Student Referral Sheet – Faculty of Computing (Final Examination Jan-June 2025)

Student ID :

Module Code :

Client server
Server-centralized system managing logic, data,files. Clients communicate only server. Communication network HTTP, SMTP, RPC. server serve multiple clients simultaneously. ()
Pros - Multiple clients can use the same server, Easy to scale clients, Centralized management - security and updates easier. **Cons** - Single point of failure, downtime of server affects all clients, Higher initial server cost
When - clients need to share access to the same resources.(Outlook Email, Gmail)

Component-based
divides a system reusable components. component has specific function, component interact through well-defined interfaces.
Pros- Reusable, Replaceable, Loosely coupled, scalability Easier to test and maintain, Can be replaced or updated independently. **Cons** - Managing large number of components is hard, Component compatibility issues, Performance overhead. **When**- large systems using reusable parts, expect future changes, integrating third-party libraries or plug-ins.(.jar file, .ddl file)

Event-Driven Architecture (EDA)
Reacting to events – asynchronous, loosely-coupled communication.
Mediator – central event mediator that orchestrates the flow, multi-step workflows.**When**: Event processing involves sequential steps, centralized workflow logic. **Pros**:Centralized control.easier workflow management.**Cons**:Single point of failure, less scalable.
Broker – processors react independently to events, decoupling and scalability.**When**: Event processors work independently, prioritize scalability, service addition/removal without affecting others. **Pros**: High scalability, easy service addition/removal. **Cons**: No central coordination, harder to trace flows.

Domain-Driven Architecture
designing software based on complex domain(Business) logic, collaboration between domain experts and developers. uses ontology - reflect real world entities, relationship. ubiquitous language — common terms understood by all stakeholders. **Pros**- Aligns software with real-world problems. Improves understanding. scalability in business logic. **Cons**- complex. Hard to maintain if business rules change frequently
When- complex business rules and logic. require close collabarate with domain expert. deep understanding of the problem

Layered Architecture
system functionality into separate layers. Each layer handles specific responsibilities. layers Stacked. **Pros**- Separation of Concerns. Layers can be reused. independent development of layers. **Cons**- Slow due to passing through multiple layers. communicate only with own layers and below layers. Changes in lower layers may affect upper layers. **When**-requires separation between UI, logic, data. organized, maintainable, scalable code.(TCP/IP)

N-Tier Architecture
layer can be deployed on separate physical servers. distributed deployment of Presentation, Logic, and Data layers. **Pros**- each tier can scaled independently. sensitive data can be isolated at the data tier. failure in one server doesn't crash all. **Cons**- Slow due to Network latency. Complex configuration. Higher maintenance cost. **When** - build large-scale, enterprise web applications. distribute workloads across servers for better performance.(Commercial Web Applications)

Object-Oriented Architecture
system as a collection of interacting objects. Objects communicate through method calls. Based OOP principles. Extensibility. Slow due to abstraction. **pros -** objects and classes can be reused, easy to extend behavior through inheritance. **cons -** Low speed due to abstraction, Initial design complex and cost, require training for teams. **When** - model real-world entities in your software. reusable and modular components. complex systems benefit from structure and abstraction.

Pipe and Filter Architecture
Components are called filters. filter processes input and produces output. Connectors are pipes — pass data one filter to next. Suitable for stream processing and data transformation pipelines. easy to understand, Reusability. **Pros** - easy to understand and implement, filters can added/removed without affecting others, filters can run in parallel. **Cons** - no communication between filters,need common data format across all filters. **When** - series of processing steps for data, reuse filters in different combinations, concurrent processing. (Unix/Linux Shell)

Publish-Subscribe Architecture
Publishers: Send messages without knowing receivers. Subscribers: Register to receive messages. Message Broker or Proxy may manage delivery. **pros** - Loose coupling. Scalable - supports many subscribers efficiently. asynchronous. **cons** - One-way communication.Difficult to trace issues.**When**-Multiple users need to receive the same data.(News alert,Mobile App Push no)

Microkernel Architecture (Plugin Architecture)
core system that provides essential functionality. Additional features added as plug-ins. communicate via interfaces.**Pros**- Easy to extend with plug-ins. stable and simple. Plug-ins can be added/removed at runtime. **Cons** Performance low if too many plug-ins. Testing the full system can be hard. Tight integration between core and plug-ins. **When**- build customizable product, add or update features runtime without affecting the core.

