Software Design

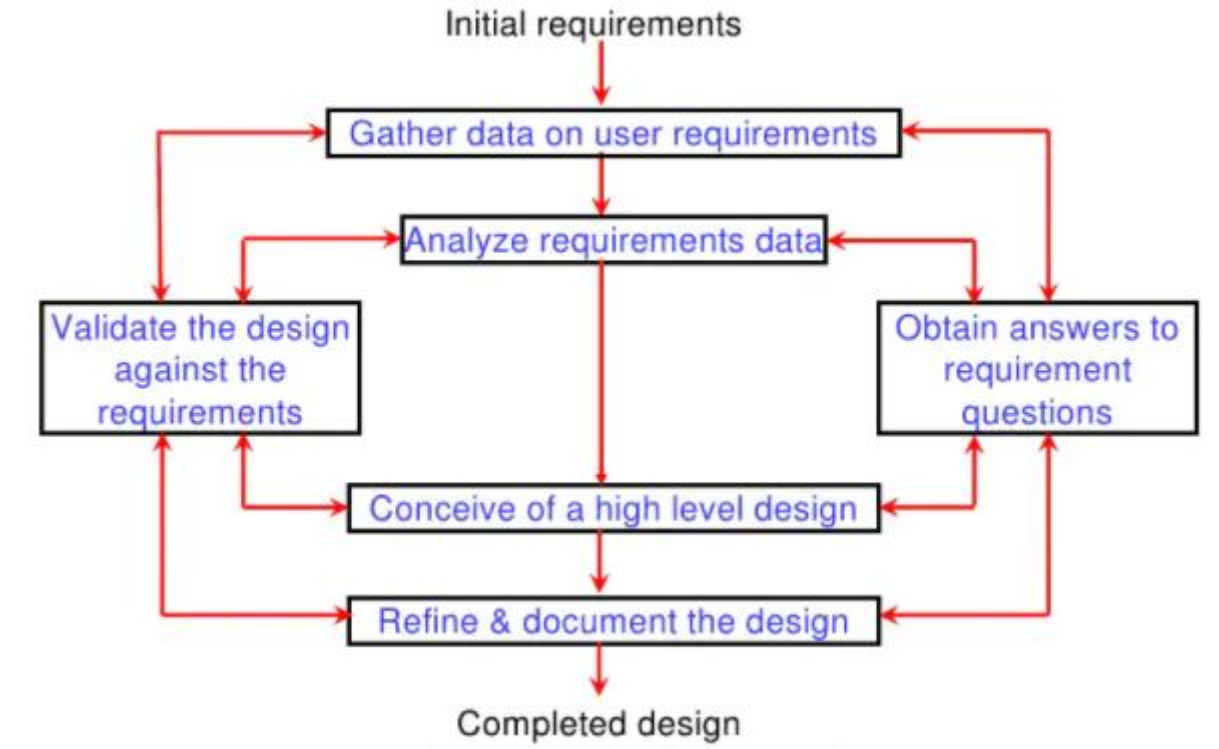
Software Design

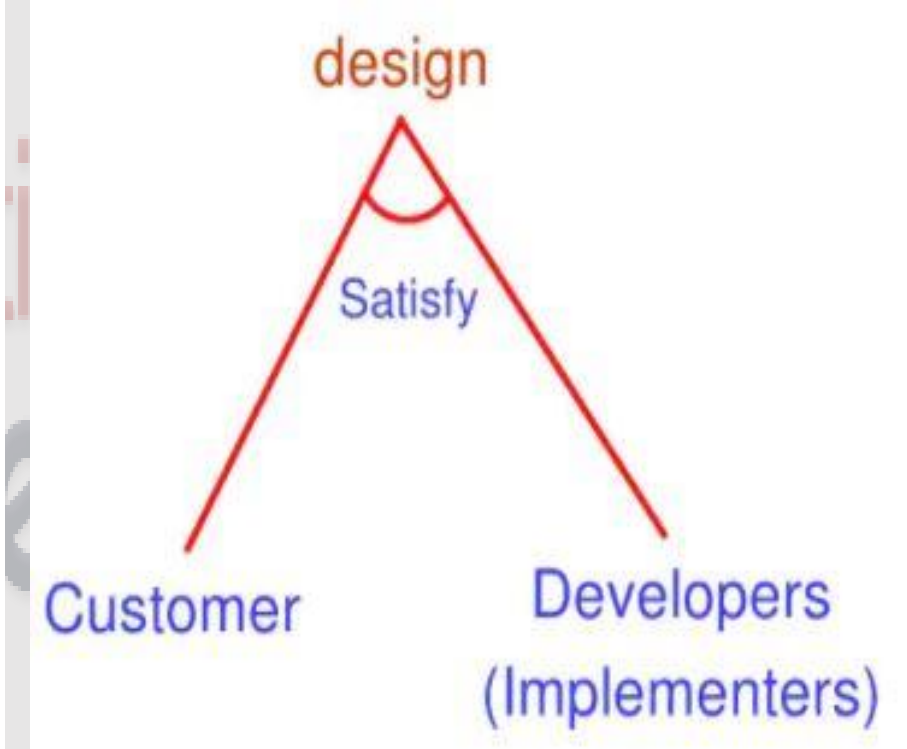
- A plan or Drawing
- Software design is an iterative process through which requirements are translated into a blueprint for constructing the software.
- It is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.
- It may be platform independent or platform specific depending on the availability of the technology called for by the design.
- It allows software engineers to produce various models that form a kind of blueprint of the solution to be implemented.

