Module 1: The Software Process

1. What is Software?

Software means a set of **programs, instructions, and data** that tell a computer what to do.

Simple Meaning:

Hardware = Physical parts of computer (keyboard, mouse, CPU)
Software = Brain of computer

Examples:

- Microsoft Word – used for typing
- WhatsApp – chatting app
- Google Chrome – internet browsing
- Windows – runs computer

Important Exam Points:

- Software is intangible (cannot touch)
- It controls hardware
- It solves user problems

---

2. What is Software Engineering?

Software Engineering means **systematic way of developing software using proper methods, planning, testing, and maintenance.**

Easy Meaning:

Making software professionally, not randomly.

Example:

If one person makes calculator app quickly = Programming
If a team plans, designs, tests, secures, updates banking app = Software Engineering

Why Needed?

- To reduce errors
- Finish on time

- Reduce cost
- Good quality software
- Easy maintenance

Definition:

Application of engineering principles to software development.

Memory Trick:

PDTM

- Planning
- Designing
- Testing
- Maintenance

---

3. Software Application Domain

Application Domain means **where software is used.**

Types:

Domain	**Meaning**	**Example**
Business	Used in offices	Billing software
Scientific	Maths/Research	Weather prediction
Embedded	Inside devices	Washing machine chip
Web Apps	Internet apps	Amazon
Mobile Apps	Smartphone apps	Instagram
AI Systems	Smart decision systems	Chatbots
Gaming	Entertainment	PUBG

Important Exam Point:

Application domain tells purpose of software.

---

4. Changing Nature of Software

Software keeps changing because user needs change.

Earlier:

- Desktop software only
- Offline systems
- Simple functions

Now:

- Cloud apps
- Mobile apps
- AI apps
- Real-time apps
- Cybersecurity focus

Example:

Paytm changed from wallet to full financial platform.

Why Changes Happen?

- New technology
- User demand
- Competition
- Security threats

Exam Point:

Software is dynamic, always evolving.

---

5. Software Development Myths

Myths = Wrong beliefs about software development.

Common Myths:

Myth 1:

"If project is late, add more people."

❌ Wrong because new people need training.

Myth 2:

“If software works once, job finished.”

❌ Wrong because maintenance needed.

Myth 3:

“Requirements can change anytime without cost.”

❌ Wrong because changes increase time and money.

Myth 4:

“Only coding matters.”

❌ Wrong because planning/testing also important.

Exam Point:

Myths create project failure.

---

6. Process Frameworks

Framework means a structured way to develop software.

Main Activities:

Communication → Planning → Modeling → Construction → Deployment

Explanation:

1. Communication

Talk with client and understand needs.

2. Planning

Time, team, cost planning.

3. Modeling

Design system.

4. Construction

Coding + Testing.

5. Deployment

Give software to users.

Example:

Food delivery app:

- Ask features
- Plan timeline
- Design screens
- Code app
- Launch to users

Exam Tip:

Remember sequence **CPMCD**

---

7. Software Umbrella Activities

These activities happen throughout project.

Main Umbrella Activities:

Activity	**Meaning**
Project Tracking	Check progress
Risk Management	Find dangers
Quality Assurance	Maintain quality
Documentation	Keep records
Configuration Management	Manage versions
Measurement	Track performance

Why Called Umbrella?

Because they cover whole project.

Example:

While coding app, testing and documentation continue side by side.

---

8. Process Patterns

Process Pattern = Reusable solution for common software problems.

Easy Meaning:

Same problem comes many times, so use standard solution.

Example:

Login page creation pattern used in many apps.

Benefits:

- Saves time
- Reduces mistakes
- Standard process

---

9. Process Assessment and Improvement

It means checking current software process and making it better.

Example:

Company notices projects always late.

Assessment:

- Why delay?
- Poor planning?
- Less testing?

Improvement:

- Better planning
- Agile method
- More automation

Benefits:

- Faster delivery
- Better quality
- Lower cost

---

10. Personal Software Process (PSP)

PSP means one developer improves their own work method.

Focus:

- Time management

- Error tracking
- Better coding habits

Example:

A programmer tracks:

- How much time coding took
- How many bugs made
- How to improve next time

Benefit:

Individual productivity improves.

---

11. Team Software Process (TSP)

TSP means improving performance of full team.

Focus:

- Team planning
- Shared responsibility
- Better communication

Example:

5 students making project:

- One coding
- One UI
- One testing
- One database
- One documentation

Benefit:

Team works efficiently.

PSP vs TSP Difference

Point	PSP	TSP
Meaning	Personal process	Team process
Used by	Individual	Group
Focus	Self improvement	Team success
Example	One coder	Full company team

Important Exam Questions (Very Likely)

1. Define software engineering.
2. Explain changing nature of software.
3. What are software myths?
4. Explain process framework activities.
5. What are umbrella activities?
6. Differentiate PSP and TSP.
7. Explain software application domains.

Quick Revision Tricks

Software Process Framework:

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

Umbrella Activities:

QRRDCM
Quality, Risk, Review, Documentation, Configuration, Measurement

---

2 Marks Ready Answers

What is Software?

Software is a collection of programs and data used to perform tasks on a computer.

What is Software Engineering?

Software engineering is systematic development and maintenance of software using engineering principles.

What is PSP?

PSP is a personal method used by individual developers to improve quality and productivity.

What is TSP?

TSP is a team-based process for planning and managing software development.

Final Easy Summary

This module is mainly about:

- What software is
- How software is built professionally
- Steps of software process
- Team and personal improvement
- Common mistakes in development

Module 2: Traditional & Agile Software Development

Introduction

Software can be developed using different methods called **Software Process Models**.

Think like this:

- If you build a house step-by-step with fixed plan = Traditional model
- If you keep improving based on feedback = Agile model

So this module has **2 parts**:

1. Traditional Development
2. Agile Development

PART A: Traditional Software Development

Traditional models follow a planned, structured sequence.

Best when:

- Requirements are clear
- Changes are less
- Big documentation needed

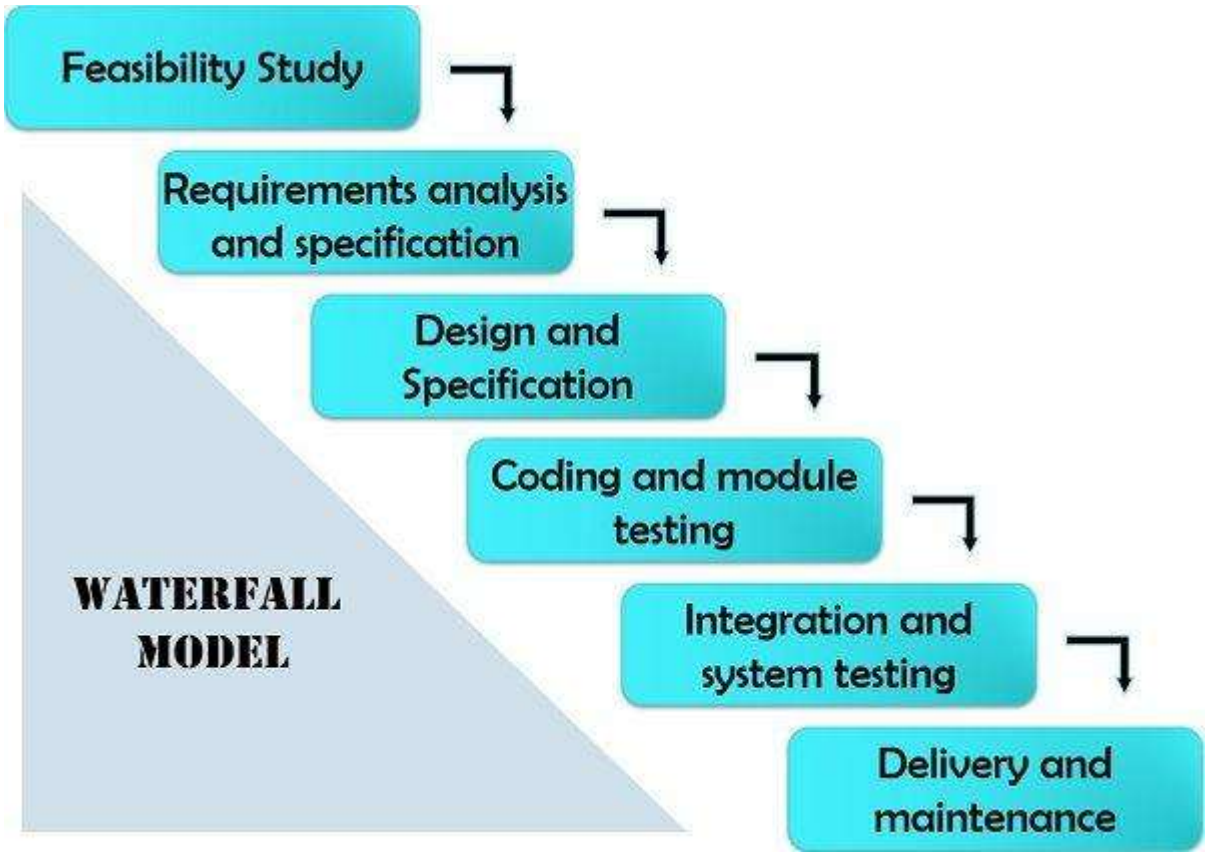
---

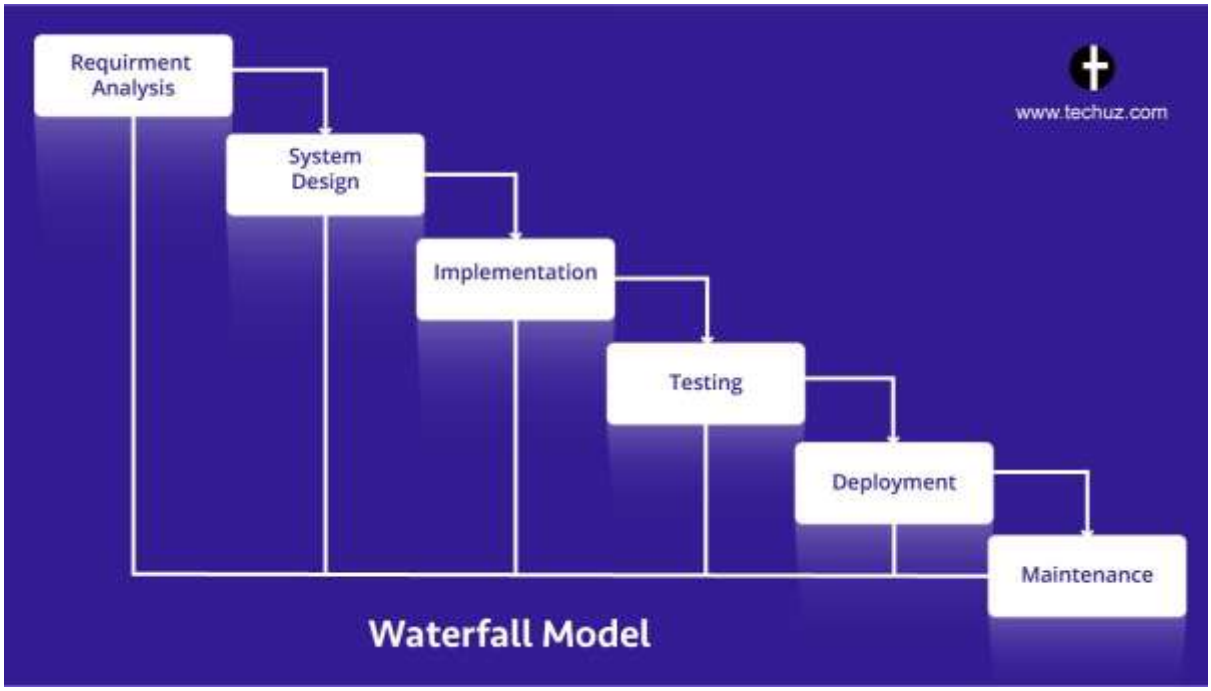
1. Waterfall Model

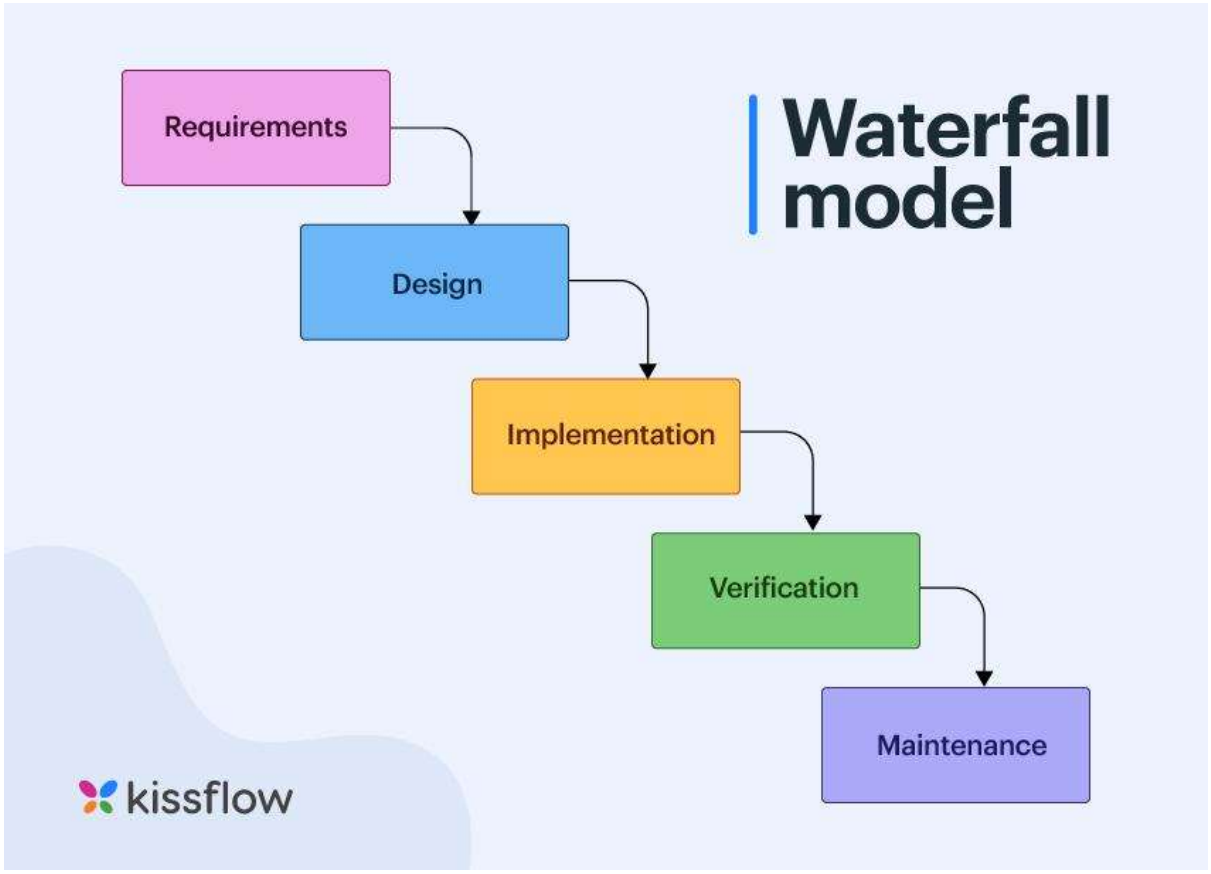
Most famous traditional model.

