

Architectural diagram of all major components

Analysis of Architectural Patterns in Software Development						
Architectural patterns	Agility	Ease of development	Testability	Performance	Scalability	Ease of deployment
Layered Pattern	High Promotes modularity and separation of concerns.	High Clear separation allows for focused development.	High Isolation of layers aids unit testing.	Low Can introduce overhead due to multiple layers.	Low Scaling might be constrained by layer dependencies.	High Clear division eases component deployment.
Client-Server Pattern	High Clear separation enables independent development.	High Clear separation allows for focused development.	High Isolation of layers aids unit testing.	Variable Can introduce overhead due to multiple layers.	High Scaling might be constrained by layer dependencies.	High Clear division eases component deployment.
Event-Driven Pattern	High Decoupled components enable flexible changes.	High Components react to events independently.	High Events can be simulated for testing.	High Efficient for asynchronous and reactive systems.	High Can scale by distributing event processing.	High Event handlers can be updated independently.
Microkernel Pattern	High Core system can adapt to various extensions.	High Extensions developed independently.	High Core and extensions can be tested separately.	Low Indirection might introduce some overhead.	High Extensions can be added or removed.	High Extensions can be deployed independently.
Microservices Pattern	High Services developed and deployed independently.	High Small teams can focus on services.	High Services can be tested in isolation.	Variable Depends on service interactions.	High Individual services can be scaled.	High Services can be deployed separately.
Broker Pattern	High Decoupling between components promotes agility.	High Components communicate through a broker.	High Components can be tested in isolation.	Variable Broker introduces a potential bottleneck.	High Broker can be optimized for scalability.	High Components connect through the broker.
Event-Bus Pattern	High Decoupled components react to events.	High Components focus on handling events.	High Events can be simulated for testing.	High Efficient for event-driven systems.	High Event processing can be distributed.	High Event handlers connect via the bus.
Pipe-Filter Pattern	High Filters can be added or modified.	High Components focus on single tasks.	High Filters can be tested independently.	High Efficient for data processing pipelines.	High Can add more filters as needed.	High Filters can be arranged and deployed.
Blackboard Pattern	High Allows diverse knowledge sources to contribute.	High Independent contributors update the blackboard.	High Blackboard's state can be observed.	Variable Depends on blackboard coordination.	Low Complex coordination might limit scalability.	High Contributors update the blackboard.
Component-Based Pattern	High Components can be developed and replaced.	High Independent development of components.	High Components can be tested in isolation.	Variable Depends on component interactions.	High Components can be replicated or replaced.	High Components are deployed individually.
Service-Oriented Architecture	High Loose coupling facilitates adaptability to changes.	High Services developed independently.	High Services can be tested in isolation.	Variable Depends on service interactions.	High Individual services can be scaled.	High Services can be deployed separately.
Monolithic Architecture	Low Tightly coupled components can hinder agility.	High Single codebase simplifies development.	High Components can be tested within the monolith.	Variable Direct method calls lead to efficiency.	Low Scaling requires duplicating the entire monolith.	Low Requires deploying the entire monolith.
Space-Based Architecture	High Decoupling allows for adaptable changes.	High Components communicate through shared space.	High Components can be tested independently.	High Efficient for distributed data processing.	High Can add more nodes to the space.	High Components connect to the shared space.
Peer-to-Peer Architecture	High Decentralized structure supports adaptability.	High Peers operate independently.	High Peers can be tested in isolation.	High Efficient for distributed applications.	High Peers can be added or removed.	High Peers can join the network dynamically.
Hybrid Architecture	High Combines advantages of multiple architectures.	Variable Depends on the hybrid approach.	Variable Depends on the hybrid components.	Variable Depends on the hybrid composition.	Variable Depends on the hybrid structure.	Variable Depends on the hybrid configuration.