What does software integration mean?

Definition

Software integration is a development process in which separate software systems—applications and components—are connected so they work together to form a new, unified system.

Phases

1.) Requirements assessment and planning

- requirements assessment: identifying user needs and business requirements
- planning: developing the integration strategy, designing the system architecture and integration points

2.) Requirements analysis and specification

- analysis: detailed assessment of required functions and the capabilities of existing systems
- specification: definition of interfaces and data exchange formats

3.) Development and implementation

- development: creation of integration code for existing systems
- implementation: deployment of new components and modification of existing ones

4.) Testing and validation

5.) Maintenance and support

Legacy Systems

Definition The term legacy system refers to IT systems that use older (possibly obsolete) technologies but are still actively operating and play an essential role in an organisation's everyday operation.

Why use legacy systems?

- Long lifetime and stability: Many legacy systems have operated reliably for years or even decades. If a system functions well and is mission-critical, there is often no compelling reason to replace it.
- Cost considerations: Replacing an entire system can be extremely expensive.
- Complexity: Legacy systems are often deeply integrated into other organizational processes and systems.
- Risk avoidance: Organizations often avoid the risk of replacement

Why Are They Not Replaced?

- Cost and lack of resources: Replacement is often not economically feasible, and risk estimation

can be difficult due to strict security requirements.
- Disruption of business processes: Introducing a new system may significantly disrupt daily operations. Many organizations prioritize operational stability (e.g., banking institutions).

Solutions

- Gradual migration: Instead of replacing the entire system at once, organizations incrementally migrate to new systems by replacing specific components or functionalities.
- Outsourcing maintenance and operation: Maintenance and operation of legacy systems may be delegated to external service providers, reducing internal costs and specialized staffing requirements.

Overview of Integration Strategies

Point to Point connection

Components connect directly to each other, typically via file transfer or direct database access. There is no intermediary layer, therefore communication is fast. Initially, it is easy to implement.

flowchart LR %% Nodes R[Radiology] EMR[EMR] CDB[Central Database] PS[Patient Search] PDB[Patient DB] ER[Emergency Dept.] FIN[Billing / Finance] PHARM[Pharmacy] %% Layout helpers (optional) %% Try to mimic the original positions by grouping subgraph Left[ ] direction TB R PS FIN end subgraph Middle[ ] direction TB EMR PDB PHARM end subgraph Right[ ] direction TB CDB ER end %% Connections (based on the diagram) R <--> PS PS <--> EMR PS --> PDB PS --> FIN EMR --> PDB EMR --> FIN EMR <--> CDB PDB <--> ER PDB <--> PHARM ER --> PHARM PHARM --> ER CDB --> ER ER --> CDB

Disadvantages – Challenges

- Difficult to scale
- Future expansion can become complex
- The number of connections grows exponentially → $n(n-1)/2$ connections
- Fragile architecture, as monitoring and troubleshooting errors are difficult

---

Middleware Integration

Components do not connect to each other directly; instead, they communicate through a central intermediary (e.g., API Gateway, Application Server, Enterprise Service Bus – ESB).

The intermediary layer handles different communication protocols.

- Monitoring capabilities: communication tracking and centralized supervision
- Improved scalability

- Centralized functions: authorization management and transaction handling

Disadvantages – Challenges

- Complexity: system development costs are high
- Monolithic architecture – one central server serving many clients

flowchart TB EMR[EMR] RAD[Radiology] FIN[Billing and Finance] ER[Emergency Department] MW[Message Oriented Middleware] PS[Patient Search] PDB[Patient Database] CDB[Central Database] PHARM[Pharmacy] EMR --> MW RAD --> MW FIN --> MW ER --> MW MW --> PS MW --> PDB MW --> CDB MW --> PHARM

Message Queue-Based Integration

Components do not connect to each other directly; instead, they communicate via message queues.

Messages are processed asynchronously.

- Monitoring capabilities: use of specialized queues (e.g., Dead Letter Queue – DLQ)
- Highly scalable architecture
- Naturally suited for cloud-based systems

Disadvantages – Challenges

- Complexity: system development costs are high
- Requires advanced expertise and careful architectural design

flowchart TB %% Top layer systems RAD[Radiology] EMR[EMR] FIN[Billing] ER[Emergency Department] %% Message Queues Q1[[Queue]] Q2[[Queue]] Q3[[Queue]] %% Bottom layer systems PS[Patient Search] PDB[Patient Database] CDB[Central Database] PHARM[Pharmacy] %% Top -> Queues RAD --> Q1 EMR --> Q1 FIN --> Q2 ER --> Q3 %% Queue interconnection Q2 <--> Q3 %% Queues -> Bottom Q1 --> PS Q2 --> PDB Q3 --> CDB Q3 --> PHARM

Data Sharing

A simple approach to integration is data sharing.

Data sharing–based integration aims to transfer and share data between systems. This enables individual systems to access and utilize data stored in other systems.

Data sharing can take several forms:

Comparison of Data Sharing Approaches

- **Data Migration** is typically a one-time process: It is suitable for system replacement or major upgrades, but it does not ensure continuous consistency.
- **Data Synchronization** provides ongoing consistency between systems: It is appropriate when multiple systems must maintain aligned datasets over time.
- **Data Sharing Services** enable real-time access to shared data: They are ideal for modern distributed and cloud-based architectures that require immediate data availability.

File-based data sharing