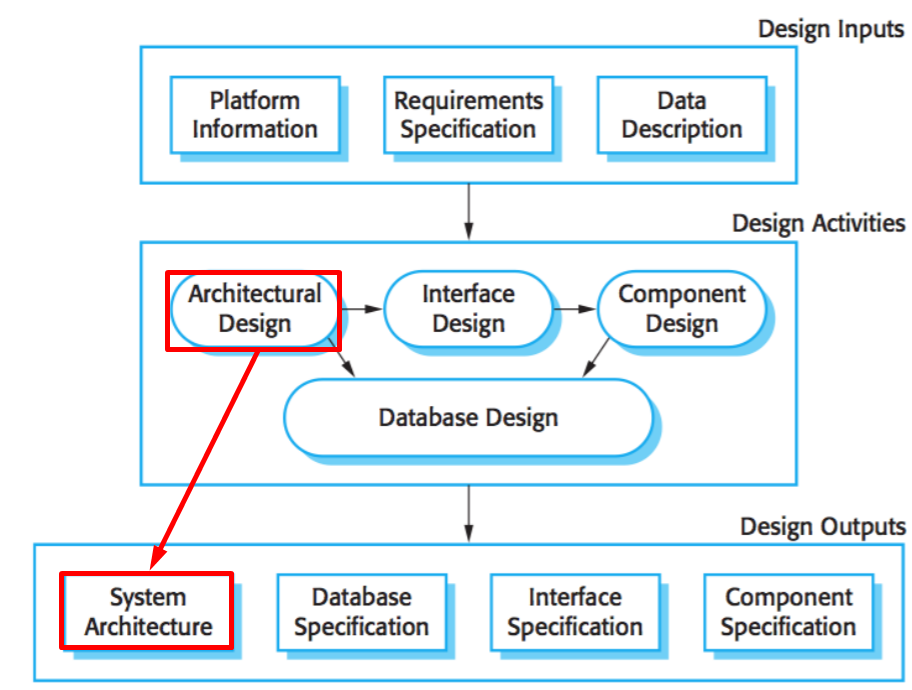
# Software Architecture

This refers to fundamental structures of a software system and the discipline of creating it. Architecture is a metaphor, analogous to the architecture of a building – it also talks about how the system is built and what are its building blocks consisting of. Also, it denotes what technologies, techniques and approach is used to build the system.

**Architectural Design**

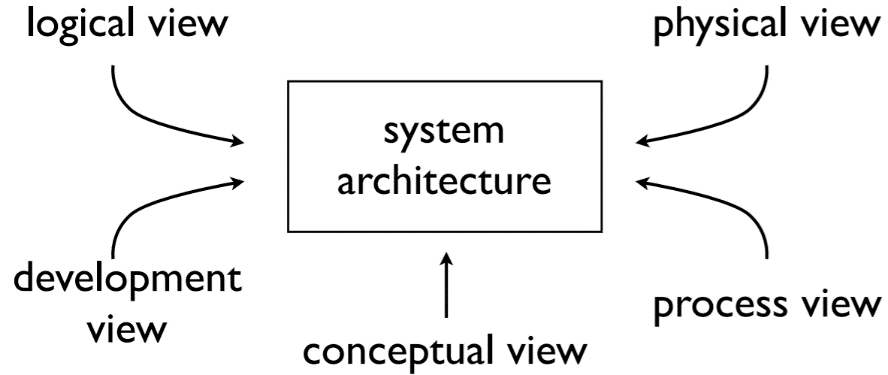
Understanding how system should be organized

Designing overall structure, principle components, relationships and distribution.

Architecture basically addresses **non-functional requirements**, while Components address **functional requirements**.

**Architectural Design** addresses structure and behavior.

It is like blueprints – looking at the system from particular perspective – you cannot represent all aspect of system in one diagram.

Different stakeholders need different views – displayed in a different diagram.

## 4+1 Architectural View Model

**Logical View** – concerned with the functionality the system provides to end users. UML diagrams are used to represent logical view – Class Diagram, state diagrams

**Process View** – process view deals with dynamic aspects of a system, explains the system processes and how they communicate. Also, it focuses on the run time behavior of the system. UML diagrams as Activity diagram are used to represent it visually.

