SOFTWARE ARCHITECTURE NOTES

Q.1 Describe the various models of software development with their issues.

✅ **Answer 1: Hinglish Format (7 Marks Level)**

Software development models wo frameworks hote hain jinke through software ko plan, design, develop, test aur deliver kiya jata hai. Har model ka apna process aur apni problems hoti hain. Neeche most important models aur unke issues diye gaye hain.

---

1. Waterfall Model

Description:

Ye ek sequential model hai jisme phases ek line me aage badhti hain – Requirements → Design → Coding → Testing → Deployment.

Ek phase complete hone ke baad hi next phase start hota hai.

Issues:

- Changes ko handle karna mushkil hota hai (rigid).
- End user ko final product bahut late dekhne milta hai.
- Agar requirement galat nikli to pura project risky ho jata hai.
- Complex aur large projects ke liye slow ho jata hai.

---

2. Incremental Model

Description:

Project ko chote-chote increments ya modules me divide karke develop kiya jata hai. Har increment ek mini-version of product hota hai.

Issues:

- Good planning aur module division required hota hai.
- Integration problems aati hain jab multiple increments jodne hote hain.
- Poor architecture design se system unstable ho sakta hai.
- Time estimation accurate nahi hota.

---

3. Spiral Model

Description:

Risk-based model hai. Har phase me Planning → Risk Analysis → Engineering → Evaluation hota hai. Complex aur risky projects ke liye best.

Issues:

- Bohot costly hota hai.
- Risk analysis experienced team ke bina possible nahi.
- Small projects ke liye suitable nahi.
- Time-consuming process.

---

4. Agile Model

Description:

Iterative + collaborative process jisme software short sprints me banaya jata hai. Customer se continuous feedback liya jata hai.

Issues:

- Documentation kam hoti hai.
- Client involvement high required hoti hai.
- Scope frequently change hota rehta hai.
- Large teams me coordination tough hota hai.

---

5. V-Model (Verification & Validation Model)

Description:

Waterfall ka advanced version jisme development ke har phase ka testing phase se direct relation hota hai. V shape me activities hoti hain.

Issues:

- Changes accommodate karna mushkil.
- Early prototypes nahi milte.
- Complex aur evolving requirements ke liye suitable nahi.

⭐ 7-Marks Conclusion (Hinglish)

Har software development model ki apni strengths aur weaknesses hoti hain. Project requirement, time, cost aur complexity ke basis par model choose kiya jata hai. Agar requirements stable hain to Waterfall/V-model sahi hai, aur agar frequent changes expected hain to Agile best choice hota hai.

✅ Answer 2: English Format (7 Marks Level)

Software development models define the structured approach to plan, design, develop, test, and deliver a software system. Each model follows a unique process and faces different issues. Below are the major models and their issues.

---

1. Waterfall Model

Description:

A linear and sequential model where each phase (Requirements → Design → Coding → Testing → Deployment) is completed before moving to the next.

Issues:

- Very difficult to handle changing requirements.
- End users see the working product very late.
- High risk if initial requirements are wrong.
- Slow for large and complex systems.

---

2. Incremental Model

Description:

Development happens in small increments or modules. Each increment adds some functionality to the system.

Issues:

- Requires good planning and modular design.
- Integration issues may occur between increments.
- Poor architecture can lead to unstable system.
- Time estimation is often inaccurate.

---

3. Spiral Model

Description:

A risk-driven development model involving repeated cycles of Planning, Risk Analysis, Engineering, and evaluation. Best for large and high-risk projects.

Issues:

- Very expensive to implement.
- Requires highly skilled risk-analysis experts.
- Not suitable for small projects.
- Time-consuming process.

---

4. Agile Model

Description:

An iterative and collaborative model where software is developed in short sprints with continuous customer feedback.

Issues:

- Documentation is minimal.
- High customer involvement is needed.
- Requirements may change frequently.
- Coordination becomes difficult in large teams.

---

5. V-Model (Verification & Validation Model)

Description:

An extension of the Waterfall model where each development phase has a corresponding testing phase. All activities are arranged in a V shape.

Issues:

- Difficulty in handling requirement changes.
- No early working prototypes.
- Not suitable for dynamic or unclear requirements.

⭐ 7-Marks Conclusion (English)

Each software development model has its own advantages and limitations. The selection of a model depends on factors like project size, requirements stability, customer involvement, and risk level. Waterfall/V-model is suitable for stable requirements, whereas Agile is preferred for frequently changing and customer-driven projects.

Q.2 Explain architectural business cycle.

✅ Answer 1: Hinglish Format (7 Marks Level)

Architectural Business Cycle (ABC)

Architectural Business Cycle ek aisa concept hai jo batata hai ki **software architecture kaise banti hai, kis-kis factors se effect hoti hai, aur architecture khud baad me kisko impact karti hai**.

Simply bolen to:

Architecture ek cycle hai jisme stakeholders, business goals, technical decisions, architecture, aur final system ek-dusre ko effect karte rehte hain.

⭐ Main Components of Architectural Business Cycle

1. Stakeholders

Stakeholders wo log hote hain jinka project se directly ya indirectly interest hota hai. Jaise:

- Customers
- Developers
- Managers
- Investors
- End Users

Effect:

Har stakeholder ke alag goals aur expectations architecture ko influence karte hain.

---

2. Business Goals

Company ke objectives, budget, time, market competition → ye sab architecture ko shape dete hain.

Example: Agar market me competition zyada ho to company fast-delivery chahati hai → architecture ko simple aur quick-developable banana padta hai.

---

3. Technical Environment

Existing technologies, programming languages, frameworks, tools, hardware limitations, performance needs → ye sab architecture ko guide karte hain.

---

4. Software Architecture (Central Element)

Stakeholders + Business Goals + Technical environment ka influence milkar **final architectural design** banta hai.

Architecture me decisions include hote hain:

- Layered architecture
- Client-server
- Microservices
- Modules, components
- Interfaces, data flow, etc.

---

5. Developed System

Architecture ke base par actual **software system banaya** jata hai.

System ke qualities depend karte hain:

- Usability
- Performance
- Security
- Scalability
- Maintainability

---

6. Impact Back on Stakeholders

Jab system ready hota hai, wo phir se stakeholders ko influence karta hai:

- Customer satisfaction badhta ya ghatta hai
- Business ka profit impact hota hai
- Developers future me architecture ko improve karte hain
- Market reputation change hoti hai

Is tarah poora cycle dobara start hota hai.

⭐ Hinglish Conclusion (7-Marks)

Architectural Business Cycle ek continuous loop hai jisme **stakeholders → business needs → technical environment → architecture → final system** ek-dusre ko affect karte hain. Architecture koi one-time activity nahi hoti, balki ek ongoing cycle hoti hai jisme improvements aur feedback lagatar chalte rehte hain.

📌 Diagram

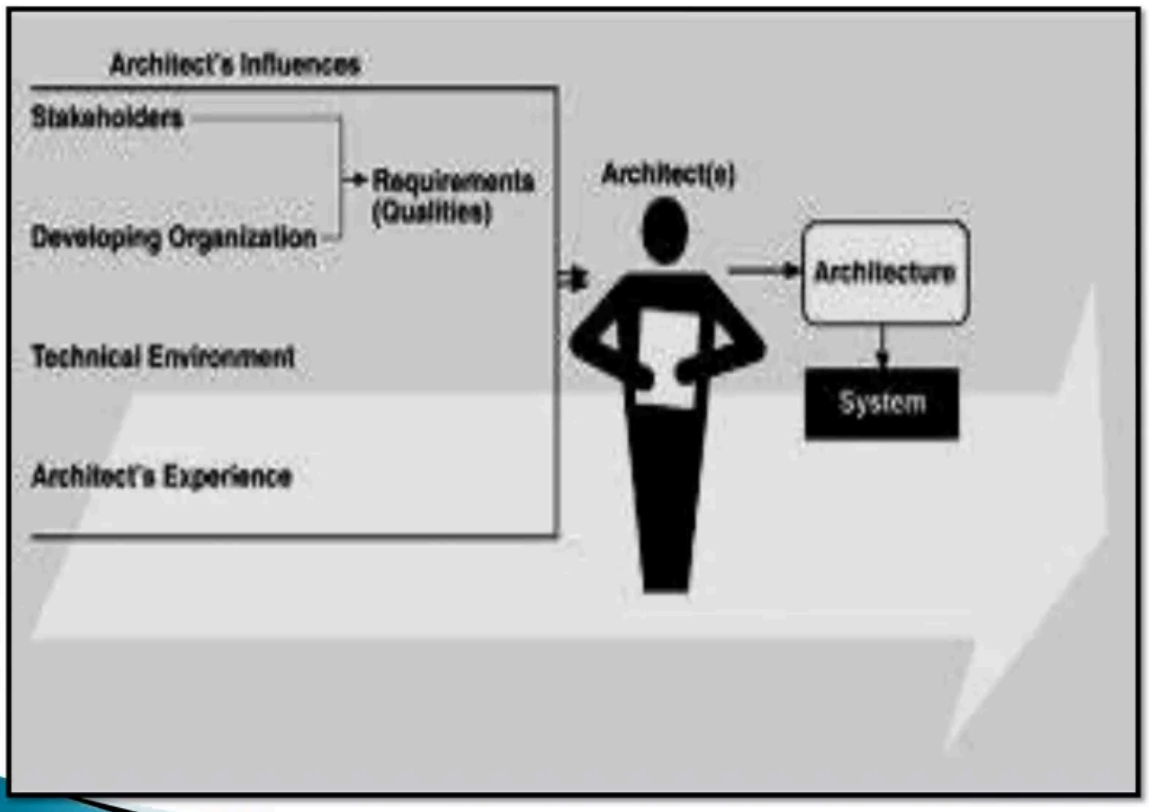


Figure 1: Influences on the architecture

5

✅ Answer 2: English Format (7 Marks Level)

Architectural Business Cycle

The **Architectural Business Cycle (ABC)** describes how software architecture is created, what factors influence it, and how architecture itself affects the final system and business environment.

It is a **continuous cycle** showing interactions between **stakeholders, business goals, technical constraints, the architecture, and the final system**.

⭐ Main Elements of the Architectural Business Cycle

1. Stakeholders

Stakeholders are individuals or groups who have an interest in the project:

- Customers
- Developers
- Project managers
- Investors
- End users

Influence:

Each stakeholder has different expectations that directly influence architectural decisions.

---

2. Business Goals

Business needs such as cost limits, deadlines, product quality, customer demands, and market competition influence the architecture.

Example:

If the business wants a quick delivery, the architect chooses a simple and fast-developable architecture.

---

3. Technical Environment

Existing technology stack, available tools, hardware constraints, platform limitations, and performance needs influence how the architecture is designed.

---

4. Software Architecture

Architecture is the **central output** of the business cycle.

It is formed based on the requirements received from

stakeholders + business goals + technical constraints.

Architecture includes decisions about:

- Layers
- Modules
- Components
- Communication patterns
- Styles (Client-server, Microservices, etc.)

---

5. Developed System

Based on the architecture, the actual software system is built.

System qualities depend on the architecture, such as:

- Performance
- Reliability
- Maintainability
- Security
- Scalability

---

6. Feedback/Impact on Stakeholders

Once the system is deployed, it affects stakeholders again:

- Customer satisfaction
- Business profits
- Developer experience
- Future improvements

This feedback creates a loop, making the cycle continuous.

⭐ English Conclusion (7-Marks)

The Architectural Business Cycle is an ongoing process where **stakeholders**, **business goals**, and **technical environment** influence the software architecture, and the architecture further influences the final system and future business decisions. It shows that software architecture is not static; it evolves continuously through feedback and changing requirements.

Q.3 compare structural model, the framework model, and dynamic model

✅ Answer 1: Hinglish Format (7 Marks Level)

Compare: Structural Model, Framework Model, and Dynamic Model

Software architecture me teen important types ke models use hote hain: **Structural Model**, **Framework Model**, aur **Dynamic Model**. Ye tino system ko different viewpoint se represent karte hain.

⭐ 1. Structural Model

Meaning:

Structural model system ka **static view** dikhata hai—yaani system ka structure, components, modules, unke relations aur data organization.

Focus:

- Components
- Subsystems
- Data flow
- Connections between modules

Use:

- High-level architecture design
- Database design
- Module-level understanding

⭐ 2. Framework Model

Meaning:

Framework model ek **reusable skeleton** provide karta hai jisme predefined classes, libraries, patterns aur components hote hain. Developers is framework ke upar system build karte hain.

Focus:

- Reusability
- Predefined architecture patterns
- Standard guidelines
- Plug-and-play components

Examples:

React, Angular, .NET, Spring Framework

⭐ 3. Dynamic Model

Meaning:

Dynamic model system ka **runtime behavior** show karta hai—yaani system kaise behave karta hai jab user interact karta hai, data flow hota hai, aur processes run hoti hain.

Focus:

- Sequence of events
- State changes
- Interactions of objects

- Runtime message passing

Use:

- Sequence diagrams
- State machine diagrams
- Activity diagrams

⭐ Comparison Table (Hinglish)

Feature	Structural Model	Framework Model	Dynamic Model
Nature	Static	Semi-static (predefined structure)	Dynamic / Runtime
Focus	Components & relations	Reusable architectural skeleton	Changing behavior of system
Represents	Modules, subsystems	Prebuilt classes & patterns	Events, states, interactions
Usage	Architecture & database planning	Fast development, industry apps	Process flow, runtime design
Tools/Diagrams	Class & component diagrams	Libraries, APIs	Sequence, Activity, State diagrams
When used	Starting phase	During implementation	Behavior analysis & testing

---

⭐ Hinglish Conclusion (7 Marks)

Structural model system ka **static architecture** show karta hai, framework model **predefined reusable structure** provide karta hai, aur dynamic model **runtime behavior** dikhata hai. Ye tino models architect ko system ko alag-alag angle se samajhne aur design karne me help karte hain.

✅ Answer 2: English Format (7 Marks Level)

Compare: Structural Model, Framework Model, and Dynamic Model

These three models represent different architectural perspectives that help in understanding and designing a complete software system.

⭐ 1. Structural Model

Meaning:

The structural model represents the **static structure** of the system, including components, modules, classes, and their relationships.

Key Focus:

- System organization
- Component interactions
- Data organization
- Subsystem structure

Used For:

- High-level architecture
- Database and class design

⭐ 2. Framework Model

Meaning:

A framework model provides a **reusable architectural template** or skeleton used to build software efficiently. It includes predefined components, patterns, and libraries.

Key Focus:

- Reusability
- Standardization
- Predefined architecture rules
- Integration of plug-in components

Examples:

Spring, Django, React, Angular, .NET

⭐ 3. Dynamic Model

Meaning:

The dynamic model describes the **runtime behavior** of the system—how components interact when the system is executing.

Key Focus:

- State changes
- Event flow
- Object interactions
- Message sequence and workflows

Used For:

- Sequence diagrams
- Activity diagrams
- State machine modeling

⭐ Comparison Table (English)

Feature	Structural Model	Framework Model	Dynamic Model
Type	Static	Semi-static	Runtime / Behavioral
Represents	Components & relationships	Reusable architecture structure	Event flow & state transitions
Main Focus	Organization of system	Reusability & predefined skeleton	System behavior during execution
Diagrams/Tools	Class, Component diagrams	APIs, Libraries	Sequence, Activity, State diagrams
Used During	Early design phase	Implementation phase	Behavior analysis & testing
Goal	Describe architecture	Speed up development	Understand runtime behavior

⭐ English Conclusion (7 Marks)

The structural model shows the **static architecture**, the framework model provides a **reusable foundation**, and the dynamic model explains the **runtime behavior**. Together, they give a complete perspective of how the system is built, organized, and behaves during execution.

Q.4 what is the importance of call and return architecture?

✅ Answer 1: Hinglish Format (7 Marks Level)

Importance of Call and Return Architecture

Call and Return Architecture ek aisa traditional architectural style hai jisme software system ko multiple functions, procedures, ya modules me divide kiya jata hai. Ek module dusre module ko “call” karta hai aur job complete hone ke baad “return” hoti hai.

Is architecture ka use almost sabhi programming languages (C, Java, Python) aur operating systems me hota hai.

⭐ Importance (7-Marks Points)

1. Simple and Easy to Understand

Is architecture ka structure bohot simple hota hai. Modules ek sequence me calls karte hain, jisse beginners aur developers ko design samajhna easy hota hai.

---

2. Modular Development Support

Call and return style me system ko **independent modules** me divide kiya ja sakta hai.

Isse:

- Code reuse hota hai
- Maintenance easy hota hai
- Development faster hoti hai

---

3. Easy Debugging and Testing

Har module separately test kiya ja sakta hai. Agar bug aata hai to pata chal jata hai ki **kaun sa module call hua tha aur kis module se error aaya**.

---

4. Good for Layered Architecture

Ye layered design ko naturally support karta hai.

Example:

- Presentation layer → Business layer → Data layer

Each layer calls the next layer, making system organized.

---

5. Supports Top-Down and Bottom-Up Design

Call-return architecture software development ke dono popular approaches support karta hai:

- **Top-down**: High-level functions niche wale modules ko call karte hain
- **Bottom-up**: Chote modules ko combine karke big module banaya jata hai

---

6. Easy to Maintain and Modify

Agar koi functionality change karni ho to sirf us module ko update karo jise call kiya ja raha hai.

Baaki system par impact bohot kam hota hai.

---

7. Suitable for Most Programming Languages

Almost har programming language functions/procedures support karti hai, isliye call-return architecture **universally acceptable** hai.

⭐ Hinglish Conclusion (7 Marks)

Call and return architecture system ko **modular**, **maintainable**, **easy to debug**, aur **simple** banata hai. Ye layered systems, traditional programming, aur large applications ke liye ek reliable architectural approach hai. Iska use almost har software system me dekhne ko milta hai.

✅ Answer 2: English Format (7 Marks Level)

Importance of Call and Return Architecture

Call and Return Architecture is a fundamental architectural style where a program is divided into multiple procedures or modules, and each module calls another to perform a specific task and returns control after completion. It is widely used in conventional programming and layered systems.

⭐ Importance (7-Marks Points)

1. Simple and Clear Design

This architecture is easy to understand because modules call each other in a clear and structured way. Developers can easily navigate and understand program flow.

---

2. Supports Modular Development

The entire system can be divided into small, independent modules. This allows:

- Reusability
- Parallel development
- Clean design
- Faster implementation

---

3. Easy Debugging and Testing

Since each module performs a specific function, it can be individually tested. When errors occur, it is easy to trace which module was called and where the problem occurred.

---

4. Supports Layered Architecture

It naturally fits into layered system design.

Example:

- UI layer
- Business layer

- Data layer

Each layer calls the next layer, improving structure and maintainability.

---

5. Useful in Top-Down and Bottom-Up Approaches

Developers can follow either design method:

- **Top-down design:** Main function calls lower-level modules.
- **Bottom-up design:** Small modules are combined to form higher-level functions.

