Most Common C++ Architectures for All Projects

1. Layered (n-Tier) Architecture

Structure: Presentation → Application Logic → Business Logic → Data Access
Use In C++: GUI apps, game engines, system tools
Pros: Separation of concerns, testable, scalable
Example: Qt applications, layered rendering engines

2. Component-Based Architecture

Structure: Systems are built from interchangeable components with well-defined interfaces.
Use In C++: Game engines (ECS), plug-in systems, device drivers
Pros: Flexible, reusable, decoupled
Example: Unity-like ECS engines (Entity-Component-System)

3. Event-Driven Architecture

Structure: Components communicate through asynchronous events.
Use In C++: GUIs, games, real-time systems
Pros: Highly decoupled, scalable for async handling
Example: Qt signal-slot system, SDL event loops

4. Microkernel (Plug-in) Architecture

Structure: Core system with independently loadable plug-ins or extensions.
Use In C++: IDEs, interpreters, CAD systems
Pros**: Extensibility, modularity
Example: LLVM/Clang, Visual Studio extensions

5. Model-View-Controller (MVC)

Structure: Separation of data (Model), UI (View), and input control (Controller)
Use In C++: GUI applications, web servers, tools
Pros: Clear separation, maintainable
Example: Qt MVC framework, ImGui bindings

6. Model-View-ViewModel (MVVM)

Structure: Similar to MVC but with binding logic in the ViewModel
Use In C++: GUI applications with reactive UIs
Pros: Testable, clean UI logic separation
Example: Used with Qt/QML

7. Hexagonal Architecture (Ports and Adapters)

Structure: Core logic (domain) is surrounded by adapters (UI, databases, etc.)
Use In C++: Highly portable systems, embedded software
Pros: Loose coupling, testable
Example: Embedded systems with multiple interfaces (serial, network, UI)

8. Service-Oriented (Modular Service) Architecture

Structure: Applications are made of loosely coupled services.
Use In C++: Backends, modular systems
Pros: Scalable, testable, reusable
Example: Distributed applications, C++ daemons using ZeroMQ or gRPC

9. Client-Server Architecture

Structure: Clients request services from central servers
Use In C++: Multiplayer games, distributed tools
Pros: Networked, scalable
Example: Game clients communicating with a game server (UDP/TCP)

10. Data-Oriented Design (DoD)

Structure: Organizes data for cache efficiency and performance.
Use In C++: Game engines, real-time simulations
Pros: High performance, predictable memory usage
Example: ECS (Entity Component System), SIMD-friendly processing