Software development moves step-by-step like waterfall.

Requirements → Design → Coding → Testing → Deployment → Maintenance







6

Explanation:

Each phase must finish before next starts.

Example:

College result system where requirements are already fixed.

Advantages:

- Easy to understand
- Clear stages
- Good documentation

Disadvantages:

- Hard to change later
- Testing comes late
- Slow for dynamic projects

Memory Trick:

R D C T D M

---

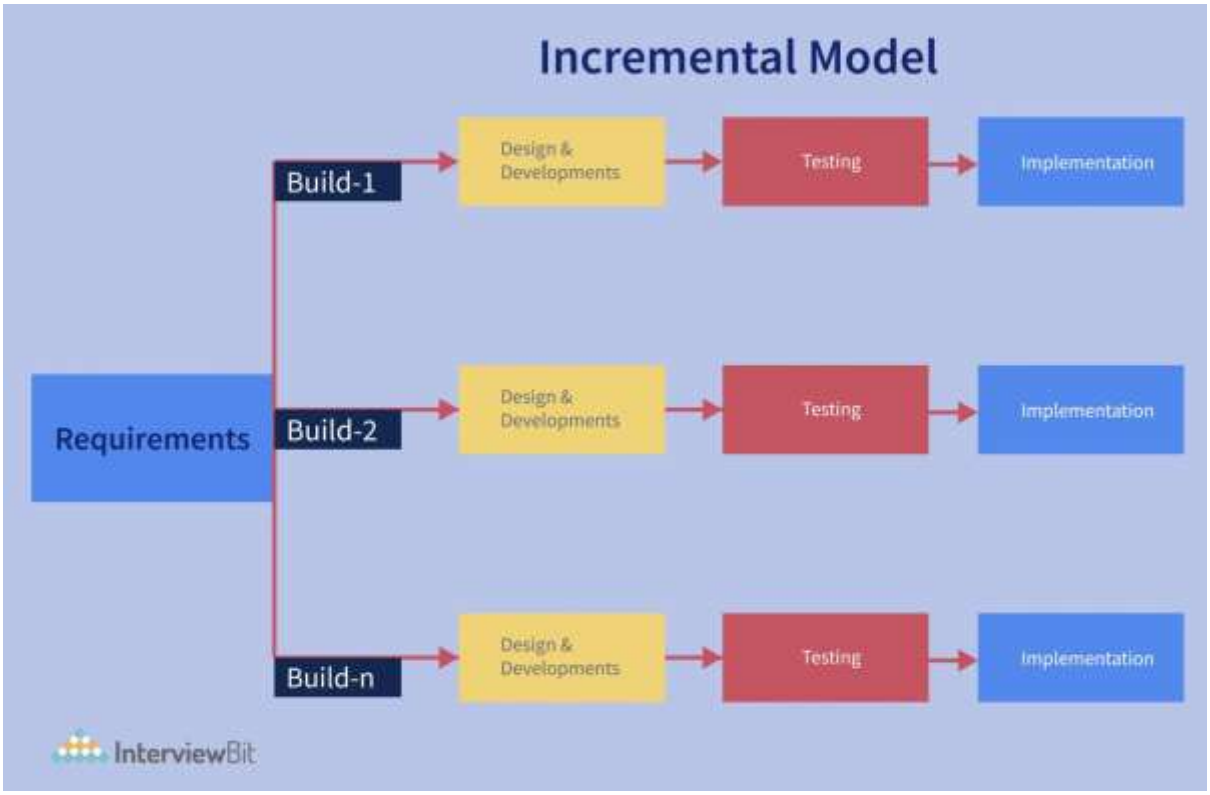
2. Incremental Model

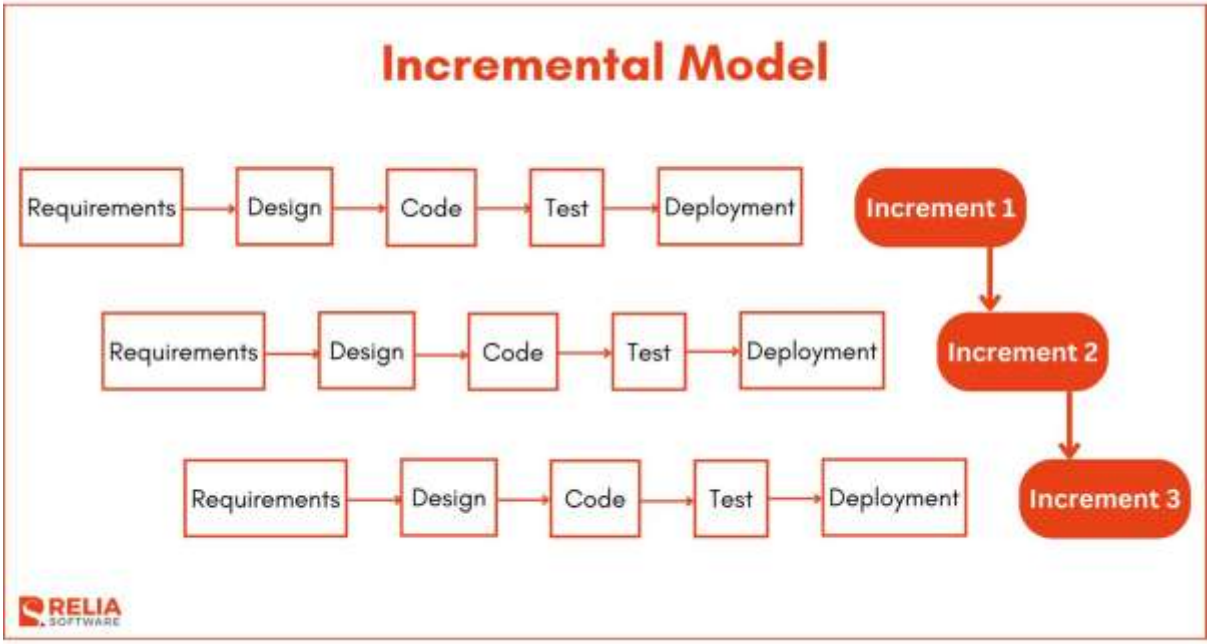
Software is built in small parts called increments.

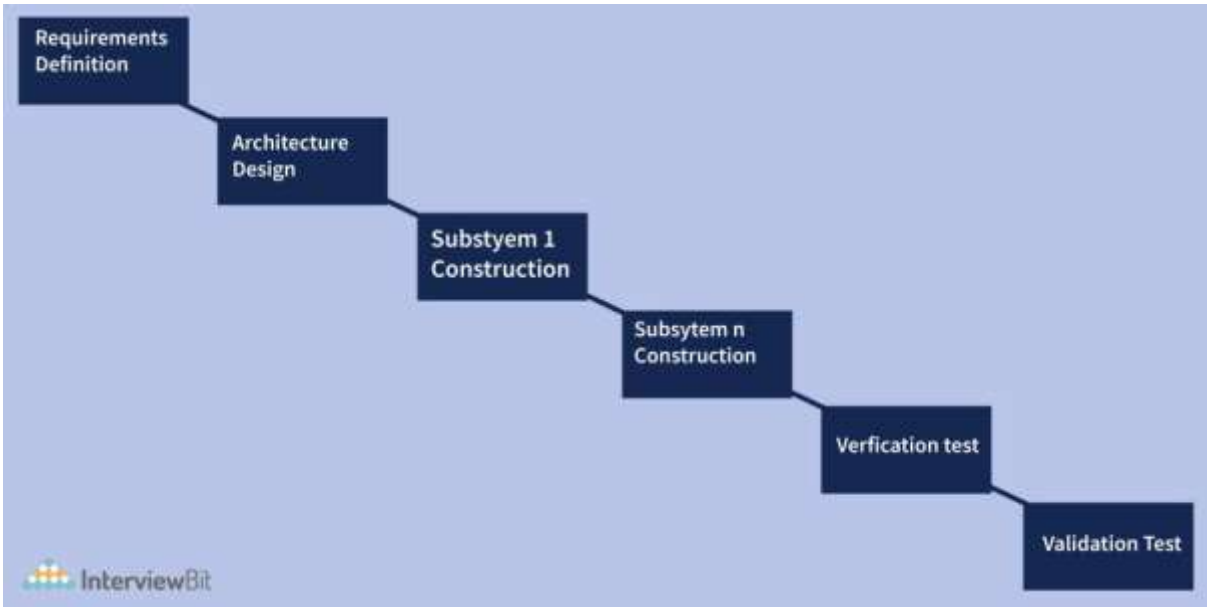
Example:

Food app release:

- Version 1 = Login
- Version 2 = Order food
- Version 3 = Tracking


Incremental Model
Requirements
Build-1
Build-2
Build-n
Design & Developments
Design & Developments
Design & Developments
Testing
Testing
Testing
Implementation
Implementation
Implementation
InterviewBit


Incremental Model
Requirements
Design
Code
Test
Deployment
Increment 1
Requirements
Design
Code
Test
Deployment
Increment 2
Requirements
Design
Code
Test
Deployment
Increment 3
RELIA SOFTWARE



4

Advantages:

- Early usable product
- Easy updates
- Lower risk

Disadvantages:

- Needs good planning
- Integration issues possible

---

3. RAD Model (Rapid Application Development)

RAD means fast development using reusable tools/components.

Focus:

- Speed
- Fast prototypes
- User feedback

Example:

Internal office management app built quickly.

Advantages:

- Quick delivery
- Less manual coding
- User involvement

Disadvantages:

- Not good for huge systems
- Needs skilled team

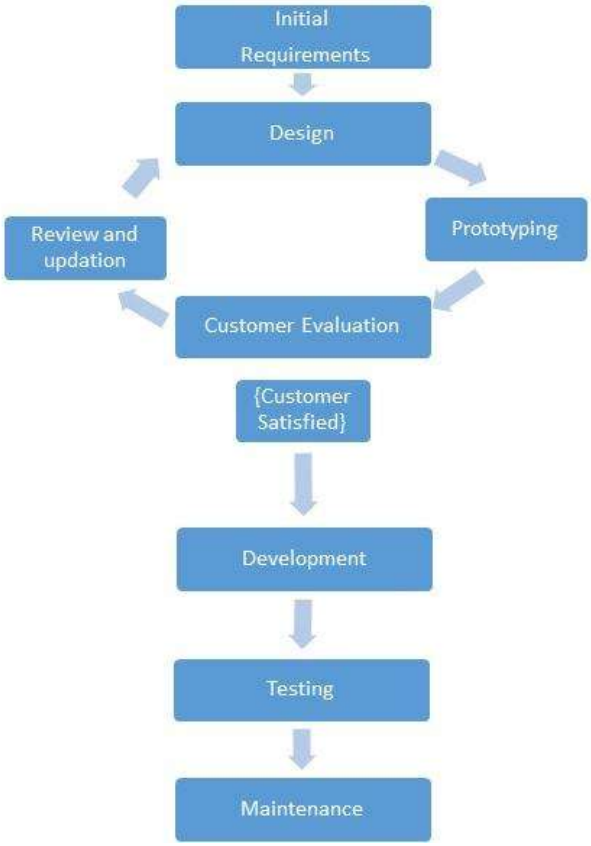
---

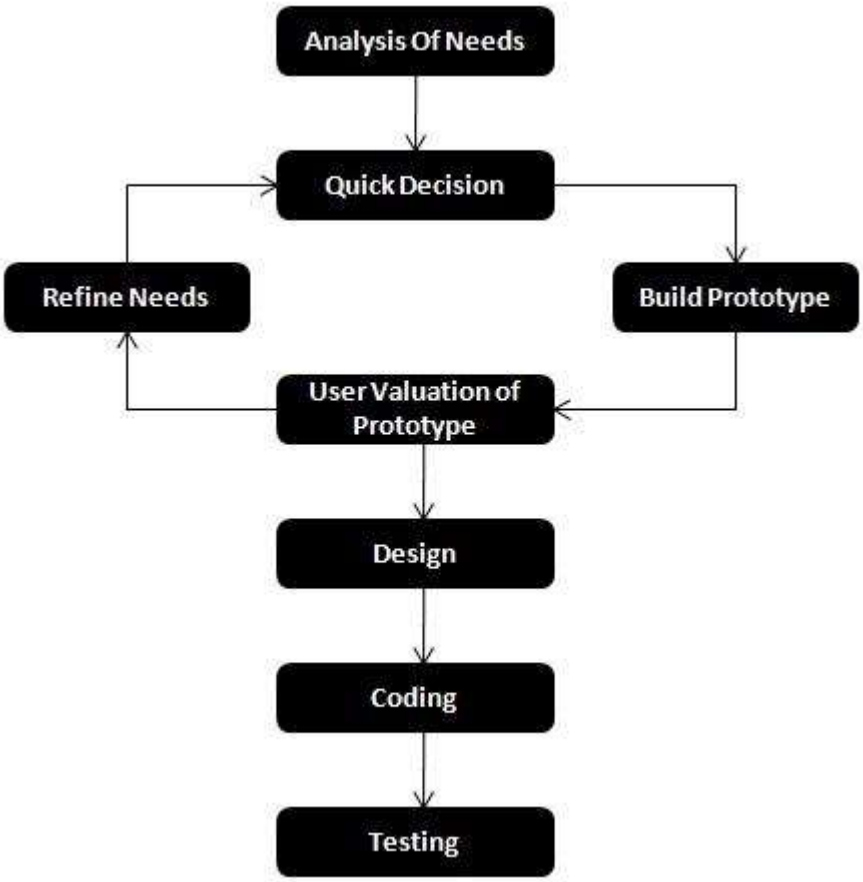
4. Prototyping Model

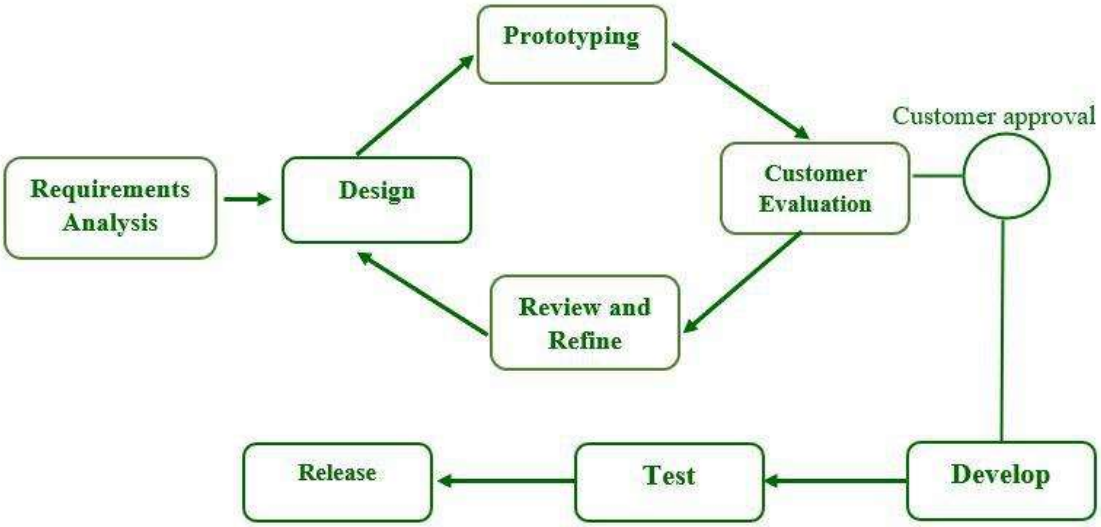
Prototype = Sample version before final product.