Quality Attributes of a good Design

- **Correctness:** Software design should be correct as per requirement.
- **Completeness:** The design should have all components like data structures, modules, and external interfaces, etc.
- **Efficiency:** Resources should be used efficiently by the program.
- **Flexibility:** Able to modify on changing needs.
- **Consistency:** There should not be any inconsistency in the design.
- **Maintainability:** The design should be so simple so that it can be easily maintainable by other designers.

Design Framework

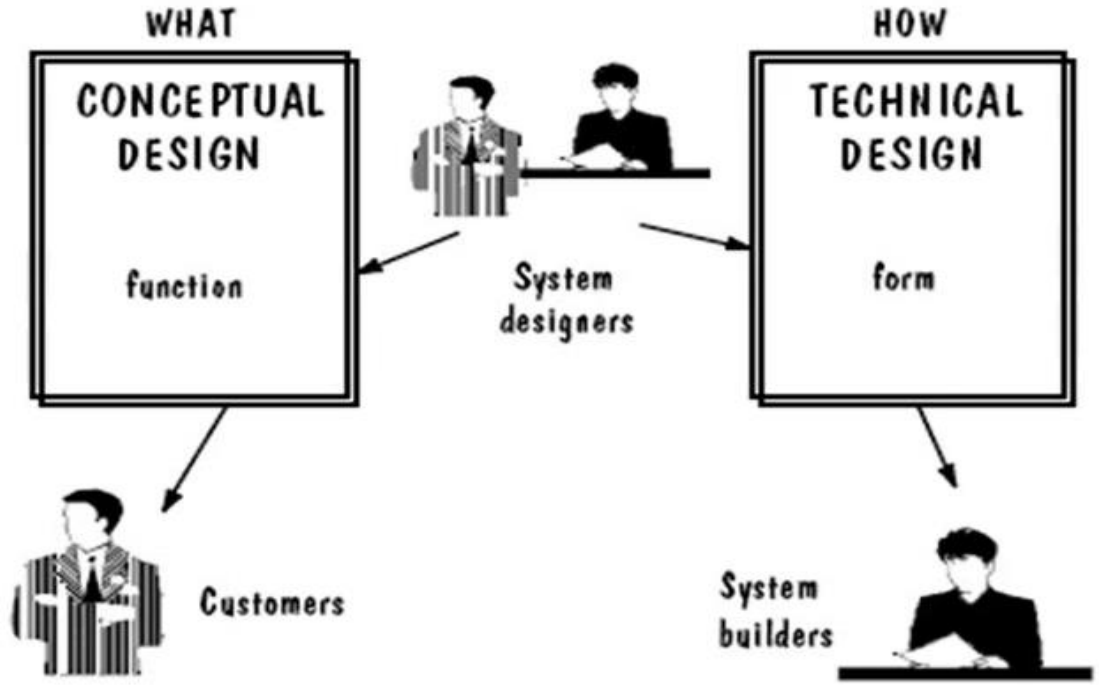




Conceptual and Technical Design

Technical design:

- Tells the programmers what the system will do
- It includes the functioning and working of the conceptual design.
- It describes any other thing that converts the requirements into a solution to the customer's problem.
- Includes:
  - major hardware components and their function
  - hierarchy and function of software components
  - data structures, data flow
  - Any other things that translates requirement into solution to customer problem



Conceptual design:

- It gives the answers to customer queries.
- It shows the conceptual model i.e., what a system should look like.
- It is written in the customer's language and designed according to the client's requirements.

