**Activite-01**

**Software Architecutre**

**What Is Software Architecture?**

A big part of our daily lives, from using a smartphone to sending an email, depends heavily on the software architecture of systems we use. Without it, so much of what we use and what we know would not be possible at all. Software architecture is what brings innovation to an organization.

**Definition**

The definition of software architecture has long been argued. For some people, it’s the way essential compounds are wired together or the fundamental organization of a system. In this regard, the abstract concept proposed by Ralph Johnson, Associate Professor, University of Illinois, is something noteworthy: **“Architecture is about the important stuff. Whatever that is.”**

**Why Does Software Architecture Matter?**

Software architecture is designed for a specified reason — it’s all about identifying the components that will be directly responsible for the success or failure of our system and creating the system to serve and protect those essential components. An organized system architecture design helps in maintaining the internal quality that further improves the software.

In **Netflix**, it’s their **microservice architecture** that empowers them to manage availability, whereas, in **Salesforce**or **Google**, it’s a domain-driven design that helps them run **domain logic complexity**.

**two scenarios are there.**

1.Launch — 31st May: The code is messy and tangled. The users have nothing to do with it but tracing the scope of change and incorporating it has become tricky.

2.Launch — 30th June: The code is perfectly organized. Users have nothing to do with it, but the software development team can easily handle and implement the changes.

**Software Architecture Patterns**

Think about a software development project for designing an online shopping app. The foremost important thing you need is to define its programming architecture and design to build this app. These are the foundation pillars upon which your app is going to be built. For example, how will the algorithm function for product suggestions? How will the shopping cart work? The list continues.

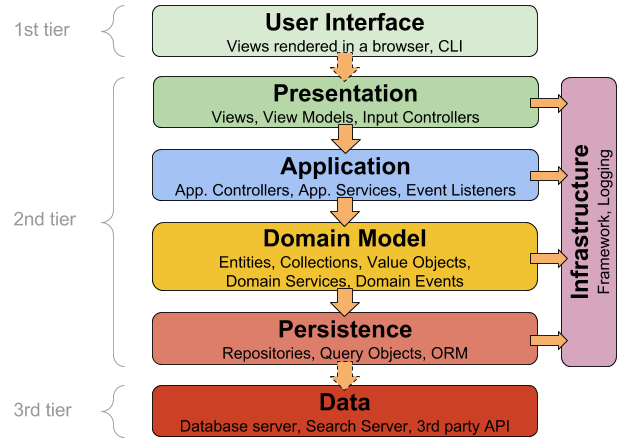
5 most common software architecture patterns.

**1. Layered Architecture Pattern**

This pattern is one of the widely used architectural patterns in the software industry for its easy-to-develop-and-maintain feature. As the name suggests, it follows a tiered approach wherein the code is organized in layers. This pattern is used to structure programs that can be broken down into groups of subtasks, each at a specified level of abstraction. Each layer here provides services to the higher layer.

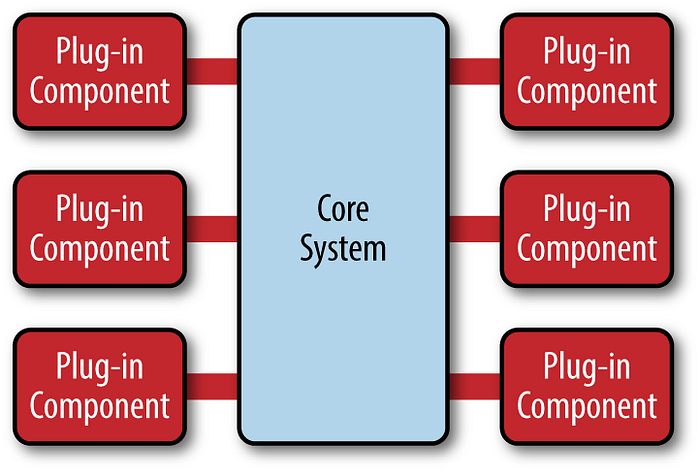
In a general information system, these 4 are the most used layers:

* Presentation Layer or UI Layer
* Application Layer or Service Layer
* Business Logic Layer or Domain Layer
* Data Access Layer or Persistence Layer



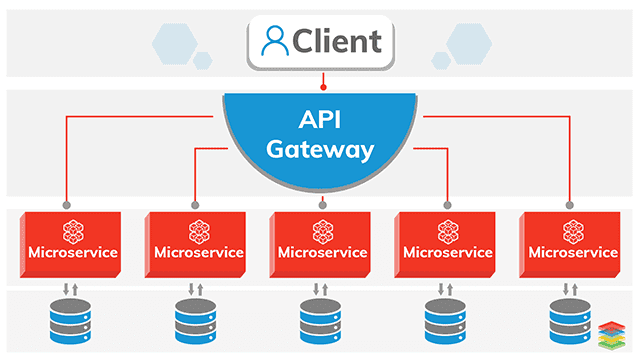
## ****2. Microkernel Architecture Pattern****

This pattern is best suited for apps that need to be adaptive and flexible enough to frequently changing system requirements. It’s separated into an extended functionality (plug-ins) and a minimal functional core. The core system comprises standard business logic without any custom code for complex conditional processes or exceptional cases.



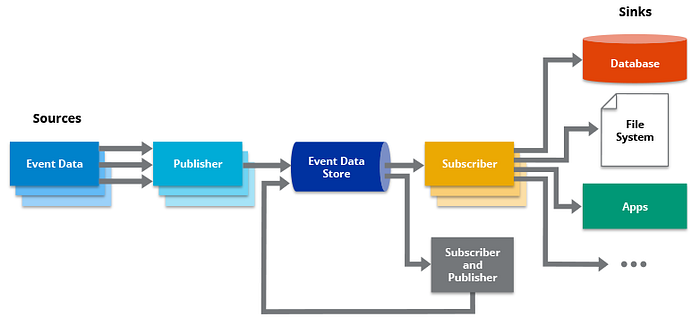
## 3. Microservices Architecture Pattern

This pattern approaches the building of multiple small and independent apps that work together under an entire system. Each app or microservice has its own responsibility, and the only dependence between them is to communicate.



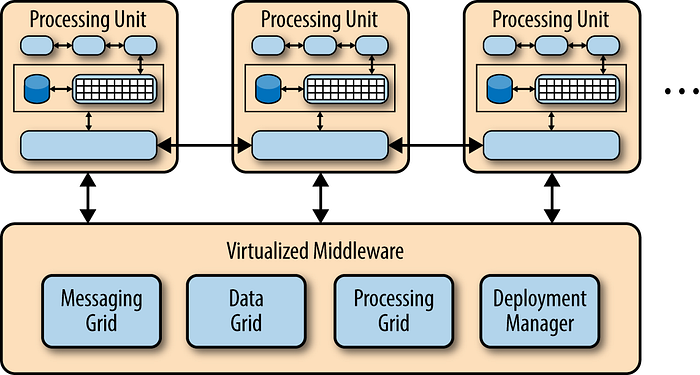
**4. Event-based Architecture Pattern**

This is the most commonly allocated asynchronous architecture pattern for developing highly scalable systems. Its approach is based on data that defines ‘events’, such as moving the scroll bar, clicking a button, etc., and processes them asynchronously



**5. Space-based Architecture Pattern**

The space-based pattern has particularly been designed for addressing and solving concurrency and scalability issues. It also proves to be useful for apps that have unpredictable and variant concurrent user volumes. Here, high scalability is accomplished by eliminating the central database constraint and utilizing copied in-memory data grids.



**What Makes A Good Software Architecture?**

An efficient programming architecture has a set of quality attributes as mentioned below:

1. **Functionality:** Refers to the extent the software performs against its needed purposes.
2. **Usability:**Refers to the level the software can be used with ease and convenience.
3. **Reliability:**Refers to the capability of the product to provide intended functionality under given circumstances.
4. **Supportability:**Refers to the facility with which developers can transfer the software from one platform to another with minimal or no changes.
5. **Performance:**Refers to the approximation by considering resource utilization, processing speed, response time, productivity, and throughput.
6. **Self-Reliance:**refers to the capability of independent activities for optimal performance even if one is going through a downtime.