Steps:

1. Build rough model
2. Show user
3. Take feedback
4. Improve final software







6

Example:

E-commerce website design mockup before full coding.

Advantages:

- Better requirement understanding
- User satisfaction

Disadvantages:

- User may think prototype is final product

---

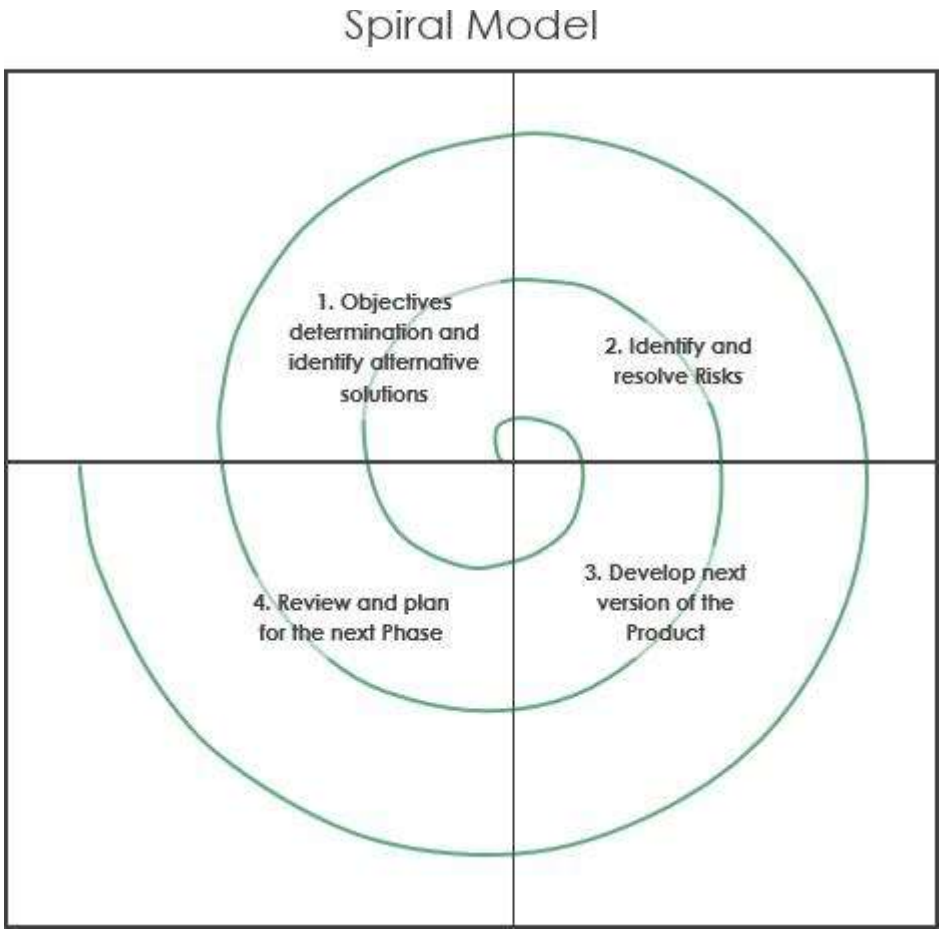
5. Spiral Model

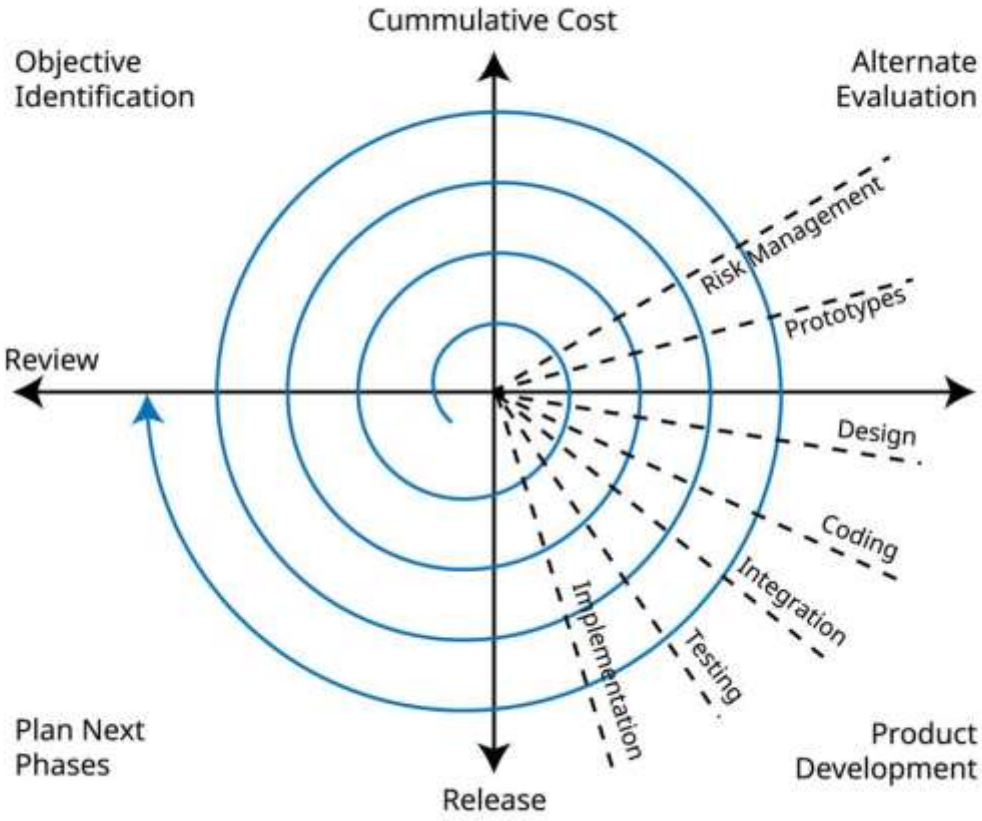
Combines design + risk analysis + repeated cycles.

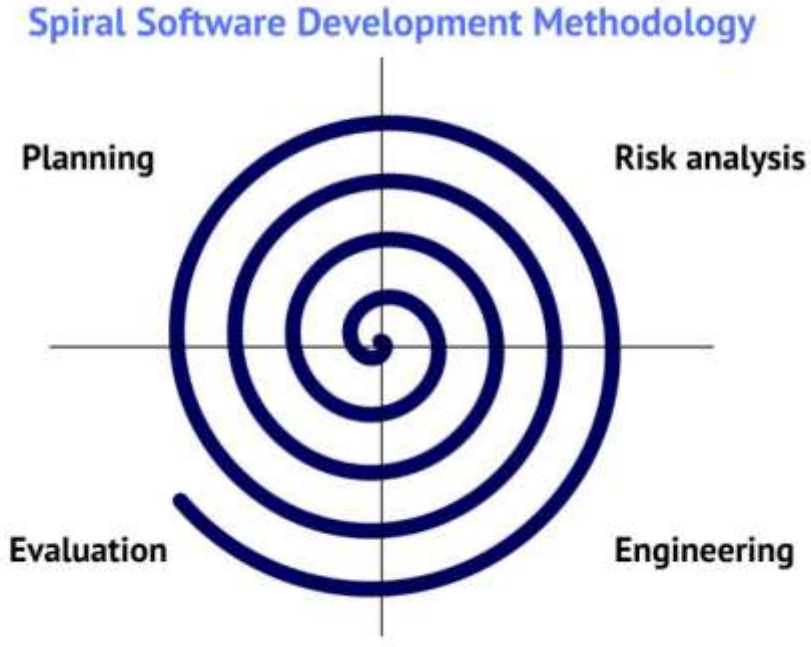
Used for expensive/high-risk projects.

Each Spiral Loop:

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




Cummulative Cost
Objective Identification
Alternate Evaluation
Risk Management
Prototypes
Review
Design
Coding
Integration
Implementation
Testing
Plan Next Phases
Product Development
Release


Spiral Software Development Methodology
Planning
Risk analysis
Evaluation
Engineering

5

Example:

Banking software, defense systems.

Advantages:

- Risk handled early
- Flexible

Disadvantages:

- Expensive
- Complex

---

6. Specialized Models

Special models for special situations.

Examples:

- Component-Based Development
- Formal Methods
- Aspect-Oriented

---

7. Component-Based Development Model

Software made using ready-made reusable parts.

Example:

Website using login module + payment module + chat module.

Advantage:

- Saves time
- Less coding

---

8. Concurrent Development Model

Many activities happen together at same time.

Example:

- Team A coding
- Team B testing

- Team C UI designing

Good for large teams.

Traditional Models Comparison

Model	Best For	Main Feature
Waterfall	Fixed requirements	Sequential
Incremental	Growing product	Small releases
RAD	Fast delivery	Speed
Prototype	Unclear requirements	Sample first
Spiral	High risk projects	Risk analysis

PART B: Agile Development

Agile means flexible, fast, customer-focused development.

Instead of waiting months, software is delivered in small working parts.

Agile Values:

- Working software > Too much paperwork
- Customer collaboration > Contract fights
- Responding to change > Fixed rigid plan

Agile Manifesto

Created in 2001.

Main idea: Build software smarter and faster.

Four Values:

1. Individuals & interactions over processes/tools
2. Working software over documentation
3. Customer collaboration over contracts
4. Responding to change over following plan

Exam Tip:

Write any 4 values.

Agile Principles (Easy)

- Deliver software frequently
- Welcome changing requirements
- Teamwork daily
- Simplicity matters
- Continuous improvement

Agile Models

---

1. Scrum

Most popular Agile framework.

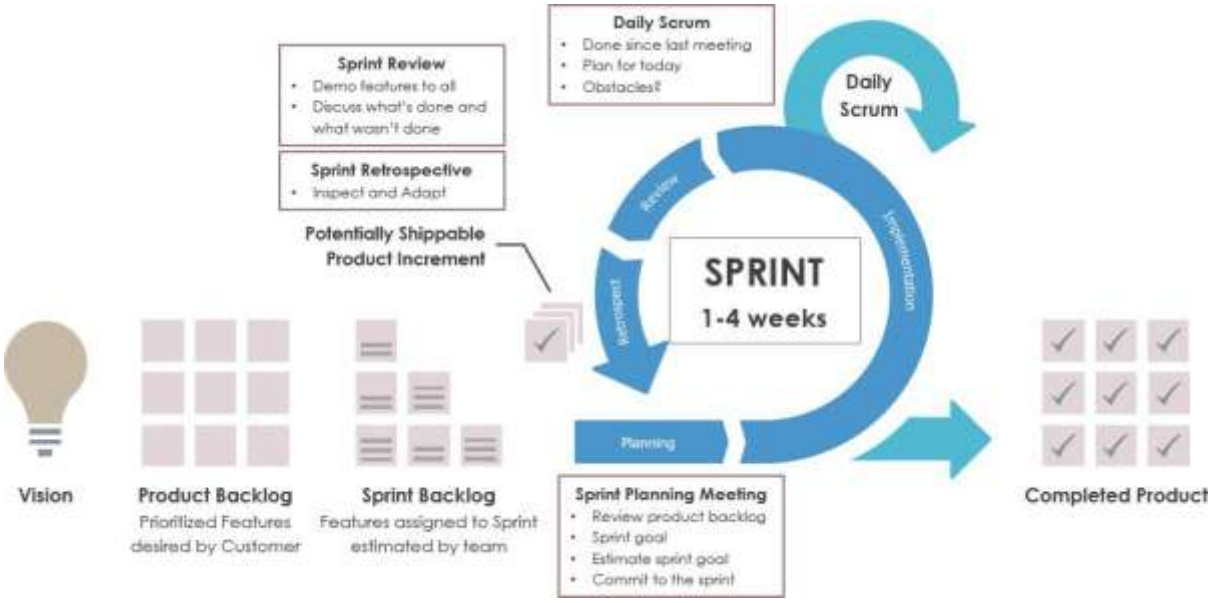
Work divided into short cycles called **Sprints** (1–4 weeks).