---

6. Easy Maintenance and Upgradation

Changing or modifying a feature becomes easier because only the specific called function needs modification without affecting the entire codebase.

---

7. Language and Platform Independent

Almost every programming language supports the concept of function calls.

Therefore, this architecture is **universal** and widely used.

⭐ English Conclusion (7 Marks)

Call and Return Architecture is important because it enables a system to be modular, maintainable, easy to debug, and simple to understand. It supports layered design, structured programming, and remains one of the most commonly used architectural styles in software development.

Q.5 How Node.js is different from Angular.js ?

✅ Answer 1: Hinglish Format (7 Marks Level)

How Node.js is different from Angular.js?

Node.js aur Angular.js dono JavaScript technologies hain, par dono ka purpose, working, aur use-case bilkul different hota hai. Node.js mainly **server-side** kaam karta hai, jabki Angular.js **client-side** me use hota hai.

⭐ 1. Purpose

- **Node.js:** Server-side JavaScript runtime environment.
- **Angular.js:** Client-side JavaScript framework for building dynamic web UIs.

⭐ 2. Where They Run

- **Node.js:** Server par run hota hai (backend).
- **Angular.js:** Browser par run hota hai (frontend).

⭐ 3. Type of Technology

- **Node.js:** Platform + runtime environment
- **Angular.js:** MVC-based frontend framework

⭐ 4. Main Use

- **Node.js:** API creation, server building, backend logic, databases se connect karna.
- **Angular.js:** Single Page Applications (SPA), dynamic forms, data-binding, UI rendering.

⭐ 5. Programming Style

- **Node.js:** Asynchronous, event-driven programming.
- **Angular.js:** Two-way data binding aur MVC pattern use karta hai.

⭐ 6. Modules & Packages

- **Node.js:** NPM (Node Package Manager) use karta hai.
- **Angular.js:** Uses built-in directives, controllers, and services.

⭐ 7. Language Used

- **Node.js:** Only JavaScript on server.
- **Angular.js:** HTML templates + JavaScript for UI.

⭐ Comparison Table (Hinglish)

Feature	Node.js	Angular.js
Runs on	Server	Browser
Use-case	Backend development	Frontend development
Type	Runtime environment	Framework
Pattern	Event-driven	MVC-based
Main Feature	Fast, scalable servers	Dynamic UI, two-way binding
Package System	NPM	Built-in directives
Best For	APIs, real-time apps	SPAs, dynamic interfaces

⭐ Hinglish Conclusion (7 Marks)

Node.js backend ke liye fast, scalable aur efficient environment deta hai, jabki Angular.js frontend ke liye dynamic, interactive, aur responsive UI banane ke kaam

aata hai. Dono JavaScript world me important hain, par dono ka role bilkul alag hai.

✅ Answer 2: English Format (7 Marks Level)

How Node.js is different from Angular.js?

Node.js and Angular.js are both JavaScript-based technologies, but they are completely different in terms of use, execution, and purpose. Node.js works on the **server-side**, while Angular.js works on the **client-side**.

⭐ 1. Purpose

- **Node.js:** Used for server-side development.
- **Angular.js:** Used for creating dynamic client-side interfaces.

⭐ 2. Execution Environment

- **Node.js:** Runs on the server.
- **Angular.js:** Runs inside the web browser.

⭐ 3. Type of Technology

- **Node.js:** A runtime environment built on Chrome's V8 engine.
- **Angular.js:** A front-end JavaScript MVC framework.

⭐ 4. Usage

- **Node.js:** Backend logic, APIs, connecting to databases, real-time apps.
- **Angular.js:** Single Page Applications (SPA), UI rendering, data binding.

⭐ 5. Programming Approach

- **Node.js:** Asynchronous, event-driven.
- **Angular.js:** Uses MVC structure with two-way data binding.

⭐ 6. Modules & Packages

- **Node.js:** Uses NPM for thousands of modules.
- **Angular.js:** Uses built-in directives, filters, controllers.

⭐ 7. Development Focus

- **Node.js:** Performance, scalability on backend.
- **Angular.js:** User experience and interactive UI on frontend.

⭐ Comparison Table (English)

Feature	Node.js	Angular.js
Environment	Server-side	Client-side
Technology Type	Runtime platform	JS Framework
Design Pattern	Event-driven	MVC
Used For	APIs, server apps	Single Page Applications
Runs In	Server (OS level)	Browser
Main Strength	Fast I/O operations	Dynamic UI rendering
Packages	NPM	Built-in components

⭐ English Conclusion (7 Marks)

Node.js is ideal for creating fast and scalable server-side applications, while Angular.js is suitable for building interactive and dynamic front-end UI. Both are essential in modern web development but serve completely different roles.

Q.6 Define JSP. Describe the life cycle of servlets.

✅ Answer 1: Hinglish Format (7 Marks Level)

1. Define JSP

JSP (Java Server Pages) ek server-side technology hai jo **dynamic web pages** banane ke liye use hoti hai. JSP file HTML, CSS aur Java code ka combination hoti hai. Jab browser request karta hai, JSP page server par Java Servlet me convert ho jata hai aur dynamic output generate karta hai.

Hinglish Definition:

JSP ek technology hai jisme hum HTML ke andar Java code likh kar dynamic web content generate karte hain. JSP pages server par Servlet me convert hote hain aur fir user ko response dete hain.

✅ 2. Describe the Life Cycle of Servlets

Servlet ka life cycle servlet container (Tomcat, GlassFish, JBOSS) manage karta hai. Servlet ke 3 main life cycle methods hote hain:

⭐ 1. Loading and Instantiation

- Servlet container servlet class ko load karta hai.
- Fir uska **ek hi object** (instance) banaya jata hai.
- Ye kaam application start hote hi ya pehli request par hota hai.

⭐ 2. Initialization (init() method)

- Container servlet ka **init()** method call karta hai.
- Ye method sirf **ek baar** call hota hai.
- Isme resources initialize kiye jaate hain, jaise database connection, config parameters, etc.

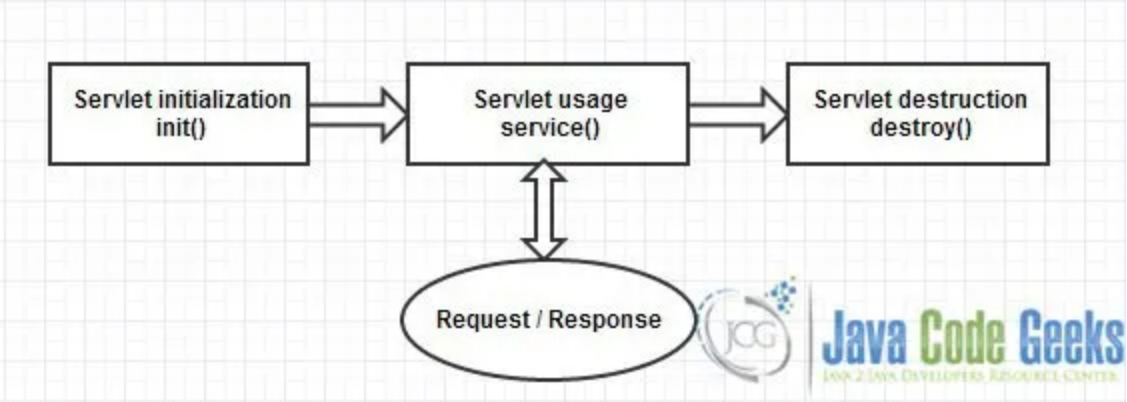
⭐ 3. Request Handling (service() method)

- Jab bhi user request bhejta hai, servlet ka **service()** method execute hota hai.
- Ye HTTP request ke type ke according **doGet()** ya **doPost()** call karta hai.
- Har request ke liye service() bar-bar call hota hai.

⭐ 4. Destruction (destroy() method)

- Jab server band hota hai ya servlet unload hota hai, **destroy()** method call hota hai.
- Isme resources close kiye jaate hain, jaise files, database connections, threads, etc.

⭐ Servlet Life Cycle Diagram



⭐ Hinglish Conclusion (7 Marks)

Servlet ka life cycle 4 steps me hota hai: **loading, init(), service(), destroy()**. Servlet container complete control rakhta hai. JSP dynamic web pages banane ki technique hai jo internally servlet me convert hoti hai.

✅ Answer 2: English Format (7 Marks Level)

1. Define JSP

JSP (Java Server Pages) is a server-side technology used to create **dynamic web pages**. A JSP file contains HTML along with Java code. When a client sends a request, the JSP file is converted into a servlet and executed to generate dynamic output.

English Definition:

JSP is a technology used to embed Java code inside HTML to generate dynamic content. JSP pages are automatically translated into servlets by the server.

✅ 2. Life Cycle of Servlets

The life cycle of a servlet is managed by the servlet container (like Apache Tomcat). A servlet goes through the following phases:

⭐ 1. Loading and Instantiation

- The servlet container loads the servlet class into memory.
- It creates **a single instance** of the servlet.
- This happens at application startup or at the first request.

⭐ 2. Initialization (init() method)

- The container calls the **init()** method.
- This method executes **only once** during the servlet's lifetime.
- Used to initialize resources such as database connections and configuration settings.

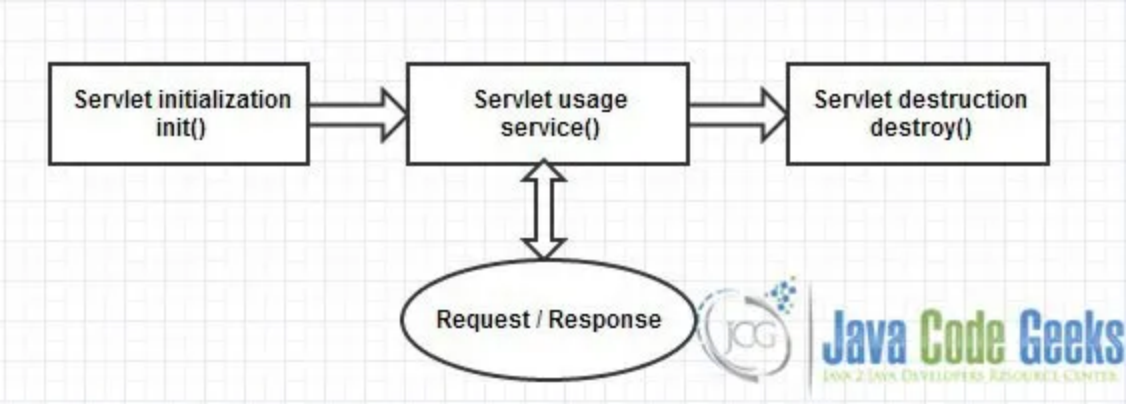
⭐ 3. Request Handling (service() method)

- For every incoming request, the container calls the **service()** method.
- Based on the request type, service() calls **doGet()** or **doPost()**.
- This method is executed **multiple times** for multiple requests.

⭐ 4. Destruction (destroy() method)

- When the server shuts down or the servlet is removed, the **destroy()** method is called.
- Used to release resources, close files or database connections.

⭐ Life Cycle Diagram



⭐ English Conclusion (7 Marks)

Servlet life cycle consists of class loading, initialization, request processing, and destruction using **init(), service(), and destroy()** methods. JSP is used to build dynamic pages and works internally as a servlet.

Q.7 Define ARID with its characteristics.

✅ Answer 1: Hinglish Format (7 Marks Level)

Define ARID

ARID (Active Reviews for Intermediate Designs) ek architecture evaluation method hai jiska use **design ke intermediate stage** par kiya jata hai.

Ye method design reviewers, stakeholders aur architects ko involve karta hai taaki system ki **accuracy, completeness, risks, aur achhe design decisions** identify kiye ja sake.

Hinglish Definition:

ARID ek review technique hai jisme software architecture ko uske beech ke design stage par actively review kiya jata hai taaki problems early detect ho sakein aur architecture strong ban sake.

⭐ Characteristics of ARID (7-Marks Points)

1. Early Design Evaluation

ARID ko system ke complete hone ka wait nahi hota.

Ye **intermediate design** ko hi review karta hai, jisse early issues pakde jaate hain.

---

2. Active Participation

Stakeholders, reviewers, developers, testers — sab actively participate karte hain.

Ye method brainstorming jaisa hota hai.

---

3. Scenario-Based Evaluation

ARID me **use-case scenarios** ko lekar design evaluate kiya jata hai.

Example:

“How will the system react if 10,000 users login at once?”

---

4. Identifies Risks and Gaps

Design me jo bhi **missing elements, risks, performance issues** ya architecture flaws hote hain, ARID unhe early stage me identify kar deta hai.

5. Cost-Effective

Early design review se late-stage bugs aur rework nahi hota.

Isliye ARID **time aur cost dono bachata hai**.

6. Useful for Incomplete Designs

ARID ki khas baat ye hai ki **design fully complete hona zaroori nahi**.

Even 50-60% complete design bhi ARID se review ho sakta hai.

7. Improves Communication

Architecture reviewers aur designers ke beech strong communication hota hai.

Ye misunderstandings ko kam karta hai aur design clarity improve karta hai.

⭐ Hinglish Conclusion (7 Marks)

ARID ek effective architectural review method hai jo **intermediate design**, **risks**, **scenarios**, aur **stakeholder feedback** ko consider karke software architecture ko strong aur reliable banata hai.

✅ Answer 2: English Format (7 Marks Level)

Define ARID

ARID (Active Reviews for Intermediate Designs) is an architectural evaluation method used to review a design **during its intermediate stages**. It helps architects and stakeholders identify flaws, gaps, risks, and improvement areas before the architecture is finalized.

English Definition:

ARID is an early-stage architecture review method that actively evaluates intermediate designs using scenarios and stakeholder participation.

⭐ Characteristics of ARID (7-Marks Points)

1. Early Design Review

ARID reviews the architecture when it is only partially complete. This allows early detection of problems.

---

2. Active Stakeholder Participation

It involves architects, developers, reviewers, managers, and even users who actively participate in design discussion and evaluation.

---

3. Scenario-Driven Approach

Real-life usage scenarios are used to check whether the design can handle expected operations effectively.

---

4. Identifies Design Risks

ARID helps discover architectural risks, incomplete aspects, bottlenecks, and weaknesses early in development.

---

5. Reduces Cost and Rework

Early identification of issues reduces later rework, making ARID cost-effective and time-efficient.

6. Works with Incomplete Designs

ARID can be applied even when the architecture is not fully ready. It is suitable for reviewing **partial or intermediate designs**.

7. Improves Understanding and Communication

ARID creates a platform for better communication among team members, ensuring everyone understands the design clearly.

⭐ English Conclusion (7 Marks)

ARID is a powerful architecture review approach that evaluates intermediate designs using scenarios and active participation. It detects risks early, reduces cost, improves design quality, and ensures that the architecture is aligned with stakeholder needs.

Q.8 Explain software architecture analysis methods in detail.

✅ Answer 1: Hinglish Format (7 Marks Level)

Explain Software Architecture Analysis Methods in Detail

Software architecture analysis methods ka purpose hota hai **architecture ko evaluate karna**, taaki hum design ke risks, weaknesses, performance issues, aur future problems ko early stage par hi identify kar saken. Ye methods ensure karte

hain ki architecture **reliable, scalable, secure** aur **business requirements** ko satisfy kare.

In sab me se sabse important methods niche explain kiye gaye hain:

⭐ 1. ATAM (Architecture Tradeoff Analysis Method)

ATAM sabse famous analysis method hai.

Ye **quality attributes** jaise performance, security, usability, modifiability, reliability ki trade-offs ko analyze karta hai.

Features:

- Stakeholder involvement
- Scenario-based evaluation
- Risks, sensitivity points aur trade-offs identify karna
- Architecture ki strengths aur weaknesses evaluate karna

Example:

High performance chahiye? → trade-off: hardware cost badh jayega

High security chahiye? → performance thodi kam ho sakti hai

⭐ 2. SAAM (Software Architecture Analysis Method)

SAAM ka main focus **modifiability** aur **maintainability** par hota hai.

Features:

- Change scenarios create kiye jaate hain
- Evaluate kiya jata hai ki system future changes ko kitni easily handle karega
- Design complexity aur coupling check hoti hai

Use:

Long-term projects jisme frequent updates expected ho.

⭐ 3. ARID (Active Reviews for Intermediate Designs)

ARID partial ya intermediate design ko review karne ke liye use hota hai.

Features:

- Design complete hone ke pehle evaluation
- Active stakeholder participation
- Scenario-based checking
- Early design flaws identify karta hai

⭐ 4. CBAM (Cost Benefit Analysis Method)

CBAM quality attributes ka **cost vs benefit** check karta hai.

Features:

- Architecture decisions ka financial cost estimate karta hai
- Har decision ka business value evaluate karta hai
- Best option choose karna jisme benefit high aur cost low ho

Use:

Enterprises jahan architecture decisions budget se linked hote hain.

⭐ 5. Scenario-Based Analysis

Is method me **scenarios** use karke architecture ko test kiya jata hai.

Types:

- Use-case scenarios
- Growth scenarios
- Stress/load scenarios

Use:

Check karna ki architecture real-world situations me kaise behave karegi.

⭐ 6. Performance Analysis

Ye method system ki performance evaluate karta hai.

Checks:

- Response time
- Throughput
- Load handling
- Bottleneck analysis
- Latency

Use:

Large-scale web apps & real-time systems.

⭐ 7. Simulation and Modeling

Architecture ke models banake simulation tools se test kiya jata hai.

Features:

- Expected performance predict hoti hai
- Behavior under load simulate hota hai
- System ka prototype banane ki zaroorat nahi

⭐ Hinglish Conclusion (7 Marks)

Software architecture analysis methods jaise **ATAM, SAAM, ARID, CBAM, performance analysis, simulation** architecture ki quality, reliability, maintainability, aur efficiency ensure karte hain. Ye methods early stage me hi risks identify kar dete hain, jisse system strong aur cost-effective ban jata hai.

✅ Answer 2: English Format (7 Marks Level)

Explain Software Architecture Analysis Methods in Detail

Software architecture analysis methods are used to evaluate and verify whether the architecture meets the required **quality attributes** such as performance, security, modifiability, scalability, and usability. These methods identify risks, weaknesses, and improvement areas early in the design phase.

Below are the most important architecture analysis methods:

⭐ 1. ATAM (Architecture Tradeoff Analysis Method)

ATAM is the most widely used analysis method. It focuses on evaluating the **trade-offs between quality attributes**.

Key Features:

- Involves stakeholders
- Uses scenarios to test architecture

- Identifies risks, trade-offs, and sensitivity points
- Evaluates performance, security, modifiability, usability, etc.

⭐ 2. SAAM (Software Architecture Analysis Method)

SAAM mainly evaluates **modifiability** and **maintainability** of the architecture.

Key Features:

- Uses change scenarios
- Measures how architecture handles future modifications
- Identifies complexity and tight coupling

⭐ 3. ARID (Active Reviews for Intermediate Designs)

ARID is used to evaluate the **intermediate or incomplete design**.

