KEY PRINCIPLES

Software architecture is described as the organization of a system, where the system represents a set of components that accomplish the defined functions.

Architectural Style

The **architectural style**, also called as **architectural pattern**, is a set of principles which shapes an application. It defines an abstract framework for a family of system in terms of the pattern of structural organization.

The architectural style is responsible to −

- Provide a lexicon of components and connectors with rules on how they can be combined.
- Improve partitioning and allow the reuse of design by giving solutions to frequently occurring problems.
- Describe a particular way to configure a collection of components *amodulewithwell − definedinterfaces, reusable, andreplaceable* and connectors *communicationlinkbetweenmodules*.

The software that is built for computer-based systems exhibit one of many architectural styles. Each style describes a system category that encompasses −

- A set of component types which perform a required function by the system.
- A set of connectors *subroutinecall, remoteprocedurecall, datastream, andsocket* that enable communication, coordination, and cooperation among different components.
- Semantic constraints which define how components can be integrated to form the system.
- A topological layout of the components indicating their runtime interrelationships.

Common Architectural Design

The following table lists architectural styles that can be organized by their key focus area −

Category	Architectural Design	Description
Communication	Message bus	Prescribes use of a software system that can receive and send messages using one or more communication channels.
	Service–Oriented Architecture *SOA*	Defines the applications that expose and consume functionality as a service using contracts and messages.
Deployment	Client/server	Separate the system into two applications, where the client makes requests to the server.
	3-tier or N-tier	Separates the functionality into separate segments with each segment being a tier located on a physically separate computer.
Domain	Domain Driven Design	Focused on modeling a business domain and defining business objects based on entities within the business domain.
	Component Based	Breakdown the application design into reusable functional or logical components that expose well-defined communication

		interfaces.
Structure	Layered	Divide the concerns of the application into stacked groups *layers*.
	Object oriented	Based on the division of responsibilities of an application or system into objects, each containing the data and the behavior relevant to the object.

Types of Architecture

There are four types of architecture from the viewpoint of an enterprise and collectively, these architectures are referred to as **enterprise architecture**.

- **Business architecture** − Defines the strategy of business, governance, organization, and key business processes within an enterprise and focuses on the analysis and design of business processes.
- **Application *software* architecture** − Serves as the blueprint for individual application systems, their interactions, and their relationships to the business processes of the organization.
- **Information architecture** − Defines the logical and physical data assets and data management resources.
- **Information technology *IT* architecture** − Defines the hardware and software building blocks that make up the overall information system of the organization.

Architecture Design Process

The architecture design process focuses on the decomposition of a system into different components and their interactions to satisfy functional and nonfunctional requirements. The key inputs to software architecture design are −

- The requirements produced by the analysis tasks.
- The hardware architecture *thesoftwarearchitectinturnprovidesrequirementstothesystemarchitect, whoconfiguresthehardwarearchitecture*.

The result or output of the architecture design process is an **architectural description**. The basic architecture design process is composed of the following steps −

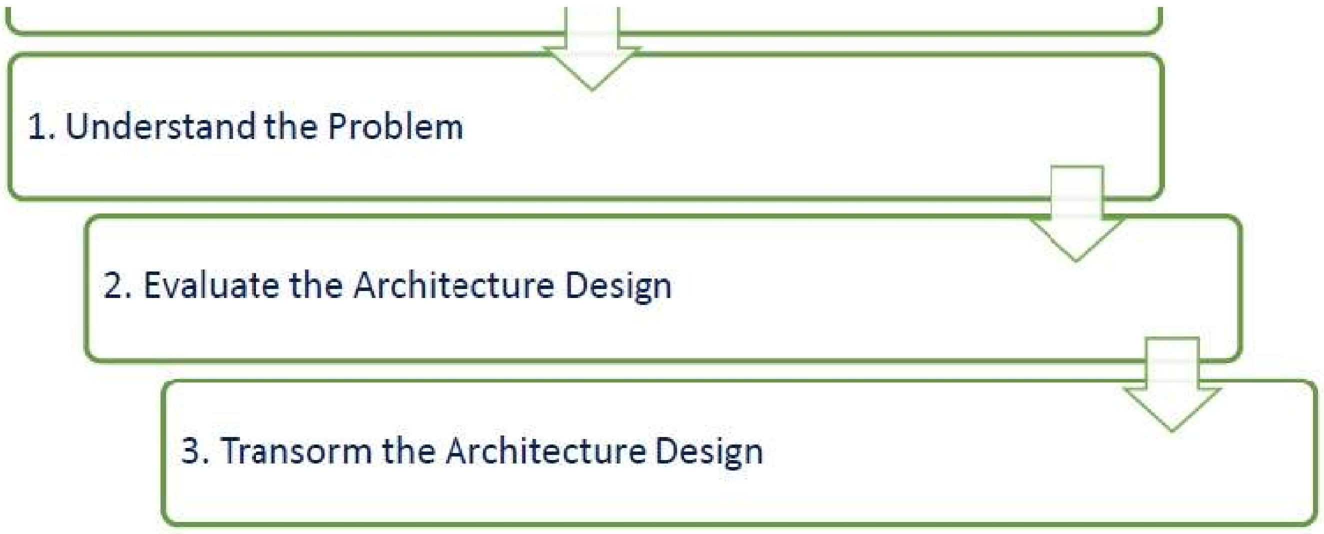
Understand the Problem

This is the most crucial step because it affects the quality of the design that follows. Without a clear understanding of the problem, it is not possible to create an effective solution. In fact, many software projects and products are considered as unsuccessful because they did not actually solve a valid business problem or have a recognizable return on investment *ROI*.

