

Verification involves:

- Reviews
- Walkthroughs
- Inspections

CHARACTERISTICS OF VERIFICATION

1. Static process.
2. No code execution.
3. Detects defects early.
4. Reduces development cost.

EXAMPLES OF VERIFICATION

1. Requirement Review
2. Design Review
3. Code Inspection
4. Technical Walkthrough

VALIDATION

Validation ensures that the developed software satisfies customer needs.

It answers:

"Are we building the right product?"

Validation involves actual software execution.

CHARACTERISTICS OF VALIDATION

1. Dynamic process.
2. Requires code execution.
3. Ensures customer satisfaction.
4. Conducted after implementation.

EXAMPLES OF VALIDATION

1. Unit Testing
2. Integration Testing
3. System Testing
4. Acceptance Testing

DIFFERENCE BETWEEN VERIFICATION AND VALIDATION

Verification	**Validation**
Checks correctness of development	Checks correctness of product
Static activity	Dynamic activity
No execution required	Execution required
Performed before coding completion	Performed after development

TESTING STRATEGIES

A testing strategy is a systematic approach used to test software.

Major testing strategies are:

1. Unit Testing

2. Integration Testing
3. Validation Testing
4. System Testing

LEVELS OF TESTING

Unit Testing

↓

Integration Testing

↓

System Testing

↓

Acceptance Testing

UNIT TESTING

Unit Testing tests individual modules or components separately.

Usually performed by developers.

OBJECTIVES OF UNIT TESTING

1. Verify individual modules.
2. Detect coding errors.
3. Improve code quality.

EXAMPLE

Testing a login function separately before integrating it with the system.

ADVANTAGES

1. Easy debugging.
2. Early defect detection.
3. Improves code reliability.

INTEGRATION TESTING

Integration Testing checks interaction between multiple modules.

Purpose:

- Detect interface defects.

EXAMPLE

Testing interaction between:

- Login Module

- Database Module

TYPES OF INTEGRATION TESTING

1. Top-Down Integration Testing

Testing starts from top-level modules.

Uses:

- Stub programs

Advantages

1. Major modules tested early.
2. Easy fault localization.

2. Bottom-Up Integration Testing

Testing starts from lower-level modules.

Uses:

- Driver programs

Advantages

1. Low-level functionality tested first.

3. Sandwich Testing

Combination of:

- Top-Down
- Bottom-Up

SYSTEM TESTING

System Testing evaluates the complete integrated software system.

It verifies that all components work together properly.

OBJECTIVES OF SYSTEM TESTING

1. Verify complete functionality.
2. Check performance.
3. Evaluate reliability.
4. Validate requirements.

TYPES OF SYSTEM TESTING

1. Performance Testing
2. Security Testing
3. Stress Testing

4. Load Testing
5. Recovery Testing

PERFORMANCE TESTING

Measures system speed and responsiveness.

Checks:

- Response time
- Throughput
- Scalability

LOAD TESTING

Tests software under expected workload.

Example:

Checking website performance with 1000 users.

STRESS TESTING

Tests software beyond normal limits.

Purpose:

- Determine breaking point.

SECURITY TESTING

Identifies security vulnerabilities.

Checks:

- Authentication
- Authorization
- Data protection

RECOVERY TESTING

Evaluates system recovery after failure.

Example:

Recovering database after server crash.

ACCEPTANCE TESTING

Acceptance Testing is performed by customers or end users.

Purpose:

- Verify that software meets business requirements.

TYPES OF ACCEPTANCE TESTING

1. Alpha Testing

Conducted at developer's site.

Performed by:

- Developers
- Testers

2. Beta Testing

Conducted at customer's environment.

Performed by:

- Real users

BLACK BOX TESTING

Black Box Testing examines software functionality without knowing internal code structure.

Tester focuses on:

- Inputs
- Outputs

CHARACTERISTICS

1. No knowledge of source code.
2. Requirement-based testing.
3. User-oriented approach.

BLACK BOX TESTING TECHNIQUES

1. Equivalence Partitioning
2. Boundary Value Analysis
3. Decision Table Testing

EQUIVALENCE PARTITIONING

Input data divided into groups.

