K. J. Somaiya College of Engineering, Mumbai-77

Batch: A1	Roll No.: 16010121045
Experiment / assignment / tutorial No	

TITLE: Architecture of any operating system, draw diagram using ADL

AIM: Study of architecture of operating system .

Expected OUTCOME of Experiment:

Using ADL represent operating system architecture

CO 1. Design the architecture of software systems in various architectural styles.

Books/ Journals/ Websites referred:

- https://www.xcubelabs.com/blog/software-architecture-understanding-styles-and-patterns-for-effective-system-design/#:~:text=Architectural Styles are high-level strategies that provide an,or homes. Examples include Layered Event-Driven and Microservices. Last visited on 24 July 2024
-

Background Theory:

A software system's high-level structure, or software architecture, is made up of the rules, patterns, and guidelines that determine how the system is organized, interacts, and is related to its components.

The design decisions will help in providing:

Modularity: allowing the system into interchangeable components that can be developed, tested, and maintained independently.

Encapsulation: Hiding the details of the components, exposing only the necessary information, thus reducing the complexity of the system.

Security: Incorporating the measures to protect the system against the unauthorized access.

Documentation: Provides clear documentation of the system architecture, thus facilitating communication and better understanding of the system.

Performance: Ensuring that the system meets the required performance metrics such as resource utilization, throughput, etc.

There are various types of Software Architectural Styles.

- Data-centered architecture
- Data-flow architecture
- Call and return architectures
- Object-oriented architectures
- Layered architectures

Different architectural patterns will be used for implementing software systems based on:

1. System Requirements

Functional Requirements: What the system needs to do, the features and capabilities it must provide.

Non-Functional Requirements: Attributes such as performance, scalability, security, maintainability, and usability.

2. Design Principles

Separation of Concerns: Dividing the system into distinct sections, each addressing a separate concern.

Modularity: Designing the system in a way that components can be developed, tested, and understood in isolation.

Abstraction: Hiding the complexity of implementation details, providing a simplified interface.

Encapsulation: Bundling data and methods that operate on the data within one unit, restricting access to some of the object's components.

3. Scalability and Performance

The ability to handle increased load, either by scaling up (adding more resources) or scaling out (adding more nodes).

Patterns like microservices and event-driven architecture are designed to improve scalability and performance.

4. Maintainability and Flexibility

Ease of modifying the system to fix bugs, add features, or adapt to changing requirements.

Patterns like layered architecture and component-based architecture promote maintainability and flexibility.

5. Security

Protecting the system from unauthorized access and ensuring data integrity and confidentiality.

Patterns like service-oriented architecture (SOA) and microservices can help isolate and secure individual components.

6. Development Team Skills and Experience

The expertise of the development team with certain technologies and architectural patterns.

Patterns should align with the team's capabilities to ensure effective implementation and maintenance.

7. Technology Stack

The programming languages, frameworks, and tools that will be used in the system.

Patterns should be chosen to leverage the strengths of the chosen technology stack.

8. Integration Requirements

How the system will interact with other systems, services, and databases.

Patterns like microservices and SOA facilitate integration with external systems and services.

9. Deployment Environment

Whether the system will be deployed on-premises, in the cloud, or in a hybrid environment.

Patterns like cloud-native architecture are optimized for cloud deployment.

10. Business Goals

Alignment with the strategic objectives of the organization, including time-to-market, cost constraints, and long-term vision.

Patterns should support achieving the business goals effectively.

11. User Experience

The expectations of end-users in terms of interface design and interaction.