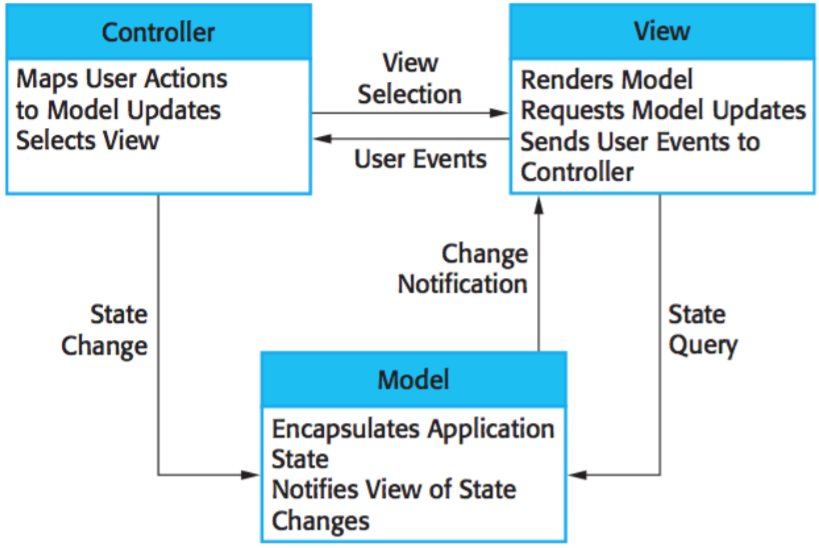
**Development View** – illustrates a system from a programmer’s implementation view. Component diagram to describe software components and Package diagram to describe the division of the domain.

**Physical View** – depicts the system from a system engineer’s point of view. It is concerned with topology of software components on the physical layer as well as the connections between them. It is also known as deployment view. Deployment diagram is used to visualize this.

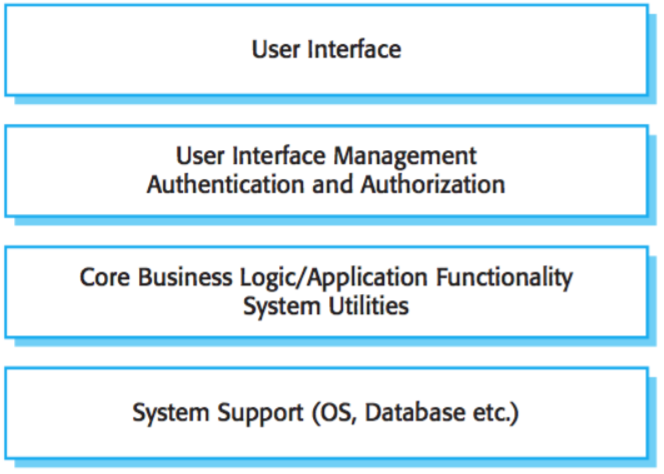
**Conceptual View (Scenarios)** – software here is broken down into series of interactions between software components and processes. Use Case view and descriptions are used to depict this view. These also serve as starting point for tests of architecture prototype.

## Architectural Patterns

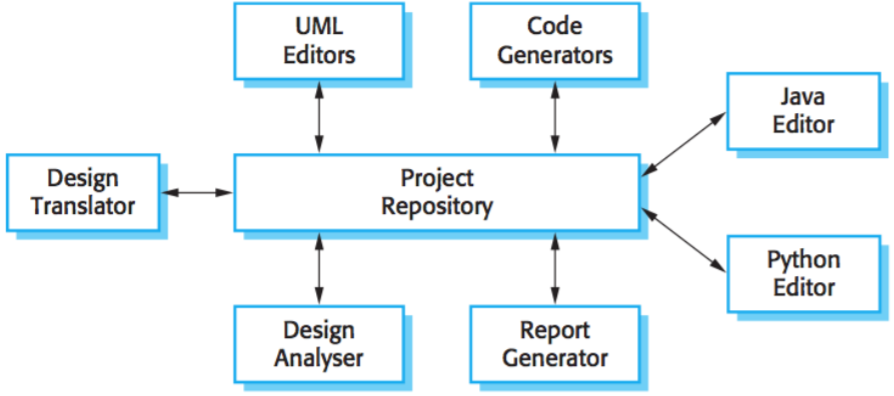
**Model-View-Controller (MVC)**

Pattern used for developing user interfaces which divides the program logic into three interconnected elements to separate internal representations of information from the ways it is presented to users. This allows for reuse of components and parallel development.

**N-Tier Architecture**

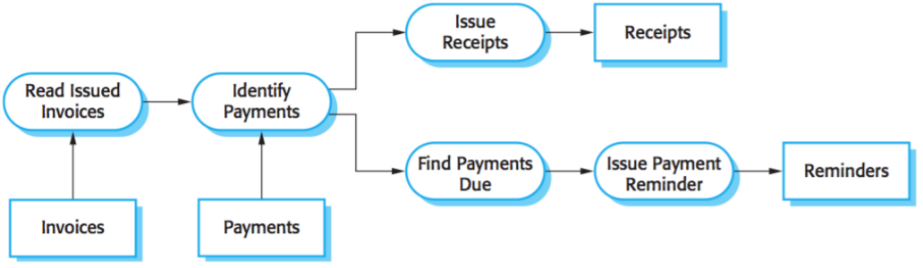
Most used is 3-tier architecture, where presentation, application processing and data management functions are physically separated, again, allowing for parallel development and reuse of software components.