Example:

Age Range:

18–60

Valid:

- 25

- 40

Invalid:

- 15
- 70

BOUNDARY VALUE ANALYSIS

Testing performed at boundary values.

Example:

Age:

18–60

Test Cases:

- 17
- 18
- 19
- 59
- 60
- 61

ADVANTAGES OF BLACK BOX TESTING

1. No programming knowledge required.
2. User perspective testing.
3. Effective for functional defects.

DISADVANTAGES

1. Limited code coverage.
2. Difficult to identify hidden errors.

WHITE BOX TESTING

White Box Testing examines internal structure and logic of software.

Tester has knowledge of source code.

CHARACTERISTICS

1. Code-based testing.
2. Internal logic verification.
3. Performed by developers.

WHITE BOX TESTING TECHNIQUES

1. Statement Coverage
2. Branch Coverage
3. Path Coverage

4. Condition Coverage

STATEMENT COVERAGE

Ensures every statement executes at least once.

BRANCH COVERAGE

Ensures every decision branch is tested.

PATH COVERAGE

Ensures all possible execution paths are tested.

ADVANTAGES OF WHITE BOX TESTING

1. Thorough testing.
2. High code coverage.
3. Detects logical errors.

DISADVANTAGES

1. Requires programming knowledge.
2. Time-consuming.

DIFFERENCE BETWEEN BLACK BOX AND WHITE BOX TESTING

Black Box Testing	White Box Testing
No code knowledge	Code knowledge required
Tests functionality	Tests logic
Performed by testers	Performed by developers
Requirement-based	Code-based

REGRESSION TESTING

Regression Testing ensures that modifications do not affect existing functionality.

Performed after:

- Bug fixes
- Enhancements
- Updates

ADVANTAGES

1. Prevents new defects.
2. Maintains software stability.

DEBUGGING

Debugging is the process of locating and fixing defects in software.

Testing identifies defects, whereas debugging removes defects.

DEBUGGING PROCESS

Error Detection

↓

Locate Error

↓

Analyze Cause

↓

Fix Error

↓

Retest

DEBUGGING APPROACHES

1. Brute Force Method

Uses print statements and logs.

2. Backtracking

Trace program backward from error.

3. Cause Elimination

Identify possible causes and eliminate one by one.

SOFTWARE QUALITY ASSURANCE (SQA)

SQA is a planned and systematic activity that ensures software quality.

OBJECTIVES OF SQA

1. Improve software quality.
2. Prevent defects.
3. Ensure standards compliance.
4. Increase customer confidence.

SOFTWARE QUALITY METRICS

Quality metrics measure software quality.

Examples:

1. Reliability

2. Maintainability
3. Efficiency
4. Portability
5. Usability

SOFTWARE RELIABILITY

Reliability is the probability that software performs correctly under specified conditions.

FACTORS AFFECTING RELIABILITY

1. Software complexity.
2. Testing quality.
3. Development practices.
4. Maintenance quality.

REAL-LIFE EXAMPLE

Online Shopping Website

Testing includes:

Functional Testing:

- Login
- Product Search
- Payment Processing

Performance Testing:

- Thousands of users accessing simultaneously

Security Testing:

- Protect customer payment information

Acceptance Testing:

- Customer approval before deployment

ADVANTAGES OF SOFTWARE TESTING

1. Improves software quality.
2. Reduces maintenance cost.
3. Increases reliability.
4. Enhances customer satisfaction.
5. Improves security.

DISADVANTAGES

1. Time-consuming.

2. Requires skilled testers.
3. Increases project cost.

UNIT – V

SOFTWARE PROJECT MANAGEMENT, COST ESTIMATION, RISK MANAGEMENT AND SOFTWARE CONFIGURATION MANAGEMENT:

INTRODUCTION TO SOFTWARE PROJECT MANAGEMENT (SPM)

Software Project Management is the process of planning, organizing, directing, monitoring, and controlling software projects to achieve project objectives within specified time, budget, and quality constraints.

A software project involves various activities such as:

- Requirement Analysis
- Design
- Coding
- Testing
- Deployment

- Maintenance

Software Project Management ensures that these activities are completed efficiently and successfully.