Service-Oriented Architecture (SOA)
built by combining reusable services. Services accessed through communication protocols - HTTP, SOAP, WSDL . XML or JSON for data exchange. interoperability across platforms, vendors, and technologies. **Pros -** services can be reused across applications, Loose coupling, Interoperability. **Cons** - Complexity in management. Performance overhead due network communication. **When** - integrate systems built in different technologies. business logic reuse

Microservices Architecture
System is broken into many small services. specific task and can be developed, deployed, and scaled independently. can use its own language, framework, and database. Communicates via REST APIs, message queues, or gRPC. **Pros** - scaling of services based on demand. (CI/CD) supported. failure in one service doesn't crash whole app. **Cons** - Testing and debugging can difficult. Complex network communication between services. **When** - teams to develop and deploy features quickly. CI/CD. scale independently.

Message Bus Architecture
communicate asynchronously by passing messages. Messages go through message bus. loose coupling between components. **Pros** -Easy to add/remove systems (extensible). integrate different technologies. **Cons** -Monitoring and debugging complex. Requires middleware setup. **When** - integrating multiple heterogeneous systems. WSO2 ESB

Cloud Architecture
provides access to a shared pool of computing resources (servers, storage, apps).multiple customers sharing the same infrastructure. **Pros** - scalable on demand. Pay-as-you-go – reduces upfront cost. No need for physical infrastructure management. global access. **Cons** - Compliance and legal issues. Vendor lock-in (hard to switch providers)
iaas - You Get -Virtual servers, storage, network - AWS EC2
paas - You Get -App platform & tools - Heroku, App Engine
SaaS - You Get -Ready-to-use software - Gmail, Salesforce
When - quickly scale resources, delivering web-based software to many users.

Monolithic Architecture
builds the entire system as single unit. functions are combined and run in one process, sharing same codebase and memory . **Pros** – Simple development ,deployment, better performance, easy to test in small applications, suitable for small teams. **Cons** – Hard to scale individual parts, difficult to maintain and update, any change requires full redeployment, tightly coupled components. When – small to medium systems, fast development,simple business logic,

Lec8 - Enterprise Application Intergration
Why SOA- Older tech like CORBA, EJB, DCOM were tightly coupled and platform-dependent. They used proprietary binary formats and lacked interoperability. Mostly relied on commercial tools. in SOA: Uses open standards like XML, Works across any platform, Can be built with open-source tools.
Web services allow different systems to communicate over the internet. They use XML to format and exchange data. platform-independent way. SOAP is the protocol that sends these XML messages, over HTTP. A SOAP message has an envelope - wraps the message, header - extra info, body - actual data. WSDL (Web Services Description Language) is an XML document that describes how to use a web service—what functions it offers, what data it needs, and where it's located. This helps systems automatically generate code to connect to the service.
In the **web services architecture**: The Service Provider creates and publishes the service,The Service Consumer uses the service, Service Registry (UDDI) stores service info so consumers can find it.
Bottom-up (code first): Write the code, then generate WSDL.
Top-down (contract first): Create the WSDL, then generate the code.

Lec 5 -Presentation Layer Patterns

Model-View-Controller (MVC)
Separates the application into - Model (data), View (UI), and Controller (logic). in J2EE - Model (EJB), View (JSP), and Controller (Servlet)Promotes clear separation of responsibilities. **When**:you need to separate UI, business logic, and data. Ideal for large applications where the UI changes often, but the underlying logic remains stable.
Pros:Clear separation of concerns, Easier maintenance and scalability, Supports parallel development. **Cons:** Complex to implement. Requires managing multiple files and classes

Front Controller. Pattern - Command pattern
Handles all incoming requests through a single controller, centralizing control logic (routing, authentication). **When**: Use when you have common logic across multiple pages (login, logging, routing). Best for complex web app(dashboard). **Pros**: Centralized request handling, Easier to manage and update shared logic, Reduces duplication.**Cons:** Can become performance bottleneck, Debugging becomes harder as complexity grows

