DAY 01: -(15/07/2024)

# System Software:-

System software is a type of computer program that is designed to run a computer’s hardware and application programs and examples of system software include operating systems (OS) (like macOS, Linux, Android, and Microsoft Windows), game engines, search engines(like google,Bing,Yahoo!), industrial automation, computational science software, and (SaS)software as a service applications.

* **Operating systems** (OS): Windows, Linux, macOS, etc.
* **Device drivers:**software that enables the communication between hardware and OS.
* **Firmware**: pre-installed low-level software that controls a device’s basic functions.
* **Utility software:** tools for system maintenance and optimization.
* **Boot loaders**: software that initializes the OS during startup.

## ****Why use  System Software?****

Here are some reasons why system software is necessary:

1.**Hardware Communication:** System software serves as an interface between the

hardware and software components of a computer, enabling them to communicate

and work together.

1. **Resource Management:** System software manages computer resources such as memory, CPU usage, and storage, optimizing their utilization and ensuring that the system operates efficiently.
2. **Security:**System software provides security measures such as firewalls, antivirus software, and encryption, protecting the system and its data from malware, viruses, and other security threats.
3. **User Interface:** System software provides a user interface that allows users to interact with the computer or computing device and perform various tasks.
4. **Application Support:** System software supports the installation and running of applications and software on the system.
5. **Customization:** System software allows for customization of the system settings and configuration, giving users greater control over their computing environment.

# Application Software:-

Application software is a type of computer program that performs a specific personal, educational, and business function. Each application is designed to assist end-users in accomplishing a variety of tasks, which may be related to productivity, creativity, or communication.

## ****Functions of Application Software:\_****

Application software programs are created to help with a wide range of tasks. Here are a few examples:

* Information and data management
* Management of documents (document exchange systems)
* Development of visuals and video
* Emails, text messaging, audio, and video conferencing, and cooperation are all options.
* Management of accounting, finance, and payroll
* Management of resources (ERP and CRM systems)
* Management of a project
* Management of business processes
* Software for education (LMS and e-learning systems)
* Software for healthcare applications

DAY 02: - (16/07/2024)

LAYERED ARCHIECTURE

CLIENT SERVER ARCHIECTURE

MICRO SERVICE ARCHIECTURE

EVENT DRIVEN ARCHIECTURE

SERVICE ORIENTED ARCHIECTURE

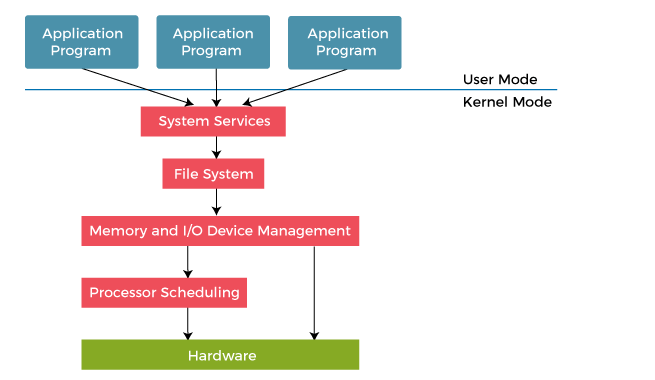
Layered Structure of Operating System:-

The operating system can be implemented with the help of various structures. The structure of the OS depends mainly on how the various common components of the operating system are interconnected and melded into the kernel. Depending on this, we have to follow the structures of the operating system.

The layered structure approach breaks up the operating system into different layers and retains much more control on the system. The bottom layer (layer 0) is the hardware, and the topmost layer (layer N) is the user interface. These layers are so designed that each layer uses the functions of the lower-level layers only. It simplifies the debugging process as if lower-level layers are debugged, and an error occurs during debugging. The error must be on that layer only as the lower-level layers have already been debugged.

Each of the layers in the operating system can only interact with the above and below layers. The lowest layer handles the hardware, and the uppermost layer deals with the user applications.

And the user layer wants to interact with the hardware layer, the response will be traveled through all the layers from n-1 to 1. Each layer must be designed and implemented such that it will need only the services provided by the layers .