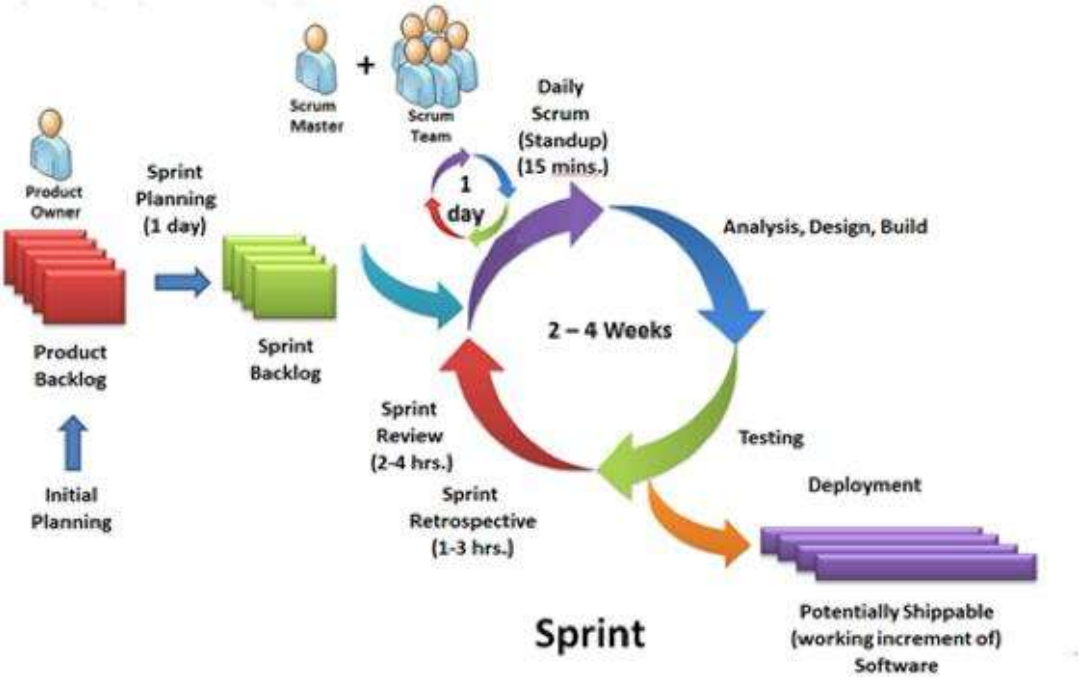
Roles:

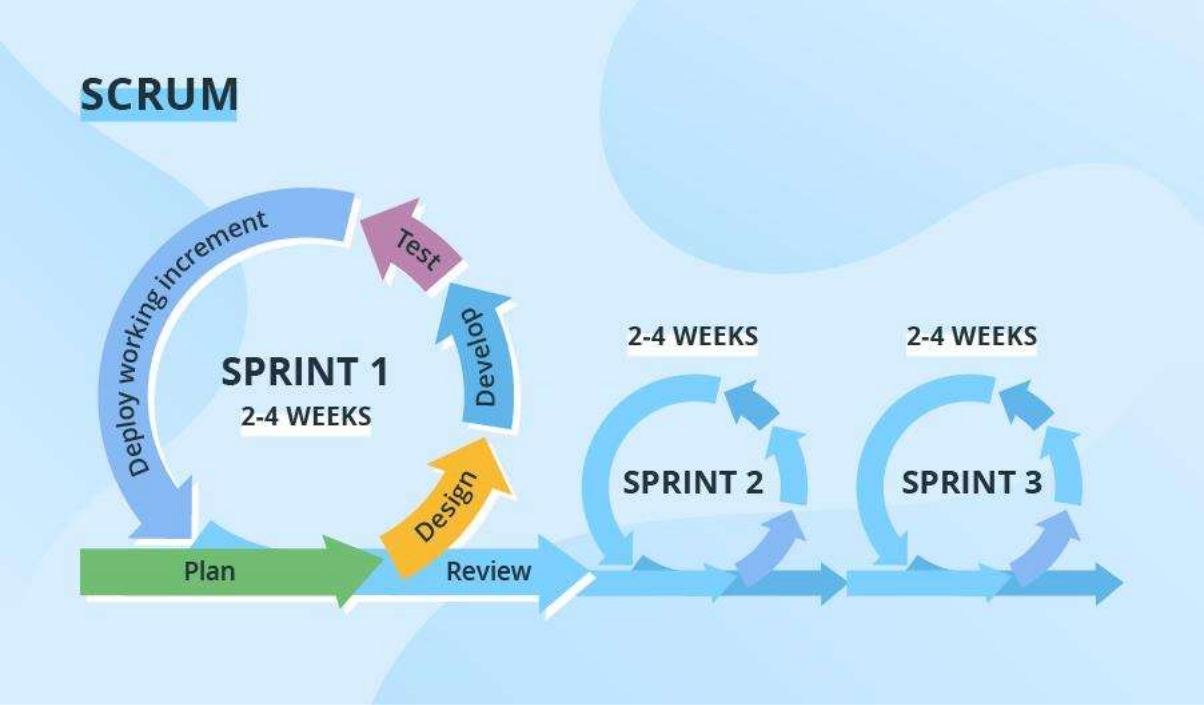
Role	Work
Product Owner	Decides features
Scrum Master	Helps team
Development Team	Builds software

Scrum Process:

Backlog → Sprint Planning → Sprint → Review → Next Sprint


Daily Scrum
Done since last meeting
Plan for today
Obstacles?
Sprint Review
Demo features to all
Discuss what's done and what wasn't done
Sprint Retrospective
Inspect and Adapt
Potentially Shippable Product Increment
Daily Scrum
Review
Implementation
Retrospect
SPRINT
1-4 weeks
Planning
Vision
Product Backlog
Prioritized Features desired by Customer
Sprint Backlog
Features assigned to Sprint estimated by team
Sprint Planning Meeting
Review product backlog
Sprint goal
Estimate sprint goal
Commit to the sprint
Completed Product


Scrum Master
+
Scrum Team
Daily Scrum (Standup) (15 mins.)
1 day
Product Owner
Sprint Planning (1 day)
Analysis, Design, Build
Product Backlog
Sprint Backlog
2 – 4 Weeks
Sprint Review (2-4 hrs.)
Testing
Initial Planning
Sprint Retrospective (1-3 hrs.)
Deployment
Potentially Shippable (working increment of) Software
Sprint



5

Example:

Building Instagram features sprint by sprint.

---

2. Extreme Programming (XP)

Focus on coding quality and customer feedback.

Practices:

- Pair programming
- Frequent testing
- Small releases
- Continuous integration

Example:

Two coders write code together.

---

3. Feature Driven Development (FDD)

Development based on features.

Example:

Banking app features:

- Transfer money
- View balance
- Statements

Each feature completed one by one.

Agile vs Traditional

Point	**Traditional**	**Agile**
Changes	Difficult	Easy
Delivery	Late	Frequent
Customer Feedback	Less	High
Documentation	Heavy	Moderate
Speed	Slower	Faster

Case Study: Agile Transformation Example – Spotify

Spotify became famous for Agile team structure.

Used:

- Scrum squads
- Small autonomous teams
- Continuous integration
- Fast releases

Challenges:

- Coordination between teams
- Scaling culture

Success Metrics:

- Faster feature releases
- Better user satisfaction

- Innovation speed

Important Exam Questions

1. Explain Waterfall Model.
2. Difference between Agile and Traditional models.
3. Explain Scrum framework.
4. What is RAD model?
5. Explain Spiral Model with advantages.
6. What is Agile Manifesto?
7. Explain XP model.

---

2 Marks Ready Answers

What is Waterfall Model?

Waterfall model is a sequential software process where one phase starts after previous finishes.

What is Agile?

Agile is a flexible software development approach with small frequent releases and customer feedback.

What is Sprint?

Sprint is a short fixed time period in Scrum to complete selected work.

What is Prototype?

Prototype is an early sample model of software.

Quick Memory Tricks

Traditional Models:

WIRPSC
Waterfall, Incremental, RAD, Prototype, Spiral, Component

Agile Models:

SXF
Scrum, XP, FDD

Final Easy Summary

This module teaches:

- Old software development methods
- New Agile methods
- How companies build software faster today
- Difference between rigid and flexible development

Module 3: Requirements Analysis with Cost Estimation

Introduction

Before making software, we must know:

1. What customer wants
2. What system should do
3. How much time it takes
4. How much cost it needs

This module is about understanding needs and estimating project cost.

PART A: Software Requirements

What is Requirement?

Requirement means **need or demand** of user from software.

Example:

For ATM machine:

- Withdraw money
- Check balance
- Print receipt

These are requirements.

Types of Requirements

1. Functional Requirements

These tell **what software should do**.

Examples:

- User can login
- Add to cart
- Transfer money
- Search products

Easy Meaning:

Functions/features of software.

---

2. Non-Functional Requirements

These tell **how software should perform**.

Examples:

- Fast speed
- Secure system
- Easy to use
- Available 24/7

Easy Meaning:

Quality standards.

Functional vs Non-Functional

Point	Functional	Non-Functional
Meaning	What system does	How system performs
Example	Login feature	Fast login
Focus	Features	Quality

---

3. User Requirements

Written in simple language for users.

Example:

“The app should allow customer to order food online.”

---

4. System Requirements

Detailed technical requirements for developers.

Example:

- Database must store orders
- Server response under 2 sec

---

5. Interface Specification

Interface means connection between systems/users.

Types:

- User Interface (screens/buttons)
- Hardware Interface
- Software Interface
- API Interface

Example:

Paytm connects with bank server.

---

6. Software Requirements Specification (SRS)

SRS = Document containing all requirements clearly.

Full Form:

Software Requirements Specification

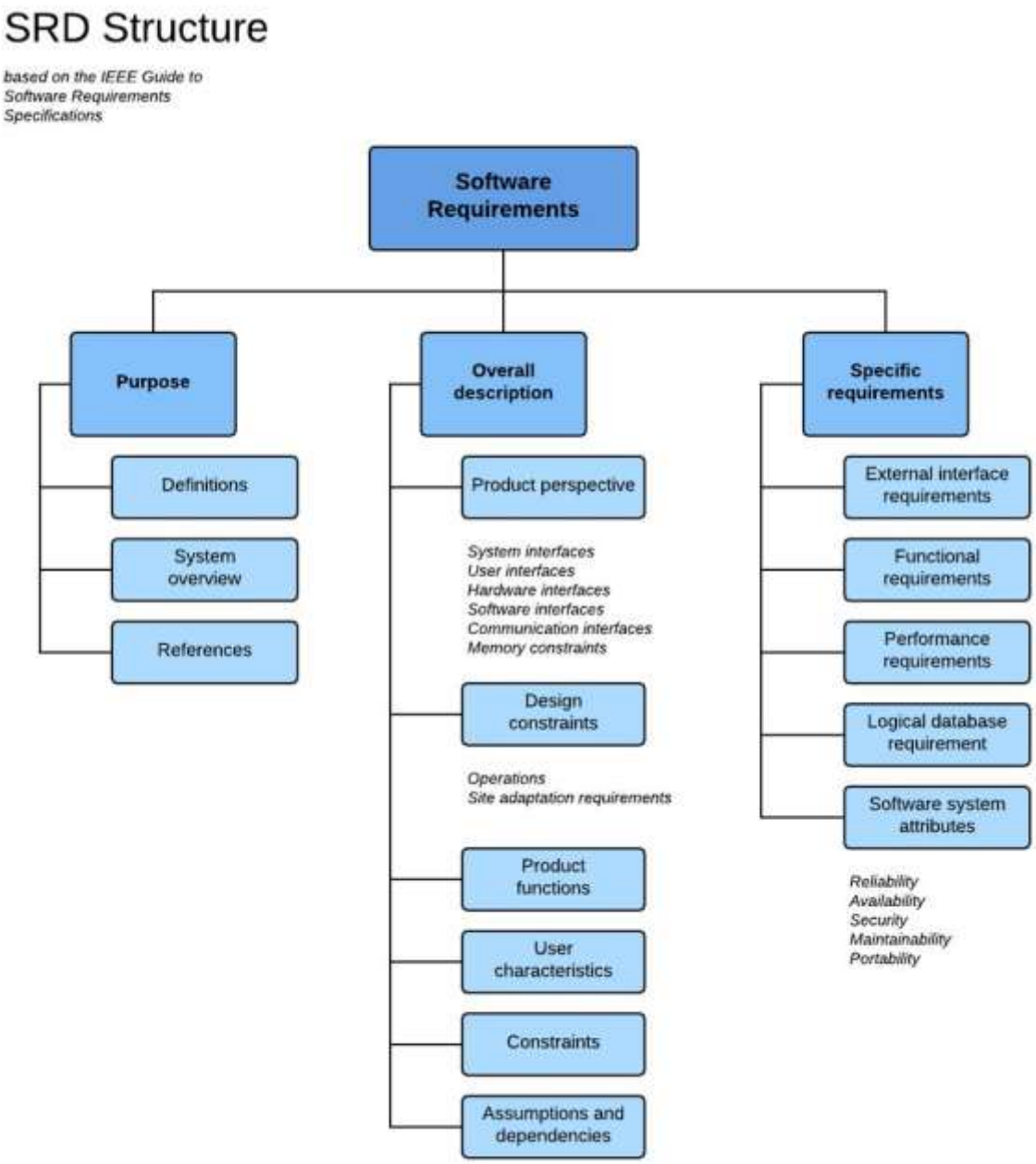
Contains:

- Project overview

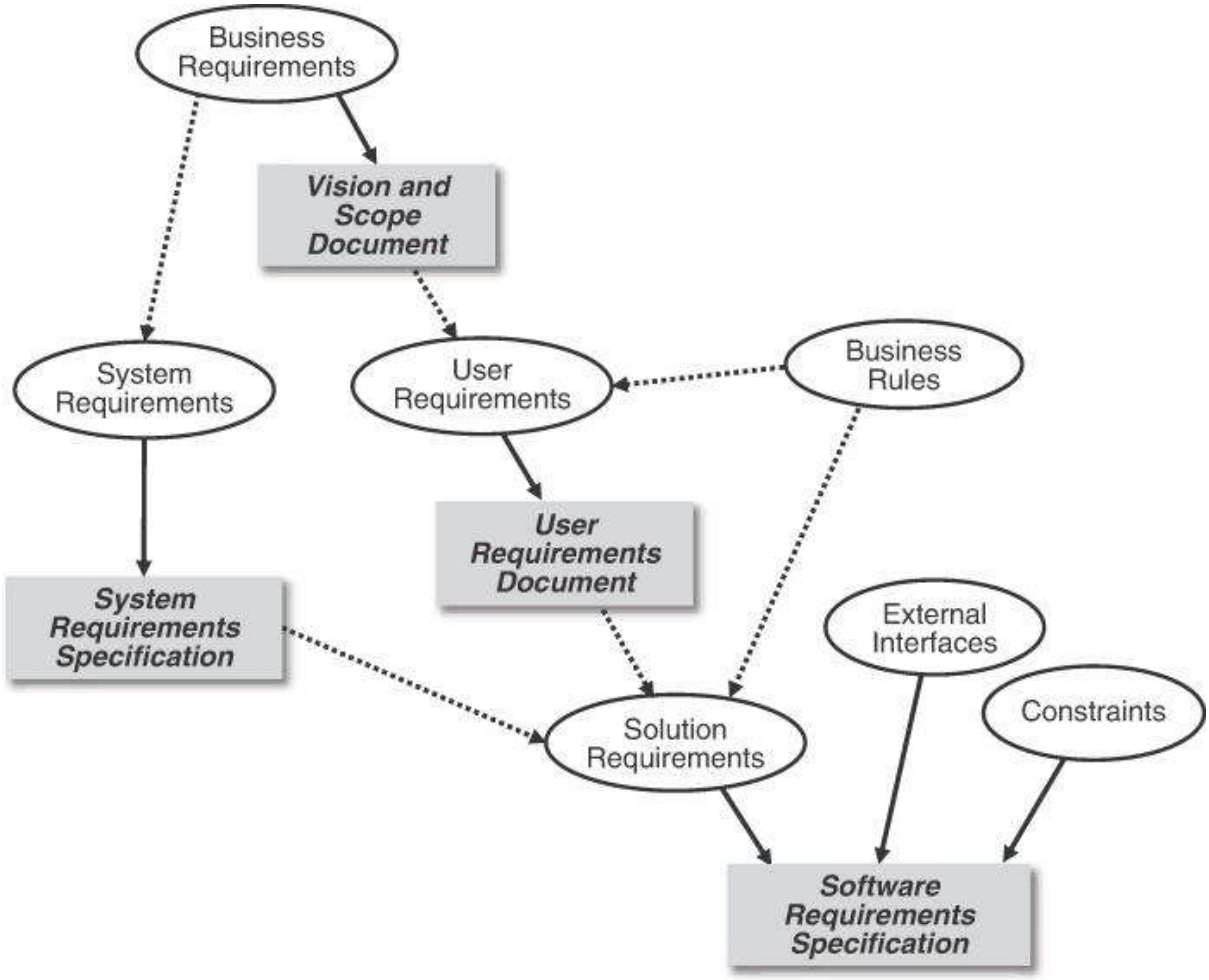
- Functional requirements
- Non-functional requirements
- Constraints
- Interface details

Why Important?