Key Features:

- Active participation of stakeholders
- Scenario-based analysis
- Detects design flaws early
- Does not require complete architecture

⭐ 4. CBAM (Cost Benefit Analysis Method)

CBAM evaluates the **cost and economic impact** of architectural decisions.

Key Features:

- Calculates business value of each architectural option
- Compares cost vs benefit
- Helps choose cost-effective architectural strategies

⭐ 5. Scenario-Based Analysis

This method evaluates architecture using different types of scenarios.

Types of Scenarios:

- Use-case scenario
- Growth scenario
- Failure scenario
- Stress/load scenario

Purpose:

To check real-world behavior under different workloads.

⭐ 6. Performance Analysis

Used to evaluate the performance characteristics of the architecture.

Checks:

- Response time
- Throughput
- Resource utilization
- Bottlenecks

Used For:

Large enterprise systems and real-time applications.

⭐ 7. Simulation and Modeling

Architecture is modeled and tested using simulation tools to predict behavior.

Key Features:

- Predicts performance
- Tests load behavior
- No need to build full prototype

⭐ English Conclusion (7 Marks)

Software architecture analysis methods such as **ATAM, SAAM, ARID, CBAM, performance evaluation, and scenario-based analysis** ensure that the architecture is reliable, cost-effective, modifiable, and performs well under real-world conditions. These methods improve quality and reduce risks at the early stages of development.

Q.9 Define refinements. Explain the principles of sound documentation.

✅ Answer 1: Hinglish Format (7 Marks Level)

Define Refinements

Refinement ka matlab hota hai **system ke design ko step-by-step aur zyada detail me convert karna**.

Jab hum kisi high-level architecture ko break karke usme choti-choti details, modules, interfaces, aur algorithmic steps add karte hain, ise refinement bolte hain.

Hinglish Definition:

Refinement ek process hai jisme high-level design ko progressively detail me convert kiya jata hai taaki final implementation clear aur correct ho.

Example:

High-level module → sub-modules → functions → code structure

(Yeh sab refinement ke steps hain.)

⭐ Explain the Principles of Sound Documentation

Software documentation ko “sound” tab maana jata hai jab wo **clear, correct, useful, maintainable** ho.

Neeche software architecture ki **sound documentation ke principles** explain kiye gaye hain:

⭐ 1. Completeness (Poora hona)

Documentation me sari necessary information honi chahiye —

modules, interfaces, responsibilities, constraints, data flow, design decisions, trade-offs.

Incomplete documentation se misunderstanding hoti hai.

⭐ 2. Consistency (Ek jaisa hona)

Terminology, symbols, diagrams, naming conventions har jagah same hone chahiye.

Agar documentation inconsistent ho, to confusion badh jata hai.

⭐ 3. Correctness (Sahi hona)

Jo system actual me bana hai, documentation exactly wahi show kare.

Galat information developers ko misguide karti hai.

⭐ 4. Clarity and Simplicity (Saaf aur simple)

Documentation ko simple language me likhna chahiye, unnecessary technical jargon avoid karna chahiye.

Clear diagrams aur examples use karna chahiye.

⭐ 5. Traceability (Track karna easy ho)

Requirements → Design → Code → Testing

Sab ek-dusre se linked hone chahiye.

Yeh ensure karta hai ki koi requirement miss na ho.

⭐ 6. Organization and Structure

Documentation acchi tarah organize honi chahiye:

- Sections
- Headings
- Subheadings
- Diagrams
- Tables

Isse padna aur samajhna easy hota hai.

⭐ 7. Modifiability / Maintainability

Documentation ko update karna easy hona chahiye.

Agar design change ho to documentation easily update ho sake.

⭐ 8. Use of Standard Notations

UML diagrams, standard symbols, architectural patterns —

standard notations documentation ko universally understandable banate hain.

⭐ Hinglish Conclusion (7 Marks)

Refinement design ko detail me convert karne ka process hai. Sound documentation ke principles — completeness, correctness, consistency, clarity, traceability, organization, aur standard notations — ensure karte hain ki architecture samajhne, maintain karne aur implement karne me easy ho.

✅ Answer 2: English Format (7 Marks Level)

Define Refinements

Refinement refers to the process of transforming a high-level architectural design into a more detailed and precise design.

It involves breaking down components into subcomponents, specifying interfaces, adding detailed behavior, and reducing abstraction gradually.

English Definition:

Refinement is the step-wise elaboration of a high-level design into more detailed and concrete specifications.

⭐ Principles of Sound Documentation

Sound documentation follows principles that ensure the architecture is **understandable, correct, and maintainable**.

The main principles are:

⭐ 1. Completeness

Documentation must include all important architectural details such as components, interfaces, constraints, data flow, and design decisions.

⭐ 2. Consistency

Terminology, naming conventions, diagrams, and patterns should remain consistent throughout the document to avoid confusion.

⭐ 3. Correctness

The documentation should accurately represent the implemented or intended system. Incorrect details lead to development errors.

⭐ 4. Clarity and Simplicity

Documentation should be written in simple, clear language.

Diagrams must be neat, readable, and easy to understand.

⭐ 5. Traceability

Each element of the documentation should be traceable to requirements, design elements, implementation, and test cases.

Traceability reduces the chances of missing requirements.

⭐ 6. Good Organization

Documentation must be structured logically with headings, sections, tables, diagrams, and references.

A well-organized document improves readability.

⭐ 7. Modifiability

Documentation must be easy to update whenever the architecture changes.

Flexible structure helps maintain long-term accuracy.

⭐ 8. Use of Standard Notation

Using UML diagrams, standard design patterns, and globally accepted notations makes documentation understandable to all stakeholders.

⭐ English Conclusion (7 Marks)

Refinement is the step-by-step detailing of an architectural design. Sound documentation follows principles like completeness, consistency, correctness, clarity, traceability, organization, maintainability, and use of standard notations, ensuring the architecture is well-understood and easy to implement.

Q.10 Describe the seven-part template for the documentation package.

✅ Answer 1: Hinglish Format (7 Marks Level)

Describe the Seven-Part Template for the Documentation Package

Software architecture documentation ko standard format me present karne ke liye ek **seven-part template** use kiya jata hai. Ye template ensure karta hai ki documentation **complete, clear, understandable** aur **easy to maintain** ho.

Neeche seven-part template ke sabhi parts explain kiye gaye hain:

⭐ 1. Introduction / Documentation Roadmap

Isme documentation ka **overview**, purpose, structure, aur kaise document ko use karna hai — ye sab hota hai.

Ye basically ek guide hota hai jo batata hai ki document me kya-kya milega.

⭐ 2. View Catalog

Architecture ko different viewpoint se represent kiya jata hai.

Is part me mention hota hai ki kaun-kaun se views include hain:

- Module view
- Component & connector view
- Deployment view
- Allocation view

Each view ka short description diya hota hai.

⭐ 3. Views (Detailed Architecture Views)

Ye documentation ka sabse **important part** hota hai.

Har view ko detail me explain kiya jata hai:

- Components
- Connectors
- Interfaces
- Responsibilities
- Behaviors
- Restrictions

Isme **diagrams, tables, relationships** sab include hote hain.

⭐ 4. Cross-View Consistency

Different views ek-dusre se consistent hone chahiye.

Is part me bataya jata hai ki:

- Views ek-dusre ko contradict na kare
- Common elements same ho
- Naming conventions consistent ho

⭐ 5. Architectural Rationale

Is part me bataya jata hai ki architect ne **kya decisions kyu liye**.

Example:

Microservices architecture kyun choose ki?

Layered architecture ya monolithic kyun nahi?

Yaani har design choice ka **reason** diya hota hai.

⭐ 6. Appendices

Isme extra information rakhi jati hai jo architecture ko samajhne me help karti hai:

- Glossary
- Acronyms
- Standards
- References
- Additional diagrams

⭐ 7. Stakeholder Glossary / Related Documentation

Is part me stakeholders ki list, unke roles, aur unke liye relevant documents provide kiye jate hain.

Example:

Developers ke liye API docs,

Testers ke liye test cases,

Managers ke liye requirement specs.

⭐ Hinglish Conclusion (7 Marks)

Seven-part template architecture documentation ko structured banata hai.

Isme **introduction, view catalog, detailed views, consistency checking, rationale, appendices, aur stakeholder-related info** included hoti hai, jisse documentation professional, complete, aur easy to use ban jata hai.

✅ Answer 2: English Format (7 Marks Level)

Describe the Seven-Part Template for the Documentation Package

The seven-part template provides a standard structure for organizing software architecture documentation. It ensures clarity, completeness, and easy understanding for all stakeholders.

⭐ 1. Introduction / Documentation Roadmap

This section gives an overview of the document, its purpose, structure, and how to navigate it.

It acts as a guide to the entire documentation package.

⭐ 2. View Catalog

This section lists all the architectural views included in the documentation such as:

- Module view
- Component-and-connector view
- Deployment view

It provides a short description of each view.

⭐ 3. Views (Detailed Architectural Views)

This is the core of the documentation.

Each view is described in detail, including:

- Components

- Connectors
- Responsibilities
- Interfaces
- Behavior
- Design constraints

Diagrams and tables are included in this section.

⭐ 4. Cross-View Consistency

This part ensures that all architectural views are consistent with each other.

It checks:

- No contradictions
- Consistent naming
- Matching elements across all views

⭐ 5. Architectural Rationale

This explains **why** certain architectural decisions were taken.

It includes:

- Alternatives considered
- Trade-offs
- Justifications for chosen patterns and technologies

⭐ 6. Appendices

Additional supporting information is included here, such as:

- Glossary
- Acronyms

- References
- Standards
- Supplementary diagrams

⭐ 7. Stakeholder Glossary / Related Documents

This section lists stakeholders and links them with relevant documents.

It ensures that each stakeholder can find the information they need easily.

⭐ English Conclusion (7 Marks)

The seven-part template organizes architecture documentation into introduction, view catalog, detailed views, consistency checks, rationale, appendices, and stakeholder documentation. This makes the architecture well-structured, easy to understand, and complete for both technical and non-technical stakeholders.

Q.11 Compare RMI and RPC.

✅ Answer 1: Hinglish Format (7 Marks Level)

Compare RMI and RPC

RMI aur RPC dono distributed systems me communication ke liye use hote hain, lekin dono ka working style bilkul different hota hai. RPC mainly **procedures/functions** ko call karta hai, jabki RMI **objects aur methods** ko remotely access karta hai.

⭐ RMI (Remote Method Invocation)

Meaning:

RMI Java technology hai jiske through ek JVM ka object dusri JVM me remotely available object ke method ko call kar sakta hai.

Features:

- Pure object-oriented
- Java-to-Java communication
- Supports passing objects
- Uses **stub and skeleton**
- Automatic garbage collection support

⭐ RPC (Remote Procedure Call)

Meaning:

RPC ek mechanism hai jisme ek system (client) dusre system (server) ke **procedure/function** ko remotely call karta hai.

Features:

- Procedure-oriented
- Language-independent (C, C++, Java)
- Calls functions, not objects
- Uses **marshalling/unmarshalling**
- Simple & lightweight communication

⭐ Difference Table (Hinglish Version)

Feature	RMI	RPC
Full Form	Remote Method Invocation	Remote Procedure Call
Concept	Remote **object** ke methods call hote hain	Remote **procedures/functions** call hote hain
Paradigm	Object-Oriented	Procedure-Oriented
Language Support	Only Java	Multi-language (C, C++, Java etc.)
Data Transfer	Objects + Complex data	Primitive data types mainly
Communication	Java-to-Java	Heterogeneous systems
Complexity	More complex	Simple
Use	Distributed OO systems	Client-server based apps

⭐ Hinglish Conclusion (7 Marks)

RMI object-oriented distributed systems banane ke liye best hai kyunki wo remote objects ke methods ko call karne deta hai. RPC simple, flexible aur language-independent mechanism hai jo remote functions ko call karta hai. Dono remote communication ko enable karte hain, par unka approach alag hota hai.

✅ Answer 2: English Format (7 Marks Level)

Compare RMI and RPC

RMI (Remote Method Invocation) and RPC (Remote Procedure Call) are two communication techniques used in distributed systems, but their working principles differ significantly. RMI supports **remote object invocation**, while RPC supports **remote function calling**.

⭐ RMI (Remote Method Invocation)

Meaning:

A Java-based mechanism that allows an object running in one JVM to invoke methods of an object running in another JVM.

Key Features:

- Object-oriented
- Java-to-Java communication
- Supports passing complex objects
- Uses stub and skeleton
- Supports automatic garbage collection

⭐ RPC (Remote Procedure Call)

Meaning:

A mechanism that enables a client on one machine to call a **procedure/function** on a remote machine.

Key Features:

- Procedure-oriented
- Language-independent
- Transfers primitive data
- Uses marshalling/unmarshalling
- Lightweight and simple

⭐ Difference Table (English Version)

Feature	RMI	RPC
Meaning	Calls remote object methods	Calls remote procedures/functions
Approach	Object-Oriented	Procedure-Oriented
Language Support	Java only	Multi-language
Data Handling	Objects, complex data types	Basic data types
Communication Type	Homogeneous (Java only)	Heterogeneous systems
Abstraction Level	Higher	Lower
Complexity	More complex	Simpler
Usage	Distributed object systems	Traditional client-server systems

⭐ English Conclusion (7 Marks)

RMI is suited for object-oriented distributed applications where remote objects need to interact. RPC is simpler, language-independent, and ideal for calling remote functions in a client-server environment. Both support remote communication but differ in paradigm and implementation.

Q.12 Describe the seven categories of architectural design decisions

✅ Answer 1: Hinglish Format (7 Marks Level)

Describe the Seven Categories of Architectural Design Decisions

Software architecture ban[illegible]ते समय architect ko kai तरह के **critical decisions** लेने पड़ते हैं. इन decisions को सात main categories में divide किया गया है. ये categories architecture की **structure, behavior, technologies, और quality attributes** को define करती हैं.

नीचे सभी **7 categories** simple Hinglish में explained हैं:

⭐ 1. Allocation of Responsibilities (किस part का क्या काम?)

Isme decide kiya jata hai ki **kaun sa module kya kaam karega**.

Responsibilities components, classes ya layers me allocate ki jati hain.

Ye functional + non-functional dono requirements fulfill karta hai.

Example: Authentication → security module ko

Data storage → database layer ko

⭐ 2. Organization of the System (Architecture Style/Structure)

Isme decide hota hai ki system ka **overall structure** kya hoga.

Kaunsa architectural style choose hoga:

- Layered architecture
- Client-server
- Microservices
- Pipe-and-filter
- Event-driven

Structure architecture ka backbone hota hai.

⭐ 3. Selection of Structural Elements (Kaun se components use honge?)

Is decision me architect define karta hai ki system me **kaun se components, subsystems, connectors** honge.

Examples:

- Controllers
- Services
- Repositories
- Communication connectors (REST, gRPC)

⭐ 4. Interaction Among Elements (Components kaise interact karenge?)

Components ek-dusre ke sath kaise baat karenge ye decide kiya jata hai:

- Synchronous vs Asynchronous
- Message passing
- Shared memory
- Events
- API calls

Isse system ka **behavior** decide hota hai.

⭐ 5. Technology and Platform Decisions

Isme decide hota hai:

- Programming language (Java/Python)
- Framework (Spring Boot/.NET)
- Database (MySQL/MongoDB)
- Protocols (HTTP/WebSocket)
- Cloud platform (AWS/Azure)

Ye decisions system ki performance + development speed affect karte hain.

⭐ 6. Architectural Constraints (Rules & Limitations)

Constraints woh rules hote hain jo architecture ko follow karne padte hain:

- Budget
- Time limit
- Security standards
- Hardware limitations
- Government rules
- Industry standards

Constraints architecture ke options ko restrict karte hain.

⭐ 7. Design for Quality Attributes (System ki qualities ensure karna)

Ye decisions ensure karte hain ki system desired **quality attributes** achieve kare:

- Performance
- Security
- Scalability
- Availability
- Maintainability
- Reliability

Quality attribute scenarios create karke architecture ko design kiya jata hai.

⭐ Hinglish Conclusion (7 Marks)

Architectural design decisions 7 categories me divided hote hain—responsibility allocation, structural organization, element selection, interactions, technology

decisions, constraints, aur quality attributes. Ye decisions system ki strength, performance, flexibility aur reliability ko decide karte hain.

✅ Answer 2: English Format (7 Marks Level)

Describe the Seven Categories of Architectural Design Decisions

Architectural design decisions define the core structure and behavior of a software system. These decisions are classified into **seven categories**, each focusing on a different aspect of the system's design.

⭐ 1. Allocation of Responsibilities

Determines which components or modules will handle which responsibilities.

It defines the **functional decomposition** of the system.

⭐ 2. Organization of the System (Architectural Structure)

Defines the overall **architectural style** or structural pattern of the system, such as:

Layered, Client-server, Microservices, Event-driven, Pipe-and-Filter.

⭐ 3. Selection of Structural Elements

Identifies the main **components, subsystems, connectors, and modules** that will form the system.

⭐ 4. Interaction Among Elements

Describes how components will communicate and collaborate.

Includes:

Message passing, APIs, synchronous/asynchronous calls, events, shared memory, etc.

⭐ 5. Technology and Platform Choices

Specifies the technologies required to implement architecture:

Programming language, frameworks, databases, protocols, operating systems, cloud platforms, etc.

⭐ 6. Architectural Constraints

These are the limitations and rules imposed on the architecture.

Examples: budget, deadlines, security norms, hardware limits, standards, organizational policies.

⭐ 7. Decisions Related to Quality Attributes

Ensures that architecture satisfies quality attributes such as:

Performance, security, scalability, reliability, maintainability, availability.

Architects use **scenarios and trade-offs** to achieve required quality levels.

⭐ English Conclusion (7 Marks)

The seven categories of architectural design decisions help architects define responsibilities, structure, components, interactions, technologies, constraints, and quality attributes. Together, they form the foundation of a scalable, secure, and maintainable architecture.

Q.13 Describe how the layered makes use of abstract common services, encapsulates, and uses an intermediary

✅ Answer 1: Hinglish Format (7 Marks Level)

Describe how the Layered Architecture makes use of abstract common services, encapsulates, and uses an intermediary

Layered architecture ek aisa architectural style hai jisme software ko alag-alag **layers** me divide kiya jata hai. Har layer ka apna kaam hota hai aur wo niche wali layer par depend karti hai. Ye architecture **abstraction, encapsulation, aur intermediaries** ka use karke system ko clean aur maintainable banata hai.

Neeche tina points detail me explained hain:

⭐ 1. Use of Abstract Common Services

Layered architecture me kuch **common functions** har layer ko chahiye hote hain, jaise:

- Logging
- Security
- Authentication

- Error handling
- Data validation

Inko repeat karne ke bajaye unko **abstract common services** me rakha jata hai.

Kaise use hota hai?

- Ye services ek Central layer me hoti hain.
- Upper layers unhe call kar sakti hain bina unki implementation detail jaane.
- Code duplication nahi hota, reuse badhta hai.

