Architectural Design in Software Engineering

Requirements of the software should be transformed into an architecture that describes the software's top-level structure and identifies its components. This is accomplished through architectural design (also called **system design),** which acts as a preliminary 'blueprint' from which software can be developed. **IEEE** defines architectural design as '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.' This framework is established by examining the software requirements document and designing a model for providing implementation details. These details are used to specify the components of the system along with their inputs, outputs, functions, and the interaction between them. An architectural design performs the following functions.

1. It defines an abstraction level at which the designers can specify the functional and performance behaviour of the system.

2. It acts as a guideline for enhancing the system (whenever required) by describing those features of the system that can be modified easily without affecting the system integrity.

3. It evaluates all top-level designs.

4. It develops and documents top-level design for the external and internal interfaces.

5. It develops preliminary versions of user documentation.

6. It defines and documents preliminary test requirements and the schedule for software integration.

7. The sources of architectural design are listed below.

8. Information regarding the application domain for the software to be developed

9. Using data-flow diagrams

10. Availability of architectural patterns and architectural styles.

Architectural design is of crucial importance in software engineering during which the essential requirements like reliability, cost, and performance are dealt with. This task is cumbersome as the software engineering paradigm is shifting from monolithic, stand-alone, built-from-scratch systems to componentized, evolvable, standards-based, and product line-oriented systems. Also, a key challenge for designers is to know precisely how to proceed from requirements to architectural design. To avoid these problems, designers adopt strategies such as reusability, componentization, platform-based, standards-based, and so on.

Though the architectural design is the responsibility of developers, some other people like user representatives, systems engineers, hardware engineers, and operations personnel are also involved. All these stakeholders must also be consulted while reviewing the architectural design in order to minimize the risks and errors.

Architectural Design Representation

Architectural design can be represented using the following models.

1. **Structural model:** Illustrates architecture as an ordered collection of program components
2. **Dynamic model:** Specifies the behavioral aspect of the software architecture and indicates how the structure or system configuration changes as the function changes due to change in the external environment
3. **Process model:** Focuses on the design of the business or technical process, which must be implemented in the system
4. **Functional model:** Represents the functional hierarchy of a system
5. **Framework model:** Attempts to identify repeatable architectural design patterns encountered in similar types of application. This leads to an increase in the level of abstraction.

Architectural Design Output

The architectural design process results in an **Architectural Design Document (ADD).** This document consists of a number of graphical representations that comprises software models along with associated descriptive text. The software models include static model, interface model, relationship model, and dynamic process model. They show how the system is organized into a process at run-time.

Architectural design document gives the developers a solution to the problem stated in the Software Requirements Specification (SRS). Note that it considers only those requirements in detail that affect the program structure. In addition to ADD, other outputs of the architectural design are listed below.

- Various reports including audit report, progress report, and configuration status accounts report
- Various plans for detailed design phase, which include the following
- Software verification and validation plan
- Software configuration management plan
- Software quality assurance plan
- Software project management plan.

Architectural Styles

Architectural styles define a group of interlinked systems that share structural and semantic properties. In short, the objective of using architectural styles is to establish a structure for all the components present in a system. If an existing architecture is to be re-engineered, then imposition of an architectural style results in fundamental changes in the structure of the system. This change also includes re-assignment of the functionality performed by the components.

By applying certain constraints on the design space, we can make different style-specific analysis from an architectural style. In addition, if conventional structures are used for an architectural style, the other stakeholders can easily understand the organization of the system.

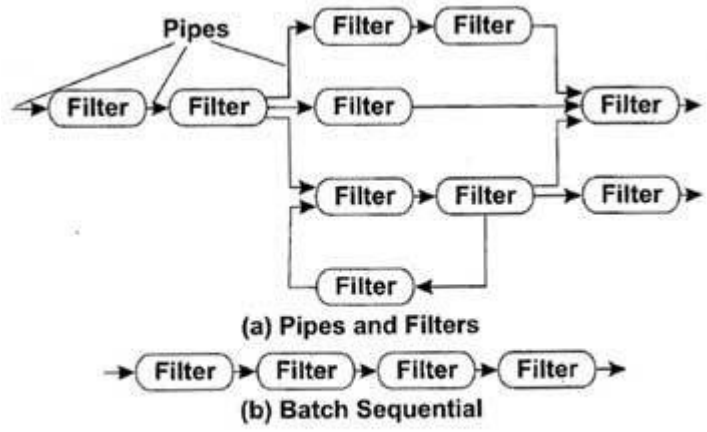
A computer-based system (software is part of this system) exhibits one of the many available architectural styles. Every architectural style describes a system category that includes the following.

