System Integration & Architecture

Week 1 & 2

Information - a complex concept that encompasses various aspects of knowledge.

System - set of interconnected components, elements, or parts that work together to achieve a common goal or function.

Characteristics of a Useful System

- A system is a whole
- Components of a system interact
- Systems are goal seeking
- Systems have input and output
- Systems must be controlled
- Systems form a hierarchy
- Systems exhibit differentiation

Information System - a set of interconnected components and process that collect, store, process, and disseminate information to support an originations' operations, management, and decision-making

Components of an Information System

- Hardware: computers, networks, peripherals
- Software: operating systems, applications, utilities
- Data: types, formats, storage, and retrieval
- People: users, developers, administrators
- Procedures: business process, workflow

Types of Information Systems

- **Transaction Processing Systems** (TPS) - handle routine transactions, such as sales, inventory management, and payroll processing.
- **Management Information Systems** (MIS) - Provide managers with relevant information to support decision-making.
- **Decision Support Systems** (DSS) - Assist decision-makers in making informed decisions by analyzing data and providing recommendations.
- **Knowledge Management Systems** (KMS) - Manage and share knowledge within an organization.
- **Enterprise Resource Planning Systems** (ERP) - Integrate various business functions, such as finance, human resources, and supply chain management.
- **Customer Relationship Management Systems** (CRM) - Manage customer interactions and relationships.
- **Supply Chain Management Systems** (SCM) - Manage the flow of goods, services, and information from raw materials to end customers.

Other systems related to IT

- **Human Resource Information Systems** (HRIS) - these systems manage HR-related data, such as employee records, benefits, and payroll.
- **Financial Management Systems** - these systems manage financial transactions, including accounting, budgeting, and forecasting.
- **Inventory Management Systems** - these systems manage inventory levels, tracking and controlling the movement of goods and materials.
- **Network Management Systems** - these systems monitor and manage network infrastructure, ensuring reliable and secure connectivity.

- **Database Management Systems** (DBMS) - these systems manage and organize data in a database, allowing for efficient retrieval and manipulation.
- **Cloud Computing Systems** - these systems provide on-demand access to a shared pool of computing resources, such as servers, storage, and applications.
- **Artificial Intelligence (AI) Systems** - these systems use machine learning algorithms to analyze data and make decisions autonomously
- **Internet of Things (IoT) Systems** - these systems connect devices and sensors to collect data and automate processes.
- **Telephony Systems** - these systems manage phone calls, voice messages, and other communication protocols.
- **Virtual Private Networks** (VPNs) - These systems provide secure and private access to networks over the internet.
- **Content Management Systems** (CMS) - these systems manage content creation, editing, and publishing for websites and other digital platforms.
- **Geographic Information System** (GIS) - these systems capture, analyze, and display geospatial data.
- **Business Intelligence Systems** - these systems analyze data to gain insights and make informed business decisions.

System Integration - is the connection of data, applications, APIs and devices across your IT organization to be more efficient, productive, and agile.

System Architecture - is the conceptual model that defines the structure, behavior, and more views of a system.

Types of System Integration

- **Internal integrations**: a company connects its internal systems to streamline specific workflows
- **Customer-facing integrations**: a company connects its product with customers' applications to help clients get more value from their solution
- **Business-to-business (B2B) integration**: a company connects its ERP system with business partners to streamline transactions
- **Enterprise service bus (i.e., horizontal integration)**: systems connect to a "bus", or a bus-like infrastructure, that facilitates communication between the various systems

Category of System Integration

- **Enterprise Application Integration (EAI)** - EAI is the process of integrating different applications within a single enterprise. This can include applications such as financial systems, CRM systems, and ERP systems.
- **Data Integration (DI)** - the process of integrating different data sources into a single system. This can include data from databases, spreadsheets, and text files.
- **Application Integration** - connects different software applications to enable them to interact and share information.
- **Business Process Integration** - aligns and optimizes business processes across different systems and departments.

Week 3

Modeling Requirements - the process of capturing, analyzing, and documenting what a system should do using models or diagrams.

Types of Requirements

- Functional Requirements - what the system should do (e.g., login, generate report)
- Non-functional Requirements - how the system should behave (e.g., performance, security)
- Business Requirements - why the system is needed (business goals)
- User Requirements - what the user expects (user stories, personas)

Importance of Modeling Requirements

- Ensures clear understanding between stakeholders
- Reduces project risk
- Serves as a blueprint for developers
- Facilitates better testing and validation
- Enhances maintenance and future development

Week 4

Software Architecture - refers to the high-level structures of a software system and the discipline of creating such structures and systems.