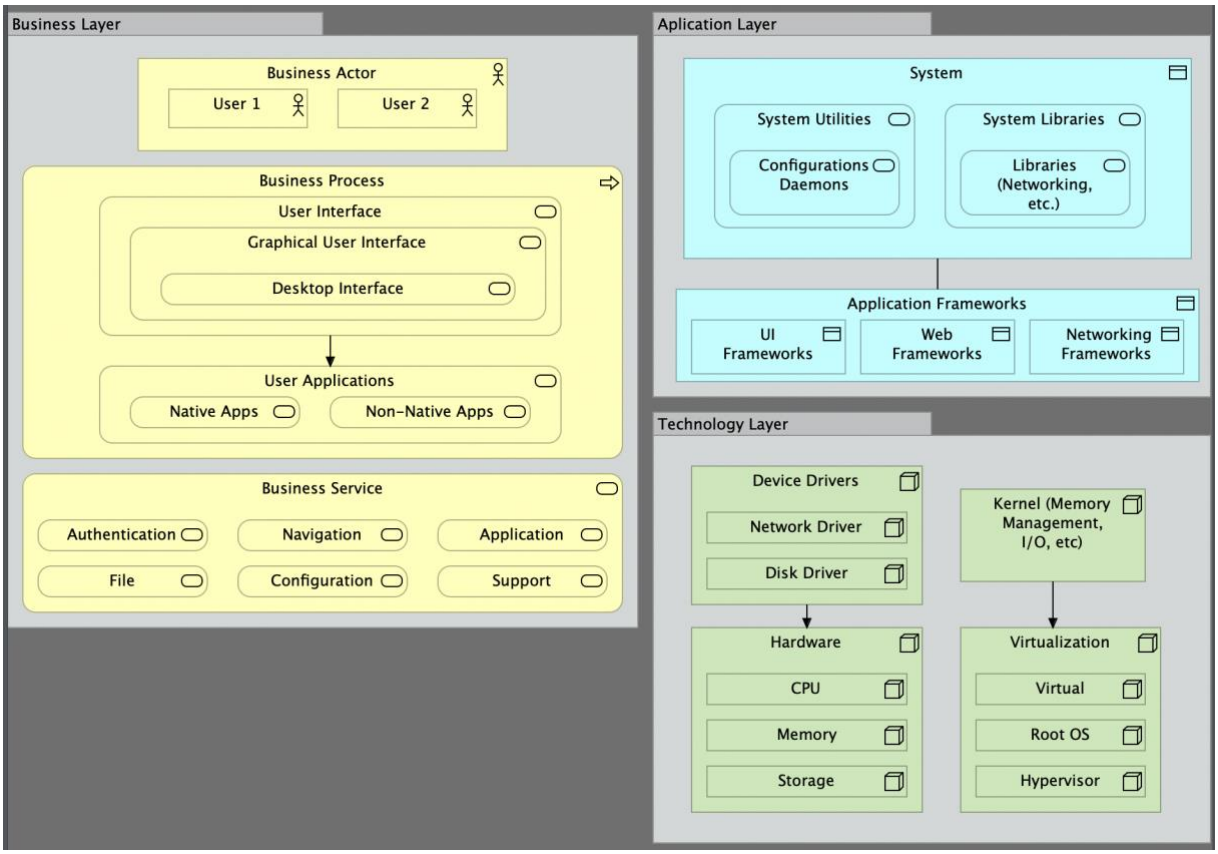
Patterns like model-view-controller (MVC) help create a better user experience by separating concerns.

Software architecture patterns offer reusable designs for various situations, to offer advantages such as improved efficiency, productivity, speed, cost optimization, and better planning.

Laboratory Work:

The previous experiment introduces ArchiMate a tool used for drawing Software architecture design. The students will be required to study architecture of any of the Operating System from the following list and draw diagram

- Ubuntu
- Windows
- macOS



Business Layer

1. **Business Actor**
    - **User 1**: Represents one type of user interacting with the system.
    - **User 2**: Represents another type of user.
2. **Business Process**
    - **User Interface**
        - **Graphical User Interface**
            - **Desktop Interface**: Part of the graphical user interface for desktop applications.
        - **User Applications**
            - **Native Apps**: Applications designed to run on Windows OS.
            - **Non-Native Apps**: Applications that are not originally designed for Windows but can run on it.
3. **Business Service**

- **Authentication**: Services related to user authentication.
- **Navigation**: Services that help users navigate through applications.
- **Application**: General application services.
- **File**: Services related to file management.
- **Configuration**: Services for system or application configuration.
- **Support**: Support services for users.

Application Layer

1. **System**
   - **System Utilities**: Utilities that assist in system management.
   - **System Libraries**
     - **Configurations Daemons**: Background processes for configuration management.
     - **Libraries (Networking, etc.)**: Libraries that provide networking and other functionalities.
2. **Application Frameworks**
   - **UI Frameworks**: Frameworks for building user interfaces.
   - **Web Frameworks**: Frameworks for developing web applications.
   - **Networking Frameworks**: Frameworks for networking functionalities.