- Acts like agreement between client and developer




01
Introduction
04
Non-functional requirements
Software requirement specifications
02
Functional requirements
03
External interface requirements
asana



PART B: Requirements Engineering Process

This is the process of collecting and managing requirements.

---

1. Feasibility Study

Checks whether project is possible or not.

Types:

Type	Meaning
Technical	Can we build it?
Economic	Is it affordable?
Operational	Will users use it?

Type	Meaning
Legal	Any law issue?
Schedule	Can finish on time?

Example:

Making delivery app in 2 days = not feasible.

---

2. Requirements Elicitation

Elicitation means collecting requirements.

Methods:

- Interview users
- Surveys
- Meetings
- Observation

Example:

Ask restaurant owner what features needed.

---

3. Requirements Analysis

Study collected requirements carefully.

Purpose:

- Remove confusion
- Remove duplicate requests
- Check priority

---

4. Requirements Validation

Validation means checking if requirements are correct.

Questions:

- Is this really needed?

- Is it clear?
- Any contradiction?

Example:

Customer asks “fast system.” Need define how fast.

---

5. Requirements Management

Managing changes in requirements.

Example:

Later client asks:
“Add dark mode.”

This change must be tracked.

Process Flow

Feasibility → Elicitation → Analysis → Validation → Management

Memory Trick:

F E A V M

PART C: Project Estimation

Estimation means predicting cost, time, effort.

---

1. LOC Based Estimation

LOC = Lines of Code

Estimate based on number of code lines.

Example:

Project may need 10,000 lines.

Advantage:

Simple

Disadvantage:

Hard before coding starts

---

2. FP Based Estimation

FP = Function Point

Estimate based on number of features/functions.

Count:

- Inputs
- Outputs
- Files
- Queries

Example:

Login + Payment + Search = more function points

Better than LOC because based on functionality.

---

3. Use Case Based Estimation

Estimate based on use cases (user actions).

Example Use Cases:

- Register
- Login
- Buy item
- Track order

More use cases = more effort.

Comparison of Estimation Methods

Method	Based On	Best Use
LOC	Code lines	Technical estimation
FP	Functions	Business systems

Method	Based On	Best Use
Use Case	User interactions	Object-oriented projects

---

4. Empirical Estimation Models

Empirical = Based on past data/formulas.

Use previous project experience to estimate new projects.

Example:

Last similar app took 3 months, so new one may take near that.

---

5. COCOMO-II Model

Famous cost estimation model.

COCOMO = Constructive Cost Model

Used to estimate:

- Effort
- Time
- Cost

Basic Idea:

Larger project = More effort

$$Effort = a \times (Size)^b$$

Where:

- Size = Project size
- a,b = constants

Exam Tip:

Write full form + used for cost estimation.

PART D: Analysis Model

Analysis model helps understand system before design.

---

1. Scenario-Based Elements

Based on user scenarios.

Example:

Customer orders food:
Login → Select item → Pay → Order confirmed

Use case diagrams used here.

---

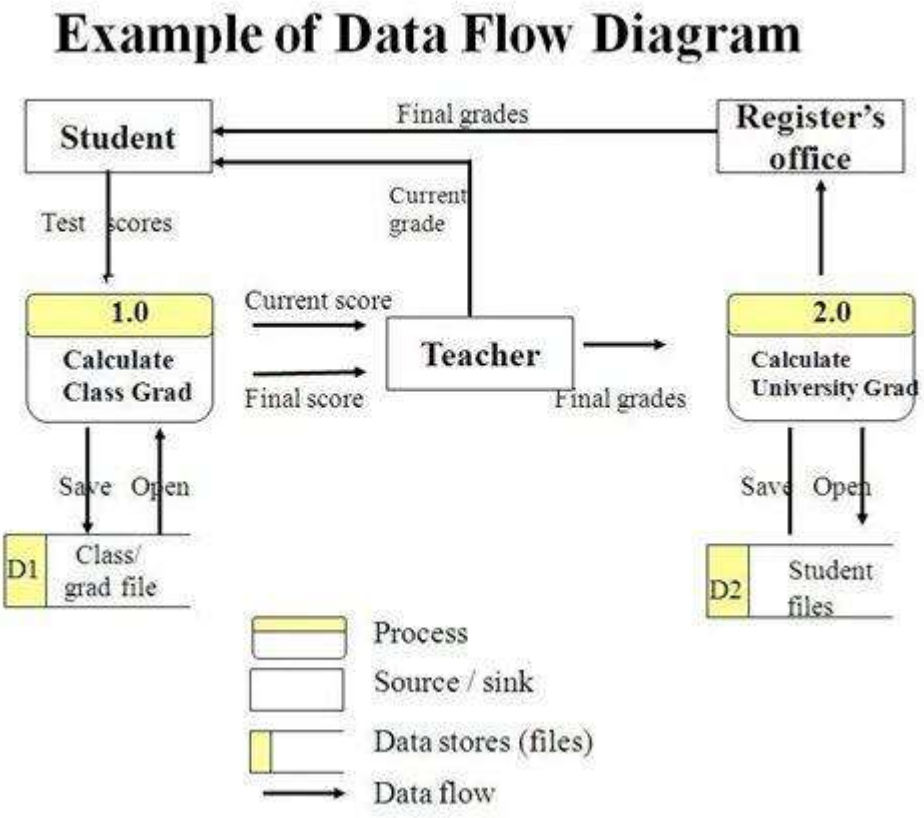
2. Flow-Based Elements

Shows flow of data.

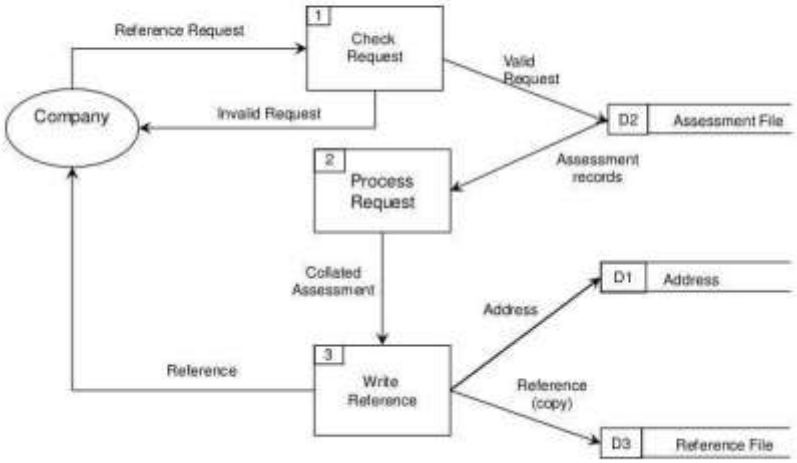
Example:

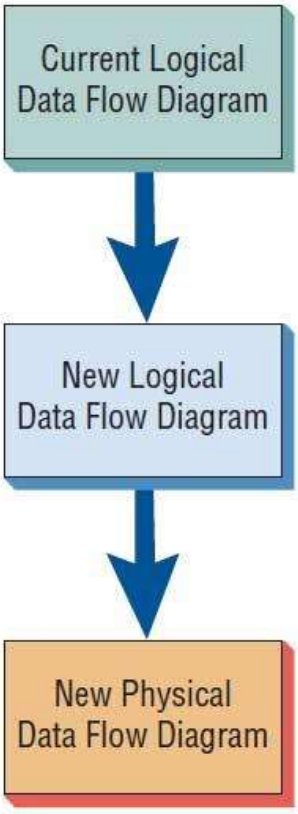
Customer enters order → system stores → kitchen receives.

(DFD used)



Level 1 DFD Example





Derive the logical data flow diagram for the current system by examining the physical data flow diagram and isolating unique business activities.

Create the logical data flow diagram for the new system by adding the input, output, and processes required in the new system to the logical data flow diagram for the current system.

Derive the physical data flow diagram by examining processes on the new logical diagram. Determine where the user interfaces should exist, the nature of the processes, and necessary data stores.

3. Behavior-Based Elements

Shows how system reacts to events.

Example:

Wrong password → Error message

---

4. Class-Based Elements

Used in object-oriented systems.

Example Classes:

- Customer
- Order
- Product
- Payment

Each class has data + functions.

Important Exam Questions

1. What is SRS? Explain contents.
2. Functional vs Non-functional requirements.
3. Explain Requirements Engineering Process.
4. What is Feasibility Study?
5. Explain LOC, FP and Use Case estimation.
6. What is COCOMO-II?
7. Explain Analysis Model elements.

---

2 Marks Ready Answers

What is Requirement?

Requirement is a need or feature expected from software.

What is SRS?

SRS is a document containing complete software requirements.

What is Feasibility Study?

It checks whether project is practical and possible.

What is LOC?

LOC means Lines of Code used for project estimation.

Quick Memory Tricks

Requirement Process:

FEAVM
Feasibility, Elicitation, Analysis, Validation, Management

Estimation:

LFU
LOC, Function Point, Use Case

Final Easy Summary

This module teaches:

- What software should do
- How to collect requirements
- How to estimate project cost/time
- How to model system before coding

Module 4: Design Engineering

Introduction

After understanding requirements, the next step is **Design**.

Design means planning **how software will be built** before coding starts.

Just like before building a house:

- First map/blueprint is made
- Then construction starts

Same in software.

What is Design Engineering?

Design Engineering means creating the structure, architecture, components, database, and interface of software in a planned way.

Easy Meaning:

Turning requirements into a software blueprint.

Example:

For an online shopping app:

- How login works
- Where product data stores
- How payment page looks
- How modules connect

All this is design engineering.

Why Design is Important?

- Saves coding time
- Reduces mistakes
- Easy maintenance
- Better performance
- Better user experience

Design Process

The design process usually happens in steps:

Requirements → Analysis → Design → Coding → Testing

In Design Stage:

1. Understand needs
2. Create system structure
3. Create database design

4. Create UI screens
5. Define modules/components

Design Principles

These are rules for making good software design.

---

1. Simplicity

Keep design simple.

Example:

Simple login page better than cluttered page.

---

2. Modularity

Break software into small parts/modules.

Example:

E-commerce app modules:

- Login
- Cart
- Payment
- Orders

Benefit:

Easy to manage.

---

3. Reusability

Use same component again.

Example:

Same button style used everywhere.

---

4. Maintainability

Software should be easy to update/fix.

Example:

Changing price formula without changing full system.

---

5. Flexibility

Easy to add future features.

Example:

Later adding dark mode.

---

6. Efficiency

Use less memory/time.

Design Concepts

Main concepts used in design.

---

1. Abstraction

Showing only important details, hiding unnecessary details.

Example:

Using car:
You use steering/brake.
You don’t need engine internal details.

In software:

User uses ATM without knowing internal code.

---

2. Refinement

Start with big idea, then break into detailed parts.

Example:

Build food app → then login module → then OTP system.

---

3. Modularity

Split system into smaller modules.

Benefit:

Easy testing and updates.

---

4. Information Hiding

Hide internal working of module.

Example:

Payment gateway internal logic hidden.

---

5. Functional Independence

Each module should do one job well.

Example:

Cart module handles cart only.

Design Model

Design model means complete plan of software design.