Factor of Software architecture

1. **Design** - A plan or specification for the construction of an object or system or for the implementation of an activity or process or the result of that plan or specification in the of a prototype, product or process
2. **Quality Attributes** - It includes correctness, reliability, adequacy, learnability, maintainability readability, testability, efficiency, portability
3. **IT Environment** - An integrated collection of technology component that servers the needs of its users and owner of the resulting system
4. **Human Dynamic** - A team-oriented activity involving engineers, developers, business analysts, domain experts, data/infrastructure architects, project manager, etc.
5. **Business Strategy** - It refers to the actions and decisions that a company takes to reach its business goals and be competitive in its industry

Software Design - provides a design plan that descripts the elements of a system, how they fit, and work together to fulfill the requirement the system.

Role of Software Architect

Software Architect provides a solution that the technical team can relate and design for the entire application

Expertise in the following areas:

Design Expertise - Expert in software design, including diverse methods and approaches such as object-oriented design, event-driven design, etc.

Domain Expertise - Expert on the system being developed and plan for software evolution.

Technology Expertise Expert on available technologies that helps in the implementation of the system

Methodological Expertise Expert on software development methodologies that may be adopted during SDLC (Software Development Life Cycle).

Common Quality Attributes

Category	Quality Attribute	Description
Design Qualities	Conceptual Integrity	Defines the consistency and coherence of the overall design. This includes the way components or modules are designed.
	Maintainability	Ability of the system to undergo changes with a degree of ease.
	Reusability	Defines the capability for components and subsystems to be suitable for use in other applications.
Run-time Qualities	Interoperability	Ability of a system or different systems to operate successfully by communicating and exchanging information with other external systems written and run by external parties.
	Manageability	Defines how easy it is for system administrators to manage the application.
	Reliability	Ability of a system to remain operational over time.
	Scalability	Ability of a system to either handle the load increase without impacting the performance of the system or the ability to be readily enlarged.
	Security	Capability of a system to prevent malicious or accidental actions outside of the designed usages.
	Performance	Indication of the responsiveness of a system to execute any action within a given time interval.
	Availability	Defines the proportion of time that the system is functional and working. It can be measured as a percentage of the total system downtime over a predefined period.
Architecture Quality	Correctness	Accountability for satisfying all the requirements of the system.
	Portability	Ability of the system to run under different computing environment.
Non-runtime Quality	Integrality	Ability to make separately developed components of the system work correctly together.
	Modifiability	Ease with which each software system can accommodate changes to its software.
Business quality attributes	Cost and schedule	Cost of the system with respect to time to market, expected project lifetime & utilization of legacy.
	Marketability	Use of system with respect to market competition.
	Supportability	Ability of the system to provide information helpful for identifying and resolving issues when it fails to work correctly.
System Qualities		
	Testability	Measure of how easy it is to create test criteria for the system and its components.
User Qualities	Usability	Defines how well the application meets the requirements of the user and consumer by being intuitive.

Architectural Style also called as *architectural pattern* - is a set of principles which shapes an application. It defines an abstract framework for a family of system in terms of the pattern of structural organization.

Types of Architecture

- **Business architecture** − Defines the strategy of business, governance, organization, and key business processes within an enterprise and focuses on the analysis and design of business processes.
- **Application (software) architecture**− Serves as the blueprint for individual application systems, their interactions, and their relationships to the business processes of the organization.
- **Information architecture** − Defines the logical and physical data assets and data management resources.
- **Information technology (IT) architecture**− Defines the hardware and software building blocks that make up the overall information system of the organization.

Architecture Design Process

1. **Understand the problem** - This is the most crucial step because it affects the quality of the design that follows.
2. **Identify design elements and their relationships** - In this phase, build a baseline for defining the boundaries and context of the system.
3. **Evaluate the architecture design** - It involves evaluating the architecture for conformance to architectural quality attributes requirements.
4. **Transform the architecture design** - This step is performed after an evaluation of the architectural design.

Key Design Principles

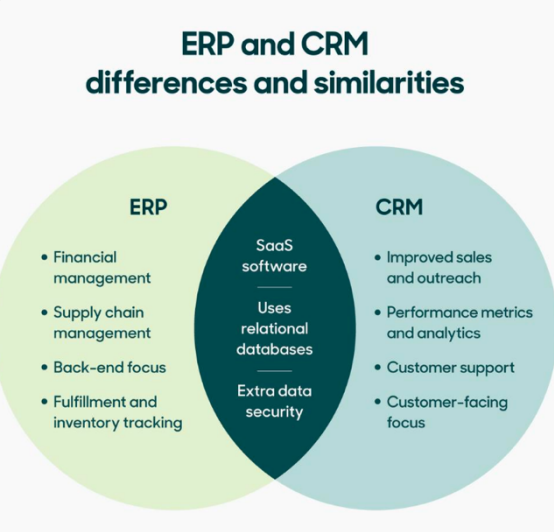
- Separation of Concerns
- Single Responsibility Principle
- Principle of Least Knowledge
- Minimize Large Design Upfront
- Do not Repeat the Functionality
- Prefer Composition over Inheritance while Reusing the Functionality
- Identify Components and Group them in Logical Layers
- Define the Communication Protocol between Layers
- Define Data Format for a Layer
- System Service Components should be Abstract
- Design Exceptions and Exception Handling Mechanism
- Naming Conventions