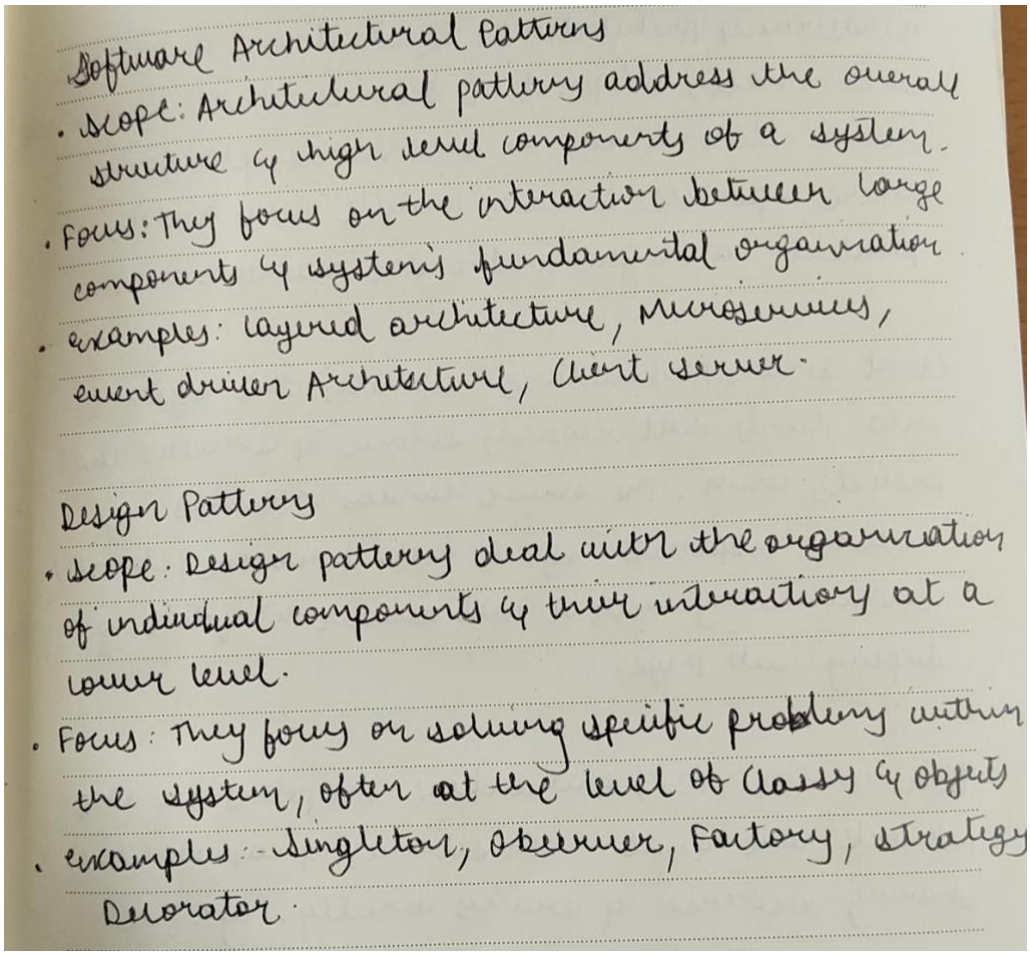
Technology Layer

1. **Device Drivers**
   - **Network Driver**: Drivers for network devices.
   - **Disk Driver**: Drivers for disk storage devices.
2. **Kernel (Memory Management, I/O, etc.)**: Core component of the OS managing memory, I/O operations, and other fundamental tasks.
3. **Virtualization**
   - **Virtual**: Virtualization of resources.
   - **Root OS**: The primary operating system in a virtualized environment.
   - **Hypervisor**: Manages virtual machines.

4. **Hardware**
    - **CPU**: Central Processing Unit.
    - **Memory**: RAM and other memory components.
    - **Storage**: Storage devices such as hard drives and SSDs.