Includes:

1. Data Design
2. Architecture Design
3. Interface Design
4. Component-Level Design

---

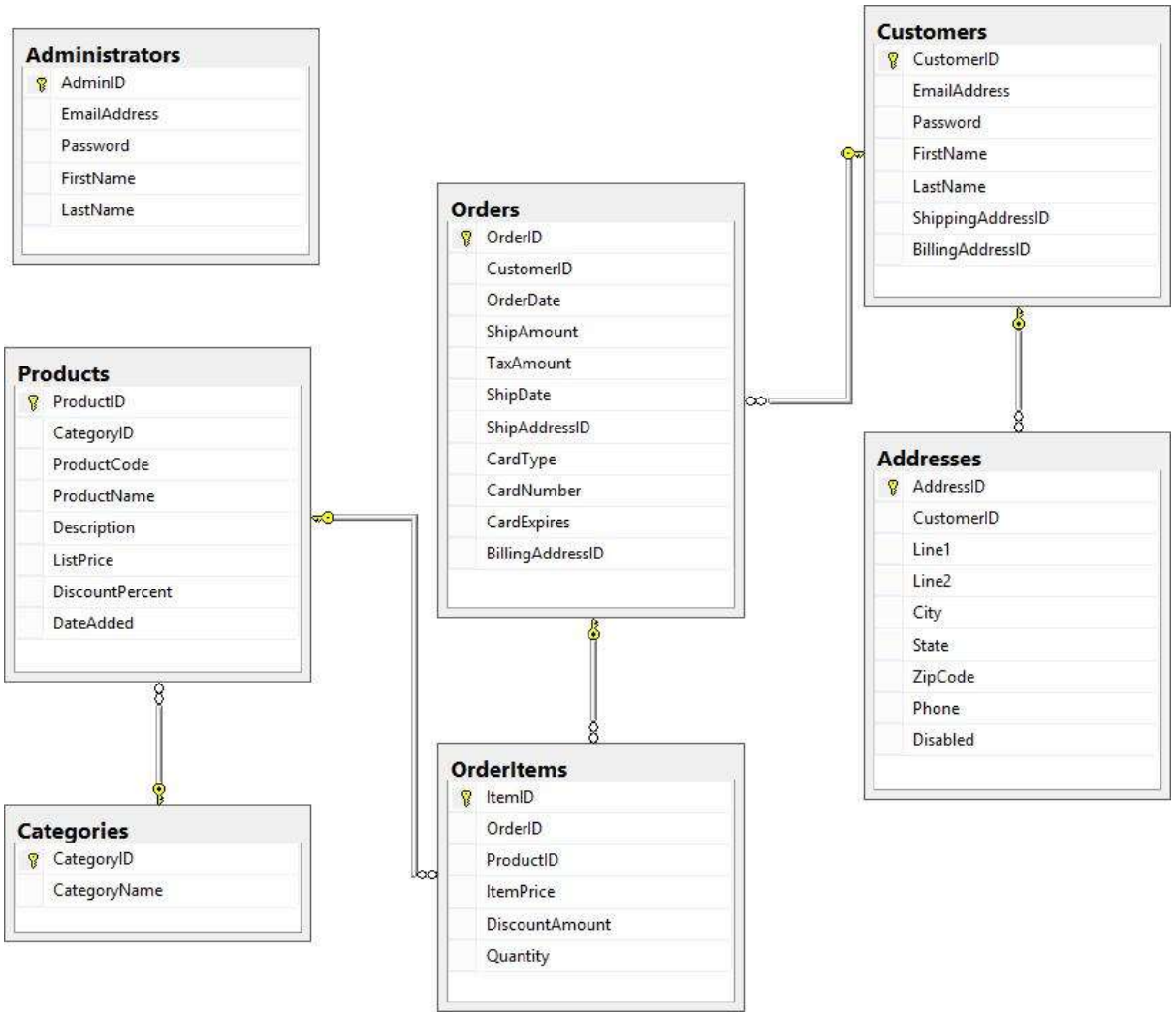
1. Data Design

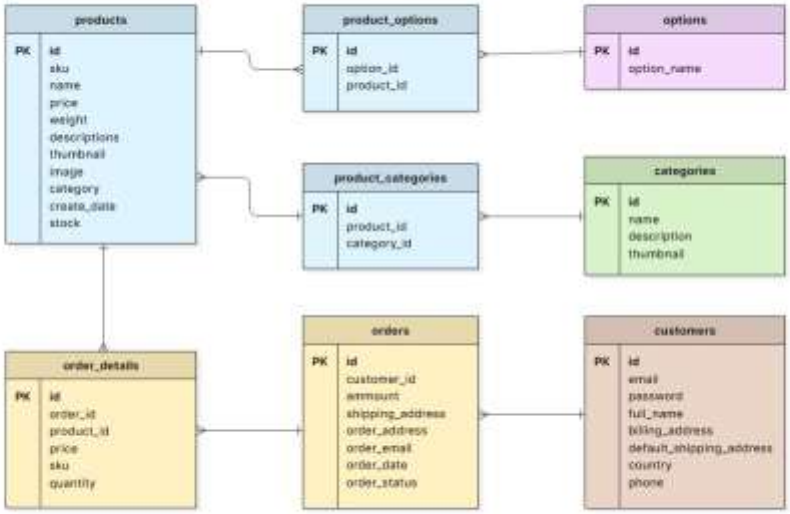
Decides how data will be stored and organized.

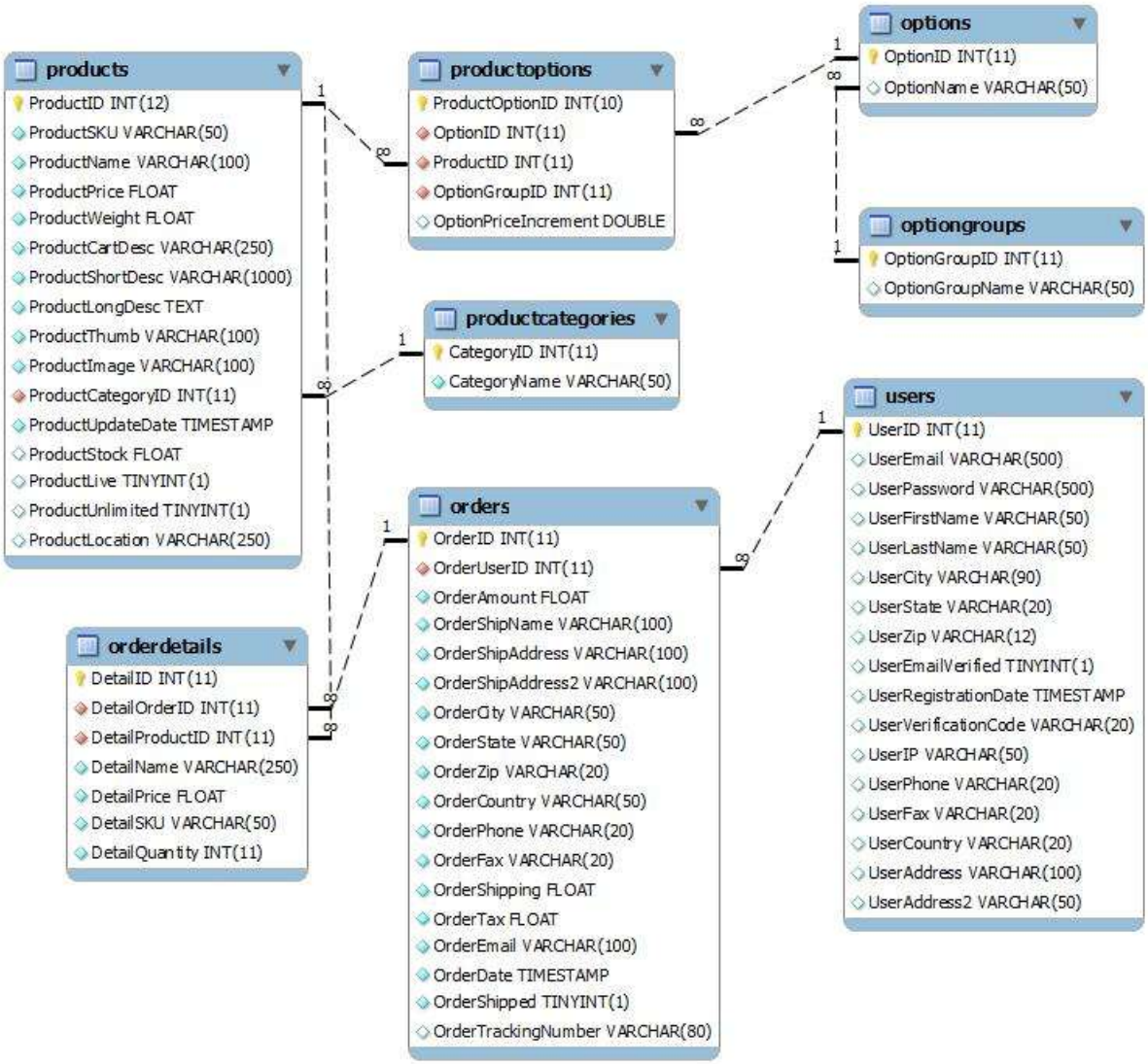
Example:

Shopping app tables:

Table	Data
Users	Name, Email
Products	Price, Stock
Orders	Item, Status




products
PK id
sku
name
price
weight
descriptions
thumbnail
image
category
create_date
stock
product_options
PK id
option_id
product_id
options
PK id
option_name
product_categories
PK id
product_id
category_id
categories
PK id
name
description
thumbnail
order_details
PK id
order_id
product_id
price
sku
quantity
orders
PK id
customer_id
ammount
shipping_address
order_address
order_email
order_date
order_status
customers
PK id
email
password
full_name
billing_address
default_shipping_address
country
phone


products
ProductID INT(12)
ProductSKU VARCHAR(50)
ProductName VARCHAR(100)
ProductPrice FLOAT
ProductWeight FLOAT
ProductCartDesc VARCHAR(250)
ProductShortDesc VARCHAR(1000)
ProductLongDesc TEXT
ProductThumb VARCHAR(100)
ProductImage VARCHAR(100)
ProductCategoryID INT(11)
ProductUpdateDate TIMESTAMP
ProductStock FLOAT
ProductLive TINYINT(1)
ProductUnlimited TINYINT(1)
ProductLocation VARCHAR(250)
productoptions
ProductOptionID INT(10)
OptionID INT(11)
ProductID INT(11)
OptionGroupID INT(11)
OptionPriceIncrement DOUBLE
options
OptionID INT(11)
OptionName VARCHAR(50)
optiongroups
OptionGroupID INT(11)
OptionGroupName VARCHAR(50)
productcategories
CategoryID INT(11)
CategoryName VARCHAR(50)
users
UserID INT(11)
UserEmail VARCHAR(500)
UserPassword VARCHAR(500)
UserFirstName VARCHAR(50)
UserLastName VARCHAR(50)
UserCity VARCHAR(90)
UserState VARCHAR(20)
UserZip VARCHAR(12)
UserEmailVerified TINYINT(1)
UserRegistrationDate TIMESTAMP
UserVerificationCode VARCHAR(20)
UserIP VARCHAR(50)
UserPhone VARCHAR(20)
UserFax VARCHAR(20)
UserCountry VARCHAR(20)
UserAddress VARCHAR(100)
UserAddress2 VARCHAR(50)
orders
OrderID INT(11)
OrderUserID INT(11)
OrderAmount FLOAT
OrderShipName VARCHAR(100)
OrderShipAddress VARCHAR(100)
OrderShipAddress2 VARCHAR(100)
OrderCity VARCHAR(50)
OrderState VARCHAR(50)
OrderZip VARCHAR(20)
OrderCountry VARCHAR(50)
OrderPhone VARCHAR(20)
OrderFax VARCHAR(20)
OrderShipping FLOAT
OrderTax FLOAT
OrderEmail VARCHAR(100)
OrderDate TIMESTAMP
OrderShipped TINYINT(1)
OrderTrackingNumber VARCHAR(80)
orderdetails
DetailID INT(11)
DetailOrderID INT(11)
DetailProductID INT(11)
DetailName VARCHAR(250)
DetailPrice FLOAT
DetailSKU VARCHAR(50)
DetailQuantity INT(11)

Goal:

Fast and safe data access.

---

2. Architecture Design

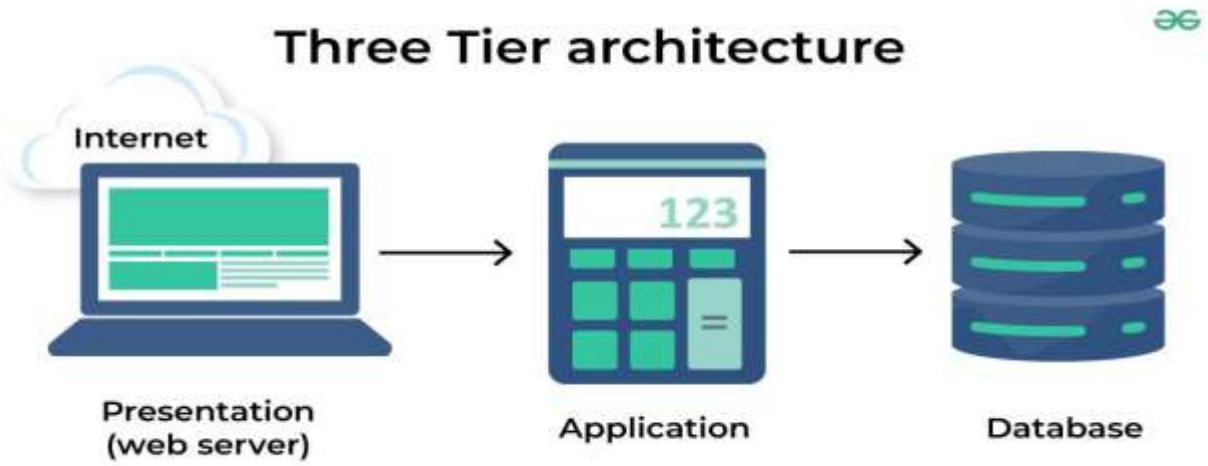
Overall structure of software.

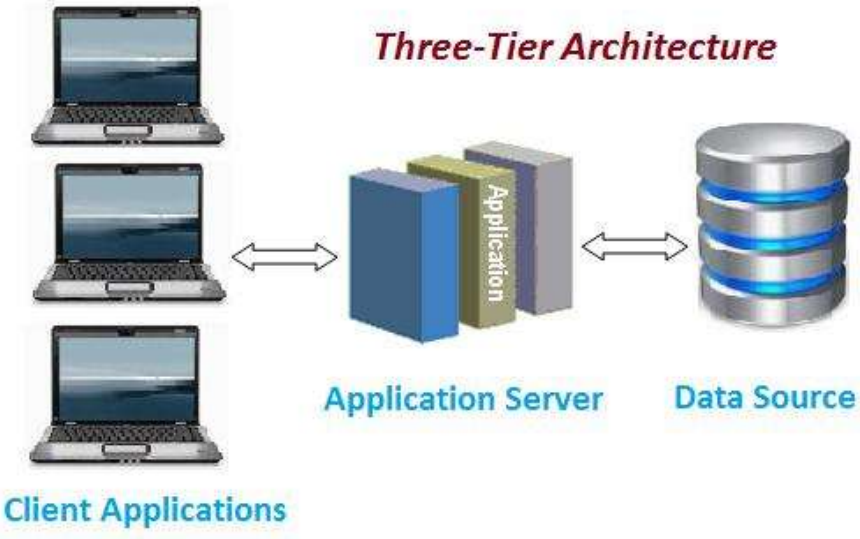
Shows how modules connect.