  Where will the data come from?
  - What will happen to the data in the system?
  - What will the system look like to users?
  - What choices will be offered to users?
  - What is the timing of events?
  - What will the reports and screens look like?

Benefits of Design

Good Design:

- Makes project easy
- Directly link to good quality software
- Minimize downstream costs

Bad Design:

- make project impossible
- can’t be rescued by good construction
- unstable system
- fail when small changes are made
- difficult to test
- quality cannot be assessed

Software Design Principles

Abstraction:

- **To hide the details to reduce complexity and increase efficiency/quality.**
- It can be used for existing element as well as the component being designed.
- For example, a class car would be made up of engine, gearbox, steering objects and many more components. To build car class, one does not need to know how the different components of the car work internally, but only how to interface with them.

There are two common abstraction mechanisms

Functional Abstraction:

- A module is specified by the method it performs.
- The details of the algorithm to accomplish the functions are not visible to the user of the function.
- Functional abstraction forms the basis for **Function oriented design approaches**.

Data Abstraction:

- Details of the data elements are not visible to the users of data.
- Data Abstraction forms the basis for **Object Oriented design approaches**.

Software Design Principles

Modularity:

- Software is divided into separately named components, often called ‘MODULES’ that are used to detect problems at ease.
- This follows the "DIVIDE AND CONQUER" conclusion. It's easier to solve a complex problem when you break it into manageable pieces.

Advantage of modularization

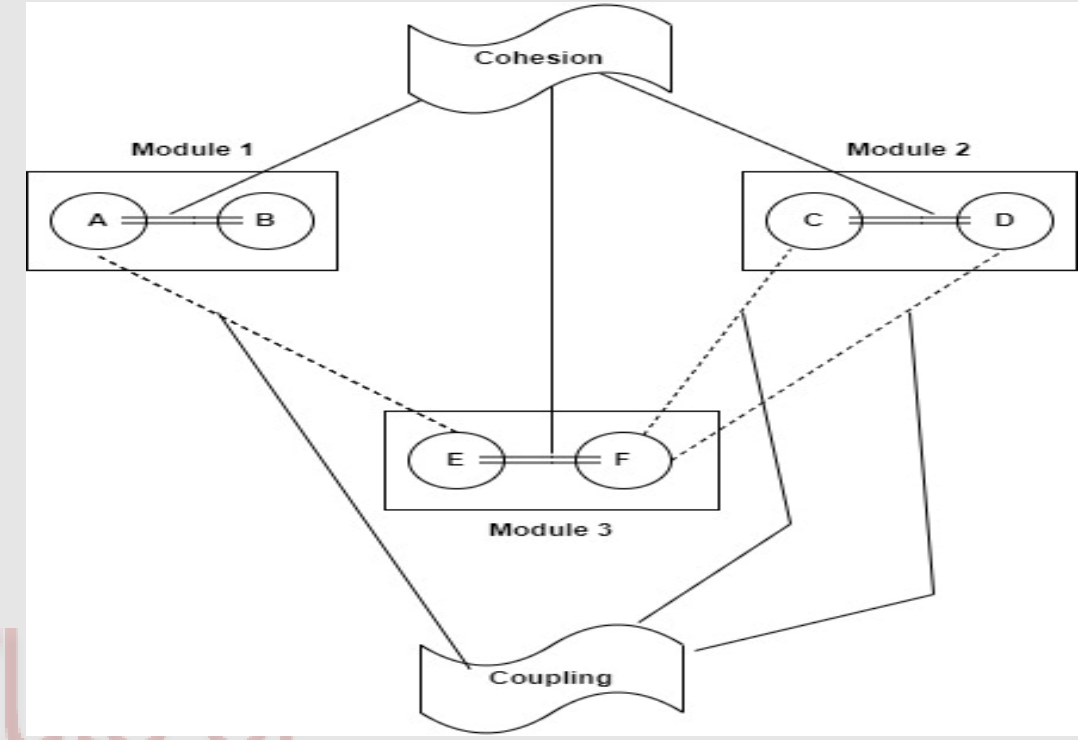
- Smaller components are easier to maintain
- Program can be divided based on functional aspects
- Desired level of abstraction can be brought in the program
- Components with high usage can be re-used.
- Concurrent execution can be made possible

Software Design Principles

Functional Independence:

- **Multiple functions/methods in the project should be independent to each other so that designing, coding and testing will be efficient.**
- Independence is important because it makes implementation more accessible and faster.
- The independent modules are easier to maintain, test, and reduce error propagation and can be reused in other programs as well.
- Functional Independence is key to design and design is key to software quality.
- Independence is assessed using two criteria: **Cohesion** and **Coupling**

Software Design Principles



Cohesion:

- Cohesion is the indication of the relationship within modules.
- It shows the module's relative functional strength.
- It is a degree (quality) to which a component/module focuses on the single thing.
- A module having **low coupling and high cohesion** is said to be **functionally independent** of other modules.