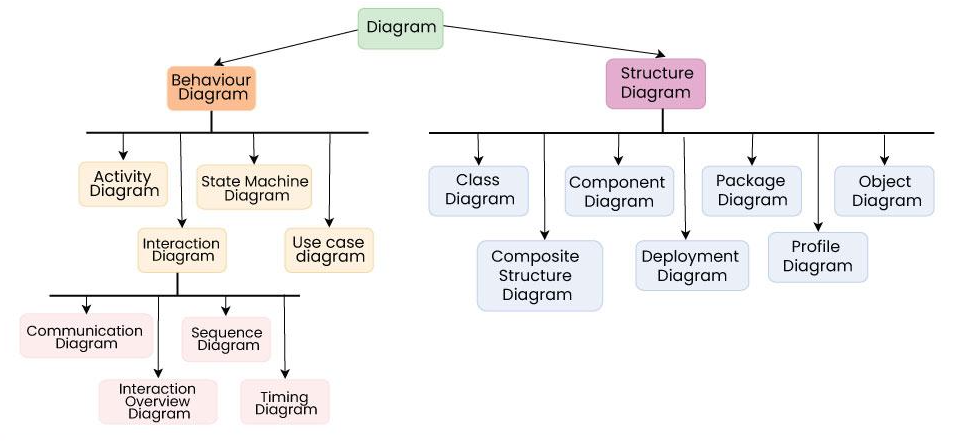


1. **Hardware:** This layer interacts with the system hardware and coordinates with all the peripheral devices used, such as a printer, mouse, keyboard, scanner, etc. These types of hardware devices are managed in the hardware layer.  
   The hardware layer is the lowest and most authoritative layer in the layered operating system architecture. It is attached directly to the core of the system.
2. **CPU Scheduling:** This layer deals with scheduling the processes for the CPU. Many scheduling queues are used to handle processes. When the processes enter the system, they are put into the job queue.
3. **Memory Management:** Memory management deals with memory and moving processes from disk to primary memory for execution and back again. This is handled by the third layer of the operating system. All memory management is associated with this layer. There are various types of memories in the computer like RAM, ROM.
4. **Process Management:** This layer is responsible for managing the processes, i.e., assigning the processor to a process and deciding how many processes will stay in the waiting schedule. The priority of the processes is also managed in this layer. The different algorithms used for process scheduling are FCFS (first come, first served), SJF (shortest job first), priority scheduling, round-robin scheduling, etc.
5. **I/O Buffer:** I/O devices are very important in computer systems. They provide users to interact with the system. This layer handles the buffers for the I/O devices and makes sure that they work correctly.  
   Suppose you are typing from the keyboard. There is a keyboard buffer attached with the keyboard, which stores data for a temporary time. Similarly, all input/output devices have some buffer attached to them. This is because the input/output devices have slow processing or storing speed. The computer uses buffers to maintain the good timing speed of the processor and input/output devices.
6. **User Programs:** This is the highest layer in the layered operating system. This layer deals with the many user programs and applications that run in an operating system, such as word processors, games, browsers, etc. You can also call this an application layer because it is concerned with application programs.

**SERVICE ORIENTED ARCHITECTURE {SOA} : -**

# What is a UML diagram?

UML, or Unified Modeling Language, is a way to visually represent the structure and behavior of systems. It uses diagrams to show how different parts of a system work together. Think of it like a blueprint that helps software developers plan, understand, and communicate about complex systems before they are built.



# TYPES OF UML DIAGRAMS:-

### Structural Diagrams:

1. **Class Diagram**:
   * Represents the static structure of a system.
   * Shows classes, attributes, operations, and relationships between classes.
   * Used for designing and understanding the domain model.
2. **Object Diagram**:
   * Shows a snapshot of objects and their relationships at a particular time.
   * Helps visualize instances of classes and their relationships.
3. **Component Diagram**:
   * Depicts the components (modules, libraries, executables) of a system and their dependencies.
   * Shows how components are wired together to form larger components or systems.
4. **Deployment Diagram**:
   * Illustrates the physical deployment of software components on hardware nodes.
   * Shows the relationships between software and hardware elements in a distributed system.

### Behavioral Diagrams:

1. **Use Case Diagram**:
   * Represents the functional requirements of a system from the user's perspective.
   * Shows actors (users, external systems) and their interactions with the system.
2. **Sequence Diagram**:
   * Shows how objects interact in a particular scenario over time.
   * Illustrates the sequence of messages exchanged between objects to accomplish a task.
3. **Activity Diagram**:
   * Represents workflows or business processes using activities (actions, decisions, and flows).
   * Shows the flow of control from one activity to another within a system.
4. **State Machine Diagram**:
   * Models the behavior of a system or object in response to events over time.
   * Shows states, events, transitions between states, and actions performed in response to events.

### Interaction Diagrams:

1. **Sequence Diagram** (again):
   * Details interactions between objects in terms of messages sent and received over time.
2. **Communication Diagram**:
   * Similar to sequence diagrams but focuses more on the relationships between objects and their interactions.

