**Team 8**

Galvo, Glenn N.  
Dela Cruz, Novrizkie Kent  
Acoymo, Kerlsan S. Jr.  
Escuña, Gilbert P.  
Niñeza, Jason Louis P.

**System Integration and Architecture 1**

**Introduction**

Software architecture defines the structure of a system, outlining how components interact and work together. Choosing the right architectural style is crucial for ensuring scalability, maintainability, and performance. Different architectures serve different purposes, depending on factors such as system complexity, scalability needs, and data processing requirements.

This document explores five major software architectural styles—**Layered Architecture, Client-Server, Event-Driven, Microservices, and Repository Style**—comparing their advantages, disadvantages, and use cases.

1. **Layered Architecture**

|  |  |
| --- | --- |
| ****Aspect**** | ****Details**** |
| ****Overview**** | Organizes software into hierarchical layers (e.g., UI, business logic, data). Each layer has a distinct function. |
| ****Advantages**** | ✅ Clear separation of concerns  ✅ Easy to maintain and test  ✅ Scalable within layers  ✅ Suitable for enterprise applications |
| ****Disadvantages**** | ❌ Performance overhead due to multiple layers  ❌ Can become rigid and monolithic  ❌ Changes in lower layers can affect higher layers |
| ****Example Use Cases**** | **E-commerce systems**, CRM applications, Enterprise software |

1. **Client-Server Architecture**

|  |  |
| --- | --- |
| ****Aspect**** | ****Details**** |
| ****Overview**** | A centralized server provides services, and multiple clients request them. Common in web and database applications. |
| ****Advantages**** | ✅ Centralized control improves security  ✅ Efficient resource utilization  ✅ Scalable with multiple clients  ✅ Easier to maintain as changes on the server affect all clients |
| ****Disadvantages**** | ❌ Single point of failure (server dependency)  ❌ Network dependency can impact performance  ❌ Scalability challenges under heavy loads |
| ****Example Use Cases**** | **Web applications**, Database systems, Online banking |

1. **Event-Driven Architecture**

|  |  |
| --- | --- |
| ****Aspect**** | ****Details**** |
| ****Overview**** | Components communicate asynchronously through events. Used in real-time systems and loosely coupled applications. |
| ****Advantages**** | ✅ Highly scalable and flexible  ✅ Efficient handling of real-time updates  ✅ Loose coupling enhances modularity  ✅ Suitable for IoT and stock trading systems |
| ****Disadvantages**** | ❌ Complex debugging and testing  ❌ Ensuring data consistency is challenging  ❌ Requires careful event management |
| ****Example Use Cases**** | **Stock trading platforms**, IoT systems, Notification services |

1. **Microservices Architecture**

|  |  |
| --- | --- |
| ****Aspect**** | ****Details**** |
| ****Overview**** | Breaks applications into small, independent services that communicate over a network. Each service handles a specific business function. |
| ****Advantages**** | ✅ Highly scalable and flexible  ✅ Independent deployment  ✅ Technology agnostic  ✅ Faster development and fault isolation |
| ****Disadvantages**** | ❌ Increased complexity in managing multiple services  ❌ Network overhead can cause latency  ❌ Higher infrastructure cost and overhead |
| ****Example Use Cases**** | **Netflix**, Amazon, Cloud-based applications |

1. **Microservices Architecture**

|  |  |
| --- | --- |
| ****Aspect**** | ****Details**** |
| ****Overview**** | Centralizes data storage and access in a shared repository, with different components interacting with it. |
| ****Advantages**** | ✅ Centralized data management  ✅ Encourages reusability and maintainability  ✅ Suitable for applications requiring data consistency |
| ****Disadvantages**** | ❌ Can become a bottleneck if not optimized  ❌ Higher dependency on repository structure  ❌ Difficult to scale if data access patterns are inefficient |
| ****Example Use Cases**** | **Version control systems (Git, SVN)**, Database-centric applications |

## **Conclusion**

Selecting the right software architecture is essential for building an efficient and scalable system. Each architecture style has its strengths and trade-offs, making them suitable for different use cases.

* **Layered Architecture** is ideal for structured applications like **enterprise software**, ensuring modularity but potentially introducing performance overhead.
* **Client-Server Architecture** is great for **web applications**, offering centralized control but with possible server dependency issues.
* **Event-Driven Architecture** enables **real-time processing**, making it effective for **IoT, stock trading, and event-based systems**, though it requires careful event management.
* **Microservices Architecture** is perfect for **large-scale distributed applications** like **Netflix and Amazon**, allowing independent deployments but requiring strong infrastructure management.
* **Repository Architecture** is beneficial for systems needing **centralized data access** like **version control** but can become a bottleneck if not optimized.