Identify Design Elements and their Relationships

In this phase, build a baseline for defining the boundaries and context of the system. Decomposition of the system into its main components is based on the functional requirements. The decomposition can be modeled by using a design structure matrix *DSM*, which shows the dependencies between design elements without specifying the granularity of the elements.

In this step, the first validation of the architecture is done by describing a number of system instances and this step is referred as functionality based architectural design.


1. Understand the Problem
2. Evaluate the Architecture Design
3. Transorm the Architecture Design

Evaluate the Architecture Design

Each quality attribute is given an estimate, so in order to gather qualitative measures or quantitative data, the design is evaluated. It involves evaluating the architecture for conformance to architectural quality attributes requirements.

If all the estimated quality attributes are as per the required standard, the architectural design process is finished. If not, then the third phase of software architecture design is entered: architecture transformation. However, if the observed quality attribute does not meet its requirements, then a new design must be created.

Transform the Architecture Design

This step is performed after an evaluation of the architectural design. The architectural design must be changed until it completely satisfies the quality attribute requirements. It is concerned with selecting design solutions to improve the quality attributes while preserving the domain functionality.

Further, a design is transformed by applying design operators, styles, or patterns. For transformation, take the existing design and apply design operator such as decomposition, replication, compression, abstraction, and resource sharing.

Moreover, the design is again evaluated and the same process is repeated multiple times if necessary and even performed recursively. The transformations $i.e. quality attribute optimizing solutions$ generally improve one or some quality attributes while they affect others negatively.

Key Architecture Principles

Following are the key principles to be considered while designing an architecture −

Build to Change Instead of Building to Last

Consider how the application may need to change over time to address new requirements and challenges, and build in the flexibility to support this.

Reduce Risk and Model to Analyze

Use design tools, visualizations, modeling systems such as UML to capture requirements and design decisions. The impacts can also be analyzed. Do not formalize the model to the extent that it suppresses the capability to iterate and adapt the design easily.

Use Models and Visualizations as a Communication and Collaboration Tool

Efficient communication of the design, the decisions, and ongoing changes to the design is critical to good architecture. Use models, views, and other visualizations of the architecture to communicate and share the design efficiently with all the stakeholders. This enables rapid communication of changes to the design.

Identify and understand key engineering decisions and areas where mistakes are most often made. Invest in getting key decisions right the first time to make the design more flexible and less likely to be broken by changes.

Use an Incremental and Iterative Approach

Start with baseline architecture and then evolve candidate architectures by iterative testing to improve the architecture. Iteratively add details to the design over multiple passes to get the big or right picture and then focus on the details.

Key Design Principles

Following are the design principles to be considered for minimizing cost, maintenance requirements, and maximizing extendibility, usability of architecture −

Separation of Concerns

Divide the components of system into specific features so that there is no overlapping among the components functionality. This will provide high cohesion and low coupling. This approach avoids the interdependency among components of system which helps in maintaining the system easy.

Single Responsibility Principle

Each and every module of a system should have one specific responsibility, which helps the user to clearly understand the system. It should also help with integration of the component with other components.

Principle of Least Knowledge

Any component or object should not have the knowledge about internal details of other components. This approach avoids interdependency and helps maintainability.

Minimize Large Design Upfront

Minimize large design upfront if the requirements of an application are unclear. If there is a possibility of modifying requirements, then avoid making a large design for whole system.

Do not Repeat the Functionality

It specifies that functionality of the components should not to be repeated and hence a piece of code should be implemented in one component only. Duplication of functionality within a single application can make it difficult to implement changes, decrease clarity, and introduce potential inconsistencies.

Prefer Composition over Inheritance while Reusing the Functionality

Inheritance creates dependency between children and parent classes and hence it blocks the free use of the child classes. In contrast, the composition provides a great level of freedom and reduces the inheritance hierarchies.

Identify Components and Group them in Logical Layers

Identity components and the area of concern that are needed in system to satisfy the requirements. Then group these related components in a logical layer, which will help the user to understand the structure of the system at a high level. Avoid mixing components of different type of concerns in same layer.

Define the Communication Protocol between Layers

Understand how components will communicate with each other which requires a complete knowledge of deployment scenarios and the production environment.

Define Data Format for a Layer

Various components will interact with each other through data format. Do not mix the data formats so that applications are easy to implement, extend, and maintain. Try to keep data format same for a layer, so that various components need not code/decode the data while communicating with each other. It reduces a processing overhead.

System Service Components should be Abstract

Code related to security, communications, or system services like logging, profiling, and configuration should be abstracted in the separate components. Do not mix this code with business logic, as it is easy to extend design and maintain it.

Design Exceptions and Exception Handling Mechanism

Defining exceptions in advance, helps the components to manage errors or unwanted situation in an elegant manner. The exception management will be same throughout the system.

Naming Conventions

Naming conventions should be defined in advance. They provide a consistent model that helps the users to understand the system easily. It is easier for team members to validate code written by others, and hence will increase the maintainability.