Example:

Business layer ko security check chahiye → wo directly **Security Service** use karegi.

⭐ 2. Encapsulation in Layered Architecture

Encapsulation ka matlab hota hai **implementation ko hide karna** aur sirf required interface expose karna.

Layered architecture me:

Kaise hota hai?

- Har layer sirf apna interface expose karti hai (methods).
- Internal logic aur data hidden hota hai.
- Upper layer ko pata hi nahi hota ki lower layer internally kaise kaam kar rahi hai.

Benefits:

- Changes karna easy
- Maintenance easy
- Code secure aur clean hota hai

Example:

UI layer ko pata nahi hota data database me kaise store ho raha hai — ye **encapsulation** hai.

⭐ 3. Use of an Intermediary (Mediator Role)

Layered architecture ek **layer ko mediator/intermediary** ki tarah use karta hai.

Kaise?

- Direct communication allowed nahi hota (UI → Database directly nahi ja sakta).
- Har communication ek beech wali layer ke through hota hai.
- Ye intermediary data ko process karke aage forward karta hai.

Example:

UI Layer → (intermediary) Business Layer → Data Layer

⭐ Hinglish Conclusion (7 Marks)

Layered architecture **abstract common services** ka use karke common functions share karta hai, **encapsulation** se internal details hide karta hai, aur **intermediary layer** ke through communication manage karta hai. Isse system zyada reusable, secure, maintainable aur organized ban jata hai.

✅ Answer 2: English Format (7 Marks Level)

Describe how the layered architecture makes use of abstract common services, encapsulates, and uses an intermediary

Layered architecture organizes software into multiple layers, each having specific responsibilities. It uses abstraction, encapsulation, and intermediary layers to create a clean, maintainable, and scalable structure.

⭐ 1. Use of Abstract Common Services

Layers often need common functionalities such as:

- Security
- Logging
- Error handling
- Validation
- Transaction management

Instead of duplicating these functions in every layer, they are placed into **abstract common services**.

How it works?

- These services exist in a shared or lower layer.
- Upper layers access them via interfaces.
- They hide implementation details and promote reuse.

Benefit:

Reduces redundancy and increases maintainability.

⭐ 2. Encapsulation in Layered Architecture

Encapsulation means hiding the internal implementation of a component and exposing only required functionalities.

How layered architecture uses encapsulation?

- Each layer exposes only its public interface.
- Internal logic, algorithms, and data remain hidden from other layers.
- Higher layers do not know *how* lower layers perform tasks; they just call the interface.

Benefit:

Improves security, simplifies changes, and reduces system complexity.

⭐ 3. Use of an Intermediary

Layers communicate **only through the layer below them**, meaning a layer acts as an intermediary between the one above and below.

How it works?

- No direct communication between far apart layers (e.g., UI cannot directly access Database).
- Each layer receives requests, processes them, and forwards them.
- This creates a controlled and structured flow of information.

Benefit:

Improves modularity, simplifies debugging, and reduces coupling.

⭐ English Conclusion (7 Marks)

Layered architecture uses **abstract common services** to share common functionalities, applies **encapsulation** to hide internal details of each layer, and uses **intermediary layers** to organize communication. These principles make the architecture cleaner, more secure, maintainable, and scalable.

Q.14 Describe the steps of the attribute-driven design method

✅ Answer 1: Hinglish Format (7 Marks Level)

Describe the Steps of the Attribute-Driven Design (ADD) Method

ADD (Attribute-Driven Design) ek architecture design method hai jiska main focus **quality attributes** (performance, security, scalability, modifiability, availability etc.) ko achieve karna hota hai.

Ye method system ko step-by-step refine karta hai aur architecture ko quality attribute scenarios ke base par banata hai.

Neeche **ADD ke important steps** simple Hinglish me explain kiye gaye hain:

⭐ Step 1: Confirm the Inputs (Requirements Gather Karna)

Is step me architect system ki sari important information gather karta hai:

- Functional requirements
- Quality attribute requirements
- Constraints (budget, technology, time)
- Business goals

Ye inputs architecture design ki foundation bante hain.

⭐ Step 2: Choose the Architectural Drivers

Architectural drivers woh requirements hote hain jinka architecture par **strong impact** hota hai.

Drivers include:

- Key functional requirements
- Quality attribute scenarios (performance, security, etc.)
- Technical constraints
- Business priorities

Architecture primarily inhi drivers ke basis par banaya jata hai.

⭐ Step 3: Choose an Architectural Pattern / Decomposition Strategy

Ab architect decide karta hai ki kaunsa **architecture style** use hoga:

- Layered architecture
- Client-server
- Microservices
- Event-driven
- Pipe-and-filter
- MVC

Ye step system ke **overall structure** define karta hai.

⭐ Step 4: Decompose the System into Modules

System ko chote-chote **modules, components, responsibilities** me divide kiya jata hai.

In sub-per focus hota hai:

- Responsibilities allocation
- Interfaces
- Communication rules
- Data flow
- Dependencies

Ye step architecture ka main structure banata hai.

⭐ Step 5: Identify and Address Quality Attribute Scenarios

Architect har quality attribute (jaise performance, security, scalability) ke liye specific **scenarios** banata hai.

Example:

"System must handle 10,000 requests per second" → performance scenario.

Phir architecture me solutions add kiye jate hain:

- Caching
- Load balancing
- Encryption
- Replication
- Modularization

⭐ Step 6: Create and Refine Interfaces

Har module ke public interfaces define kiye jate hain:

- Methods
- Events
- APIs
- Data formats

Isse modules ek-dusre se clearly communicate kar sakte hain.

⭐ Step 7: Validate and Review the Architecture

Final step me architects aur stakeholders architecture ko review karte hain.

Check kiya jata hai ki:

- Quality attributes achieve ho rahe hain ya nahi
- Constraints satisfied hain ya nahi
- Design complete aur consistent hai ya nahi

Agar issue mile, architecture refine kiya jata hai.

⭐ Hinglish Conclusion (7 Marks)

ADD method architecture ko **quality attribute-driven** banata hai. Steps jaise input gathering, drivers selection, decomposition, scenario handling, interface design aur validation architecture ko strong, efficient aur maintainable banate hain.

✅ Answer 2: English Format (7 Marks Level)

Describe the Steps of the Attribute-Driven Design (ADD) Method

The Attribute-Driven Design (ADD) method is a systematic architectural design technique that focuses on achieving **quality attributes** such as performance, modifiability, security, and reliability. It refines the architecture step by step using quality attribute scenarios.

Below are the **steps of the ADD method**:

⭐ Step 1: Confirm Inputs (Gather Requirements)

This step collects all important inputs required for architecture:

- Functional requirements
- Quality attribute requirements
- Technical and business constraints
- Environmental considerations

These inputs form the basis for architectural decisions.

⭐ Step 2: Identify Architectural Drivers

Architectural drivers are the most influential requirements.

They include:

- Critical functional requirements
- Quality attribute scenarios
- Constraints
- Business goals

Drivers guide architectural structure and decisions.

⭐ Step 3: Select Architectural Patterns / Styles

Architect chooses appropriate patterns and styles based on drivers:

- Layered
- Client-server
- Service-oriented / Microservices
- Event-driven
- Repository
- MVC

This defines the **overall structure** of the system.

⭐ Step 4: Decompose the System

The system is broken into **modules, components, and subsystems**.

This includes:

- Allocating responsibilities
- Defining component behavior
- Specifying data flow
- Establishing communication methods

Decomposition forms the initial architecture.

⭐ Step 5: Address Quality Attribute Scenarios

Each quality attribute has specific scenarios.

Examples:

- Performance: “Handle 10,000 requests/second”

- Security: “All data must be encrypted”
- Modifiability: “Add new payment method easily”

Architectural tactics are applied such as caching, load balancing, encryption, modularization, replication, etc.

⭐ Step 6: Define and Refine Interfaces

Interfaces are specified for communication among components:

- APIs
- Methods
- Events
- Data formats

Clear interfaces reduce coupling and improve maintainability.

⭐ Step 7: Validate and Review the Architecture

The architecture is evaluated to ensure:

- Quality attribute goals are met
- Constraints are satisfied
- Design is consistent and complete

Feedback is used to refine the architecture.

⭐ English Conclusion (7 Marks)

The ADD method is a structured approach to architecture design driven by quality attributes. By following steps such as identifying drivers, choosing patterns, decomposing the system, handling scenarios, defining interfaces, and validating the design, ADD ensures a robust, maintainable, and high-quality architecture.

Q.15 Short notes

A) variability :

Variability in Software Architecture

Variability ka matlab hota hai ki software architecture me aise points available hon jahan se system ko **change, customize, modify, ya extend** kiya ja sake — bina puri system ko change kiye. Ye capability architecture ko **flexible**, **reusable** aur **future-proof** banati hai.

Variability specially **software product lines**, multi-version software, aur long-term systems me bahut important hoti hai.

⭐ 1. Meaning of Variability

Variability define karti hai ki system me **kaunsi cheezein change ho sakti hain** aur wo changes kaise implement kiye ja sakte hain.

Examples:

- Different themes in an app
- Optional features
- Multiple ways to implement a function
- Version-specific modules

⭐ 2. Need for Variability

- Different customer requirements handle karne ke liye
- Multiple product versions banane ke liye
- Future updates ko easy banane ke liye

- Reusability improve karne ke liye
- Cost aur development time bachane ke liye

⭐ 3. Types of Variability

a) Functional Variability

Functions ya features change/enable/disable kiye ja sakte hain.

Example: e-commerce me coupon system optional ho sakta hai.

b) Structural Variability

Modules, components ya layers ko add, remove ya replace kiya ja sakta hai.

Example: Payment module: UPI add karna ya remove karna.

c) Deployment Variability

System ko different environments me deploy kiya ja sakta hai.

Example: Cloud vs On-premise deployment.

d) Behavioral Variability

System ka behavior change ho sakta hai.

Example: Different countries ke liye tax rules alag hona.

⭐ 4. Mechanisms to Support Variability

a) Configuration Files

Features ko on/off karna.

b) Plugins / Modules

New features plug-in form me add kiye jaate hain.

Example: Browser plugins.

c) Inheritance / Polymorphism

Different implementations for the same interface.

d) Feature Flags

Runtime me functionality enable/disable karna.

e) Design Patterns

Strategy, Factory, and Adapter pattern variability achieve karte hain.

⭐ 5. Variability Benefits

a) Reusability Increase Hoti Hai

Ek base architecture multiple products ke liye use hoti hai.

b) Faster Development

Naye versions banana easy hota hai.

c) Cost-Effective

Har baar system scratch se build nahi karna padta.

d) Flexibility and Adaptability

System different requirements ko easily handle kar leta hai.

e) Future-Proof Architecture

New technologies add karna easy hota hai.

⭐ 6. Example for Variability

A single mobile app “Basic, Pro, Premium” versions:

- Basic → Limited features
- Pro → More features

- Premium → All features

Architecture same rehta hai, only features enable/disable hote hain.

⭐ Hinglish Conclusion (7 Marks)

Variability software architecture ko flexible, reusable, aur customizable banati hai. Ye multiple product versions, changing requirements, aur future upgrades ko easily support karta hai. Proper mechanisms jaisa configuration, plugins, design patterns variability ko powerful aur practical banate hain.

✅ 7-Marks Answer on Variability (English Format)

Variability in Software Architecture

Variability refers to the ability of a software system to **adapt, customize, and support multiple versions or configurations** without changing the entire architecture. It defines what parts of the system may vary and how these variations can be implemented.

⭐ 1. Meaning of Variability

It specifies the **variation points** in the architecture where change is possible, such as features, modules, interfaces, or behaviors.

⭐ 2. Need for Variability

- To meet diverse customer requirements

- To support different product versions
- To reduce development time and cost
- To improve reusability
- To adapt to future changes easily

⭐ 3. Types of Variability

a) Functional Variability

Different features can be enabled, disabled, or modified.

b) Structural Variability

Components or modules may be added, removed, or replaced.

c) Deployment Variability

System may be deployed in different technological environments.

d) Behavioral Variability

System may behave differently under different conditions.

⭐ 4. Mechanisms Supporting Variability

a) Configuration mechanisms

Using configuration files or feature flags.

b) Plugin or modular architecture

Allows new features to be added as external modules.

c) Object-oriented techniques

Inheritance, polymorphism, and interfaces.

d) Design Patterns

Factory, Strategy, and Adapter patterns.

⭐ 5. Benefits of Variability

- **Higher reusability** of software components
- **Faster product development**
- **Cost savings** through reuse
- **Better flexibility and adaptability**
- **Helps create scalable and future-ready architectures**

⭐ 6. Example

A software product line offering Basic, Standard, and Enterprise versions uses the same architecture but varies features based on customer needs.

⭐ English Conclusion (7 Marks)

Variability is crucial for building flexible, reusable, and adaptable software architecture. It supports customization, multiple versions, and future enhancements while reducing cost and effort. Through mechanisms like configuration, plugins, OO concepts, and design patterns, variability becomes easier to manage and implement.

B) CORBA and RMI :

✅ Answer 1: Hinglish Format (7 Marks Level)

CORBA and RMI

CORBA aur RMI dono technologies **distributed systems** me remote communication enable karne ke liye use hoti hain. Dono ka main purpose remote method/procedure calling hai, lekin unka working, language support, architecture aur protocols bahut alag hote hain.

Neeche dono ko alag-alag explain kiya gaya hai aur phir comparison diya hai.

⭐ 1. CORBA (Common Object Request Broker Architecture)

Meaning:

CORBA ek **language-independent** distributed object technology hai jo OMG (Object Management Group) ne develop ki hai.

Ye different programming languages me likhe systems ko communicate karne deta hai.

Key Features:

- **Language-independent** (C++, Java, Python, Ada, etc.)
- Uses **ORB (Object Request Broker)** for communication
- Defines interfaces using **IDL (Interface Definition Language)**
- Supports heterogeneous systems
- Platform-independent communication

Use Case Example:

A C++ server and Java client communicating through CORBA.

⭐ 2. RMI (Remote Method Invocation)

Meaning:

RMI Java ka mechanism hai jisme ek JVM ka object dusri JVM me present remote object ke method ko call kar sakta hai.

Key Features:

- **Only Java-to-Java** communication
- Pure **object-oriented**
- Supports passing objects
- Uses **stub and skeleton** mechanism
- Built on top of TCP/IP

Use Case Example:

A Java-based distributed application calling methods of remote Java objects.

⭐ Difference Table (Hinglish Version)

Feature	CORBA	RMI
Full Form	Common Object Request Broker Architecture	Remote Method Invocation
Language Support	Multi-language (C++, Java, Python etc.)	Only Java
Communication	Heterogeneous systems	Only Java systems
Interface Definition	Uses IDL	Java interfaces
Architecture Base	ORB (Object Request Broker)	JVM (Java VM)
Data Transfer	Structured objects via IDL	Java objects
Complexity	High (due to IDL + ORB)	Moderate
Use	Large enterprise distributed systems	Java-based distributed systems

---

⭐ Hinglish Conclusion (7 Marks)

CORBA ek powerful, language-independent distributed architecture hai, jo heterogeneous systems ko connect karta hai.

RMI sirf Java environment ke andar remote object calling provide karta hai.

Dono distributed communication ke liye use hote hain, par CORBA zyada flexible aur RMI zyada Java-specific solution hai.

✅ Answer 2: English Format (7 Marks Level)

CORBA and RMI

CORBA and RMI are two important technologies for building distributed systems. Both enable remote communication but differ significantly in design, language support, and complexity.

⭐ 1. CORBA (Common Object Request Broker Architecture)

Definition:

CORBA is a **language-independent distributed object architecture** developed by the Object Management Group (OMG).

It allows software components written in different programming languages to communicate.

Key Features:

- Supports many languages (C++, Java, Python, etc.)
- Uses **ORB (Object Request Broker)** to handle request routing
- Interfaces defined through **IDL (Interface Definition Language)**

- Enables cross-platform, heterogeneous communication
- Highly scalable for enterprise systems

⭐ 2. RMI (Remote Method Invocation)

Definition:

RMI is a **Java-based** mechanism that enables one Java virtual machine (JVM) to invoke methods on an object located in another JVM.

Key Features:

- Works only in Java environments
- Object-oriented remote communication
- Uses **stubs and skeletons**
- Supports passing serialized objects
- Easy to use within Java applications

⭐ Difference Table (English Version)

Feature	CORBA	RMI
Definition	Language-independent distributed object architecture	Java-based remote object invocation
Language Support	Multiple languages	Java only
Communication Type	Heterogeneous	Homogeneous (Java-only)
Interface Definition	IDL	Java Interface
Middleware Used	ORB	JVM RMI runtime
Complexity	High	Moderate
Use Case	Enterprise distributed systems	Java distributed apps

---

⭐ English Conclusion (7 Marks)

CORBA is suitable for large-scale, multi-language, cross-platform distributed systems.

RMI is ideal for Java-only distributed applications that require object-oriented remote invocation.

While both support distributed object communication, CORBA is more flexible whereas RMI is simpler but language-restricted.

C) Data flow architecture :

Data Flow Architecture

Data Flow Architecture ek aisa architectural style hai jisme system ka focus "**data ka flow**" aur "**data processing steps**" par hota hai. Is architecture me data input hota hai, phir wo sequence me multiple processing components se guzar kar output form me convert hota hai.

Is architecture ka main idea ye hai ki **system ko processing units (filters) and data channels (pipes)** ke form me divide kiya jaye.

⭐ 1. Definition

Data Flow Architecture wo style hai jisme **data step-by-step stages** me flow karta hai, aur har stage me data transform hota hai.

System **filters (processing elements)** aur **pipes (communication paths)** se milkar banta hai.

⭐ 2. Components of Data Flow Architecture

a) Filters (Processing Units)

- Ye independent components hote hain.
- Har filter apna specific operation perform karta hai (sort, parse, compute, convert, etc.).
- Input data leta hai, process karta hai, aur output generate karta hai.

b) Pipes (Connectors)

- Ye channels hain jinke through ek filter ka output dusre filter ka input banta hai.
- Pipes data transfer aur buffering ka kaam karti hain.

⭐ 3. Working Mechanism

Data flow architecture ka flow kuch is tarah hota hai:

Input → Filter 1 → Filter 2 → Filter 3 → ... → Output

Har filter data ko modify, transform ya compute karta hai.

Har stage ek independent module hota hai.