Week 5

Systems Integration - combines different computing systems and software applications physically or functionally to act as a coordinated whole.

Importance in Business Operation/s

1. Efficiency
2. Improved Data Accuracy
3. Enhanced Decision-making
4. Cost Savings
5. Competitive Advantage



Types of System Integration

- **Data Integration** - *involves combining data* form different sources and formats to provide a unified view.
- **Application Integration** - focuses on *connecting different software applications* and systems to enable seamless data flow and functionality.
- **System Integration** - involves *linking together various business processes* and workflows to create a cohesive and streamlined operational environment.
- **Enterprise Service Bus (ESB)** - is *a software architecture model* used for integrating different applications by establishing communication between them.
- **AI & Machine Learning Integration** - to *enhance automation, decision-making*, and predictive capabilities.
- **Cloud-based Integration** - it *enables organizations to connect disparate systems* and applications across distributed environments.
- **IOT Integration** - *devices with existing systems* is on the rise, enabling real-time data collection, monitoring, and control of connected devices.
- **Blockchain Integration** - *it enables secure and transparent data transactions*, traceability, and decentralized consensus mechanism.

Week 6: Architecture Models

Software Architecture is the *high-level structure of a software system*. It defines the components of the system, their relationships, and how they interact to achieve the desired functionality. *It acts as a blueprint for the entire software development process*, guiding decisions about design, implementation, and deployment.

Software architecture is *a critical aspect of software development, ensuring the system's quality, maintainability, and scalability*. By carefully considering the requirements and choosing an appropriate architectural style, developers can create robust and successful software systems.

Unified Modeling Language (UML) *is a standardized modeling languages* used in software engineering *for visualizing, specifying, constructing, and documenting the artifacts of software styles. It is a general-purpose of modeling language that provides a standard way to visualize the design of a system*.
UML includes a set of graphic notations techniques to create visual models of object-oriented software-intensive systems.

UML Diagrams (two main types)

1. **Structural Diagrams** are used to depict the static structure of a system. They represent the components that make up a system and the relationship between these components.

Class Diagram:
Represents the structure of a system by showing the classes of the system, their attributes, methods, and relationships between the classes.

Object Diagram:
Shows a snapshot of the instances of classes in a system at a specific point in time, along with their relationships.

Component Diagram:
Illustrates the components of a system and their dependencies. It shows how the physical components are wired together.

Composite Structure Diagram:
Describes the internal structure of a class and how its parts are interconnected, including the collaboration between part

Package Diagram:
Organizes the elements of a system into related groups (packages) to show the dependencies between different packages.

Profile Diagram:
Defines custom stereotypes, tagged values, and constraints for UML elements, allowing for extension and customization of UML.

Deployment Diagram:
Models the physical deployment of artifacts on nodes (hardware) in a distributed system, showing how software and hardware components are connected.

2. Behavioral Diagrams are used to capture the dynamic aspects of a system. They illustrate how the components of a system interact with each other and how the system's behavior changes over time

Use Case Diagram:
Describes the interactions between the system and its users (actors) to represent the functional requirements of the system.

Activity Diagram:
Represents the flow of control within a system, showing the sequence of activities or actions that need to be performed.

State Machine Diagram:
Models the behavior of an object or system in different states and transitions between these states based on events.

Sequence Diagram:
Illustrates how objects interact in a particular sequence, showing the messages exchanged between objects over time.

Communication Diagram:
Similar to a sequence diagram but focuses on the interactions between objects to achieve a specific collaboration.

Interaction Overview Diagram:
Provides an overview of the flow of control between interactions, combining sequence and activity diagrams.

Timing Diagram:
Shows how objects collaborate over time, focusing on the timing constraints between messages

Architecture View Mode (4 + 1 View Model) is a software architecture description method based on the use of multiple, concurrent views. It was proposed by ***Philippe Kruchten*** to describe the architecture of software-intensive systems. This model provides different perspectives on the system.