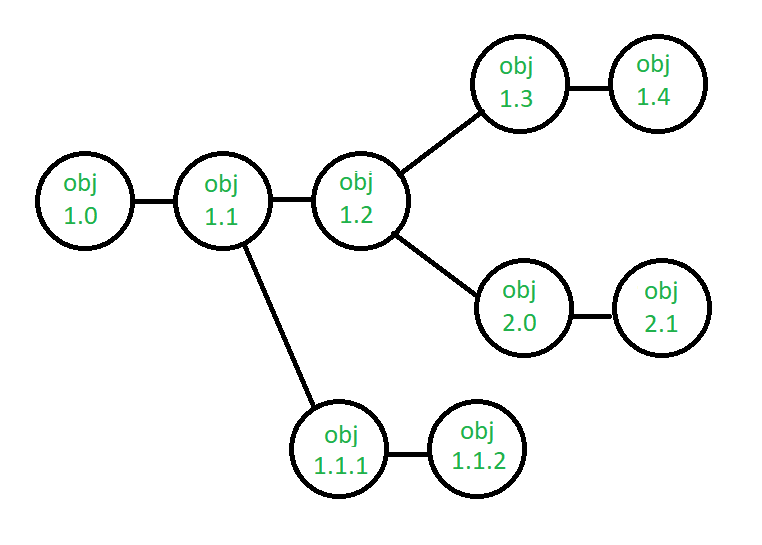
**Activite-02**

**System Configuration Management**

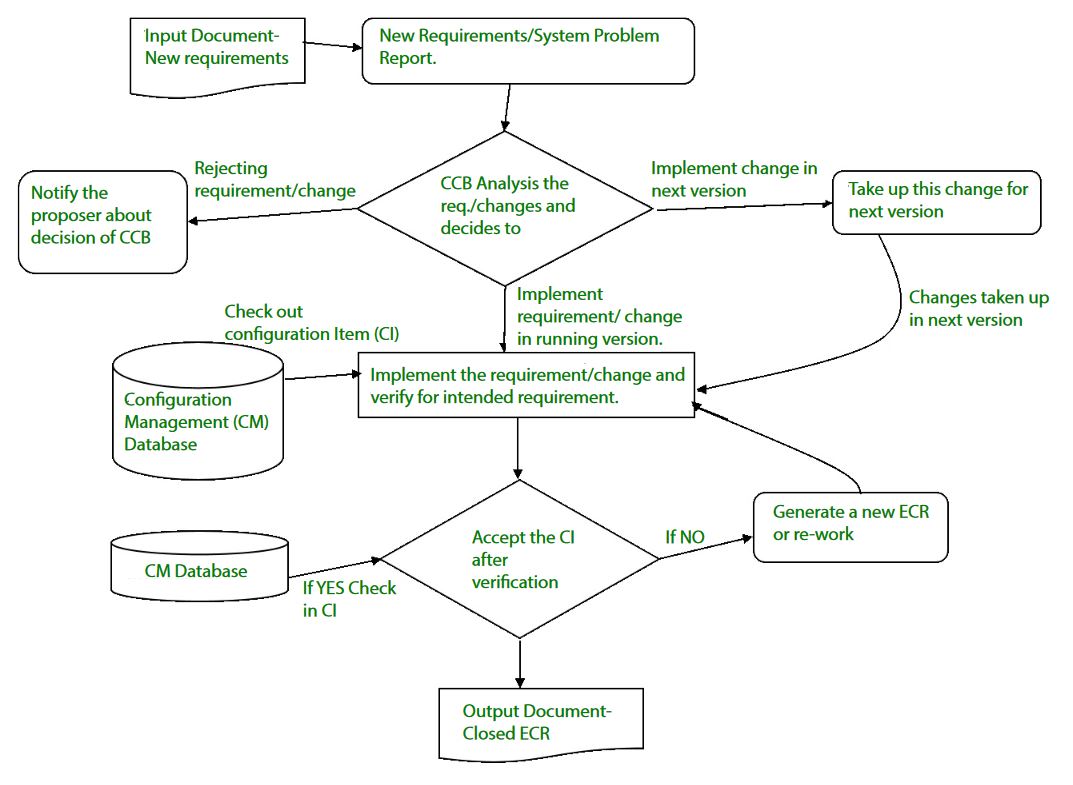
**System Configuration Management (SCM**) is an arrangement of exercises that controls change by recognizing the items for change, setting up connections between those things, making/characterizing instruments for overseeing diverse variants, controlling the changes being executed in the current framework, inspecting and revealing/reporting on the changes made. It is essential to control the changes because if the changes are not checked legitimately then they may wind up undermining a well-run programming. In this way, SCM is a fundamental piece of all project management activities.