Intercepting Filter .Pattern - Chain of Responsibility, Strategy, Decorator ,Command
Applies reusable filters for pre-processing and post-processing of requests, such as logging, authentication, and compression. **When**: Use when requests must pass through multiple checks or transformations before reaching core application logic.(pre-proccessing/ post-processing). **Pros**: Reusable and pluggable filters. Centralized and modular request processing Improves code organization.**Cons**:Debugging is challenging. Filter execution order must carefully managed

View Helper. pattern - Strategy, Factory Method, Template Method
Moves logic related to data formatting or business operations out of the view and into reusable helper classes(java bean).**When**: Use when view templates (e.g., HTML or JSP) are coplex with logic or calculations, and you want cleaner, more maintainable views. **Pros**: Cleaner, more readable views, Promotes reusability of helper code, Enhances separation of concerns
Cons: Risk of overusing helpers. May still lead to mixed complex if not managed properly

Composite View
Builds a complex UI from smaller, reusable parts like headers, footers, and content sections.
When:Use when pages consist of multiple dynamic sections or share common layout components across different views (e.g., dashboards).**Pros**:Promotes UI component reuse. Easier maintenance and updates. Suitable for dynamic content rendering.**Cons**:Can negatively affect performance. Layout management can become complex

Lec 6 -Business Layer Patterns

Business Delegate
Acts as an intermediary between the presentation layer and business services to reduce direct coupling(decoupling business and presernation server). without delegater cause security issue.
When: presentation layer frequently interacts with business services and you want to avoid tight coupling and centralize control. **Pros**: Decouples UI from business logic. Centralizes access and control. **Cons**: Adds an extra layer. Can lead to unnecessary complexity if overused

Service Locator
Provides a centralized registry for locating and caching business services.**When**: services are accessed repeatedly, and service lookup (e.g., via JNDI) is expensive or time-consuming. **Pros**: Improves performance by caching service references. Reduces lookup overhead **Cons**: Can hide service dependencies. May introduce a global state, making testing harder.

Session Facade
Wraps multiple business objects to expose a unified interface for clients, simplifying their interaction.**When**:Use when a client needs to access many fine-grained business components, and you want to reduce network traffic and simplify access.(without session facde want many netwark calls) **Pros**: Simplifies client-side code. Reduces network calls and dependencies. **Cons**:Facade can become overly complex. May turn into a bottleneck if not well-managed ex-ESB

Composite Entity
Manages a group of interrelated persistent objects as a single entity.**When**: Use when multiple dependent objects share a common lifecycle and should be treated as a unit. **Pros**: Reduces database access. Simplifies management of related entities.**Cons**:Tight coupling of components. Difficult to change or reuse individual objects independently
Lazy Loading in Composite Entity- In a Composite Entity, you might have a main object (Customer) and related objects (Order/Address). With lazy loading, the related objects (orders,address) are not loaded immediately(dynamically) when you load the customer.

Transfer Object (DTO)
Carries data between layers, especially across remote calls, to avoid multiple fine-grained interactions(multiple method call). **When**:Use when data needs to be transferred frequently between client and server to reduce communication overhead. **Pros**: Minimizes remote calls. Encapsulates data in a single object.**Cons**: May include unused data. Can cause versioning issues during system evolution

Transfer Object Assembler
Assembles data from multiple business objects into a single transfer object for client use.
When:Use when clients need composite data from multiple sources or layers.**Pros**: Combines data into a single object. Simplifies client-side data handling. **Cons**: Increases complexity. Creates more dependencies between layers

Value List Handler
Manages retrieval and handling of large data sets, including sorting, filtering, and pagination.
When: Use when the UI requires working with large lists that need to be efficiently paginated or filtered.**Pros**: Improves UI performance and responsiveness. Efficiently handles large result sets
Cons: Complex state management. Requires additional logic for paging and filtering