Coupling:

- It is the indication of relationship between modules.
- It shows relative dependence/ interdependence among the modules.
- It is a degree to which a component/module is connected to other modules.
- While designing you should strive for low coupling i.e dependency between modules should be less.

Software Design Principles

Refactoring [reconstruct something]

- Refactoring simply means reconstructing something in such a way that it does not affect the behavior of any other features.
- It is the process of changing a software system in such a way that it does not alter the external behavior of the code, yet improves its internal structure.
- It reduces complexity and simplify it without affecting the behavior or its functions.

Software Design Principles

Patterns

- Pattern simply means a repeated form or design in which the same shape is repeated several times to form a pattern.
- Design pattern can be used throughout software design. One analysis model has been developed, the designer can examine a detailed representation of the problem to be solved.
- It is a reusable solution to a commonly occurring design problem.
- Pattern in the design process means the repetition of a solution to a common recurring problem within a certain context.
- Design patterns are adapted for the unique characteristics of the particular problem

Software Design Principles

Refinement

- Refinement simply means to refine something to remove any impurities if present and increase the quality.
- The refinement concept of software design is actually a process of developing or presenting the software or system in a detailed manner that means to elaborate a system or software.
- Refinement is very necessary to find out any error if present and then to reduce it.
- **Stepwise refinement or decomposition**: It is the most effective way to solve a complex problem is to break it down into successively simpler sub problems.
- You start by breaking the whole task down into simpler parts.

Software Design Principles

Architecture:

- Architecture serves as a **blueprint for a system**. It provides an abstraction to manage the system complexity and establish a communication and coordination mechanism among components.
- It is the process of defining a collection of hardware and software components and their interfaces to establish the framework for the development of a computer system.
- It defines its overall structure, the interactions between its components, and the principles and guidelines that guide its development.

Importance of Architecture in software engineering

- **Ensures Quality**: A well-designed architecture ensures that software systems are reliable, efficient, and maintainable, reducing the risk of defects, errors, and other issues.
- **Facilitates Collaboration**: An architecture provides a common language and framework for developers, architects, and other stakeholders to collaborate effectively, reducing misunderstandings and errors.
- **Enables Scalability**: A well-designed architecture allows software systems to be scalable, meaning they can handle an increasing number of users, requests, and data volumes without compromising performance or stability.
- **Improves Maintainability**: A well-designed architecture makes it easier to maintain software systems over time, as it provides a clear understanding of the system's structure and how its components are related.

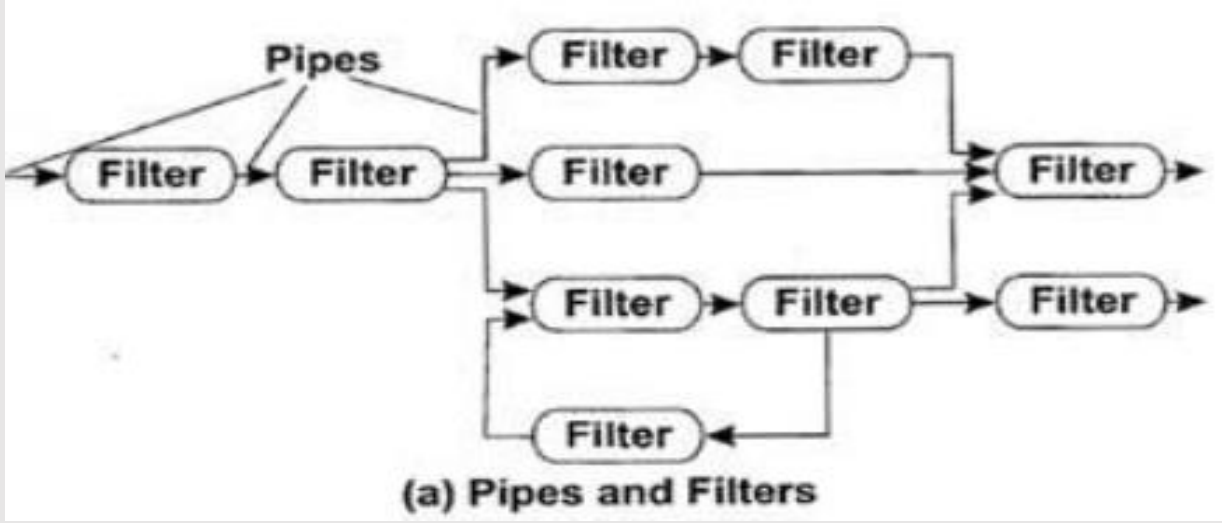
Architectural Styles

- It is a pattern or set of principles that guide the design and development of software systems.
- the objective of using architectural styles is to establish a structure for all the components present in a system.