### Other Diagrams:

1. **Package Diagram**:
   * Organizes elements (such as classes and components) into groups to illustrate the dependencies between packages.
2. **Profile Diagram**:
   * Extends UML to define custom stereotypes, tagged values, and constraints for modeling specific domains or architectures.
3. **Composite Structure Diagram**:
   * Describes the internal structure of a classifier (class, component, or use case) and how its parts collaborate to achieve functionality.

# When and Why we use UML !!??

UML is used to improve the understanding, communication, and development process of complex software systems. It provides a standardized approach to designing and documenting software systems, making it an essential tool in modern software engineering practices.

1. Top of Form
2. Bottom of Form

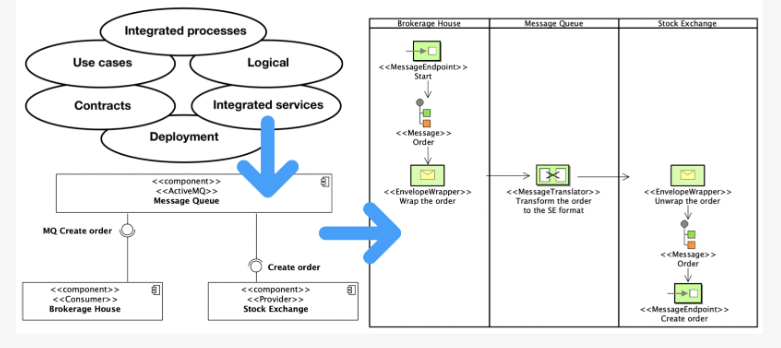
Case Study of SOA: -

# Example :- Online Banking System

An online banking system is being developed to provide banking services to customers over the internet. The system aims to integrate various banking functionalities into a cohesive platform using SOA principles.

Online banking typically involves web applications that customers access through their web browsers or mobile apps. These web applications provide a user-friendly interface for customers to perform various banking activities such as checking account balances, transferring funds, paying bills, applying for loans, and more.

In the context of Service-Oriented Architecture (SOA), these web applications interact with backend services that are implemented as independent modules or components. Each service within the SOA framework encapsulates specific business logic (e.g., account management, transaction processing) and exposes well-defined interfaces that the web application can interact with.



DAY 03:-(17/07/2024)

# PPT PRESENTATION :-

# SERVICE ORIENTED ARCHIECTURE

# CASE STUDY :- ONLINE BANKING .

DAY 04:-(18/07/2024)

# WHAT IS DESIGN PATTERN ?

## The design pattern is communicating objects and classes that are customized to solve a general design problem in a particular context.

# ****TYPES OF DESIGN PATTERNS :--****

# CREATIONAL PATTERN

# STRUCTURAL PATTERRN

# BEHAVIORAL PATTERN

# Creational pattern :-

Creational patterns help developers manage how new objects are created. They focus on:

* Controlling the object creation process.
* Ensuring objects are created efficiently and in a way that suits the application's needs.

Examples:-

* Singleton
* Factory based
* Abstract factory
* Builder
* Prototype

# Structural Patterns:-

Structural patterns deal with how objects and classes are composed to form larger structures. They help in defining relationships between objects to simplify design and ensure flexibility.

Examples:-

* Adaptor
* Composite
* Decorator
* Proxy

# Behavioral Patterns:-

Behavioral patterns focus on communication between objects, how they interact, and how they distribute responsibilities. They help in defining algorithms and the assignment of responsibilities between objects.

Examples:-

* Command
* Interpreter
* Iterator

### Benefits of Using Design Patterns:

* **Reusability:** Design patterns promote reuse of proven solutions, reducing redundancy and promoting consistency in software design.
* **Scalability:** They help in managing complex designs and evolving software systems by providing structured solutions to common problems.
* **Maintainability:** Design patterns improve code readability and maintainability by providing a common language and structure for developers.