Lec8 - Enterprise Application Intergration Cont.
Integration Challenges: Point-to-Point - Number of connections: - N(N-1)/2 ,Difficult to manage, especially if endpoint details (like IP addresses) change. This impacts Service Transparency.
Enterprise Service Bus (ESB) -**Solution to Point-to-Point Complexity**:A centralized approach (Hub-Spoke Model) using a Service Proxy or Gateway,If an endpoint changes, only the proxy configuration needs updating,Components communicate with the ESB, which handles routing. Key Functions of an ESB: Promotes asynchronous message mediation,Message identification and routing.
Allows messages across different transport protocols,Message transformation.Secure, reliable communications,Extensible architecture (pluggable components).
EsB work flow- Incoming Request: First received by the Proxy Service. Sequence Begins: The request goes through a series of steps. Mediators (Synapse) handle the request. They can: Log the request ,Transform or Filter data ,Validate the message ,Error Handling: If there's an error, it's sent to a Fault Sequence, Routing: The message is routed to the correct Destination.
Enterprise Integration Patterns : 1.Message Channel: Establishes a path for applications to send messages to each other. , **2.Message**: Defines the piece of information that applications exchange over a message channel.([App A] --> [Message] --> [App B]) **3.Pipes and Filters:** Breaks down complex message processing into a series of independent steps (filters) connected by channels (pipes) - **intercepting filter.** ([Input] --->[filter manager ([Filter 1] , [Filter 2] , [Filter 3]] --> [Output]) **4.Message Router:** Examines a message and sends it to different channels based on certain conditions, without changing the message itself. **5.Message Translator:** Converts a message from one format to another, allowing systems with different data formats to communicate.-**adpter pattern**. ([Format A Message] --> [Translator] --> [Format B Message]) **6.Message Endpoint**: Provides the specific connection point for an application to send or receive messages through a channel.([App] <---> [Message Endpoint] <---> [Channel])

Mediator pattern - A mediator is the basic, full-powered message processing unit in the ESB. • A mediator can take a message, carry out some predefined actions on it, and output the modified message. • Usually, a mediator is configured using XML. • Different mediators have their own XML configurations. • At the run-time, a message is injected in to the mediator with the ESB run-time information. • Then this mediator can do virtually anything with the message.

Lec04 - Architectural Activities & Design Process

ABC Cycle

1.Stakeholders- People who affected by the system

2.Business Goals- Organization's goals that architecture support. **3.Technical Environment -** Existing technologies systems must considered. **4.Architect's Influence -** Decisions made to balance business goals, tech constraints, and stakeholder needs. **5.Resulting Architecture -** Final design chosen to meet all requirements. **6.System Qualities -** Non-functional goals the architecture supports.

Key Architectural Activities

1. Creating Business Case - Assessing goals: cost, market, timeline, limitations **2. Understanding Requirements -** Capturing functional and non-functional needs **3. Creating/Selecting Architecture -** Design/select architecture pattern. **4. Documenting & Communicating -** Making sure all stakeholders understand the design(Creating system diagrams). **5. Analyzing/Evaluating -** Comparing options to pick the best design.(Use ATAM , SAAM) **6. Implementing -** Building the system as per architecture. **7. Ensuring Conformance -** Making sure the final system matches the original design

Alternative Design Strategies

1. **Standard**: Uses a common, predefined approach. Follows best practices or well-known methods.
2. **Linear Model**: Design flows in one direction, stage by stage. No going back to previous steps. Simple but rigid.
3. **Cyclic Model**: Repeats stages in cycles.Allows continuous improvement.Useful for iterative development.
4. **Reversible Process:** Can go back to earlier stages if needed, Helps correct mistakes or make changes later.
5. **Parallel Design:** Multiple solutions or ideas are developed at the same time. Increases creativity and comparison between alternatives.
6. **Adaptive ("Lay tracks as you go"):** Next step is decided only after the current stage is complete. Allows flexibility and adjustments on the go.
7. **Incremental Design:** Design improves gradually in small steps. Each stage adds value to the previous version.Good for evolving systems.

Lec 9 - Architectural Structures and Views

View: A view is how the system appears to a particular stakeholder based on a specific aspect. Different people see the system in different ways depending on what ther perspective.

***4+1 View model* : Logical View -** Functional requirements(Users, Designers). **Development View** - Code organization(developer). **Process View** - Runtime behavior & performance (Integrators, DevOps). **Physical View -** Deployment and hardware(System Engineers). **Scenarios -** Use cases to tie everything.