DEFINITION OF SOFTWARE PROJECT MANAGEMENT

Software Project Management is the application of knowledge, skills, tools, and techniques to software development activities in order to meet project requirements.

OBJECTIVES OF SOFTWARE PROJECT MANAGEMENT

1. Complete project on time.
2. Complete project within budget.
3. Deliver quality software.
4. Manage project risks.
5. Ensure customer satisfaction.
6. Efficient utilization of resources.

PROJECT

A Project is a temporary activity undertaken to create a unique product, service, or result.

Characteristics of a Project:

1. Defined objective.
2. Specific start and end date.
3. Limited resources.
4. Unique outcome.

Example:

Development of a College Management System.

ROLE OF SOFTWARE PROJECT MANAGER

A Software Project Manager is responsible for planning, monitoring, and controlling software development activities.

RESPONSIBILITIES OF PROJECT MANAGER

1. Project Planning
2. Team Management
3. Risk Management
4. Cost Estimation
5. Resource Allocation
6. Progress Monitoring
7. Quality Assurance

8. Client Communication

PROJECT PLANNING

Project Planning is the process of deciding:

- What to do?
- How to do?
- When to do?
- Who will do?

Proper planning is essential for project success.

PROJECT PLANNING ACTIVITIES

1. Scope Definition
2. Resource Planning
3. Cost Estimation
4. Scheduling
5. Risk Analysis
6. Quality Planning

PROJECT SCHEDULING

Project Scheduling determines the sequence and timing of project activities.

Scheduling helps in:

- Monitoring progress
- Resource utilization
- Timely completion

OBJECTIVES OF PROJECT SCHEDULING

1. Complete project on time.
2. Allocate resources efficiently.
3. Identify critical activities.
4. Track project progress.

GANTT CHART

A Gantt Chart is a graphical representation of project activities against time.

EXAMPLE OF GANTT CHART

Task Week1 Week2 Week3 Week4

ADVANTAGES OF GANTT CHART

1. Easy visualization.
2. Progress tracking.
3. Simple scheduling.

PERT (PROGRAM EVALUATION REVIEW TECHNIQUE)

PERT is a project management technique used to analyze and schedule project activities.

PERT helps determine:

- Minimum completion time
- Critical activities
- Project delays

PERT COMPONENTS

1. Event
2. Activity
3. Network Diagram

ADVANTAGES OF PERT

1. Better planning.
2. Risk identification.
3. Efficient scheduling.

COST ESTIMATION

Cost Estimation is the process of predicting the effort, time, and resources required for software development.

Accurate estimation is essential for successful project management.

IMPORTANCE OF COST ESTIMATION

1. Budget preparation.
2. Resource allocation.
3. Project planning.
4. Risk reduction.

FACTORS AFFECTING COST ESTIMATION

1. Project Size
2. Complexity
3. Team Experience
4. Technology Used
5. Development Environment

ESTIMATION TECHNIQUES

1. Expert Judgment
2. Delphi Technique
3. COCOMO Model
4. Function Point Analysis

DELPHI TECHNIQUE

Delphi Technique uses opinions from experts to estimate project cost and effort.

Process:

1. Experts provide estimates.
2. Estimates are reviewed.
3. Consensus is achieved.

FUNCTION POINT ANALYSIS (FPA)

Function Point Analysis estimates software size based on functionality provided to users.

Measures:

- Inputs
- Outputs
- Queries
- Files
- Interfaces

COCOMO MODEL

COCOMO stands for:

Constructive Cost Model

Developed by:

Barry Boehm (1981)

It is used to estimate:

- Development effort
- Project cost
- Development time

BASIC COCOMO MODEL

Effort Calculation:

Effort = a(KLOC)^b

Where:

- KLOC = Thousand Lines of Code
- a, b = Constants

PROJECT TYPES IN COCOMO

1. Organic Project

Small projects with experienced teams.

Example:

- College Management System

2. Semi-Detached Project

Medium-sized projects.

Example:

- Inventory Management System

3. Embedded Project

Large and complex projects.

Example:

- Air Traffic Control System

ADVANTAGES OF COCOMO

1. Easy estimation.
2. Scientific approach.
3. Widely used.

DISADVANTAGES OF COCOMO

1. Depends on code size estimation.
2. Less suitable for modern agile projects.

SOFTWARE RISK MANAGEMENT

Risk is the possibility of an event that may negatively affect project objectives.