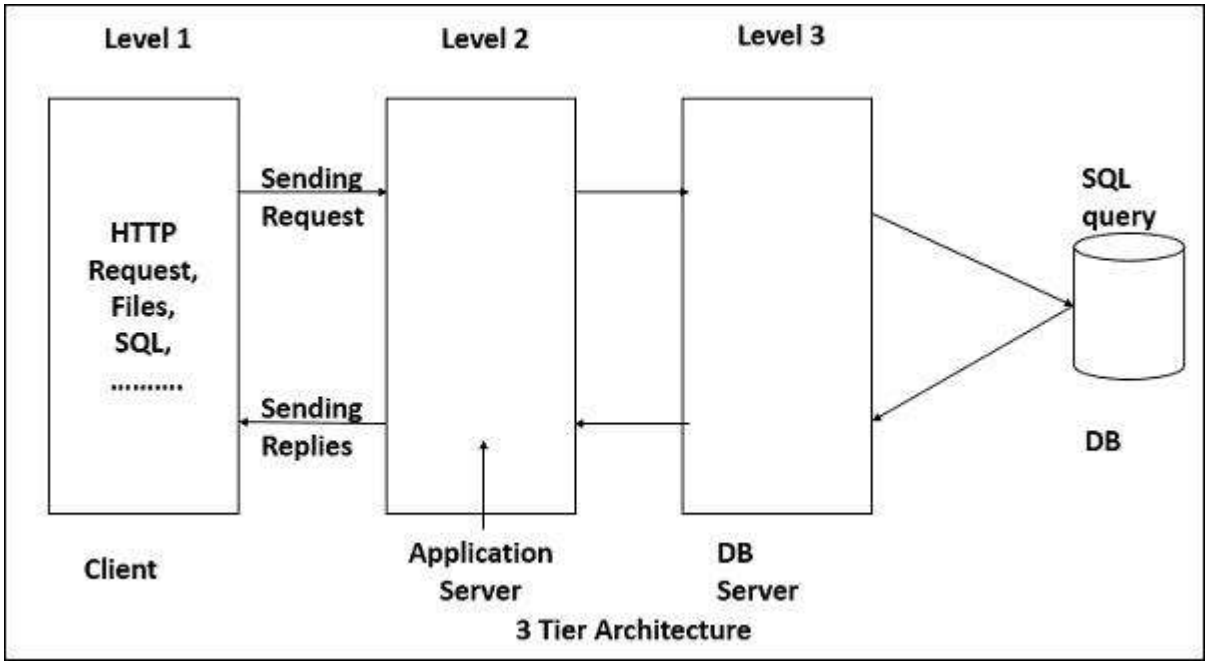
Example:

3-Tier Architecture:

Presentation Layer (UI)
Business Logic Layer
Database Layer







6

Example:

Amazon uses large scalable architecture.

---

3. User Interface Design

Designing screens user sees.

Includes:

- Buttons
- Menus
- Forms
- Colors
- Navigation

Good UI Should Be:

- Simple
- Fast
- Attractive
- Easy to understand

Example:

Instagram simple UI.

---

4. Component-Level Design

Detailed design of each module/component.

Example:

Login component:

- Input email
- Validate password
- Check database
- Allow access

Coupling and Cohesion

Very important exam topic.

What is Coupling?

Coupling means dependency between modules.

If one module heavily depends on another = high coupling.

Example:

Payment module cannot work without cart code directly linked badly.

Good or Bad?

Low coupling is good.

What is Cohesion?

Cohesion means how strongly related tasks inside one module are.

Example:

Login module only does login tasks.

Good or Bad?

High cohesion is good.

Coupling vs Cohesion

Point	Coupling	Cohesion
Meaning	Connection between modules	Relatedness inside module
Good Type	Low	High
Focus	Inter-module	Intra-module

Memory Trick:

- **Low Coupling Good**
- **High Cohesion Good**

Example to Remember

Good Design:

Login module:

- Validate email
- Check password

(High cohesion)

Works independently from payment system.

(Low coupling)

Important Exam Questions

1. Define Design Engineering.
2. Explain Design Principles.
3. What is Abstraction?
4. Explain Design Model components.
5. What is Data Design?
6. Explain Architecture Design.
7. Difference between Coupling and Cohesion.

8. What is User Interface Design?

---

2 Marks Ready Answers

What is Modularity?

Modularity means dividing software into smaller manageable modules.

What is Abstraction?

Abstraction means showing important details and hiding unnecessary details.

What is Coupling?

Coupling is the dependency between software modules.

What is Cohesion?

Cohesion is how closely related functions inside a module are.

Quick Memory Tricks

Design Model:

D A U C
Data, Architecture, UI, Component

Good Software Rule:

Low Coupling + High Cohesion

Final Easy Summary

This module teaches:

- How to plan software before coding
- How data/UI/modules are designed
- How to make clean maintainable software
- Coupling and cohesion concepts

Send **Module 5** and I'll explain it same easy exam-oriented style.

next

Module 5: Risk Management & Software Configuration Management

(Explained in very easy language for exams + understanding)

Introduction

When making software, many problems can happen:

- Project delay
- Budget increase
- Team member leaves
- Bugs increase
- Client changes requirements
- Server crashes

These possible future problems are called **Risks**.

So this module teaches:

1. How to handle risks
2. How to manage software changes/versions

PART A: Risk Management

What is Risk?

Risk means an uncertain event that may cause loss or project problems.

Example:

A developer quits in middle of project.

Important:

Risk may happen or may not happen.

Why Risk Management Needed?

- Avoid project failure
- Save money
- Finish on time

- Better planning
- Reduce surprises

Reactive vs Proactive Risk Strategies

Very important exam topic.

Strategy	Meaning	Example
Reactive	Act after problem happens	Server crashes, then fix
Proactive	Find risk early and prevent	Backup server ready

Which is Better?

Proactive is better.

Memory Tip:

- Reactive = After fire
- Proactive = Before fire

Software Risks

Common risks in software projects:

1. Project Risks

Affect schedule, cost, resources.

Example:

- Late delivery
- Lack of staff

2. Technical Risks

Affect quality/performance.

Example:

- New technology fails
- Security weakness

3. Business Risks

Affect company/project value.

Example:

- Product no market demand
- Funding stops

Risk Identification

Finding possible risks before they happen.

Methods:

- Team meetings
- Past project experience
- Checklists
- Expert advice

Example:

For shopping app:

- Payment gateway issue
- Data leak risk
- Delay risk

Assessing Overall Project Risk

Checking total danger level of project.

Questions:

- How many risks exist?
- How serious are they?
- Can team handle them?

Result:

Low / Medium / High Risk

Risk Components and Drivers

Risk drivers = factors causing risk.

Common Drivers:

- Tight deadline
- New technology
- Low budget
- Poor communication
- Weak team skills

Risk Projection

Risk Projection means estimating:

1. Probability (chance of happening)
2. Impact (damage if happens)

Example:

Risk	Probability	Impact
Developer leaves	Medium	High
Minor bug	High	Low

Developing a Risk Table

Risk table ranks risks by priority.

Risk	Probability	Impact	Priority
Security breach	Medium	Very High	1
Delay in UI	High	Medium	2
Printer issue	Low	Low	3

Use:

Solve top risks first.

Assessing Risk Impact

Impact means seriousness of damage.

Types:

- Cost increase
- Delay
- Quality drop
- Reputation damage

Example:

Data breach in Paytm = high impact.

Risk Refinement

Breaking a general risk into detailed smaller risks.

Example:

General Risk: Project Delay

Refined Risks:

- Testing delayed
- Coding slow
- Client approval late

Risk Mitigation

Mitigation means reducing risk chance/effect.

Examples:

- Backup developers
- Extra testing
- Security tools
- Clear communication

Risk Monitoring

Regularly checking whether risks are increasing.

Example:

Every week review schedule delay.

Risk Management

Complete cycle:

Identify → Analyze → Prioritize → Reduce → Monitor

RMMM Plan

Very Important

RMMM = Risk Mitigation, Monitoring, and Management Plan

This is a document that tells:

- Which risks exist
- How to reduce them
- How to track them
- What to do if they happen

Example:

If server fails → move to backup server.

PART B: Software Configuration Management (SCM)

What is SCM?

SCM means controlling and tracking software changes systematically.

Easy Meaning:

Managing versions, updates, files, code changes.

Example:

Version 1.0
Version 1.1 bug fix
Version 2.0 new features

Why SCM Needed?

- Prevent file confusion
- Track changes
- Team collaboration
- Restore old version
- Better release management

SCM Scenario Example

5 developers working on same project.

Without SCM:

- Files overwrite
- Confusion
- Lost code

With SCM:
Using GitHub / Git:

- Track commits
- Merge changes
- Version control

Elements of Configuration Management System

1. Version Control

Stores versions of code.

Example: Git

2. Change Control

Approve and manage changes.

3. Build Management

Combine code into working software.

4. Release Management

Publish final version.

5. Status Reporting

Shows what changed.

SCM Process

Identify Items → Control Changes → Record Status → Audit → Release

---

Step Explanation

1. Identify Configuration Items

Items = source code, docs, design files.

2. Control Changes

Only approved changes allowed.

3. Status Accounting

Keep records of versions.

4. Configuration Audit

Check if system matches records.

5. Release

Deliver official version.

Risk Management vs SCM

Point	Risk Management	SCM
Focus	Future problems	Software changes
Goal	Reduce uncertainty	Control versions
Example	Delay risk	Code version 2.0

Important Exam Questions

1. Define Risk in software engineering.
2. Reactive vs Proactive risk strategy.

3. Explain Risk Identification.
4. What is Risk Table?
5. Explain RMMM Plan.
6. What is SCM?
7. Explain SCM process.
8. Why is version control important?

---

2 Marks Ready Answers

What is Risk?

Risk is an uncertain future event that may negatively affect a project.

What is Proactive Risk Strategy?

It means identifying and preventing risks before they happen.

What is SCM?

Software Configuration Management is the process of managing software changes and versions.

What is RMMM?

RMMM stands for Risk Mitigation, Monitoring and Management Plan.

Quick Memory Tricks

Risk Cycle:

I A P R M
Identify, Analyze, Prioritize, Reduce, Monitor

SCM Cycle:

I C S A R
Identify, Control, Status, Audit, Release

Final Easy Summary

This module teaches:

- How to handle project risks

- How to avoid delays and failures
- How to manage code versions and software updates
- Importance of Git/version control

Module 6: Software Testing with Quality Assurance

Introduction

After software is developed, we must check whether it works correctly.

This checking process is called **Testing**.

Without testing:

- App crashes
- Wrong outputs
- Security issues
- Payment failures
- Bad user experience

So this module teaches:

1. Software Testing
2. Software Quality Assurance (SQA)

PART A: Software Testing

What is Software Testing?

Software testing means checking software to find errors and verify that it works as expected.

Easy Meaning:

Testing = Finding mistakes before users do.

Example:

Before launching Paytm, developers test login, payments, OTP, refunds.

Need for Testing

Why Testing Important?

- Finds bugs/errors
- Improves quality
- Increases security
- Better customer trust
- Reduces failure cost later

Example:

If ATM gives wrong money due to bug = serious issue.

Testing Tactics

Testing tactics means smart ways/strategies to test software.

Main Tactics:

1. Start from small units
2. Then combine modules
3. Then full system test
4. Repeat after changes

Black Box Testing

Tester checks software **without seeing internal code**.

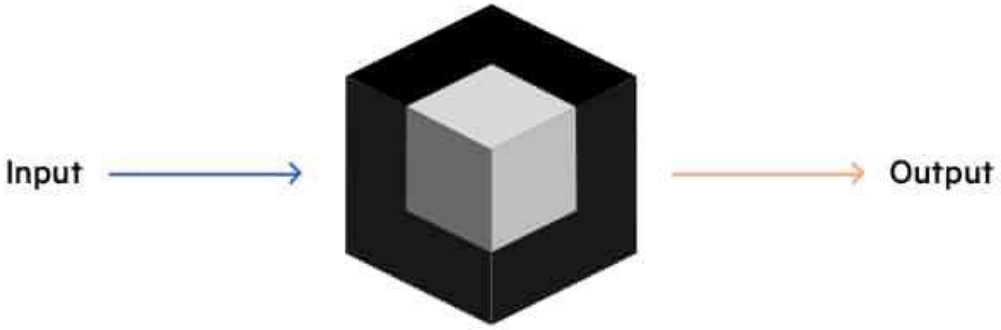
Focus on input and output.

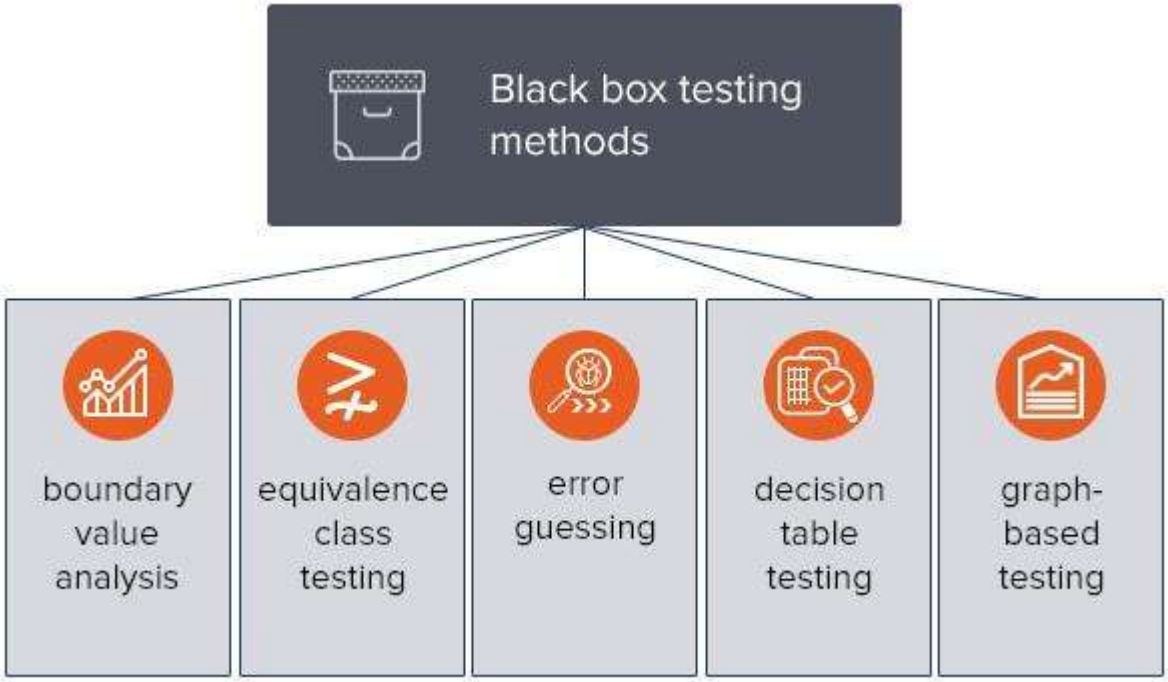
Example:

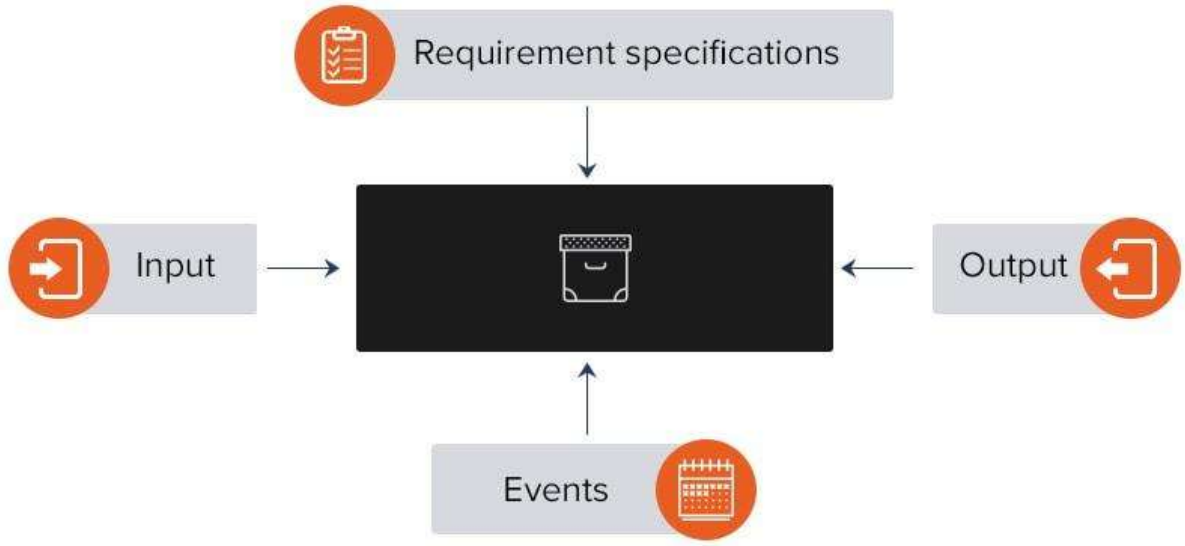
Enter wrong password → should show error.