Types of Architectural Styles

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

Data Flow Architecture



- It is mainly used in the systems that accept some inputs and transform it into the desired outputs by applying a series of transformations.
- Each component, known as **filter,** transforms the data and sends this transformed data to other filters for further processing using the connector, known as **pipe**.
- Each filter works as an independent entity, that is, it is not concerned with the filter which is producing or consuming the data.
- A pipe is a unidirectional channel which transports the data received on one end to the other end. It does not change the data in anyway; it merely supplies the data to the filter on the receiver end.

Object Oriented Architecture

- In it, components of a system encapsulate data and operations, which are applied to manipulate the data.
- In this style, components are represented as objects and they interact with each other through methods (connectors).

Layered System Architecture

- The different layers are defined in the architecture. It consists of outer and inner layer.
- The components of outer layer manage the user interface operations.
- Components execute the operating system interfacing at the inner layer.
- The inner layers are application layer, utility layer and the core layer.
- In many cases, it is possible that more than one pattern is suitable and the alternate architectural style can be designed and evaluated.

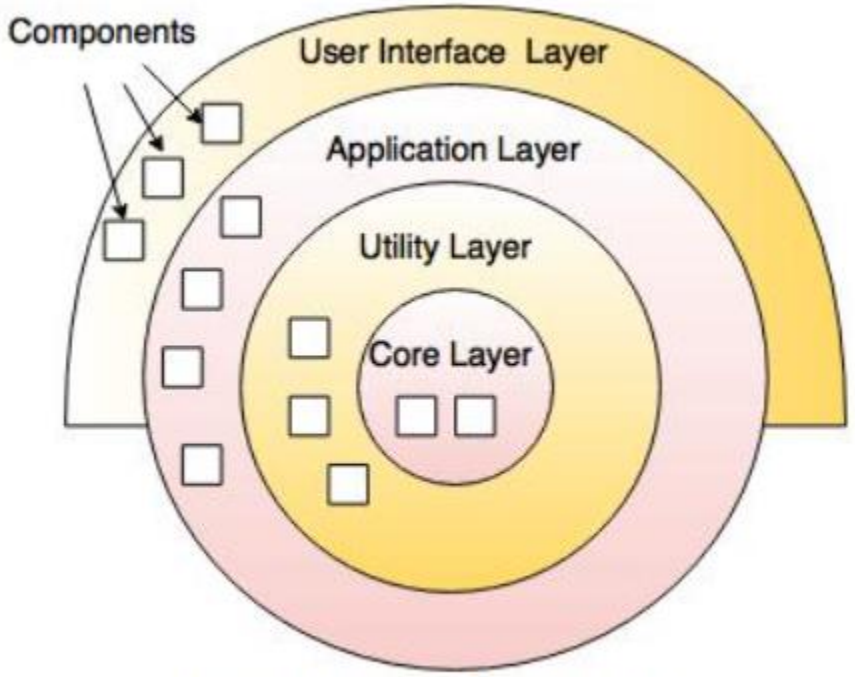
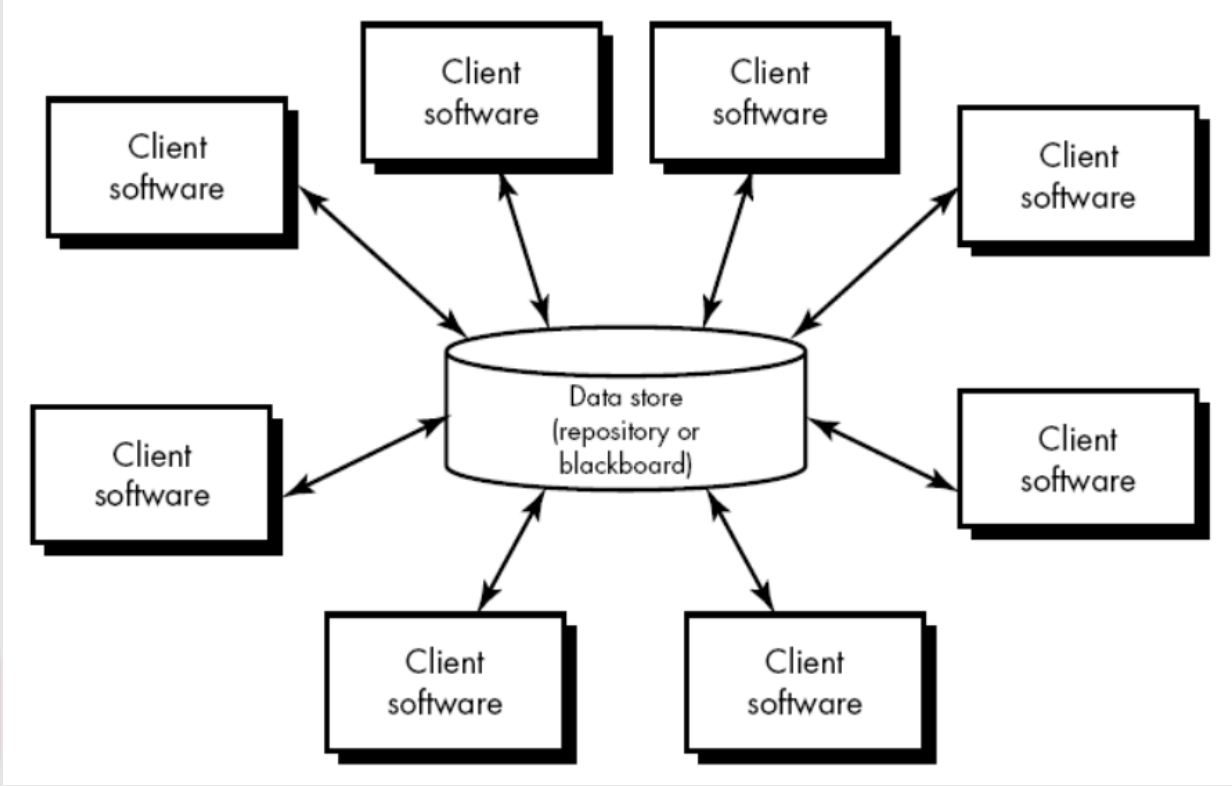