Structure: The structure is the actual organization of components, modules, and their relationships in the system. It shows how the system is built, not just how it's viewed.(actual code, modules, classes, functions, and their dependencies in the source code)

1.Module Structures - breaking a big system into smaller modules (folders and files).Each module does a specific job. Decomposition: Shows how big modules are split into smaller ones. Like splitting a big assignment into smaller parts. Uses: Shows which modules depend on which others.(Module A uses Module B). Layers: Organizes modules into layers (UI → Logic → Data). Class (Generalization): Shows inheritance or type relationships.

2.Component & Connector- describes the system as it is running,focusing on how different components communicate and interact.Process (Communicating Processes)- processes and how they talk (e.g messaging). Concurrency– Shows parts that run at the same time using logical threads. Shared Data (Repository)– Components accessing/storing persistent data (e.g databases). Client-Server– Shows clients requesting and servers responding via protocols HTTP

3.Allocation Structures – shows how software is mapped to hardware and environments (servers, files, teams). Deployment – Maps software to hardware (microservices to servers). Implementation – Maps code modules to folders/files (Git file structure). Work Assignment – Assigns modules to teams or developers.

Lec 14- Software Architecture Evaluation

helps ensure a software system's architecture aligns with its intended goals and quality attributes. It aims :Identify deficiencies early in the development process, Prevent disasters caused by poor architectural decisions, Enable cost-effective design changes during the architecture phase rather than after implementation, Support decision-making regarding schedules, budgets, team structure, and performance goals.

When : Early Evaluation: After the architecture is specified but before implementation begins. Ongoing Evaluation: During design to assess already made architectural decisions. Late Evaluation: After implementation

Evaluation methods categories:1. Qualitative Evaluation: Based on expert judgment, stakeholder interviews, and scenario analysis. 2. Quantitative Evaluation: Involves measuring metrics like performance, reliability, or modifiability using mathematical models.

Evaluation Techniques : Scenario-based Methods (e.g., ATAM, SAAM, ARID), Mathematical Model-based Methods, Simulation-based Techniques, Experience-based Reasoning

Who: Evaluation Team: Experts conducting the review. Stakeholders: Anyone with a vested interest in the system, including business owners, developers, testers, users, and architects.

Pros : Clarifies quality goals, Helps resolve conflicting priorities, Promotes cross-stakeholder communication, Improves architecture documentation. Encourages reuse across projects.

Outputs of Architecture Evaluation

Prioritized Quality Attribute Requirements - documented and ranked list of quality attributes. Mapping Approaches to Attributes - Shows how architectural decisions support quality attributes. Risks and Non-risks - Identifies architectural decisions that pose a risk or are based on fragile assumptions.

Main Evaluation Methods

1. **ARID** (Active Reviews for Intermediate Designs) - Focuses on reviewing *partial(evaluate incomplete) designs* (components or subsystems). Lightweight, stakeholder-centric, and usable even without full documentation.Helps identify if design is suitable for real-world use.

Steps: • Identify reviewers (engineers who will use the design). • Designer presents the design overview. • Stakeholders brainstorm realistic usage scenarios. • Stakeholders try to use the design in those scenarios (pseudocode). • Issues and misunderstandings are documented.

When to Use: Intermediate design phases.**Focus**: Detailed design and implementation issues.**Strengths**: Early problem detection, active stakeholder participation.**Limitations**: Limited to design details, requires preparation.

2. **SAAM** (Software Architecture Analysis Method) - Scenario-based method to analyze architecture impact from different viewpoints. Validates architecture's ability to meet quality goals *before development*.

Steps: • Specify scenarios and constraints. • Describe candidate architectures (static and dynamic views). • Brainstorm and prioritize scenarios. • Simulate and analyze architecture behavior under scenarios. • Record results and risks. **When**: Comparing multiple architectures.**Focus**: Quality attributes like modifiability, reliability.**Strengths**: Clear comparison of architectures, systematic approach.**Limitations**: Time-consuming, needs detailed scenarios.

3. **ATAM** (Architecture Tradeoff Analysis Method) - structured method for evaluating how architecture handles competing quality attributes. Identifies trade-offs, risks,sensitivity points in design decisions. Works well in large, complex systems ,*Early Evaluation,OngoingEvaluation*

Process: • Introduce stakeholders and outline business drivers. • Describe the architecture in detail. • Elicit quality attribute scenarios from stakeholders. • Analyze how the architecture supports or fails under those scenarios. • Identify trade-offs, risks, and sensitive decisions. • Propose mitigation strategies and prepare a comprehensive report.