⭐ 4. Types of Data Flow Architecture

a) Batch Sequential Model

- Input complete hone ke baad ek module run hota hai.
- Output next module ke input me pass hota hai.
- Example: Compiler phases (Lexical → Syntax → Semantic → Code Generation).

b) Pipe and Filter Model

- Continuous data flow hota rehta hai.
- Filters parallel me kaam kar sakte hain.
- Example: Unix Commands (cat | grep | sort)

⭐ 5. Advantages

- High **reusability** of filters
- Easy to **add/remove/replace** filters
- Components independent hote hain
- Parallel processing possible
- System easily extendable hota hai

⭐ 6. Disadvantages

- Real-time systems ke liye suitable nahi
- Large data buffering required ho sakti hai
- Error handling difficult hota hai
- Control flow weak hota hai (data-driven system hai)

⭐ 7. Examples of Data Flow Architecture

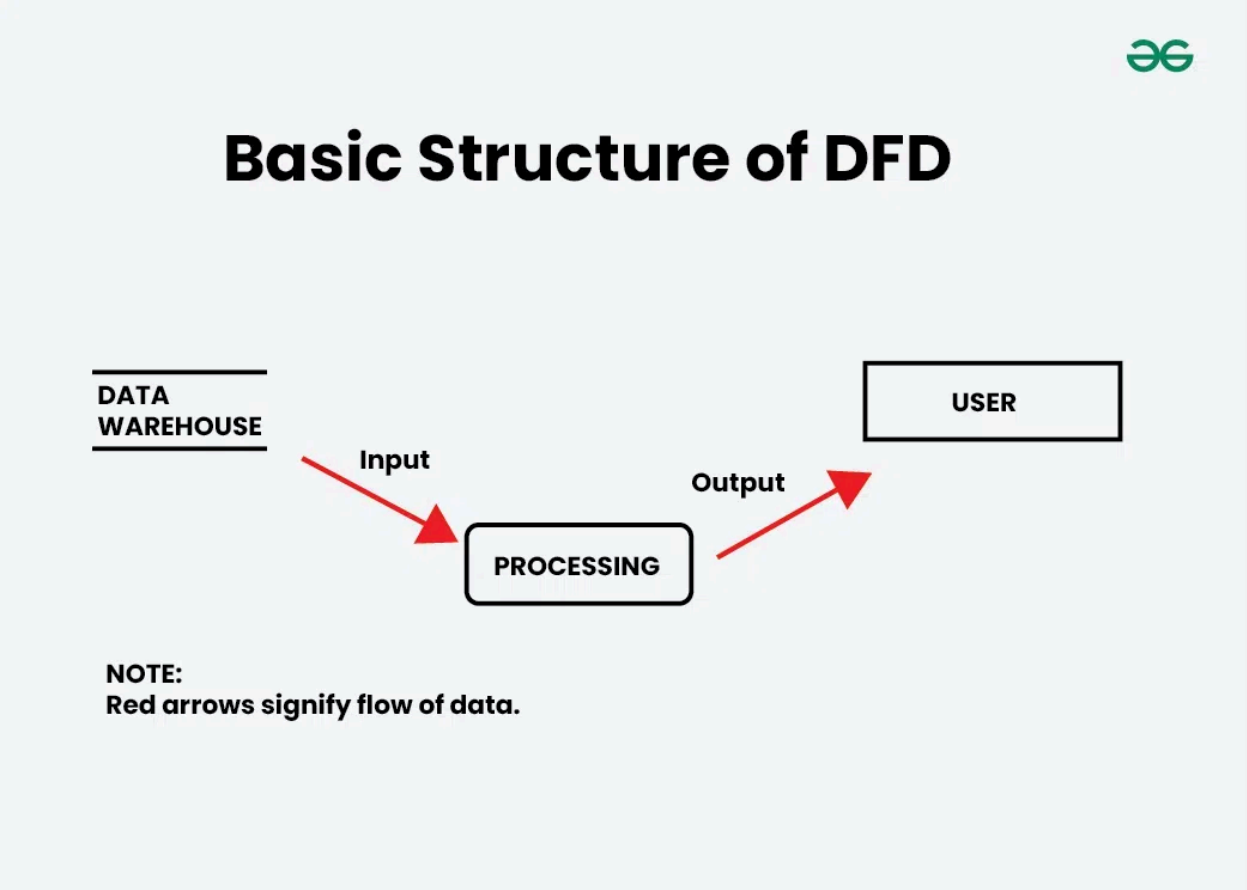
- Compilers
- Signal processing systems
- Data transformation pipelines
- Streaming data systems
- Unix pipe commands
- ETL (Extract–Transform–Load) data systems

⭐ Hinglish Conclusion (7 Marks)

Data Flow Architecture ek powerful style hai jisme system ko **filters** aur **pipes** me divide karke data ko sequentially process kiya jata hai. Ye architecture

development ko modular, reusable, scalable aur extendable banata hai, especially un systems me jahan high data processing required ho.

Data Flow Architecture Diagram :



✅ Answer 2: English Format (7 Marks Level)

Data Flow Architecture

Data Flow Architecture is a software architectural style in which the **flow and transformation of data** drive the system's behavior. The system is decomposed into independent processing components through which data passes sequentially.

⭐ 1. Definition

Data Flow Architecture organizes a system around **data processing steps**, where data flows between components (filters) via connectors (pipes).

⭐ 2. Components

a) Filters

- Independently functioning processing units
- Take input, transform it, and produce output
- Do not store global state
- Examples: parsers, converters, calculators

b) Pipes

- Connect filters to each other
- Transport data between filters
- Provide buffering and communication

⭐ 3. Working Principle

The system processes data in stages:

Input → Filter A → Filter B → Filter C → Output

Each filter transforms data without depending on other filters' internal logic.

⭐ 4. Types of Data Flow Architecture

a) Batch Sequential Architecture

- Complete batch of input processed at once
- Output passed as whole to next step
- Common in compilers and batch processing systems

b) Pipe-and-Filter Architecture

- Continuous data flow
- Allows parallel processing
- Common in streaming and UNIX pipelines

⭐ 5. Advantages

- High modularity and reusability
- Easy maintenance and extension
- Supports parallelism
- Components are easy to replace or upgrade

⭐ 6. Disadvantages

- Not suitable for real-time interactive systems
- High data buffering may be required
- Error propagation difficult to control
- Limited control flow flexibility

⭐ 7. Applications / Examples

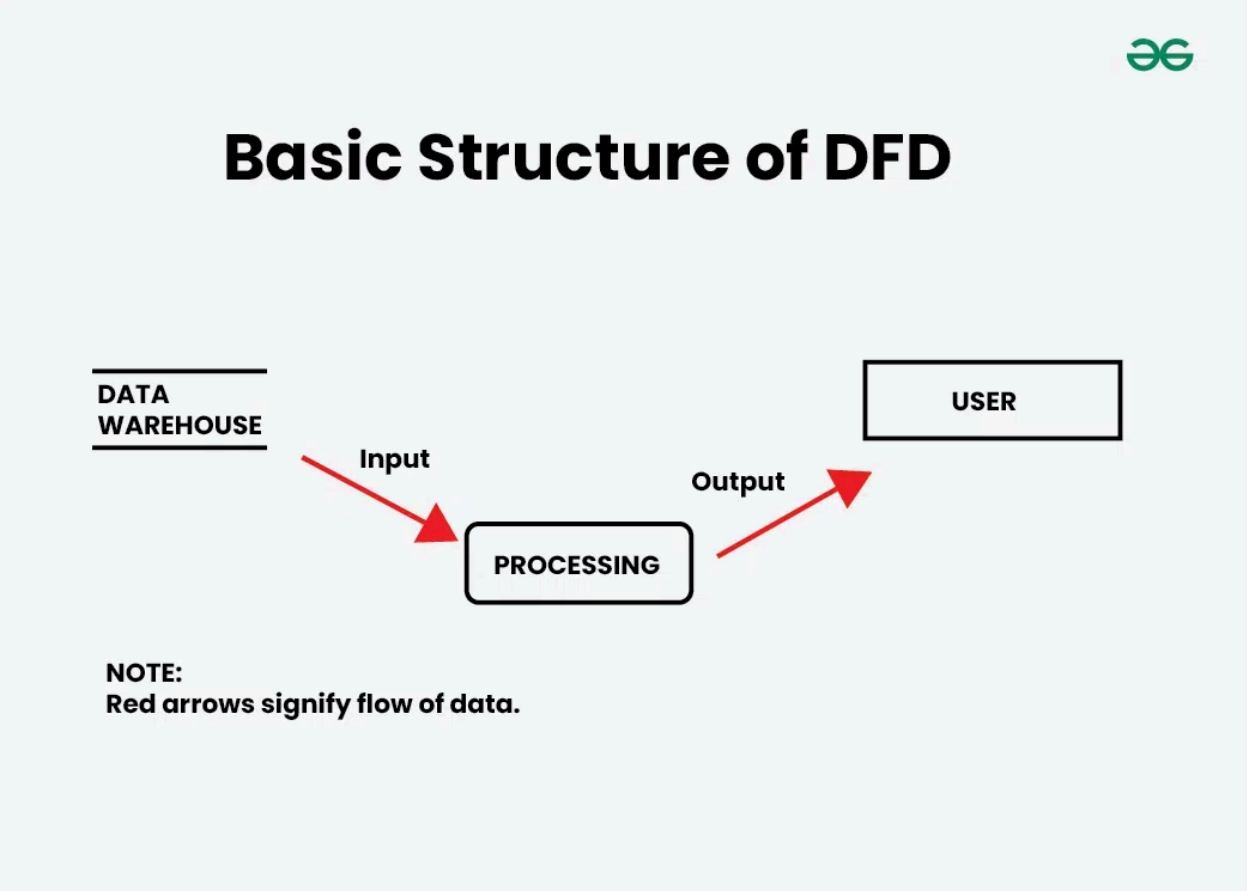
- Compiler design

- Data transformation systems
- Multimedia processing
- Streaming applications
- Sensor data processing
- Unix pipe commands

⭐ English Conclusion (7 Marks)

Data Flow Architecture emphasizes data transformation through independent filters connected via pipes. It provides strong modularity, reusability, and scalability, making it ideal for data-intensive and batch processing systems.

Data Flow Architecture Diagram :



✅ Q1(a) What is Software Architecture? How software architecture represents a system's earliest set of design decisions?

⭐ Hinglish Answer (7 Marks)

Software Architecture wo high-level structure hota hai jo software system ke main components, unke relations, communication style, constraints aur design choices ko define karta hai. Ye system ka **blueprint** hota hai jo batata hai ki system kaise kaam karega.

Architecture earliest design decisions ko kaise represent karta hai?

1. **Major components define karta hai**

   Early stage me hi decide ho jata hai ki system me kaun-kaun se modules honge — UI, business, database, middleware, etc.

2. **Communication style define hota hai**

   Components aapas me kaise interact karenge — REST, events, pipes, message queues — ye sab early stage decisions hote hain.

3. **Quality attributes decide hote hain**

   Performance, modifiability, scalability, security — architecture in sabko decide karta hai.

4. **Constraints early consider hote hain**

   Technology, hardware, budget, deployment environment — architecture in sabko early stage me lock kar deta hai.

5. **Patterns aur styles set karna**

   Monolithic hoga ya microservices? Layered hoga ya client–server?

   Ye sab sabse pehle architecture me hi decide hote hain.

Conclusion (Hinglish)

Software architecture system ke structure, communication, constraints aur quality attributes ko sabse pehle decide karta hai. Ye early decisions poore development ka future direction set kar dete hain.

⭐ English Answer (7 Marks)

Software Architecture is the high-level structure of a software system that defines major components, their relationships, interactions, and constraints. It is the **blueprint** of the entire system.

How architecture represents earliest design decisions?

1. **Defines major system components**

   The architect decides UI, business, database, network, and other modules early in the process.

2. **Decides interaction model**

   Whether components communicate via REST APIs, events, message buses, or pipes is determined at the architecture stage.

3. **Quality attributes are addressed early**

   Performance, security, modifiability, scalability—all are chosen at architecture time.

4. **Specifies constraints**

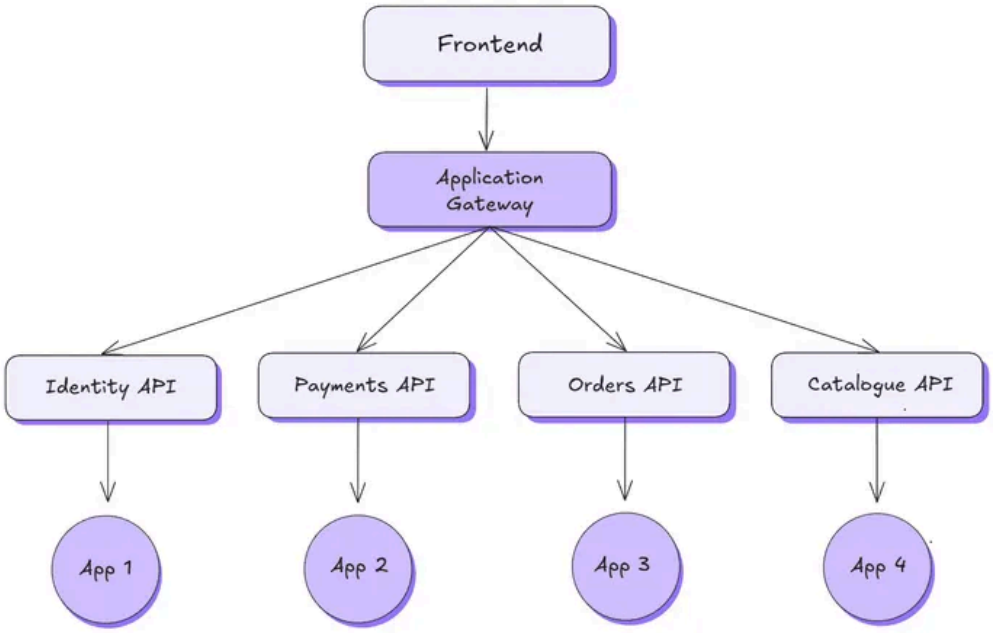
   Hardware limits, budgets, deployment platforms, and technology stacks are fixed early.

5. **Selects architectural style**

   Layered, microservices, client–server, pipe-and-filter, etc., are chosen before coding.

Conclusion (English)

Architecture defines fundamental design decisions early in the project lifecycle, guiding all future development and ensuring the system meets quality and functional expectations.



✅ Q1(b) What is Software Quality Model? Explain McCall's Model in detail.

⭐ Hinglish Answer (7 Marks)

Software Quality Model wo framework hota hai jo software ki **quality attributes** ko clearly define karta hai — jaise reliability, usability, efficiency, maintainability.

McCall's Quality Model ek classical model hai jisme software quality ko 3 categories me divide kiya gaya hai:

⭐ 1. Product Operation (Software ka behavior during execution)

Isme wo attributes aate hain jo batate hain software run hone par kaise behave karta hai.

- **Correctness** – Kya software correct output de raha hai?
- **Reliability** – Software fail kitna kam hota hai?

- **Efficiency** – CPU, memory, time kitna consume hota hai?
- **Integrity** – Unauthorized access se protection.
- **Usability** – User ke liye kitna easy hai software use karna.

⭐ 2. Product Revision (Future changes ko support karne ki capacity)

- **Maintainability** – Bugs fix karna kitna easy?
- **Flexibility** – New features add karna kitna easy?
- **Testability** – Testing kitni easily ho sakti hai?

⭐ 3. Product Transition (New environment me adapt hone ki capability)

- **Portability** – Software ko new OS/platform par chalana easy?
- **Reusability** – Components ko dusre projects me use kar sakte hain?
- **Interoperability** – Dusre systems ke sath communication possible?

Conclusion (Hinglish)

McCall Model software quality ko teen angles se judge karta hai — **operation**, **revision**, **transition** — jis se overall software quality clear aur measurable ho jati hai.

⭐ English Answer (7 Marks)

A **Software Quality Model** defines how to measure the quality of a software product through specific attributes.

McCall's Quality Model classifies software quality into 3 major categories:

⭐ 1. Product Operation

Focus: How the software performs during execution.

- **Correctness** – Accuracy of output
- **Reliability** – Frequency of failure
- **Efficiency** – Resource usage
- **Integrity** – Security and protection
- **Usability** – Ease of use

⭐ 2. Product Revision

Focus: Ability to support future changes.

- **Maintainability** – Ease of bug fixing
- **Flexibility** – Ease of enhancement
- **Testability** – Ease of testing

⭐ 3. Product Transition

Focus: Adaptation to new environments.

- **Portability** – Ability to run on different environments
- **Reusability** – Use in other applications
- **Interoperability** – Interaction with other systems

Conclusion (English)

McCall's Model provides a structured way to evaluate software quality by dividing it into **operation**, **revision**, and **transition** categories.

✅ Q2(a) What are the Architectural Constraints of RESTful API? Explain.

⭐ Hinglish Answer (7 Marks)

RESTful API ek architectural style hai jo kuch strict **constraints** follow karta hai. Ye constraints ensure karte hain ki API scalable, maintainable, secure aur lightweight bane.

Neeche REST ke **major architectural constraints** explain kiye gaye hain:

⭐ 1. Client–Server Architecture

Client aur server clearly independent hote hain.

Client sirf request bhejta hai, server process karta hai.

Isse system scalable aur maintainable banta hai.

⭐ 2. Statelessness

Har request apni **saari information** carry karta hai.

Server koi session data store nahi karta.

Benefits:

- High scalability
- Easy load balancing

⭐ 3. Cacheability

Server responses ko **cache** kiya ja sakta hai.

Isse performance improve hota hai aur bandwidth reduce hoti hai.

Example: GET response caching.

⭐ 4. Uniform Interface

REST ki sabse important constraint.

Iska matlab:

- Resources identified by URIs

- Standard HTTP methods use (GET, POST, PUT, DELETE)
- Representation JSON/XML
- HATEOAS (optional)

Uniformity API ko simple aur predictable banati hai.

⭐ 5. Layered System

Client ko ye pata nahi hota ki wo **direct server** se interact kar raha hai ya **intermediate layer** se.

Layers include:

- Proxy
- Gateway
- Load balancer
- Security layers

⭐ 6. Code-on-Demand (Optional)

Server client ko executable code bhej sakta hai (e.g., JavaScript).

Ye optional hota hai.

⭐ Hinglish Conclusion

RESTful API ki constraints — client-server, statelessness, cacheability, uniform interface, layered system — ensure karte hain ki API scalable, fast, maintainable, aur lightweight ho.

⭐ English Answer (7 Marks)

RESTful API is based on a set of **architectural constraints** that define how clients and servers communicate.

⭐ 1. Client–Server

The client and server are independent.

Client handles the UI; server handles data and logic.

⭐ 2. Stateless

Each request contains all required information.

Server does not store client session.

⭐ 3. Cacheable

Responses can be cached to improve performance.

⭐ 4. Uniform Interface

Ensures consistency and simplicity across the API.

Includes:

- Unique resource identifiers
- Standard HTTP methods
- JSON/XML representation
- Consistent conventions

⭐ 5. Layered System

Communication passes through layers like proxies, gateways, security layers.

⭐ 6. Code-on-Demand (Optional)

Server may send executable code to the client.

⭐ English Conclusion

These constraints make RESTful APIs scalable, simple, high-performance, and suitable for web-based distributed systems.

✅ Q2(b) How can we decide the quality of software architecture? Explain.

⭐ Hinglish Answer (7 Marks)

Software architecture ki quality decide karne ke liye hum **Quality Attributes** aur **evaluation methods** ka use karte hain. Good architecture wo hota hai jo functional + non-functional dono requirements satisfy kare.