Fig.- Layered Architecture

Data-centered architecture

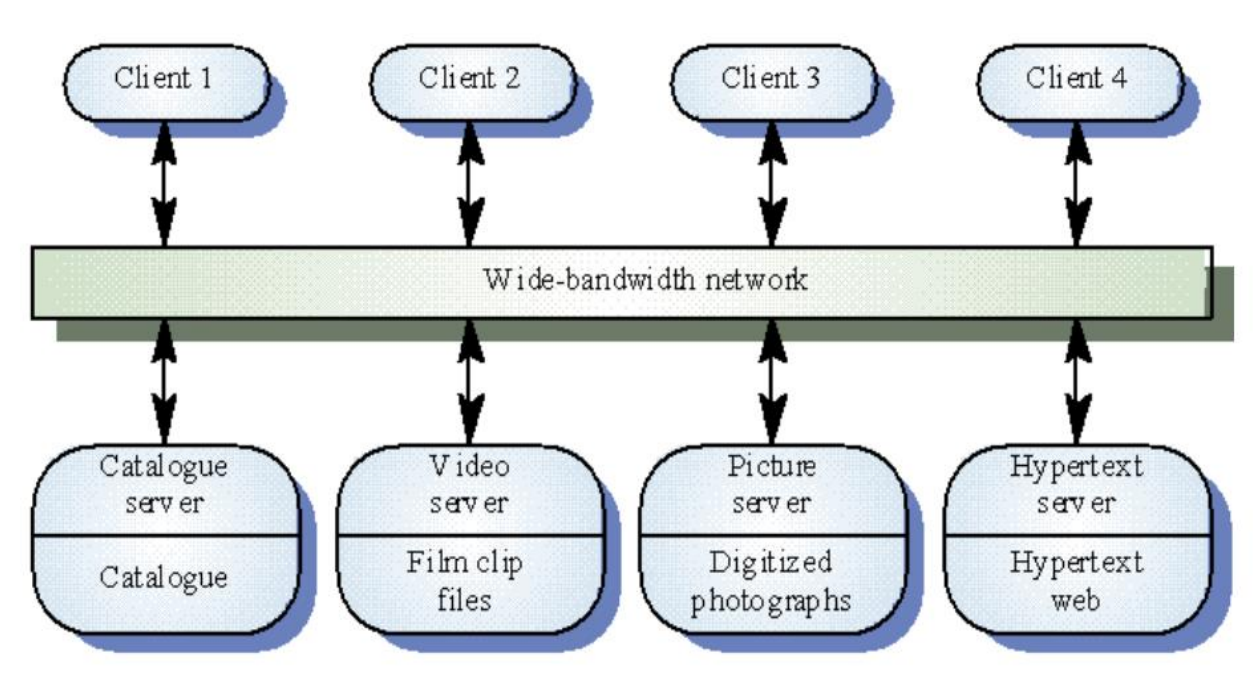


- It is a layered process which provides architectural guidelines in data center development.
- It serves as a blueprint for designing and deploying a data center facility.
- It is usually created in the data center design and constructing phase.
- In it, new components corresponding to clients can be added and existing components can be modified easily without taking into account other clients. This is because client components operate independently of one another.
- It has two distinct components: a **central data structure** or data store (central repository) and a **collection of client software.**
- The datastore (for example, a database or a file) represents the current state of the data and the client software performs several operations like add, delete, update, etc., on data stored in the data store.