When to Use: Comprehensive evaluation of architecture.**Focus**: Trade-offs between quality attributes (performance, security).**Strengths**: Identifies risks, involves diverse stakeholders. **Limitations**: Resource-intensive, needs clear business goals.

Lec 11 - Quality Attributes

Functional Requirements: Specify what the system must do, including expected behaviors and reactions.**Non-Functonality:**Non-functional requirements are not directly linked to any specific function.**Quality Attribute** - describes how well a system performs. **Business Requirements -** business level expectations and decisions . affect the scope, cost, delivery time. **Constraints-** These are fixed rules you must follow. Tech stack, Must follow regulations

Type of Qulity attribute - 1. Design-Time Qualities - **Modifiability**: Ease of changes at minimal cost. **Reusability**: Components can be reused in other systems. **Maintainability**: Ease of correcting faults, updating, or improving. 2. Runtime Qualities - **Performance**: fast system response / process request. Measured via latency (response time) and throughput (number of operations complete per time unit). **Reliability**: System work without failure. **Availability**: Ensures the system is up and running when users need it. Affected by maintenance, attacks(DOS), system load, etc. Availability = (up time / down time) * 100% **Security**: Protection from unauthorized access and attacks. **Scalability**: Ability to handle increasing loads (horizontal-Adding more servers/vertical scaling-Update one machine power.). **Interoperability**: different systems, applications, or components to work together, exchange data, and use the information. 3. System Qualities - **Testability**: Ease of writing and executing tests. how much time take to test. how much can be test. **Supportability**: Ease of providing support, debugging, and logging. 4. User Qualities - **Usability**: How user-friendly the system is (learnability, efficiency, error handling). **Accessibility**: Support for users with disabilities or different needs.

Fault tolerance - the system should is remain operational even if some module fail , consistency-maintaining data integrity across the modules.

Lec 12 - Quality Attribute Scenarios

formal and structured way to express non-functional requirements.

1. Source – Who/what initiates the event. 2. Stimulus – What event occurs. 3. Environment – When/under what conditions it occurs. 4. Artifact – What part of the system is affected. 5.Response – How the system should respond. 6. Response Measure – How to measure that the response was acceptable

Lec 13 - Quality Attribute Tactics - architect makes to help the system achieve a quality. tactics working together is called an architectural strategy.

Availability Tactics : Ping/Echo - Send a ping to a component and wait for a reply. **Heartbeat -** Component sends periodic signals to confirm it is alive. **Exceptions**:-Trigger an exception handler when a fault occurs **Active Redundancy** - All replicas process requests; output from first to respond is used. **Passive Redundancy** - A primary handles requests; backup is updated in real-time. **Spare -** Use generic backup component that replaces failed parts. **Removal from Surface** - Periodically reboot or restart to avoid long-term faults. **Transactions** - Group operations so they can be rolled back together if one fails.

Modifybility Tactics : Maintain Semantic Coherence - Keep related responsibilities together in the same module. **Anticipate Expected Changes** - Group elements change together, whan feature change. **Generalize the Module** - Design a module develop general way. **Hide Information** - Keep internal data hidden to reduce dependencies. **Maintain Existing Interfaces** - Keep interface unchanged even if internal logic changes use adapter pattern. **Use an Intermediary** - Introduce a component to handle communication between others.

Security Tactics : Increase Computational Efficiency - Use better algorithms to reduce CPU work. **Reduce Computational Overhead** - Avoid redundant computations. **Introduce Concurrency** - Handle multiple operations simultaneously. **Maintain Multiple Copies** - Cache frequently accessed data. **Increase Available Resources** - Add more or better hardware. **Scheduling Policy** - Prioritize or organize resource access

Testability Tactics : Increase Computational Efficiency - Use better algorithms to reduce CPU work**. Reduce Computational Overhead -** Avoid redundant computations**. Control Frequency of Sampling -** Reduce how often monitoring/sampling occurs**. Introduce Concurrency -** Handle multiple operations simultaneously. **Maintain Multiple Copies -** Cache frequently accessed data**. Increase Available Resources -** Add more or better hardware**. Scheduling Policy -** Prioritize or organize resource access. **Optimize Resource Usage -** Ensure resources are fairly and fully utilized