⭐ 1. Quality Attributes ka evaluation

a) Performance

Response time, throughput, load time check kiya jata hai.

b) Security

Unauthorized access, encryption, authentication.

c) Modifiability

System me change fast kar sakte hain ya nahi.

d) Reliability

System kitna stable hai? Kitni baar fail hota hai?

e) Usability

Users ko system use karna kitna easy lagta hai.

f) Scalability

System badhte data/user load ko handle kar sakta hai ya nahi.

⭐ 2. Scenario-Based Evaluation

Quality scenarios banaye jaate hain, jaise:

- "System must handle 10,000 requests/sec."
- "A new module should be added in 2 days."

Agar architecture in scenarios ko satisfy kare → quality high hai.

⭐ 3. Architecture Analysis Methods

a) ATAM

Tradeoff analysis method – risks + sensitivity find karta hai.

b) SAAM

Modifiability evaluate karta hai.

c) ARID

Intermediate designs evaluate karta hai.

⭐ 4. Design Principles Follow Kiye Gaye Hain Ya Nahi

- Low coupling
- High cohesion
- Clear modules
- Standard design patterns

Agar ye maintained ho → quality achhi hoti hai.

⭐ Hinglish Conclusion

Software architecture ki quality uske quality attributes, design principles, scenarios aur evaluation methods se decide hoti hai.

⭐ English Answer (7 Marks)

To decide the quality of software architecture, we evaluate whether the architecture satisfies important **quality attributes** and meets business goals.

⭐ 1. Evaluation of Quality Attributes

- **Performance** → response time, throughput
- **Security** → authentication, access control
- **Modifiability** → ease of change
- **Reliability** → fault tolerance
- **Usability** → user friendliness
- **Scalability** → growth handling capability

⭐ 2. Scenario-Based Evaluation

Quality attribute scenarios help in understanding real-world behavior.

If architecture supports these scenarios, quality is high.

⭐ 3. Architecture Evaluation Methods

- **ATAM** → analyzes trade-offs and risks
- **SAAM** → measures modifiability
- **ARID** → reviews intermediate design

⭐ 4. Checking Architectural Principles

- Low coupling
- High cohesion
- Abstraction and modularity
- Standard patterns

These principles ensure good architecture quality.

⭐ English Conclusion

The quality of software architecture is decided by how well it satisfies quality attributes, supports scenarios, and follows architectural principles and evaluation methods.

✅ Q3(a) What is Architecture Description Language? Outline the features of architecture description languages.

⭐ Hinglish Answer (7 Marks)

What is Architecture Description Language (ADL)?

Architecture Description Language (ADL) ek formal language hoti hai jo software architecture ko describe, model, analyze aur document karne ke liye use hoti hai.

Ye architecture ke **components, connectors, behavior, interfaces aur configurations** ko clearly define karti hai.

Simple words me:

ADL = **Architecture ko represent karne ke liye ek language.**

⭐ Features of ADLs

1. Component Specification

ADL system ke har module/component ko clearly define karte hain —

Uske functions, responsibilities, data, etc.

---

2. Connector Specification

Components kaise interact karenge (protocols, messages, calls)

ye ADL me define hota hai.

Example:

Event-based communication, REST, message queues.

---

3. Configuration / Topology Description

System ka structure yani **components ka arrangement** dikhaya jata hai.

Kaun component kis se linked hai → all defined in ADL.

---

4. Formal Syntax and Semantics

ADL ka structure mathematical aur formal hota hai.

Isse ambiguity nahi hoti aur architecture clear samajh aata hai.

---

5. Support for Analysis

ADLs allow:

- Performance analysis
- Deadlock detection
- Consistency checking
- Validating architecture

---

6. Multiple Views Support

ADLs multiple viewpoints provide karte hain, jaise:

- Structural

- Behavioral
- Deployment
- Interface view

---

7. Tool Support

Most ADLs tools ke sath आते हैं—for modeling, simulation & documentation.

Examples of ADLs:

- AADL
- Acme
- Wright
- Rapide
- Darwin

⭐ Hinglish Conclusion

ADL ek formal language hai jo architecture ko describe karne, analyze karne aur model banane me help karta hai. ADL ke features architecture ko clear, analyzable aur unambiguous banate hain.

⭐ English Answer (7 Marks)

What is an Architecture Description Language (ADL)?

An **Architecture Description Language (ADL)** is a formal language used to describe, model, analyze, and document software architecture.

It defines the system's **components, connectors, configurations, and behavior** in a precise manner.

⭐ Features of ADLs

1. Component Specification

ADLs define each architectural component, including its functions and responsibilities.

2. Connector Specification

They describe how components communicate—via protocols, messages, events, or calls.

3. Configuration / Topology Description

An ADL shows how components are arranged and connected within the system.

4. Formal Syntax and Semantics

ADLs use precise and mathematical syntax to avoid ambiguity.

5. Analysis Support

ADLs support architectural analysis for:

- Performance
- Consistency
- Deadlocks
- Correctness

6. Multiple Views

ADLs provide structural, behavioral, and deployment views.

7. Tool Support

Most ADLs have tools for visualization, simulation, consistency checking, etc.

⭐ English Conclusion

ADLs help architects describe and analyze software architecture accurately. Their structured features ensure clarity, correctness, and analyzability.

✅ Q3(b) What is Model View Architecture? Explain its characteristics in detail.

⭐ Hinglish Answer (7 Marks)

What is Model View Architecture?

Model–View Architecture ek architectural style hai jisme system ko do main parts me divide kiya jata hai:

1. **Model** – Data + business logic
2. **View** – User interface

Ye separation software ko clean, modular aur maintainable banata hai.

(MVC ka simplified form.)

⭐ Characteristics of Model View Architecture

1. Separation of Concerns

Model aur View clear alag hote hain.

UI change karne se business logic affect nahi hota.

---

2. Independent Development

UI team aur backend/business team parallel me kaam kar sakti hai.

Model change → View unaffected

View change → Model unaffected

---

3. Event-Based Communication

Model me koi change hota hai to View update hota hai.

View user input Model ko bhejti hai.

---

4. Modifiability

UI ya business logic ko easily modify kar sakte hain.

---

5. Reusability

Same Model ko multiple Views use kar sakte hain.

Example:

Web view + Mobile view + Desktop view → same Model.

---

6. Better Maintainability

Code clean, modular, readable hota hai.

Maintenance cost kam hoti hai.

---

7. Scalability

Large systems me multiple Views add karna easy hota hai.

⭐ Hinglish Conclusion

Model View Architecture UI aur business logic ko separate karta hai jisse system maintainable, reusable, scalable aur easily testable ban jata hai.

⭐ English Answer (7 Marks)

What is Model View Architecture?

Model View Architecture is a design approach where the software system is divided mainly into:

- **Model** – Manages data and business logic

- **View** – Represents the user interface

This separation increases modularity and flexibility.

⭐ Characteristics

1. Clear Separation of Concerns

Model handles logic; View handles presentation.

Each can change independently.

---

2. Independent Development

Developers can work on UI and logic separately without conflicts.

---

3. Event-Based Interaction

Model updates trigger View refresh.

User actions in the View call the Model.

---

4. Modifiability

A change in UI or logic can be made without affecting the other layer.

---

5. Reusability

Multiple Views (mobile, web, desktop) can share the same Model.

---

6. Maintainability

Better code organization results in easier maintenance.

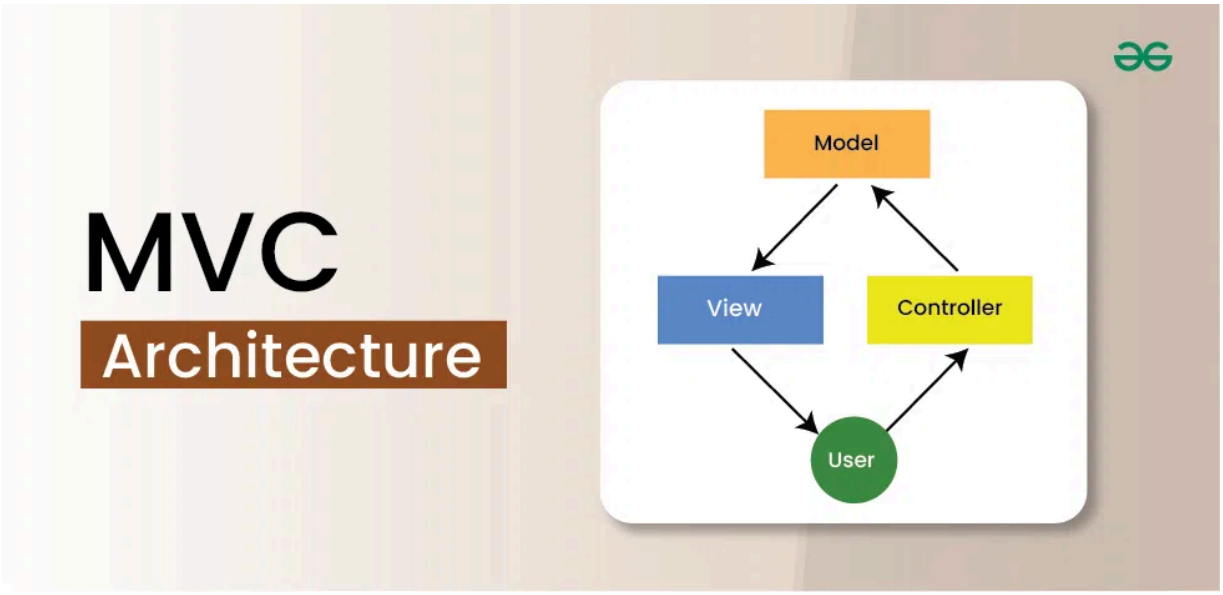
---

7. Scalability

Supports multiple interfaces and large system extensions.

⭐ English Conclusion

Model View Architecture improves flexibility, maintainability, and reusability by separating the user interface from business logic.



✅ Q4(a) What are the six characteristics of a scenario for quality attribute specification?

⭐ Hinglish Answer (7 Marks)

Quality attribute scenario ek template hota hai jiske through hum quality attributes (performance, security, usability, etc.) ko clearly define karte hain.

Ek **complete and effective scenario** me **six characteristics** hote hain:

⭐ 1. Source of Stimulus (Stimulus ka source)

Kaun trigger karega event?

Examples:

- User
- System
- External system
- Attacker

⭐ 2. Stimulus (Event ya request)

Kya cheez system me action trigger karegi?

Examples:

- User login request
- System failure
- High load
- Data update

⭐ 3. Environment (System kis condition me hai?)

Event kab aur kis situation me hoga?

Examples:

- Normal operation
- Peak load
- Failure state
- Maintenance mode

⭐ 4. Artifact (Kis part par impact hoga?)

System ka konsā component effect hoga?

Examples:

- Database
- UI
- Network module
- API server

⭐ 5. Response (System kya karega?)

Stimulus ke response me system ka expected action kya hai?

Examples:

- Authenticate user
- Retry request
- Log event
- Encrypt data

⭐ 6. Response Measure (Quality quantification)

Response ko measure karne ka standard.

Examples:

- Response time < 2 seconds
- System availability 99.99%
- Error rate < 1%
- CPU usage < 70%

⭐ Hinglish Conclusion

Ek complete scenario me **6 fixed parts** hote hain → source, stimulus, environment, artifact, response, response measure — jisse quality attribute clear, measurable aur testable ban jata hai.

⭐ English Answer (7 Marks)

A quality attribute scenario includes **six essential parts** that describe how the system should respond to a specific event under defined conditions.

⭐ 1. Source of Stimulus

Entity that initiates the event.

(Example: user, system, attacker)

⭐ 2. Stimulus

The event or request that occurs.

(Example: login request, crash, high load)

⭐ 3. Environment

Conditions under which the event occurs.

(Example: normal operation, stress load)

⭐ 4. Artifact

Part of the system affected by the stimulus.

(Example: database, UI, API server)

⭐ 5. Response

Action taken by the system.

(Example: authenticate, recover, encrypt)

⭐ 6. Response Measure

Quantification of expected response.

(Example: <2 sec response time, 99.9% uptime)

⭐ English Conclusion

A well-prepared quality attribute scenario must include all six elements so the quality requirement becomes concrete, measurable, and verifiable.

✅ Q4(b) Explain ATAM (Architecture Tradeoff Analysis Method) and its phases.

⭐ Hinglish Answer (7 Marks)

ATAM (Architecture Tradeoff Analysis Method) ek technique hai jisse software architecture ki **quality attributes**, **risks**, **trade-offs**, aur **sensitivity points** evaluate kiye jaate hain.

Ye stakeholders ko help karta hai architecture ki strengths aur weaknesses samajhne me.

⭐ Phases of ATAM

ATAM ki **four major phases** hote hain:

⭐ Phase 1: Present the ATAM

- Method explain ki jati hai
- Goals, objectives set kiye jaate hain

- Stakeholders finalize kiye jaate hain

⭐ Phase 2: Present the Business Drivers

Business team batati hai:

- Key functional requirements
- Quality attributes
- Constraints
- Business goals

Ye architecture ki direction decide karte hain.

⭐ Phase 3: Present the Architecture

Architect apna architecture explain karta hai:

- Components
- Connection styles
- Deployment diagram
- Technology stack
- Patterns used

⭐ Phase 4: Identify Architectural Approaches

Team architecture ka breakdown karti hai aur:

- Strengths
- Weaknesses
- Risks
- Sensitivity points

- Trade-offs

Identify kiye jaate hain.

⭐ Phase 5: Generate Quality Attribute Utility Tree

Quality attributes ko priority ke hisaab se organize kiya jata hai.

Example:

Security → High

Performance → Medium

Usability → Low

⭐ Phase 6: Analyze Architecture with Scenarios

Different scenarios test kiye jaate hain.

Example:

“10,000 users login at once”

“Server crash ho jaye to kya hoga?”

⭐ Phase 7: Evaluate Risks & Trade-offs

Team identify karti hai:

- High-risk decisions
- Trade-offs
- Risks impacting quality attributes

⭐ Hinglish Conclusion

ATAM ek structured method hai jisse architecture ki risk, quality aur trade-off analysis ki jati hai.

Iske phases architecture ko thoroughly evaluate karte hain.

⭐ English Answer (7 Marks)

ATAM (Architecture Tradeoff Analysis Method) is a structured technique used to evaluate the quality attributes of a software architecture and identify risks, sensitivity points, and trade-offs.

⭐ Phases of ATAM

⭐ 1. Present ATAM

Introduce the method, objectives, and evaluation plan.

⭐ 2. Present Business Drivers

Identify:

- Functional requirements
- Quality attribute priorities
- Constraints
- Business goals

⭐ 3. Present Architecture

Architect explains system design:

- Components
- Connectors
- Deployment

- Patterns
- Technologies

⭐ 4. Identify Architectural Approaches

Breakdown architecture and identify approaches used to achieve qualities.

⭐ 5. Create Quality Attribute Utility Tree

Prioritize quality attributes

(high → medium → low importance).

⭐ 6. Analyze Architecture Using Scenarios

Use real-world scenarios to test architecture behavior.

⭐ 7. Identify Risks, Non-Risks & Trade-offs

Document risks, sensitivity points, and trade-offs created by architectural choices.

⭐ English Conclusion

ATAM helps organizations evaluate architecture in terms of quality, risks, and trade-offs. Its phases provide a complete framework for architectural analysis.

✅ Q5(a) Explain design rules of User Interface Architecture.

⭐ Hinglish Answer (7 Marks)

User Interface (UI) Architecture ke design rules wo guidelines hain jo UI ko **simple, usable, consistent, fast aur user-friendly** banate hain.

Ek acchi UI architecture user experience ko improve karti hai aur system ko easy-to-use banati hai.

⭐ 1. Consistency (Har jagah same pattern)

UI ke colors, fonts, layouts, icons, navigation, behavior — sab consistent hone chahiye.

- Same button style
- Same menu placement
- Uniform error messages

Consistency se learning time kam hota hai.

⭐ 2. Simplicity (UI simple hona chahiye)

UI unnecessary information se free hona chahiye.

- Minimal design
- Important info highlight
- Clutter-free screens

Simple UI user ko comfortable banata hai.

⭐ 3. Feedback (System user ko feedback de)

User ke har action par system ko response dena chahiye.

Examples:

- Loading indicators
- Success message
- Error alerts

Feedback se confusion kam hota hai.

⭐ 4. Usability (Use karna easy ho)

UI easy-to-learn, easy-to-use aur accessible hona chahiye.

- Navigation intuitive ho
- Options clearly visible ho
- Help tips available ho

⭐ 5. Error Handling (Errors clear aur helpful ho)

System errors ko clearly explain karna chahiye.

Example:

“Invalid email format”

“Password must contain 8 characters”

Error prevention aur recovery dono important hote hain.

⭐ 6. Flexibility (User preferences support kare)

UI different devices, screens aur users ke needs ko support kare.

Example:

- Responsive design
- Light/Dark mode

- Shortcut keys

⭐ 7. User-Centered Design

UI ko user ke behavior, needs aur context ke hisab se design karna chahiye.

- User research
- Personas
- Usability testing

⭐ Hinglish Conclusion

UI Architecture ke design rules — consistency, simplicity, feedback, usability, error handling aur flexibility — user experience ko smooth, intuitive aur effective banate hain.

⭐ English Answer (7 Marks)

User Interface Architecture design rules are guidelines that help in designing a UI that is **efficient, intuitive, consistent, and user-friendly**.

⭐ 1. Consistency

Consistent use of colors, typography, layouts, icons, and navigation patterns improves usability.

⭐ 2. Simplicity

UI should be simple, minimalistic, and free from clutter.

Users should find information easily.

⭐ 3. Feedback

The system should provide immediate feedback for every user action.

- Success messages
- Error notifications
- Loading indicators

⭐ 4. Usability

UI must be easy to learn and use.

- Clear navigation
- Visible options
- Tooltips/help text

⭐ 5. Error Handling

Errors should be meaningful and helpful.

System should help users recover from errors.

⭐ 6. Flexibility

UI must adapt to different devices, screen sizes, and user preferences.

Examples: responsive layout, themes, keyboard shortcuts.

⭐ 7. User-Centered Design

UI must be designed based on user needs, behaviors, and expectations.

⭐ English Conclusion

Design rules of UI architecture ensure that the system is consistent, simple, user-centered, and accessible—providing a high-quality user experience.

✅ Q5(b) What is the Documentation Package? Explain in detail.

⭐ Hinglish Answer (7 Marks)

Documentation Package wo complete set of documents hota hai jo software architecture ko explain, justify aur communicate karta hai.

Ye architecture ke **views, diagrams, requirements, design decisions, constraints, rationale, guidelines** sab ko include karta hai.

Documentation Package ek **formal bundle** hota hai jo developers, managers, testers, aur stakeholders ko architecture samajhne me help karta hai.