**Repository**

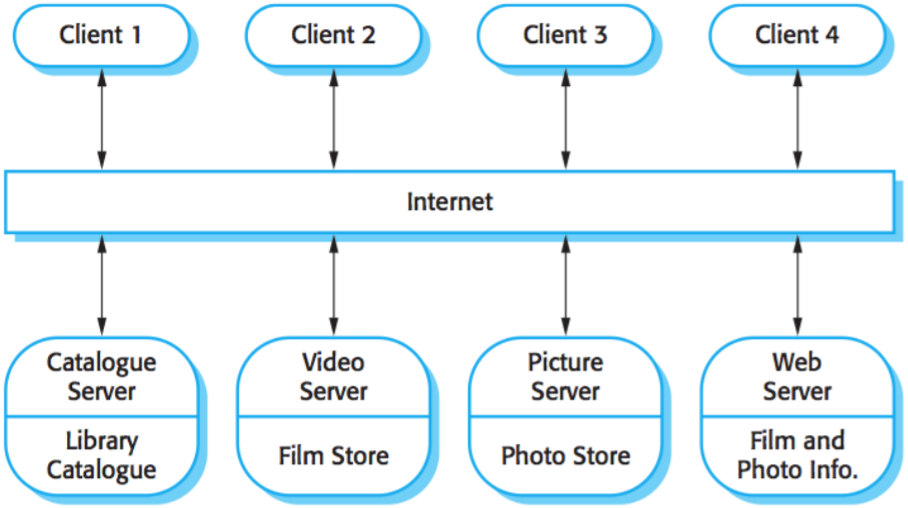
This implements a separation of concerns by abstracting the data persistence logic in applications.

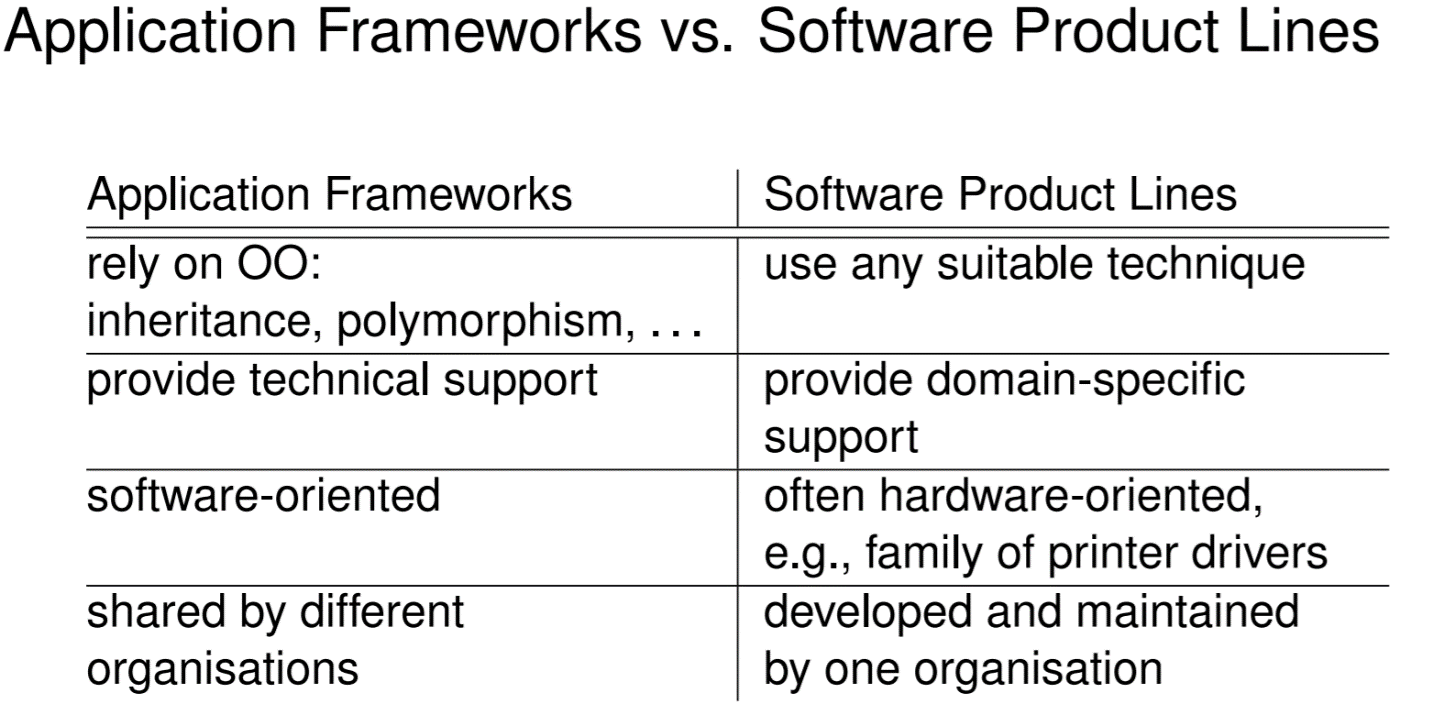
This logic is universal source of data to which all other kinds of clients and users connect (software components) to modify and operate on different things.