Post Laboratory questions:

1. **Compare Software Architectural patterns and Design Pattern**



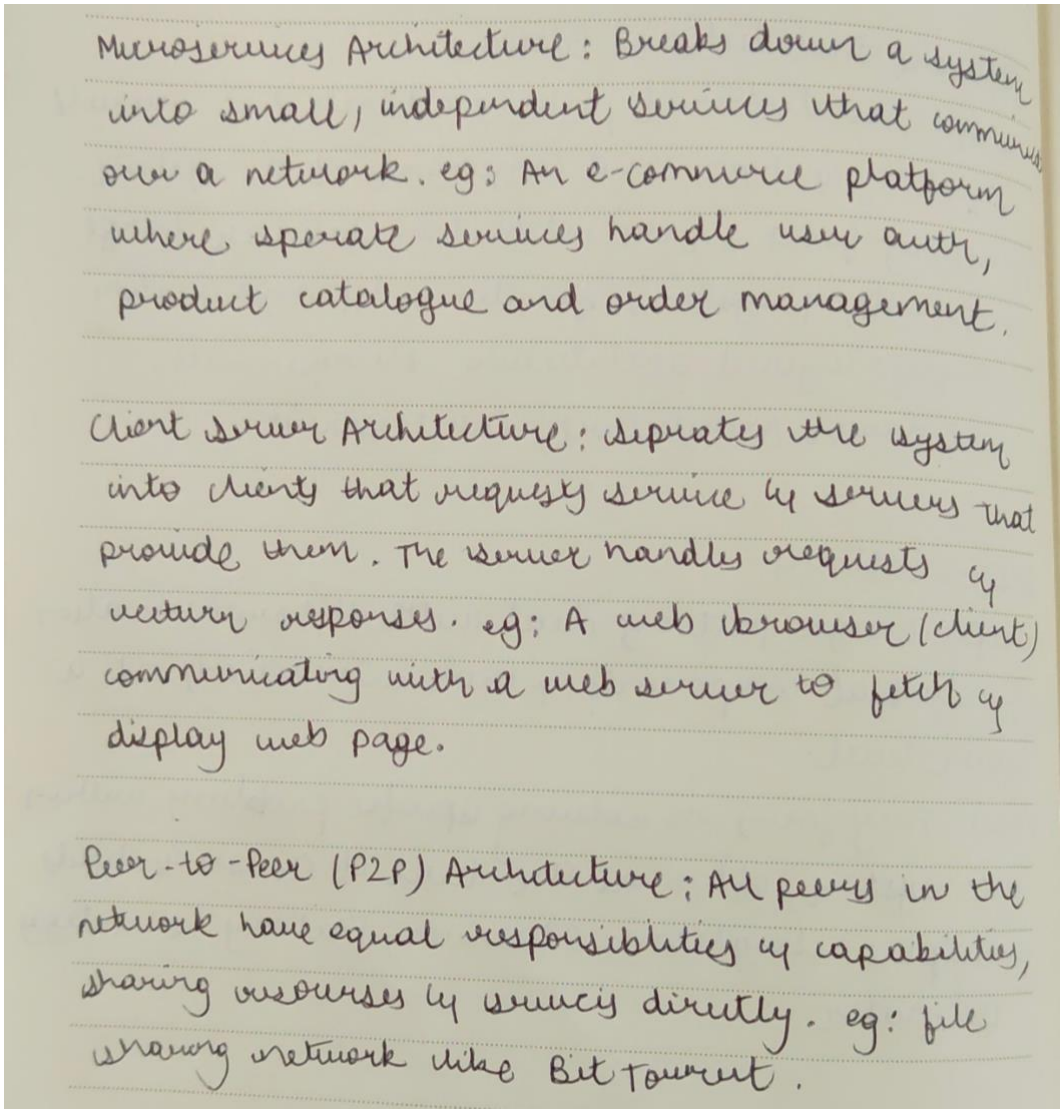
Software Architectural Patterns

- Scope: Architectural patterns address the overall structure & high level components of a system.
- Focus: They focus on the interaction between large components & system's fundamental organization.
- examples: Layered architecture, Microservices, event driven Architecture, Client server.

Design Patterns

- Scope: Design patterns deal with the organization of individual components & their interactions at a lower level.
- Focus: They focus on solving specific problems within the system, often at the level of classes & objects.
- examples: - Singleton, Observer, Factory, Strategy Decorator.