# Presentation patterns :-

Presentation patterns are design patterns that focus on how the user interface (UI) components of an application are structured, managed, and interact with each other. These patterns help in organizing the presentation layer of an application, ensuring clarity, maintainability, and separation of concerns between different parts of the UI. Here are some common presentation patterns

# **Model-View-Controller (MVC).**

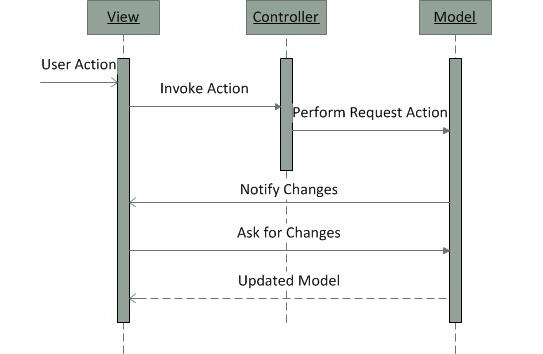
## **Model-View-Presenter (MVP).**

## 3. **Model-View-View Model (MVVM).**

### 1. ****Model-View-Controller (MVC):-****

**Purpose:** MVC separates an application into three main components: Model, View, and Controller.

* **Model:** Represents the data and business logic of the application. It manages the data and notifies observers (usually views) of any changes.
* **View:** Represents the UI components that display the data to the user. Views observe the model for changes and update themselves accordingly.
* **Controller:** Acts as an intermediary between the Model and View. It handles user input and updates the model or view as necessary. In web applications, the controller receives requests, processes them, and returns responses.

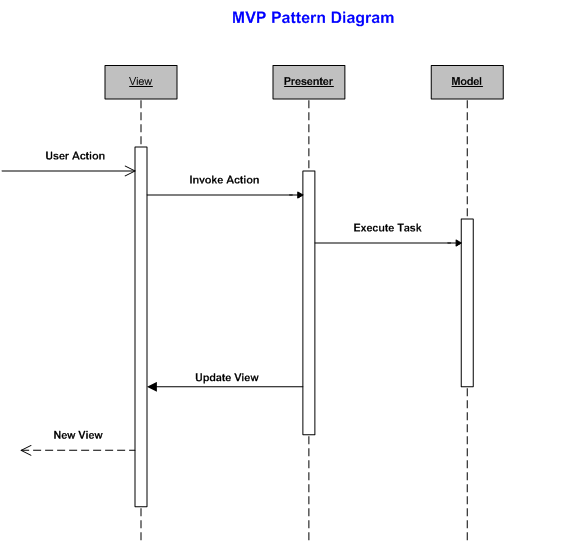


**Use Case:** Widely used in web and desktop applications to separate concerns between data (Model), presentation (View), and user input (Controller).

### 2. ****Model-View-Presenter (MVP):-****

**Purpose:** MVP is a derivative of MVC, where the presenter acts as an intermediary between the View and Model.

* **Model:** Same as in MVC, representing data and business logic.
* **View:** Responsible for displaying data to the user and sending user input to the Presenter.
* **Presenter:** Contains the presentation logic and interacts with both the View and Model. It updates the View based on changes in the Model and processes user input from the View.

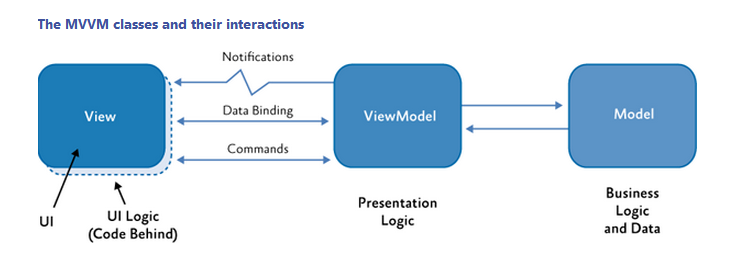


**Use Case:** Commonly used in GUI-based applications where the View is passive and controlled by the Presenter.

### 3. ****Model-View-View Model (MVVM):-****

**Purpose:** MVVM is a pattern that enhances separation of concerns by introducing a View Model layer between the View and Model.

* **Model:** Represents data and business logic.
* **View:** UI components that display data to the user and react to user input.
* **View Model:** Acts as an adapter between the View and Model. It exposes data and commands from the Model to the View in a way that is easy to bind to UI controls. It also encapsulates presentation logic that does not belong in the Model or View.



**Use Case:** Popular in modern UI frameworks and technologies, such as WPF (Windows Presentation Foundation) and Angular, where data binding and separation of concerns are emphasized.

### Benefits of Presentation Patterns:

* **Separation of Concerns:** Clear separation between data (Model), UI presentation (View), and user interaction logic (Controller/Presenter/View Model).
* **Maintainability:** Easier to maintain and update UI components without affecting other parts of the application.
* **Testability:** Facilitates unit testing of individual components (Model, View, Presenter /Controller/View Model) independently.

# CLOUD COMPUTING:-

Cloud computing refers to the delivery of computing services over the internet ("the cloud") to offer faster innovation, flexible resources, and economies of scale. These services typically include computing power, storage, databases, networking, software, and more, all delivered as a service rather than a product that needs to be managed locally.