Usability Tactics : User Initiatives - Allow users to control operations during execution. **System Initiatives -** System provides guidance, updates, or confirmations. **Mixed Initiatives -** Combine both user and system feedback for better experience**. Separate UI from Logic -** Separate user interface from business logic. **Separate UI Components -** Modularize UI for better maintainability. **Maintain Semantic Coherence** - Keep related UI functions together, anticipating areas of frequent change

Lec 15- Trade-Off

In software architecture, a trade-off happens when you increase one quality attribute , but at the decrease of another. **Stakeholder Expectations:** End User's view:-Performance, Availability, Usability, Security. Business Community's view: Time To Market, Cost and Benefits, Projected life time, Targeted Market, Integration with Legacy System, Roll back Schedule. Developer's view: Maintainability, Portability, Reusability, Testability

Quality attributes ldirectly tied to:Cost, Schedule, Scope

Trade-offs and Mitigation Plans - Complexity vs. Simplicity -(**Error**-Introducing adapters and integration layers adds architectural and maintenance complexity, **Solution**-Use clear API documentation and interface standards (OpenAPI/Swagger),Implement robust error handling and logging for debugging cross-stack communication.) Performance vs. Interoperability - (**Error** - Interoperability layers (e.g., adapters, translators) slow down system performance. **Solution-** Use lightweight formats (like JSON), async messaging, and clear API standards (e.g., OpenAPI).) Resource Allocation - (**Error** - Requires more time, skilled staff, and infrastructure. **Solution**-Use phased rollout, reuse components, and optimize resource planning.)

Lec 16- Software Architecture Frameworks

Business Benefits -•Helps an Organization achieve its business strategy, •Faster time to market for new innovations and capabilities, •More consistent business process and information across business units, •More reliability and security, less risk

I**T Benefits ,**•Better traceability of IT costs, •Lower IT costs – design, buy, operate, support, change, •Faster design and development,•Less complexity, •Less IT risk

Key architecture principles- •Build to change instead of building to last, •Model to analyze and reduce risk, •Communication and Collaboration, •Identify key engineering decisions

TOGAF (The Open Group Architecture Framework) - provides a detailed Architecture Development Method (ADM) to plan, design, implement, and govern enterprise architecture.

Benefits: Standardized and repeatable process, Supports alignment between business and IT, Helps manage complexity in large organizations, Encourages reuse of architecture components, Covers all aspects(business, data, application, and technology). **Phase -** Preliminary Phase - Define how architecture work will be done. Phase A – Architecture Vision - Identifies Concerns and Requirements. Confirms business goals, drivers, constraints High-level vision of the architecture. Phase B – Business Architecture - identify Target Business Architecture it show how the business. Phase C – Information Systems Architecture (Use ER Diagrams, Class Diagrams.)(Use Component Diagrams, App Flow). Phase D – Technology Architecture Define hardware and software platform. Phase E – Opportunities & Solutions - Identify the best way to implement the architecture. Phase F – Migration Planning - Develop detailed plan for implementation. Phase G – Implementation Governance - Ensure the implementation follows the architecture. Phase H – Architecture Change Management - Continuously monitor and adapt the architecture. Requirements Management (Central Process) - Handle changing requirements across all phases.

Zachman Framework - A structured matrix for organizing architectural artifacts (documents, models) based on 6 questions (What – Data,How – processes, Where– Network (locations), Who – People (responsibilities), When – Time (events, cycles),Why – Motivation (goals, strategy) and 6 perspectives (Contextual (Planner) – Scope / high-level overview , Conceptual (Owner) – Business model / enterprise vision, Logical (Designer) – System model / logical design, Physical (Builder) – Technology model / platform-specific, As Built (Integrator) – Actual implementation, Functioning (User) – Running system in operation). **Benefits** - Clear separation of concerns, Helps identify and fill architecture gaps, Useful for documentation and communication, Supports decision-making from multiple viewpoints, Encourages consistency across the enterprise