⭐ Main Elements of a Documentation Package

1. Introduction / Overview Section

Isme project description, scope, goals, glossary, aur document ka purpose hota hai.

---

2. View Catalog

Isme bataya jata hai ki architecture ko kaun-kaun se **views** me present kiya gaya hai:

- Module view
- Component & connector view
- Deployment view

---

3. Detailed Architectural Views

Documentation ka main part hai —

Har view ko detail me explain kiya jata hai:

- Components
- Responsibilities
- Patterns
- Interfaces
- Behavior
- Diagrams (UML, C&C diagrams)

---

4. Cross-View Consistency

Ensure karta hai ki saare views ek-dusre ke sath consistent hain.

Naming, structure, behavior same rahna chahiye.

---

5. Architectural Rationale

Isme explain kiya jata hai:

- Kyu kisi pattern ko choose kiya?
- Alternatives kya the?
- Designer ne specific decision kyu liya?

---

6. Appendices

Extra details — glossary, references, standards, supporting documents.

---

7. Stakeholder-Oriented Documentation

Different stakeholders ke hisab se custom documentation:

- Developers → API docs
- Managers → business goals
- Testers → test scenarios

⭐ Hinglish Conclusion

Documentation Package architecture ka complete blueprint provide karta hai.

Isme architecture ke views, decisions, constraints, rationale aur guidelines include hote hain jo communication aur implementation ko easy banate hain.

⭐ English Answer (7 Marks)

A **Documentation Package** is the complete set of documents that describe the software architecture in detail.

It provides all the required information to understand, implement, test, and maintain the architecture.

⭐ Elements of a Documentation Package

1. Introduction / Overview

Includes system goals, scope, definitions, and purpose of documentation.

---

2. View Catalog

Lists all architectural views included: module view, component-connector view, deployment view, etc.

---

3. Detailed Views

Each view is documented with:

- Components
- Interfaces
- Relationships
- Patterns used
- Behavior
- UML diagrams

---

4. Consistency Across Views

Ensures all views align with each other without contradictions.

5. Architectural Rationale

Explains *why* particular decisions were made—trade-offs, alternatives, constraints.

6. Appendices

Glossary, standards, references, additional diagrams, etc.

7. Stakeholder Documentation

Specific documents for developers, testers, project managers, customers.

⭐ English Conclusion

The documentation package is a complete architectural blueprint that communicates design decisions, structure, behavior, and constraints to all stakeholders.

✅ Q6(a) Discuss the layer architecture style with a neat diagram.

⭐ Hinglish Answer (7 Marks)

Layered Architecture Style ek aisa model hai jisme software ko **multiple layers** me divide kiya jata hai.

Har layer ka **specific responsibility** hota hai aur wo sirf apne niche wali layer ke through hi interact karti hai.

Ye architecture **simplicity, reusability, maintainability, testability** ko improve karta hai.

⭐ Main Features of Layered Architecture

1. Separation of Concerns

System ko logical layers me divide kiya jata hai:

- Presentation Layer
- Business Layer
- Data Access Layer
- Database Layer

Har layer ka apna kaam hota hai.

---

2. Encapsulation

Layer ke internal details hidden hote hain.

Upper layer ko sirf lower layer ka interface pata hota hai.

---

3. Modifiability

Agar business logic change ho, to UI aur Database ko change nahi karna padta.

---

4. Reusability

Layers ka code dusre systems me bhi reuse ho sakta hai.

---

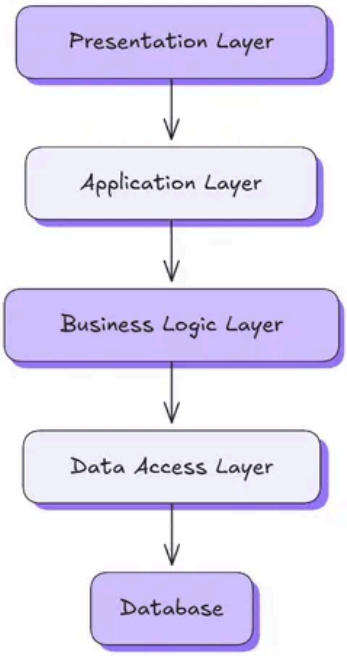
5. Independent Development

Different teams alag-alag layers par parallel kaam kar sakti hain.

⭐ Typical Layers

1. **Presentation Layer** – UI, user interaction
2. **Business Layer** – Business rules
3. **Data Access Layer** – Queries, repository
4. **Database Layer** – Actual DB

⭐ Diagram Link (Layered Architecture)



⭐ Hinglish Conclusion

Layered architecture clean, maintainable aur scalable system banane ke liye best hai. Har layer independent hoti hai aur apni responsibility handle karti hai.

⭐ English Answer (7 Marks)

Layered Architecture Style organizes software into multiple layers, each of which has a specific role. Communication flows **from the top layer down**, and each layer interacts only with the layer directly below it.

⭐ Key Characteristics

1. Clear Separation of Concerns

Each layer handles a specific concern (UI, business rules, data).

2. Encapsulation

Internal details of each layer are hidden from other layers.

3. Modifiability

Changing one layer does not affect others.

4. Reusability

Common services in a layer can be reused across systems.

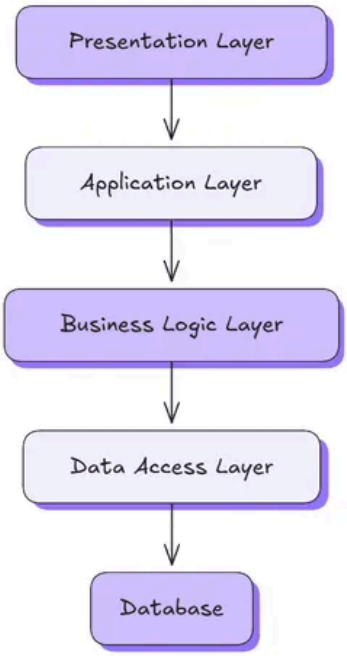
5. Testability

Each layer can be tested independently.

⭐ Typical Layers

- **Presentation Layer**
- **Business Layer**
- **Data Access Layer**
- **Database Layer**

⭐ Diagram Link



⭐ English Conclusion

Layered architecture enhances organization, reusability, maintainability, and clarity in large systems.

✅ Q6(b) Discuss the client–server architecture style with its benefits and drawbacks.

⭐ Hinglish Answer (7 Marks)

Client–Server Architecture ek model hai jisme system ko do parts me divide kiya jata hai:

- **Client** → Request send karta hai
- **Server** → Request ko process karke response bhejta hai

Client user interface create karta hai, aur server business logic + database manage karta hai.

⭐ Benefits of Client–Server Architecture

1. Centralized Control

Server par saare data aur logic store hote hain — easy management.

2. Scalability

Agar load badh jaye to server ko scale kiya ja sakta hai (vertical/horizontal).

3. Security

Data centralized hone ke karan access control aur encryption easy hota hai.

4. Maintenance Easy

Server update karne par sab clients automatic latest data use karte hain.

5. Multi-user Environment Support

Server ek saath multiple clients ki requests handle karta hai.

⭐ Drawbacks of Client–Server Architecture

1. Single Point of Failure

Server down → Entire system down.

2. High Server Cost

Powerful server hardware ki requirement hoti hai.

3. Network Dependency

System pura network connectivity par depend karta hai.

4. Performance Issues

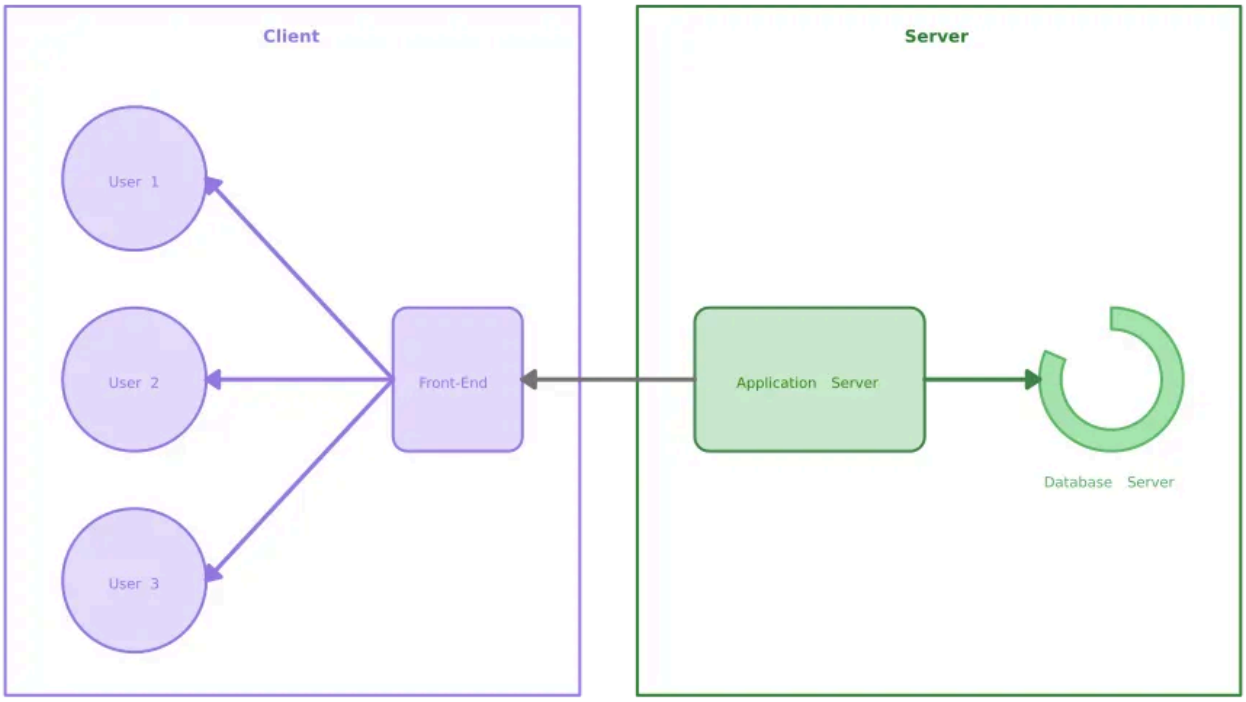
Agar too many clients request bheje to server slow ho sakta hai.

5. Security Risks

Server par attack → All data compromised.

⭐ Hinglish Conclusion

Client–server architecture simple, secure aur manageable hota hai, par server failure aur network dependency iske major drawbacks hain.



Client-Server architecture

⭐ English Answer (7 Marks)

Client–Server Architecture divides the system into two main components:

- **Client:** Initiates requests
- **Server:** Processes requests and sends responses

⭐ Benefits

1. Centralized Management

All data and logic reside on the server.

2. Scalability

Server can be upgraded to support more clients.

3. Security

Centralized data allows strong security control.

4. Easy Maintenance

Updating the server updates the whole system automatically.

5. Supports Multi-user Access

Many clients can communicate with the server simultaneously.

⭐ Drawbacks

1. Single Point of Failure

If the server fails, the entire system becomes unavailable.

---

2. Expensive Server Hardware

High-performance servers are needed.

---

3. Network Dependency

System performance depends on network reliability.

4. Server Overload Issues

Too many client requests slow down the server.

5. Security Challenges

A server attack can compromise all data.

⭐ English Conclusion

Client–server architecture is widely used due to its simplicity and manageability, but it requires strong servers and reliable networks.

✅ Q7(a) Explain the importance and need for software architecture documentation.

⭐ Hinglish Answer (7 Marks)

Software Architecture Documentation system ka **blueprint** hota hai. Ye batata hai system ka structure, components, connections, design rules, constraints, patterns aur behavior. Documentation development, communication, maintenance aur future upgrades ke liye extremely important hota hai.

⭐ Importance & Need of Software Architecture Documentation

1. Communication Among Stakeholders

Developers, testers, managers, designers, clients—sab ko same understanding milti hai.

Documentation misunderstandings ko eliminate karta hai.

2. Clear Understanding of System Structure

Documentation se pata chalta hai system me kaun se components, layers, modules, interfaces hai aur wo kaise interact karte hain.

3. Supports Development and Implementation

Developers coding karte time architecture ko follow kar sakte hain.

Consistency maintained rehti hai.

4. Helps in Maintenance & Future Modifications

Documentation se future changes easy ho jate hain kyunki system ka structure clear hota hai.

5. Helps in Training New Team Members

New developers jaldi system samajh lete hain.

Onboarding fast hota hai.

6. Evaluation & Analysis Easy Hota Hai

Architecture ka performance, security, scalability, risks easily evaluate kiya ja sakta hai.

7. Acts as a Reference for Long-Term Projects

Large projects me architecture documentation long-term reference ka kaam karta hai.

⭐ Hinglish Conclusion

Software architecture documentation project ko organized, maintainable, consistent aur future-ready banata hai. Without documentation, system ko samajhna aur maintain karna bahut mushkil ho jata hai.

⭐ English Answer (7 Marks)

Software Architecture Documentation is essential because it provides a **clear description** of the system's structure and behavior. It supports communication, development, and long-term maintenance.

⭐ Importance & Need

1. Communication Tool

It gives a common understanding among stakeholders and avoids confusion.

2. Clear Structure Representation

It explains components, layers, interfaces, and interactions.

3. Supports Implementation

Developers follow documented architecture to maintain consistency.

4. Simplifies Maintenance

Future changes and debugging become easier.

5. New Member Training

Helps new team members quickly understand the system.

6. Supports Analysis

Enables evaluation of performance, risks, and quality attributes.

7. Long-Term Reference

Large projects need documentation for future updates and enhancements.

⭐ English Conclusion

Architecture documentation is necessary for communication, clarity, consistency, maintenance, and long-term system success.

✅ Q7(b) Discuss the concept of Quality Attribute Workshop with its advantages.

⭐ Hinglish Answer (7 Marks)

Quality Attribute Workshop (QAW) ek structured workshop hota hai jisme stakeholders milkar system ke **quality attributes** define, prioritize aur clarify karte hain.

Is workshop ka goal hota hai system ke quality requirements ko sahi tareeke se capture karna.

Examples of quality attributes:

Performance, security, availability, modifiability, usability, scalability.

⭐ Steps / Working of QAW

1. Business Drivers Identify Karna

Stakeholders project ke goals, constraints aur priorities batate hain.

---

2. Quality Attributes Identify Karna

Project ko kin quality attributes ki need hai — ye decide hota hai.

---

3. Create Quality Attribute Scenarios

Har attribute ke liye scenarios banaye jate hain:

Example:

“System 10,000 requests per second handle kare.”

---

4. Prioritize the Scenarios

High-risk and high-priority scenarios shortlist kiye jate hain.

5. Discuss Risks, Challenges & Trade-offs

Scenarios ke basis pe risks aur architectural decisions identify hote hain.

6. Output Report

Workshop ke end me documentation ready hota hai jisme

goals, scenarios, priorities, risks aur decisions hote hain.

⭐ Advantages of QAW

1. Early Identification of Quality Requirements

Development se pehle hi system ke expectations clear ho jate hain.

2. Involves All Stakeholders

Business, developers, testers, managers—sab participate karte hain.

3. Reduces Risks

Quality attribute risks early detect ho jate hain.

4. Improves Architecture Design

Architect proper decisions le sakta hai based on captured scenarios.

5. Better Communication

Team members ke beech clear understanding banti hai.

6. Saves Time & Cost

Late-stage rework avoid hota hai.

⭐ Hinglish Conclusion

QAW ek powerful method hai jo quality attributes ko early capture karke architecture ko strong, risk-free aur predictable banata hai.

⭐ English Answer (7 Marks)

Quality Attribute Workshop (QAW) is a structured meeting that helps identify, refine, and prioritize the system's quality attribute requirements.

It ensures that all stakeholders clearly understand the expected quality attributes.

⭐ Working / Steps

1. Identify Business Drivers

Stakeholders define goals, constraints, and priorities.

2. Identify Quality Attributes

Determine which quality attributes matter for the system.

3. Generate Scenarios

Real-world scenarios are created for each attribute.

4. Prioritize Scenarios

Select the most critical and high-impact scenarios.

5. Analyze Risks & Trade-offs

Identify potential problems and architectural risks.

6. Final Report

Documentation is prepared with scenarios, risks, and decisions.

⭐ Advantages of QAW

1. Early Clarification of Requirements

Quality expectations become clear before development starts.

2. Stakeholder Collaboration

Brings all stakeholders together.

3. Early Risk Identification

Architecture risks are discovered early.

4. Improves Design Decisions

Architects make better decisions using identified scenarios.

5. Better Communication

Improves understanding among team members.

6. Cost & Time Savings

Prevents rework and misunderstandings.

⭐ English Conclusion

QAW helps capture the correct quality attributes early, improving design decisions and reducing risks throughout development.

✅ Q8(a) Describe the software architecture analysis process.

⭐ Hinglish Answer (7 Marks)

Software Architecture Analysis ka main goal hota hai architecture ki **quality, risks, performance, maintainability, security, scalability** jaisi qualities ko evaluate karna.

Ye process ensure karta hai ki architecture business goals aur technical requirements ko fulfill kare.

⭐ Steps in Software Architecture Analysis Process

1. Collect Requirements

Functional + non-functional requirements, constraints aur business goals ko gather kiya jata hai.

---

2. Identify Architectural Drivers

Important decisions impacting architecture choose kiye jaate hain:

- Performance
- Scalability
- Security
- Modifiability

---

3. Select Scenarios

Quality attribute scenarios banaye jaate hain like:

- “System 10,000 users handle kare.”
- “System crash hone par 2 seconds me recover ho.”

---

4. Choose Evaluation Method

Popular methods:

- **ATAM**
- **SAAM**
- **ARID**
- **CBAM**

---

5. Analyze Architectural Views

Module view, component-connector view, deployment view ko deeply review kiya jata hai:

- Dependencies
- Communication
- Performance hotspots
- Potential failures

---

6. Identify Risks & Sensitivity Points

Risks ko detect kiya jata hai:

- Single point of failure
- Wrong design pattern
- Performance bottlenecks

---

7. Generate Report

Final report me:

- Findings
- Risks
- Trade-offs
- Improvement suggestions

document kiye jaate hain.

⭐ Hinglish Conclusion

Software architecture analysis ek structured process hai jo risks, quality issues, trade-offs ko identify karke architecture ko strong aur reliable banata hai.

⭐ English Answer (7 Marks)

Software Architecture Analysis evaluates the architecture to ensure that it satisfies the required **quality attributes, business goals, and constraints**.

⭐ Steps in Architecture Analysis

1. Requirement Gathering

Collect functional requirements, quality attributes, constraints.

2. Identify Architectural Drivers

Drivers such as performance, modifiability, security, and business goals.

3. Develop Scenarios

Prepare quality attribute scenarios to test architectural behavior.

4. Select an Evaluation Method

Use ATAM, SAAM, ARID, or CBAM based on the system.

5. Analyze Architectural Views

Evaluate module view, component-and-connector view, deployment view.

6. Identify Risks & Sensitivity Points

Detect architecture risks, bottlenecks, and critical decisions.