2. **Explain Various type of Software Architecture Patterns giving example of each**

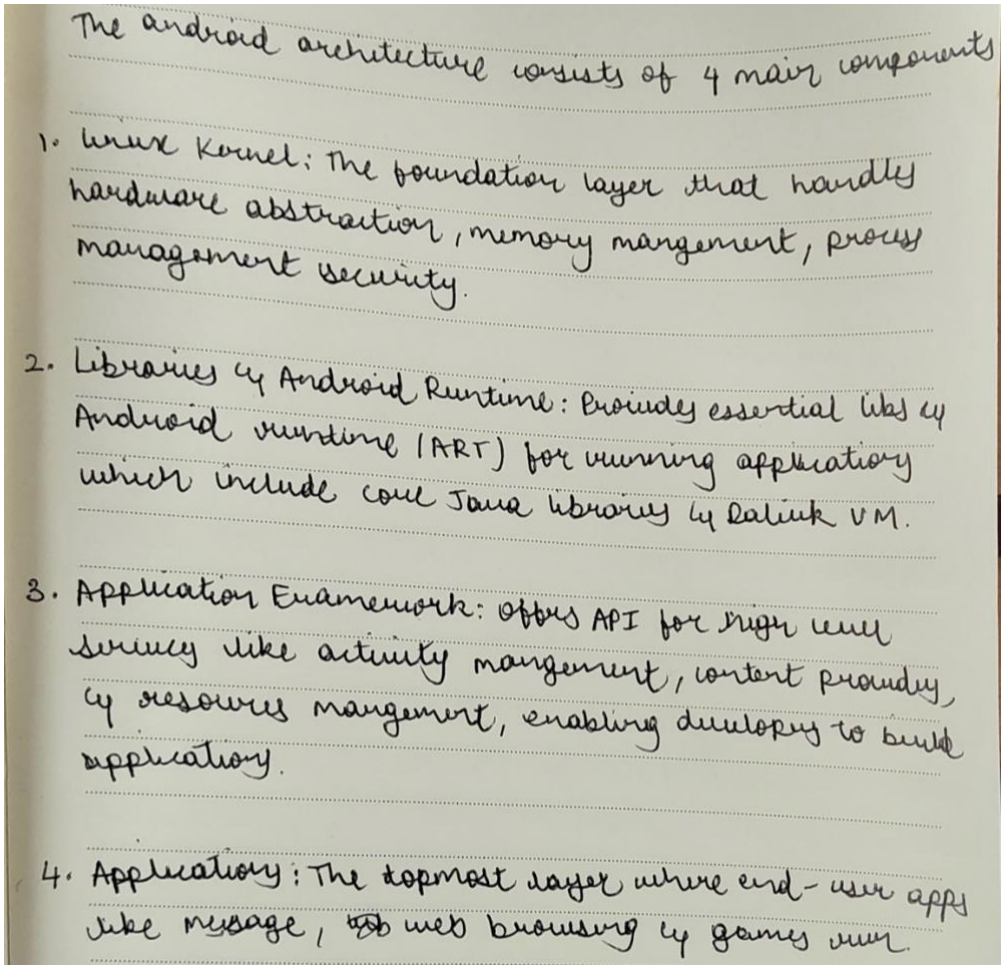


Microservices Architecture: Breaks down a system into small, independent services that communicate over a network. eg: An e-commerce platform where sperate services handle user auth, product catalogue and order management.

Client Server Architecture: Sepreates the system into clients that requests service by servers that provide them. The server handles requests & return responses. eg: A web browser (client) communicating with a web server to fetch & display web page.

Peer-to-Peer (P2P) Architecture: All peers in the network have equal responsibilities & capabilities, sharing resources & services directly. eg: file sharing network like Bit Torrent.

3. Explain in brief the architecture of Android Operating system.



The android architecture consists of 4 main components

1. Linux Kernel: The foundation layer that handles hardware abstraction, memory mangement, process management security.

2. Libraries & Android Runtime: Provides essential libs & Android runtime (ART) for running applications which include core Java libraries & Dalvik VM.

3. Application Framework: Offers API for high level services like activity mangement, content provider, & resources mangement, enabling developers to build applications.

4. Applications: The topmost layer where end-user apps like message, web browsing & games run.

References:

https://www.dragon1.com/downloads/microsoft%20architecture%20overview.pdf

https://www.slideshare.net/Stacksol/windows-architecture-explained-by-stacksol