1. Computational components such as clients, server, filter, and database to execute the desired system function
2. A set of *connectors* such as procedure call, events broadcast, database protocols, and pipes to provide communication among the computational components
3. Constraints to define integration of components to form a system
4. A *semantic* model, which enable the software designer to identify the characteristics of the system as a whole by studying the characteristics of its components.

Some of the commonly used architectural styles are data-flow architecture, object oriented architecture, layered system architecture, data-centered architecture, and call and return architecture. Note that the use of an appropriate architectural style promotes design reuse, leads to code reuse, and supports interoperability.

Data-flow Architecture

Data-flow architecture 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.



Data-flow Architecture

Most of the times, the data-flow architecture degenerates a batch sequential system. In this system, a batch of data is accepted as input and then a series of sequential filters are applied to transform this data. One common example of this architecture is UNIX shell programs. In these programs, UNIX processes act as filters and the file system through which UNIX processes interact, act as pipes. Other well-known examples of this architecture are compilers, signal processing systems, parallel

programming, functional programming, and distributed systems. Some advantages associated with the data-flow architecture are listed below.

1. It supports reusability.
2. It is maintainable and modifiable.
3. It supports concurrent execution.
4. Some disadvantages associated with the data-flow architecture are listed below.
5. It often degenerates to batch sequential system.
6. It does not provide enough support for applications requires user interaction.
7. It is difficult to synchronize two different but related streams.

Object-oriented Architecture

In object-oriented architectural style, 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). This architectural style has two important characteristics, which are listed below.

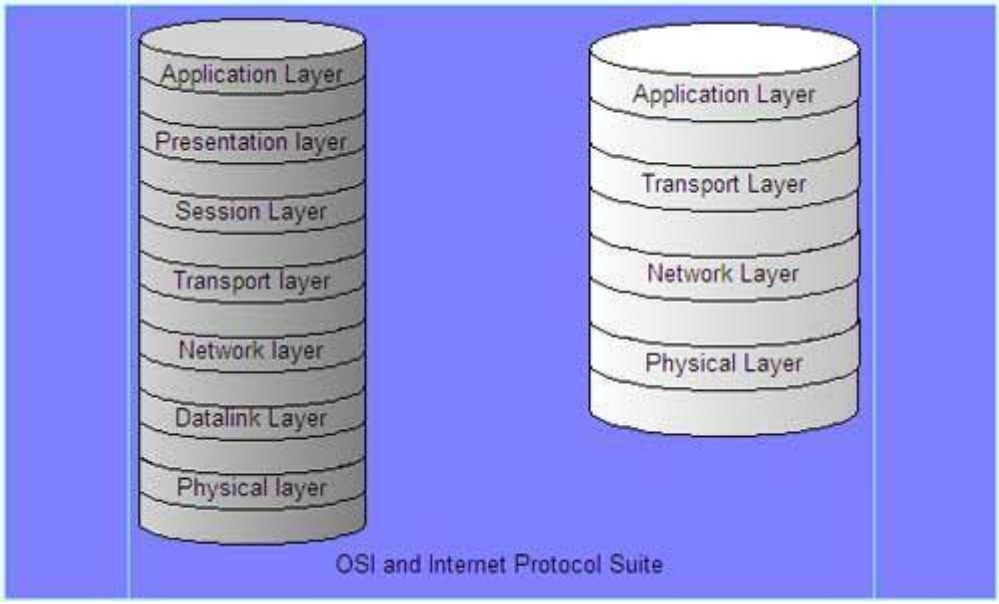
1. Objects maintain the integrity of the system.
2. An object is not aware of the representation of other objects.
3. Some of the advantages associated with the object-oriented architecture are listed below.
4. It allows designers to decompose a problem into a collection of independent objects.
5. The implementation detail of objects is hidden from each other and hence, they can be changed without affecting other objects.

Layered Architecture

In layered architecture, several layers (components) are defined with each layer

performing a well-defined set of operations. These layers are arranged in a hierarchical manner, each one built upon the one below it. Each layer provides a set of services to the layer above it and acts as a client to the layer below it. The interaction between layers is provided through protocols (connectors) that define a set of rules to be followed during interaction. One common example of this architectural style is OSI-ISO (Open Systems Interconnection-International

Organization for Standardization) communication system.