Risk Management identifies, analyzes, and controls project risks.

TYPES OF RISKS

1. Project Risks

Affect project schedule and resources.

Examples:

- Staff shortage
- Budget issues
- Schedule delays

2. Technical Risks

Affect software quality and performance.

Examples:

- Technology failure
- Design problems
- Integration issues

3. Business Risks

Affect business success.

Examples:

- Market competition
- Customer dissatisfaction
- Product rejection

RISK MANAGEMENT PROCESS

Risk Identification

↓

Risk Analysis

↓

Risk Planning

↓

Risk Monitoring

RISK IDENTIFICATION

Potential risks are identified.

Examples:

- Hardware failure
- Requirement changes
- Team turnover

RISK ANALYSIS

Probability and impact of risks are evaluated.

RISK MITIGATION

Mitigation means reducing the effect of risks.

Example:

Risk:

- Key employee leaves project.

Mitigation:

- Cross-training team members.

SOFTWARE CONFIGURATION MANAGEMENT (SCM)

Software Configuration Management is the process of controlling and managing software changes throughout the software life cycle.

SCM helps maintain consistency and integrity of software products.

OBJECTIVES OF SCM

1. Control software changes.
2. Maintain version history.
3. Improve team coordination.
4. Prevent unauthorized modifications.

NEED FOR SCM

During development:

- Requirements change.
- Code changes occur.
- Bugs are fixed.

SCM ensures that these changes are properly tracked and managed.

CONFIGURATION ITEM (CI)

A Configuration Item is any software artifact that needs control.

Examples:

1. Source Code
2. Design Documents
3. Test Cases
4. Requirement Documents
5. User Manuals

SCM ACTIVITIES

1. Configuration Identification
2. Version Control
3. Change Control
4. Configuration Audit
5. Status Accounting

CONFIGURATION IDENTIFICATION

Identifies software components that require management.

VERSION CONTROL

Version Control manages different versions of software.

Examples:

- Git
- GitHub
- GitLab
- Bitbucket

BENEFITS OF VERSION CONTROL

1. Tracks changes.
2. Supports teamwork.
3. Easy rollback.
4. Prevents data loss.

CHANGE CONTROL

Change Control evaluates and approves software modifications.

Process:

Change Request

↓

Review

↓

Approval

↓

Implementation

CONFIGURATION AUDIT

Configuration Audit verifies that software complies with specified requirements.

Types:

1. Functional Audit
2. Physical Audit

STATUS ACCOUNTING

Maintains records of software configurations and changes.

Benefits:

- Better tracking
- Improved management

SOFTWARE QUALITY MANAGEMENT

Quality Management ensures that software meets required standards.

QUALITY MANAGEMENT ACTIVITIES

1. Quality Planning
2. Quality Assurance
3. Quality Control

QUALITY PLANNING

Defines quality standards.

QUALITY ASSURANCE

Ensures processes are followed correctly.

QUALITY CONTROL

Detects defects in software products.

SOFTWARE METRICS

Software Metrics are quantitative measures used to evaluate software quality and performance.

TYPES OF SOFTWARE METRICS

1. Product Metrics

Measure software product characteristics.

Examples:

- Reliability
- Maintainability

2. Process Metrics

Measure development process effectiveness.

Examples:

- Defect Density
- Development Time

3. Project Metrics

Measure project performance.

Examples:

- Cost
- Schedule
- Productivity

REAL-LIFE EXAMPLE

Online Banking Project

Project Manager Responsibilities:

- Estimate project cost.
- Schedule development tasks.
- Monitor risks.
- Manage software versions.
- Ensure quality standards.

Tools Used:

- Microsoft Project
- Jira
- GitHub
- Jenkins

ADVANTAGES OF SOFTWARE PROJECT MANAGEMENT

1. Better planning.
2. Timely project completion.
3. Improved quality.

4. Reduced risks.
5. Customer satisfaction.

DISADVANTAGES

1. Requires experienced managers.
2. Additional documentation effort.
3. Management overhead.