**Pipeline**

Just read from the picture and make something up.

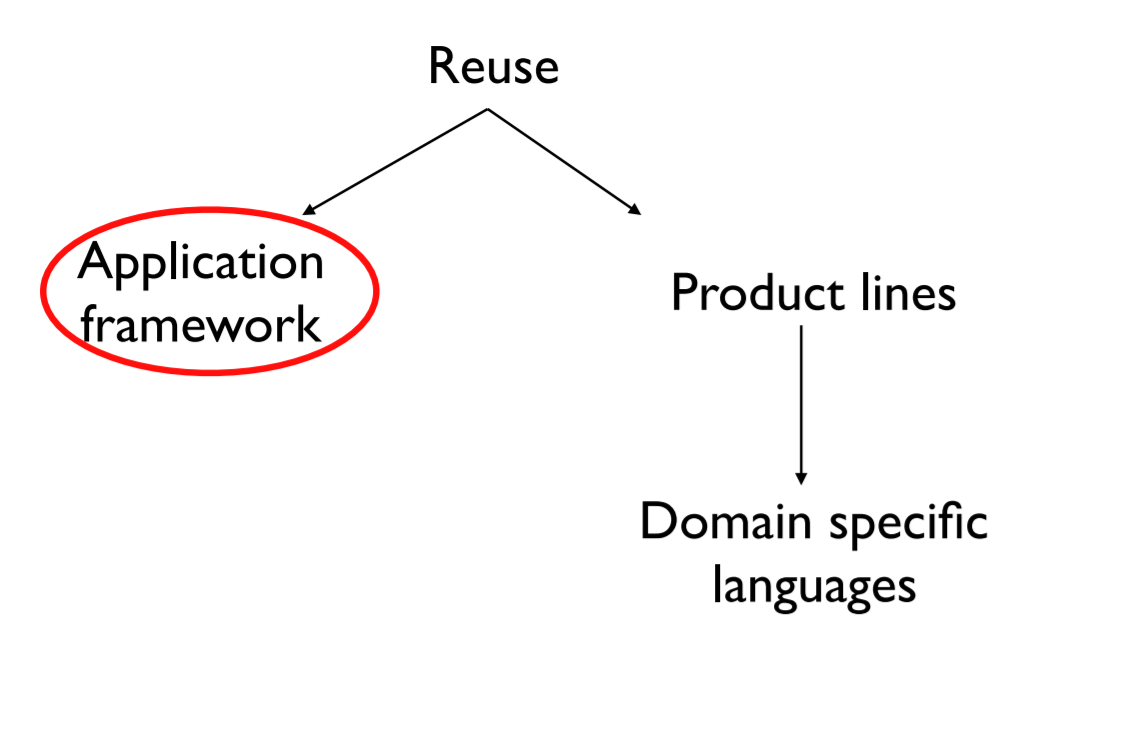
**Client-Server**

[Distributed](https://en.wikipedia.org/wiki/Distributed_application) architecture structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system.



**Reuse**

• reuse-based software engineering - reuse existing software as much as possible   
  
- system reuse - number of applications incorporated into larger “system of systems”   
• application reuse - existing application incorporated into other system (e.g. software product lines)   
• component reuse - entire subsystems or just small modules   
• object, function reuse - single functions, standard libraries offer this  
 - concept reuse - e.g. “design patterns”



**Application framework**

idea: reuse software at higher level of abstraction   
• not fine-grained objects, but larger-grained “framework”   
• application framework: provides a generic structure of an application   
• you “extend” (in sense of OO) to implement your functionality   
• it is software - a collection of abstract and concrete classes (if it's an OO framework)  
  
**Product lines**when company needs to support many similar (but not identical) systems   
• e.g. printer manufacturer   
• every printer has its own control software   
• control software very similar   
• so make core product - adapt for each printer

* **Common architecture, shared components, some specialisations**