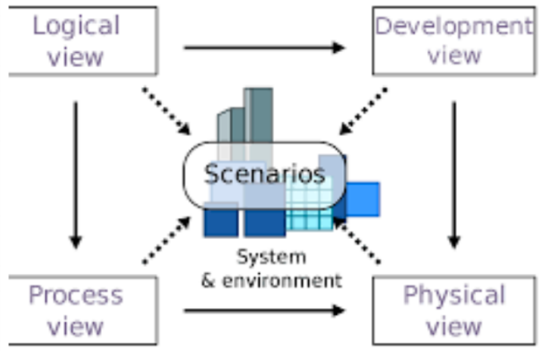
Logical View
This view focuses on the functionality that system provides to end-users. It describes the high-level design of the system in terms of components, modules and their Interactions

Development View:
This view is concerned with the software development process. It describes the software modules, components, and their relationships from a developer's perspective.

Process View:
The process view focuses on the dynamic aspects of the system, describing the system's runtime behavior. It includes information about concurrency, distribution, performance, and scalability.

Physical View:
The physical view describes the mapping of software components onto the hardware infrastructure. It includes details about the deployment of software components on physical nodes.

Scenarios (or Use Cases):
The fifth view in the 4+1 view model is the set of scenarios or use cases that describe sequences of interactions between users and the system. These scenarios provide concrete examples of how the system functions in different situations.



Architecture Description Language (ADL) is a specialized language used to formally describe the architecture of a software-intensive system.

Key features and purposes of ADLs
Formalism: ADLs often provide a formal structure for describing the architecture of a system, enabling precise and unambiguous representations.

Abstraction: They allow architects to abstract away from implementation details and focus on the high-level structure and behavior of the system.

Analysis: ADLs support analysis activities such as performance evaluation, reliability assessment, and other architectural quality attributes.

Tool Support: Many ADLs come with tools that facilitate the creation, visualization, and analysis of architectural descriptions.

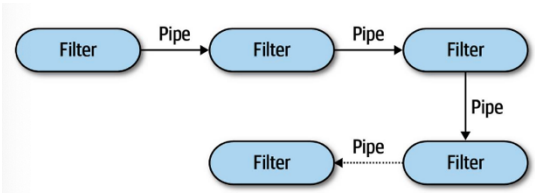
Standardization: Some ADLs are standardized languages used in the software engineering community, promoting consistency and interoperability in architectural descriptions.

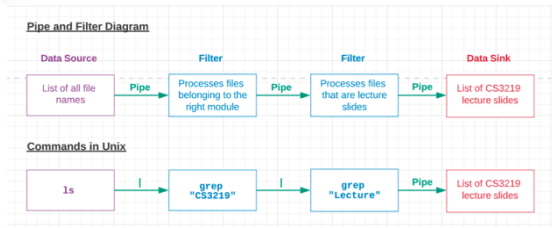
Data Flow Architecture is a design pattern used in software *engineering to describe the flow of data within a system*. It focuses on how data is input, processed, stored, and output within the software application

Three types of execution sequences

1. **Batch Sequential** is a classical data processing model, in which a data transformation subsystem can initiate process only after its previous subsystem completely through.

2. **Pipe and filter or non - sequential pipeline mode** is a design pattern used in software engineering to process data sequentially through a series of processing steps or filter.
   - **Filters** are components responsible for processing data
   - **Pipes** serve as communication channels between filters





3. **Process Control Architecture** typically refers to the design and structure of systems that *manage and control industrial processes*. These architectures are *essential for monitoring and regulating physical processes* such as manufacturing.

Types of Subsystems
- **A process control** architecture would have a processing unit for changing the process control variables and controller unit for calculating the number of changes.
- **A controller units** must have the following elements:
- **Controlled Variables** - it provides values for the underlying system and should be measured by sensors. For example, speed in cruise control system

Component-based Architecture is a design paradigm in software engineering where complex systems are decomposed into smaller, reusable pieces called components.

Components - is modular, portable, replaceable, and reusables set of well-defined functionalities that encapsulates its implementation and exporting it's as a higher-level interface. It is also a software object, intended to interact with other components.

Views of a Component

1. **Object-oriented View** - in this view, a component is seen as a set of one or more cooperating classes.

2. **Conventional View** - considers a component as a functional element or module of a program. It integrates the processing logic, the internal data structures required for the processing logic, and an interface that allows the component to be invoked and data to be passed to it

3. **Process-related View** - the focus is on building the system from existing components maintained in a library, rather than creating each component from scratch.

Popular frameworks and technologies that follow a component-based architecture

- **React.js**: A JavaScript library for building user interfaces with reusable components.

- **Angular**: A TypeScript-based framework for building web applications with a component-based architecture.

- **Vue.js**: A progressive JavaScript framework for building interactive web interfaces with reusable components.