Tester does not care how code works.

Black Box Testing







6

Used By:

QA testers, users

Equivalence Partitioning

Input data divided into groups.

Test one value from each group.

Example:

Age allowed 18–60

Groups:

- Below 18 (invalid)
- 18 to 60 (valid)
- Above 60 (invalid)

Instead of testing every age, test sample values.

Boundary Value Analysis

Errors happen at limits/boundaries.

So test edge values.

Example:

Age allowed 18–60

Test:

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

Memory Tip:

Check borders first.

White Box Testing

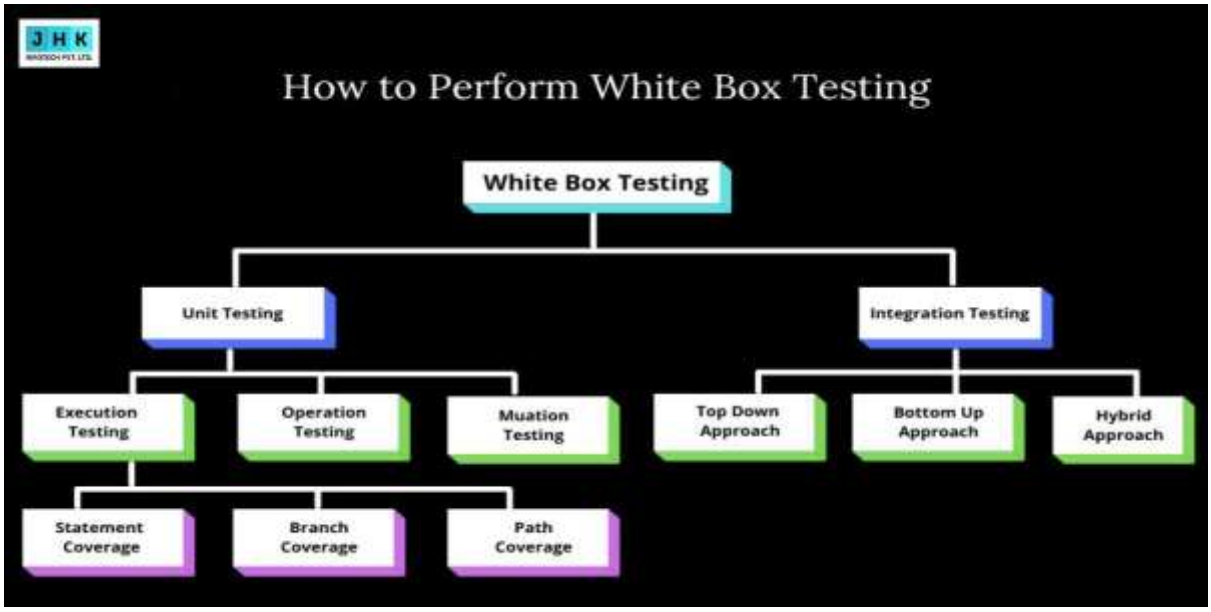
Tester checks internal code logic.

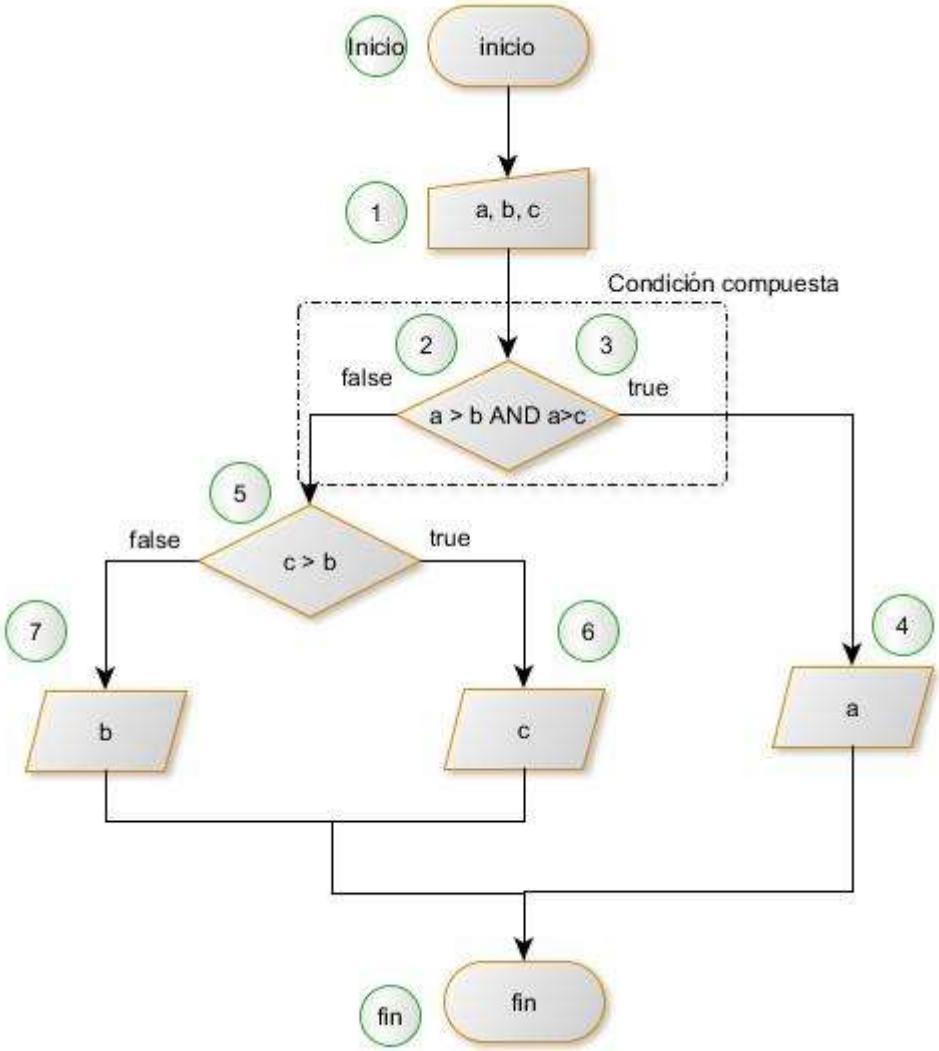
Focus on:

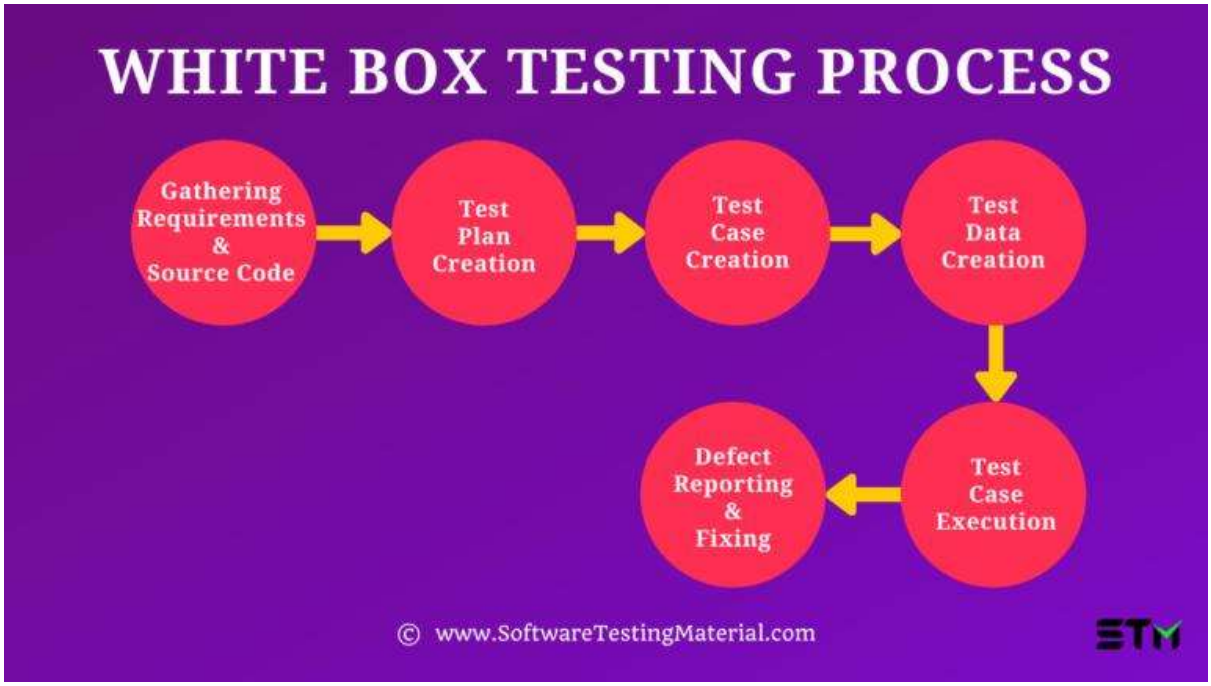
- Conditions
- Loops
- Paths
- Statements

Example:

Check all branches of login code.


How to Perform White Box Testing
White Box Testing
Unit Testing
Integration Testing
Execution Testing
Operation Testing
Muation Testing
Top Down Approach
Bottom Up Approach
Hybrid Approach
Statement Coverage
Branch Coverage
Path Coverage


Inicio
inicio
1
a, b, c
Condición compuesta
2
3
false
true
a > b AND a>c
5
false
c > b
true
7
6
4
b
c
a
fin
fin



6

Black Box vs White Box

Point	Black Box	White Box
Focus	Input/Output	Internal code
Code knowledge needed	No	Yes
Done by	Tester	Developer/Test engineer

Basis Path Testing

White-box technique.

Tests all independent logic paths in program.

Example:

If password correct → login
Else → error

Need to test both paths.

Strategic Approach of Testing

Testing usually happens in order:

Unit Testing → Integration Testing → Validation Testing → System Testing

---

1. Unit Testing

Test smallest component/module.

Example:

Login function only.

---

2. Integration Testing

Check combined modules.

Example:

Login + Database connection.

---

3. Validation Testing

Check software meets user requirements.

Example:

Can customer really place order?

---

4. System Testing

Test complete software system.

Example:

Whole shopping app end-to-end.

Test Strategies for Conventional Software

Traditional projects follow planned testing stages:

- Unit test first
- Integration next
- Full system later

- Final acceptance testing

PART B: Software Quality Assurance (SQA)

What is Quality?

Quality means software works correctly, efficiently, securely, and satisfies users.

What is SQA?

SQA = Software Quality Assurance

It is a planned process to ensure software quality during development.

Easy Meaning:

Prevent defects, not only detect defects.

Elements of SQA

Main Elements:

1. Standards
2. Reviews
3. Testing
4. Audits
5. Documentation
6. Process improvement

SQA Processes

Activities done continuously:

- Define quality standards
- Review work
- Test software
- Measure defects
- Improve process

SQA Activities

Examples:

- Requirement review
- Code review
- Test planning
- Defect tracking
- Quality reports

McCall's Quality Factors

Very Important Topic

Model used to measure software quality.

Main Factors:

Factor	Meaning
Correctness	Gives right output
Reliability	Works consistently
Efficiency	Fast and uses less resources
Integrity	Secure from unauthorized access
Usability	Easy to use
Maintainability	Easy to fix/update
Flexibility	Easy to modify
Portability	Runs on different systems
Reusability	Can reuse parts

Example:

Instagram should be usable, reliable, fast.

Case Study: Testing in FinTech Apps (Paytm / Razorpay)

Financial apps need strong testing.

---

1. Unit Testing

Test payment calculation function.

---

2. Integration Testing

Check bank API + payment app connection.

---

3. Regression Testing

After update, old features should still work.

Example:
New cashback feature added, but payment must still work.

---

4. Security Testing

Very important in finance apps.

Check:

- Hacking risks
- Data leaks
- OTP bypass
- Fraud attacks

Important Exam Questions

1. Define Software Testing.
2. Need for testing.
3. Black Box vs White Box testing.
4. Explain Equivalence Partitioning.
5. Explain Boundary Value Analysis.
6. Explain Basis Path Testing.

7. What is SQA?
8. Explain McCall's Quality Factors.
9. Explain testing strategy in FinTech apps.

---

2 Marks Ready Answers

What is Testing?

Testing is the process of checking software for errors and correctness.

What is Black Box Testing?

Testing software based on input and output without seeing internal code.

What is White Box Testing?

Testing internal code logic, paths, and conditions.

What is SQA?

Software Quality Assurance is a process to ensure software quality standards.

Quick Memory Tricks

Testing Levels:

U I V S
Unit, Integration, Validation, System

Quality Factors:

CREI UMFPR
Correctness, Reliability, Efficiency, Integrity, Usability, Maintainability, Flexibility, Portability, Reusability

---

Final Easy Summary

This module teaches:

- How to test software properly
- Different testing methods
- How to maintain software quality
- Why finance/payment apps need strong testing

Full Subject Overview

Your subject mainly covers:

1. Software basics & process
2. Development models (Waterfall, Agile)
3. Requirements & cost estimation
4. Design principles
5. Risk + SCM
6. Testing + Quality