7. Document Findings

Prepare a report summarizing evaluation results, risks, and recommendations.

⭐ English Conclusion

Architecture analysis ensures that the system will meet desired quality levels and helps identify improvements before implementation.

Q.8 What is rapid application development methodology ?

✅ Hinglish Answer (7 Marks)

What is Rapid Application Development (RAD) Methodology?

Rapid Application Development (RAD) ek **software development methodology** hai jisme focus fast development, quick delivery, aur continuous user feedback par hota hai.

RAD ka main goal hai **software ko jaldi develop karna**, prototypes banana, aur user se frequent feedback lekar product ko improve karna.

Ye traditional Waterfall model se faster aur flexible hota hai.

⭐ Key Characteristics of RAD

1. Fast Development

RAD me development process short hota hai.

Tools, reusable components, aur automation se development speed badhti hai.

---

2. Prototyping-Based Approach

RAD me working prototypes create kiye jate hain.

Users prototype test karke immediate feedback dete hain.

---

3. User Involvement

Users development ke har stage me active role play karte hain.

Isse requirement clarity achhi hoti hai.

4. Iterative Development

System choti-choti iterations me banaya jata hai.

Har iteration ek improved version hota hai.

5. Reusable Components

Existing software components reuse karke time save hota hai.

⭐ Phases of RAD

1. Requirement Planning Phase

Users + developers milkar high-level requirements finalize karte hain.

2. User Design Phase

Prototype ban kar users ko show kiya jata hai.

Users UI, workflow, design par feedback dete hain.

3. Construction Phase

Actual coding hoti hai using rapid tools:

- GUI builders
- Code generators
- Reusable modules
- Automation tools

4. Cutover Phase (Final Implementation)

Testing, data conversion, user training, aur system deployment hota hai.

⭐ Advantages

- Fast product delivery
- User satisfaction high
- Better requirement understanding
- Low development time

⭐ Disadvantages

- Not suitable for large complex systems
- High involvement of users required
- Poor documentation
- Requires skilled developers + tools

⭐ Hinglish Conclusion

RAD ek fast aur flexible methodology hai jisme prototypes, iterations aur user feedback ka use karke software quickly deliver kiya jata hai. Ye small to medium projects ke liye best hota hai.

✅ English Answer (7 Marks)

What is Rapid Application Development (RAD) Methodology?

Rapid Application Development (RAD) is a software development methodology that emphasizes **quick development**, **prototyping**, and **continuous user involvement**.

It reduces development time by using iterative cycles, reusable components, and automated tools.

⭐ Characteristics of RAD

1. Speed of Development

RAD focuses on delivering software faster than traditional models.

2. Prototyping

Working prototypes are developed early and refined based on user feedback.

3. High User Involvement

Users actively participate in design and evaluation.

4. Iterative Process

The product is built in multiple short iterations.

5. Reusability

RAD relies on reusable components and automation tools.

⭐ Phases of RAD

1. Requirement Planning

Users and analysts define high-level requirements.

2. User Design Phase

Prototypes are developed and refined with user feedback.

3. Construction Phase

Actual system development takes place using rapid tools and components.

4. Cutover Phase

Final testing, training, data migration, and deployment are performed.

⭐ Advantages

- Faster delivery
- Better user satisfaction
- Improved requirement clarity
- Reduced development time

⭐ Disadvantages

- Not ideal for large, complex projects
- Requires constant user involvement
- Documentation may be weak
- Needs highly skilled developers and tools

⭐ English Conclusion

RAD is a fast-paced, user-driven development methodology suitable for small and medium-sized projects where quick delivery and frequent feedback are essential.

Q.1 Explain pipes and filters in detail ?

Pipes and Filters ek **data flow architectural style** hai jisme system ko chote-chote independent processing units (filters) me divide kiya jata hai, aur data in filters ke beech **pipes** ke through flow karta hai.

Is architecture ka main goal hai data ko **step-by-step transform** karna.

⭐ 1. Filters (Processing Units)

Filters independent modules hote hain jo:

- Input data receive karte hain
- Data ko process/transform karte hain
- Output data next filter ko bhejte hain

Characteristics of Filters:

- Self-contained hote hain
- No shared state
- Order-independent operations
- Easily replaceable
- Can run in parallel

Examples:

Parsing filter, sorting filter, calculation filter.

⭐ 2. Pipes (Connectors)

Pipes wo channels hote hain jo filters ke beech data send karte hain.

Characteristics of Pipes:

- Data transfer + buffering ka kaam karte hain
- Format consistent hota hai
- Filters ko independent banate hain

Examples:

Unix pipes, message streams, queues.

⭐ 3. Working of Pipes and Filters

Data flow example:

Input → Filter 1 → Filter 2 → Filter 3 → Output

Har filter apna operation karta hai aur next filter ko data de deta hai.

⭐ 4. Advantages of Pipes and Filters

a) High Reusability

Har filter independent hota hai, isliye reuse ho sakta hai.

b) Parallel Execution

Filters parallel me run ho sakte hain → fast processing.

c) Modifiability

Koi ek filter change karne par baaki filters affect nahi hote.

d) Maintainability

Small modules hone ki wajah se maintenance easy hota hai.

e) Scalability

Naye filters add karna easy hota hai.

⭐ 5. Disadvantages of Pipes and Filters

a) Not Suitable for Interactive Systems

Yeh batch data processing ke liye best hai, real-time ke liye nahi.

b) Data Format Rigid

Agar format change ho, to sab filters update karne padenge.

c) Large Data Buffering

Agar data zyada hai to buffering issues aa sakte hain.

⭐ 6. Applications / Examples

- **Compilers** (Lexical → Syntax → Semantic → Code generation)
- **Unix pipelines** (cat | grep | sort)
- **Data processing tools**
- **Signal processing systems**
- **Streaming architectures**

⭐ Hinglish Conclusion

Pipes and Filters architecture system ko modular, reusable aur scalable banata hai. Har filter independently kaam karta hai, aur pipes ke through data flow hota hai. Ye especially data-centric aur batch processing systems me best hai.

✅ English Answer (7 Marks)

Explain Pipes and Filters in Detail

Pipes and Filters is a **data flow architectural style** where a system is composed of independent processing elements called **filters**, and data is passed between them through **pipes**.

⭐ 1. Filters

Filters are independent processing components that:

- Receive input data
- Process or transform it
- Send output to the next filter

Characteristics:

- No shared state
- Replaceable
- Reusable
- Can operate in parallel
- Perform specific, well-defined tasks

⭐ 2. Pipes

Pipes are connectors that transfer data between filters.

Characteristics:

- Provide data flow channels
- Enable loose coupling
- May support buffering
- Maintain consistent data format

⭐ 3. How It Works

Input → Filter A → Filter B → Filter C → Output

Each filter transforms the data, and pipes forward it to the next filter.

⭐ 4. Advantages

1. Reusability

Filters can be reused in different systems.

2. Modularity

Each filter is a separate component.

3. Parallel Processing

Filters can run concurrently.

4. Easy Maintenance

Small independent units are easy to update.

5. Scalability

More filters can be added as needed.

⭐ 5. Disadvantages

1. Not Suitable for Interactive Systems

It works best for batch processing.

2. Rigid Data Formats

All filters must accept the same data structure.

3. Buffering Overhead

Large datasets may require heavy buffering.

⭐ 6. Applications

- Compiler design
- ETL pipelines
- Unix command pipeline
- Audio/video signal processing
- Data transformation tools

⭐ English Conclusion

Pipes and Filters is an effective architectural style for transforming data through sequential processing. It offers modularity, scalability, and reusability, making it

suitable for data-driven and batch processing systems.

Q.2 Discuss the role of UML in software architecture ?

✅ Hinglish Answer (7 Marks)

Discuss the Role of UML in Software Architecture

UML (Unified Modeling Language) ek standardized modeling language hai jo software architecture ko **visually represent**, **communicate**, **design**, aur **document** karne me help karta hai.

Ye architecture ke major components, interactions, behavior aur structure ko clear diagrams ke form me show karta hai.

⭐ 1. Architecture ko Visual Banata Hai (Visualization)

UML architecture ko diagrams ke through easily samajhne layak banata hai.

Text-based description ki jagah visual representation mile to stakeholders fast understand kar lete hain.

⭐ 2. Architecture Documentation

UML architectural views ko formally document karne ke liye use hota hai:

- Structural views
- Behavioral views
- Interaction views
- Deployment views

Yeh documentation maintenance aur development me help karta hai.

⭐ 3. Communication Between Stakeholders

Developers, testers, managers, designers, clients sab UML diagrams ke zariye ek hi architecture ko same tarah understand karte hain.

Isse misunderstandings kam hoti hain.

⭐ 4. Design Decisions Ko Represent Karna

UML important architecture decisions jaise

inheritance, decomposition, component design, data flow, workflows ko clearly dikhata hai.

⭐ 5. Multiple Views Provide Karta Hai

Software architecture ko multiple angles se show kiya ja sakta hai:

- **Class Diagram** (structure)
- **Component Diagram** (architecture)
- **Sequence Diagram** (interaction)
- **State Machine Diagram** (behavior)
- **Deployment Diagram** (runtime environment)

Ye views architecture ko well-defined banate hain.

⭐ 6. Consistency Checking

UML diagrams me easily identify ho jata hai ki

architecture consistent hai ya nahi.

Koi contradiction quickly visible hota hai.

⭐ 7. Design, Analysis & Code Generation Support

Modern tools like Enterprise Architect, StarUML, Visual Paradigm UML se:

- Code generation
- Reverse engineering
- Simulation

possible hota hai, jo architecture ko reliable banata hai.

⭐ Hinglish Conclusion

UML software architecture ko clearly visualize, document, communicate aur analyze karne ka strong tool hai.

Iske diagrams architecture ko understandable, maintainable aur error-free banate hain.

✅ English Answer (7 Marks)

Discuss the Role of UML in Software Architecture

UML (Unified Modeling Language) plays a major role in software architecture by providing standard notations to model, visualize, document, and analyze the architecture of a software system.

⭐ 1. Visualization of Architecture

UML turns architectural ideas into clear diagrams that help understand system structure and behavior easily.

⭐ 2. Architectural Documentation

UML supports documentation of different architectural views:

- Structural view
- Behavioral view
- Interaction view
- Deployment view

This helps in maintenance and knowledge transfer.

⭐ 3. Improved Communication

UML ensures that developers, architects, testers, and clients share a **common understanding** of the architecture, reducing misunderstandings.

⭐ 4. Representation of Design Decisions

Architectural decisions such as component decomposition, inheritance, data flow, and workflows are represented clearly through UML diagrams.

⭐ 5. Support for Multiple Views

UML provides various diagrams:

- Class Diagram
- Component Diagram
- Sequence Diagram

- State Diagram
- Activity Diagram
- Deployment Diagram

Each view helps explain a particular aspect of architecture.

⭐ 6. Ensures Consistency

UML diagrams help check consistency across components, modules, and relationships.

⭐ 7. Supports Design, Analysis & Tools

UML tools provide:

- Code generation
- Reverse engineering
- Simulation
- Automatic consistency checks

This strengthens architectural quality.

⭐ English Conclusion

UML plays a crucial role in software architecture by offering standard visual tools to design, document, communicate, and validate architecture. It improves clarity, reduces errors, and supports effective system development.

Q.3 Write a note on Active reviews for intermediate design

✅ Hinglish Answer (7 Marks)

Active Reviews for Intermediate Designs (ARID)

ARID (Active Reviews for Intermediate Designs) ek architecture evaluation method hai jiska use **intermediate design stage** par kiya jata hai.

Ye method designers, reviewers aur stakeholders ko involve karta hai taaki design ke issues, risks, aur gaps ko **early stage par identify** kiya ja sake.

ARID ka main focus hota hai design ki **sahi samajh**, **completeness**, aur **usability** ko check karna — even before the full architecture is completed.

⭐ Key Points of ARID

1. Works on Incomplete Designs

ARID ki khas baat ye hai ki design **fully complete hone ki zaroorat nahi hoti**.

50–60% ready design bhi ARID ke through review ho sakta hai.

---

2. Scenario-Based Evaluation

Is method me real-world usage scenarios use kiye jate hain.

Reviewers check karte hain ki design in scenarios ko kaise handle karega.

---

3. Active Participation

Architects, developers, testers, stakeholders sab active role play karte hain.

Ye brainstorming-style review hota hai.

---

4. Identifies Risks Early

Design me jo:

- missing parts
- inconsistencies
- performance issues

- architectural flaws

hote hain, wo ARID immediately identify kar deta hai.

---

5. Cost-Effective Method

Design stage par hi errors pakad liye jate hain, isliye later rework kam hota hai → time & cost dono save hote hain.

---

6. Improves Design Quality

ARID design ki understandability, clarity, aur maintainability improve karta hai.

---

7. Strong Communication Tool

Team members ke beech misunderstanding kam hoti hai aur architecture ki clear picture milti hai.

⭐ Hinglish Conclusion

ARID ek powerful review method hai jo intermediate design ke flaws, risks aur improvements ko early detect karta hai, jisse final architecture strong, consistent aur high-quality ban jata hai.

✅ English Answer (7 Marks)

Active Reviews for Intermediate Designs (ARID)

ARID (Active Reviews for Intermediate Designs) is an architecture evaluation technique used to assess a system's design **during the intermediate stages**.

It helps stakeholders identify risks, gaps, weaknesses, and misunderstandings **before the architecture is fully completed**.

⭐ Key Points

1. Evaluates Incomplete Designs

ARID is specifically meant for partially completed or intermediate designs.

The entire architecture need not be ready.

2. Scenario-Based Review

Evaluation is based on real usage scenarios to understand how well the design supports required behaviors.

3. Active Stakeholder Participation

Architects, developers, reviewers, testers, and users participate actively in reviewing the design.

4. Early Risk Identification

ARID identifies architectural risks, missing elements, inconsistencies, and design flaws early in development.

5. Cost and Time Efficient

Catching issues early reduces rework, development time, and cost.

6. Improves Design Quality

It enhances the clarity, accuracy, and completeness of the architecture.

7. Enhances Communication

ARID ensures all stakeholders share a common understanding of design decisions and system behavior.

⭐ English Conclusion

ARID plays a crucial role in improving architecture quality by evaluating incomplete designs early, identifying risks, and ensuring effective communication among stakeholders.

Q.4 What are the basic principles of sound documentation?

✅ Hinglish Answer (7 Marks)

Basic Principles of Sound Documentation

Sound documentation ka matlab hai aisa documentation jo **clear, accurate, complete, consistent, and easy to maintain** ho.

Software architecture documentation ko effective banane ke liye kuch important **principles** follow kiye jaate hain.

⭐ 1. Completeness (Poora hona)

Documentation me **saari zaroori information** honi chahiye:

- Components
- Interfaces
- Design decisions
- Constraints
- Diagrams

Incomplete documentation misunderstandings create karta hai.

⭐ 2. Consistency (Har jagah same pattern)

Terminology, naming conventions, symbols, diagrams sab **consistent** hone chahiye.

Agar documentation inconsistent ho → confusion hota hai.

⭐ 3. Correctness (Sahi hona)

Documentation me jo likha hai wo architecture ke actual behavior se match hona chahiye.

Galat information developer ko misguide kar sakti hai.

⭐ 4. Clarity and Simplicity (Saaf aur simple)

Documentation clear language me likha hona chahiye.

Unnecessary jargon avoid karna chahiye.

Diagrams aur examples se readability improve hoti hai.

⭐ 5. Organization (Structured hona)

Content acchi tarah organize hona chahiye:

- Headings / Subheadings
- Sections
- Tables
- Diagrams

A well-organized document easy to navigate hota hai.

⭐ 6. Traceability (Track karna easy ho)

Requirements → Design → Code → Testing

Sab ek chain me linked hona chahiye.

Isse pata chalega ki har requirement ka implementation kahan hai.

⭐ 7. Modifiability (Update karna easy ho)

Documentation ko update karna simple hona chahiye.

Design change hote hi documentation bhi update karna easy ho.

⭐ 8. Use of Standard Notations

UML diagrams, standard symbols, templates ka use documentation ko professional aur widely understandable banata hai.

⭐ Hinglish Conclusion

Sound documentation ke principles ensure karte hain ki architecture clearly explained, easily maintainable, accurate, aur consistent ho. Ye long-term project success ke liye essential hote hain.

✅ English Answer (7 Marks)

Basic Principles of Sound Documentation

Sound documentation is documentation that is **clear, correct, complete, consistent, and maintainable**.

To achieve high-quality architectural documentation, the following principles must be followed:

⭐ 1. Completeness

Documentation must include all required architectural information—components, connections, decisions, diagrams, and constraints.

⭐ 2. Consistency

Terminology, naming conventions, diagram formats, and symbols should be uniform throughout the document.

⭐ 3. Correctness

The documentation should accurately represent the real architecture.

Incorrect information leads to implementation errors.

⭐ 4. Clarity and Simplicity

Documentation should be written in simple and clear language.

Diagrams and examples improve readability.

⭐ 5. Organization

The document should be logically structured with proper headings, sections, diagrams, and tables for easy navigation.

⭐ 6. Traceability

Each architectural element should be traceable back to requirements and forward to design, implementation, and testing.

⭐ 7. Modifiability

Documentation must be easy to update when architecture changes.

⭐ 8. Use of Standard Notations

Using UML diagrams and standard modeling conventions ensures the documentation is understandable by all stakeholders.

⭐ English Conclusion

The basic principles of sound documentation ensure that architectural documents are effective communication tools, easy to maintain, and accurate representations of the system.

Q.5 write a short on Architectural patterns

✅ Short Note on Architectural Patterns (Hinglish + English Combined)

Hinglish Version

Architectural Patterns software architecture ke ready-made solutions hote hain jo system ke structure, communication, aur behavior ko define karte hain.

Ye patterns reusable hote hain aur common architectural problems ko solve karte hain.

Architectural patterns high-level blueprints provide karte hain jisse system **maintainable, scalable, reliable aur structured** ban jata hai.

Common Architectural Patterns:

- **Layered Architecture** – System ko layers me divide karna.

- **Client–Server Architecture** – Client request bhejta hai, server process karta hai.
- **MVC (Model–View–Controller)** – UI aur logic ko separate karta hai.
- **Microservices** – System ko small independent services me todna.
- **Pipe and Filter** – Data processing ko filters ke through pipeline me pass karna.

Importance:

- Reusability badhata hai
- Design decisions simplify karta hai
- System ka structure clear hota hai
- Best practices follow karna easy hota hai

English Version

Architectural Patterns are high-level design solutions that provide a reusable way to structure software systems. They describe fundamental organization principles, communication styles, and component interactions.

They help architects design systems that are **scalable, maintainable, modular, and easy to evolve**.

Examples include:

- Layered Architecture
- Client–Server
- MVC
- Microservices
- Pipe-and-Filter

Why They Are Important:

- Provide proven solutions
- Improve system quality

- Reduce design complexity
- Promote consistency and best practices