**Processes involved in SCM –** Configuration management provides a disciplined environment for smooth control of work products. It involves the following activities:

1. **Identification and Establishment –** Identifying the configuration items from products that compose baselines at given points in time (a baseline is a set of mutually consistent Configuration Items, which has been formally reviewed and agreed upon, and serves as the basis of further development). Establishing relationships among items, creating a mechanism to manage multiple levels of control and procedure for the change management system.
2. **Version control –** Creating versions/specifications of the existing product to build new products with the help of the SCM system. A description of the version is given below:



**3.Change control –** Controlling changes to Configuration items (CI). The change control process is explained in Figure below:



A change request (CR) is submitted and evaluated to assess technical merit, potential side effects, the overall impact on other configuration objects and system functions, and the projected cost of the change. The results of the evaluation are presented as a change report, which is used by a change control board (CCB) —a person or group who makes a final decision on the status and priority of the change.

1. **Configuration auditing –** A software configuration audit complements the formal technical review of the process and product. It focuses on the technical correctness of the configuration object that has been modified. The audit confirms the completeness, correctness, and consistency of items in the SCM system and tracks action items from the audit to closure.
2. **Reporting –** Providing accurate status and current configuration data to developers, testers, end users, customers, and stakeholders through admin guides, user guides, FAQs, Release notes, Memos, Installation Guide, Configuration guides, etc.

System Configuration Management (SCM) is a software engineering practice that focuses on managing the configuration of software systems and ensuring that software components are properly controlled, tracked, and stored. It is a critical aspect of software development, as it helps to ensure that changes made to a software system are properly coordinated and that the system is always in a known and stable state.

## Importance of Software Configuration Management

1. Effective Bug Tracking: Linking code modifications to issues that have been reported, makes bug tracking more effective.
2. Continuous Deployment and Integration: SCM combines with continuous processes to automate deployment and testing, resulting in more dependable and timely software delivery.
3. Risk management: SCM lowers the chance of introducing critical flaws by assisting in the early detection and correction of problems.
4. Support for Big Projects: Source Code Control (SCM) offers an orderly method to handle code modifications for big projects, fostering a well-organized development process.
5. Reproducibility: By recording precise versions of code, libraries, and dependencies, source code versioning (SCM) makes builds repeatable.
6. Parallel Development: SCM facilitates parallel development by enabling several developers to collaborate on various branches at once.

## The main advantages of SCM

1. Improved productivity and efficiency by reducing the time and effort required to manage software changes.
2. Reduced risk of errors and defects by ensuring that all changes were properly tested and validated.
3. Increased collaboration and communication among team members by providing a central repository for software artifacts.
4. Improved quality and stability of software systems by ensuring that all changes are properly controlled and managed.

## The main disadvantages of SCM

1. Increased complexity and overhead, particularly in large software systems.
2. Difficulty in managing dependencies and ensuring that all changes are properly integrated.
3. Potential for conflicts and delays, particularly in large development teams with multiple contributors.