Call and Return Architecture

- This architectural style enables a software designer (system architect) to achieve a program structure that is relatively easy to modify and scale.
- A number of sub styles exist within this category as below:
  - **Main program/subprogram architectures:** This classic program structure decomposes function into a control hierarchy where a “main” program invokes a number of program components, which in turn may invoke still other components.
  - **Remote procedure call architectures:** The components of a main program/subprogram architecture are distributed across multiple computers on a network.

Client and Server architecture



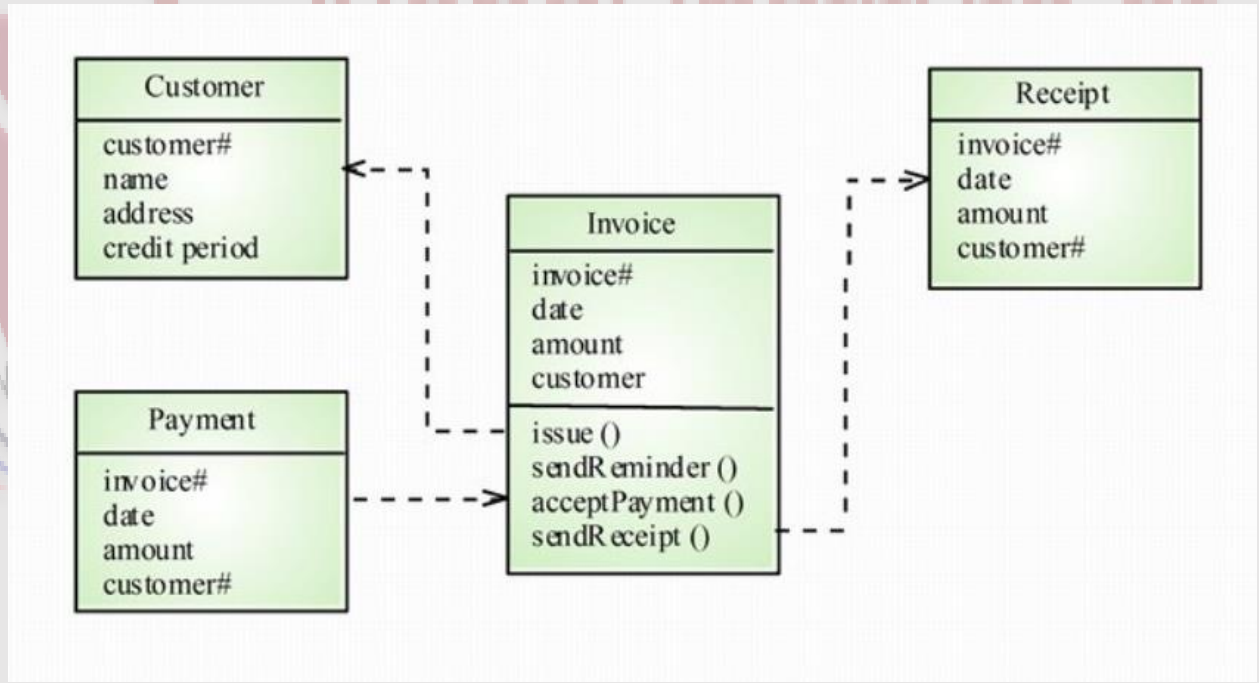
- It separates the application into two parts, the client and the server, and allows them to communicate over a network.
- The client sends requests to the server, and the server processes those requests and sends back responses.