### Types of Cloud Computing Services:-

Cloud computing services are broadly categorized into three main types based on the level of abstraction and control they offer to users:

1. **Infrastructure as a Service (IaaS):**
   * **Definition:** IaaS provides virtualized computing resources over the internet. It offers virtualized hardware resources such as virtual machines (VMs), storage, and networking capabilities.
   * **Use Cases:** Ideal for organizations that want to manage their own operating systems, applications, and data, while outsourcing the management of underlying infrastructure to the cloud provider.
   * **Examples:** Amazon Web Services (AWS) EC2, Microsoft Azure Virtual Machines, Google Compute Engine.
2. **Platform as a Service (PaaS):**
   * **Definition:** PaaS provides a platform and environment for developers to build, deploy, and manage applications without worrying about underlying infrastructure details.
   * **Use Cases:** Suitable for developers who want to focus on application development without managing the underlying infrastructure, including scaling, load balancing, and database management.
   * **Examples:** AWS Elastic Beanstalk, Microsoft Azure App Service, Google App Engine.
3. **Software as a Service (SaaS):**
   * **Definition:** SaaS delivers software applications over the internet on a subscription basis. Users can access these applications via a web browser without needing to install or maintain software locally.
   * **Use Cases:** Ideal for end-users who want to access software applications without the complexity of software installation, maintenance, and management.
   * **Examples:** Salesforce (CRM), Google Workspace (formerly G Suite), Microsoft Office 365, Dropbox.

### Benefits of Cloud Computing Services:-

* **Scalability:** Cloud services can scale up or down based on demand, allowing organizations to accommodate fluctuating workloads efficiently.
* **Cost Efficiency:** Pay-as-you-go pricing models and economies of scale reduce costs associated with hardware, maintenance, and upgrades.
* **Flexibility and Agility:** Rapid deployment of resources and services enables faster innovation and time-to-market for applications and solutions.
* **Reliability and Availability:** Cloud providers offer robust infrastructure and data redundancy to ensure high availability and reliability of services.

**Docker:**

Docker is a containerization platform that allows developers to package, ship, and run applications in containers. Containers are lightweight and portable, providing a consistent and reliable way to deploy applications across different environments.

TYPES:-

* **Images**: A Docker image is a lightweight, standalone, and executable package that includes everything an application needs to run, such as code, libraries, and dependencies.
* **Containers:** A Docker container is a runtime instance of an image, providing a isolated environment for the application to run in.
* **Volumes**: A Docker volume is a directory that is shared between the host machine and a container, allowing data to be persisted even after the container is deleted.
* **Ports**: Docker containers can expose ports to the host machine, allowing incoming requests to be routed to the container.

**Benefits**:-

* **Lightweight**: Containers are much lighter than virtual machines, making them faster to spin up and down.
* **Portable:** Docker containers are highly portable, allowing developers to deploy applications across different environments with minimal modifications.

**Docker Architecture**:-

# Docker ClienT.

# Docker Daemon.

# Docker Hub.

**Kubernetes:**

Kubernetes (also known as K8s) is an open-source container orchestration system for automating the deployment, scaling, and management of containerized applications. It was originally designed by Google, and is now maintained by the Cloud Native Computing Foundation (CNCF).

**TYPES:-**

* **Cluster**: A group of machines, called nodes, that run Kubernetes components and are used to deploy and manage applications.
* **Node:** A machine in the cluster that runs Kubernetes components and containers.
* **Pod**: The basic execution unit in Kubernetes, consisting of one or more containers.
* **ReplicaSet**: Ensures a specified number of replicas (identical pods) are running at any given time.
* **Deployment:** Manages the rollout of new versions of an application, including rolling updates and rollbacks.
* **Service**: Provides a network identity and load balancing for accessing applications.
* **Namespace**: A logical partitioning of resources, providing isolation and management of applications.

**Components:**

* API Server: The central management point for the cluster, providing a RESTful API for interacting with the cluster.
* Controller Manager: Runs and manages control plane components, such as the replication controller and deployment controller.
* Scheduler: Responsible for scheduling pods on nodes based on resource availability and constraints.
* etcd: A distributed key-value store used for storing cluster state and configuration.
* kubelet: An agent running on each node, responsible for managing pods and containers.

**Deployment**:

* Rolling Update: Gradually replaces old pods with new ones, ensuring minimal downtime.
* Recreate: Deletes old pods and creates new ones, suitable for stateless applications.
* Blue-Green: Deploys a new version of an application alongside the old one, allowing for quick rollbacks.