Data-centered Architecture

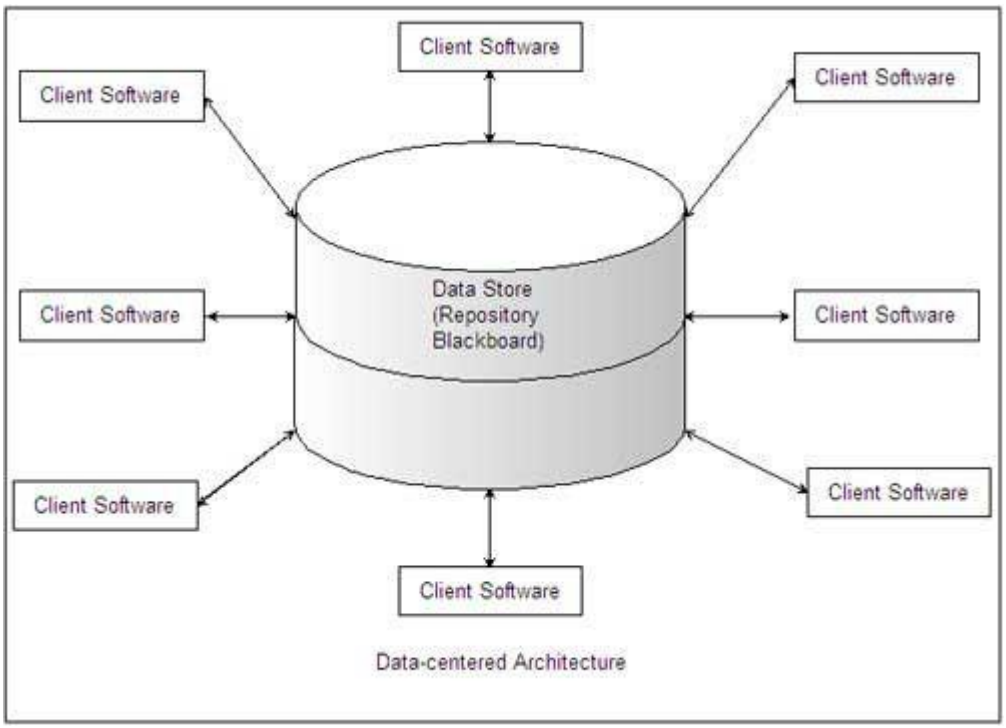
A data-centered architecture has two distinct components: a **central data structure** or data store (central repository) and a **collection of client software.** The datastore (for example, a database or a file) represents the current state of the data andthe client software performs several operations like add, delete, update, etc., onthe data stored in the data store. In some cases, the data storeallows the client software to access the data independent of any changes or theactions of other client software.

In this architectural style, 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.

A variation of this architectural style is blackboard system in which the data store is transformed into a blackboard that notifies the client software when the data (of their interest) changes. In addition, the information can be transferred among the clients through the blackboard component.

Some advantages of the data-centered architecture are listed below.

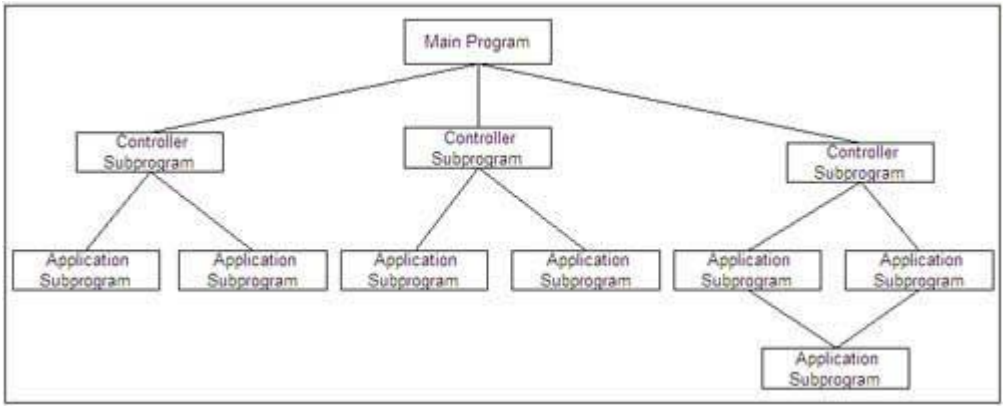
- Clients operate independently of one another.
- Data repository is independent of the clients.
- It adds scalability (that is, new clients can be added easily).
- It supports modifiability.
- It achieves data integration in component-based development using blackboard.



Call and Return Architecture

A call and return architecture enables software designers to achieve a program structure, which can be easily modified. This style consists of the following two substyles.

- **Main program/subprogram architecture:** In this, function is decomposed into a control hierarchy where the main program invokes a number of program components, which in turn may invoke other components.



- **Remote procedure call architecture:** In this, components of the main or subprogram architecture are distributed over a network across multiple computers.