Modular Decomposition

- Modular Decomposition means dividing each subsystem in modules and identifying interconnection between modules.
- Two modular decomposition models covered:
  - **Object model** where the system is decomposed into interacting objects
  - **Data-flow model** where the system is decomposed into functional modules which transform inputs to outputs. Also known as the pipeline model

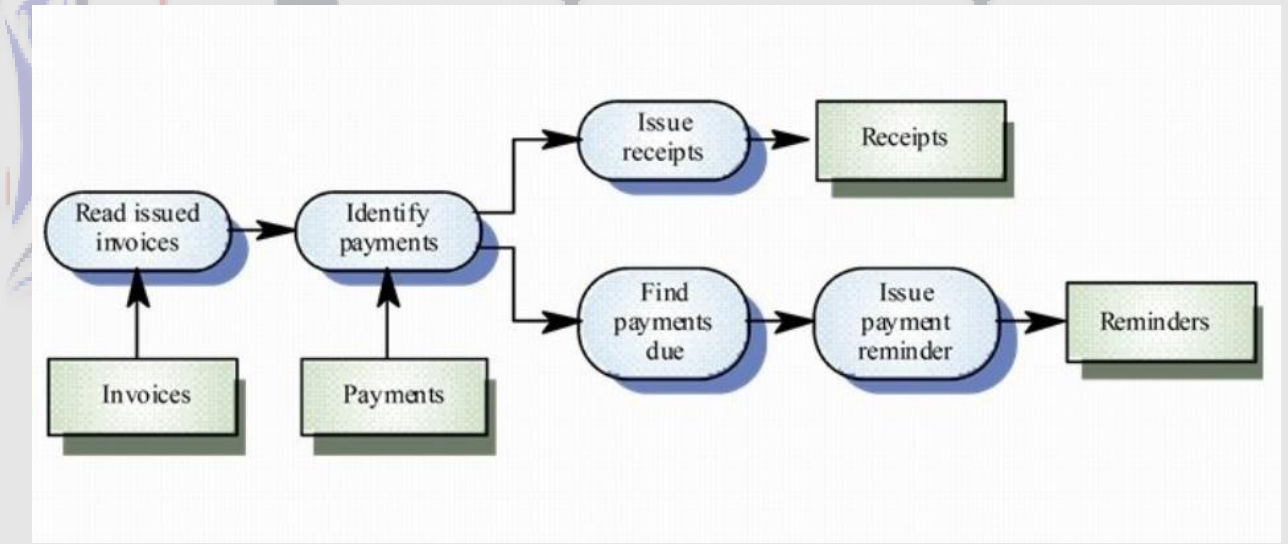
Object Models

- Object-oriented decomposition is concerned with identifying object classes, their attributes and operations.
- When implemented, objects are created from these classes and some control model used to coordinate object operations.



Data-Flow Models

- Functional transformations process their inputs to produce outputs.
- May be referred to as a pipe and filter model (as in UNIX shell)
- Variants of this approach are very common. When transformations are sequential, this is a batch sequential model which is extensively used in data processing systems
- Not really suitable for interactive systems



Real time software design

- A real-time system is a software system where the correct functioning of the system depends on the results produced by the system and the time at which these results are produced
- A ‘soft’ real-time system is a system whose operation is degraded if results are not produced according to the specified timing requirements
- A ‘hard’ real-time system is a system whose operation is incorrect if results are not produced according to the timing specification

Real time software design

- It refers to the process of creating software systems that must meet strict timing constraints and deadlines.
- are used in a wide range of applications, including aviation, defense, medical devices, and industrial control systems.
- The design process typically involves analyzing the system's requirements, identifying the critical paths and potential bottlenecks, and developing algorithms and data structures that can meet the system's timing constraints.
- Real-time software design involves understanding the system's performance and timing requirements from the early stages of development.

Component-based software engineering (CBSE)

- It is an approach to software development that emphasizes the use of reusable software components to build complex software systems.
- In it, developers assemble software systems by selecting and integrating pre-existing components, rather than developing everything from scratch.
- It increase productivity, reduce development costs, and improve the quality of the software system.

Component-based software engineering (CBSE)

Advantages:

- improved software quality
- better reuse of existing components
- faster development, and easier maintenance

Disadvantages:

- need for well-defined component interfaces,
- compatibility issues
- the potential for increased complexity due to the integration of multiple components.


THANK
YOU