What is Systems Design – Learn System Design

Systems Design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. It involves translating user requirements into a detailed blueprint that guides the implementation phase. The goal is to create a well-organized and efficient structure that meets the intended purpose while considering factors like scalability, maintainability, and performance.

Mastering Systems Design is crucial for anyone looking to build robust and scalable systems. Our comprehensive **Systems Design course** provides you with the knowledge and skills to excel in this area. Through practical examples and expert insights, you'll learn how to effectively translate user requirements into detailed designs that can be successfully implemented.

Important Topics for System Design

Why learn System Design?

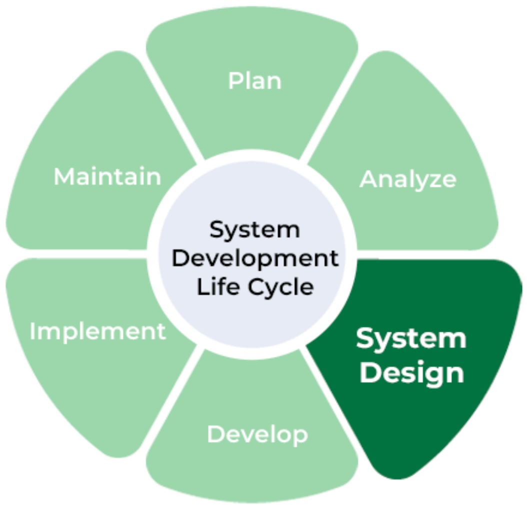
In any development process, be it Software or any other tech, the most important stage is **Design** . Without the designing phase, you cannot jump to the implementation or the testing part. The same is the case with the System as well.

Systems Design not only is a vital step in the development of the system but also provides the backbone to handle exceptional scenarios because it represents the business logic of software.

From the above SDLC steps, it is clear that system design acts as a backbone because no matter how good the coding part is executed, it, later on, becomes irrelevant if the corresponding design is not good. So here we get crucial vital information as to why it is been asked in every Product Based Company.

Below are the main 5 reasons why to learn system design:

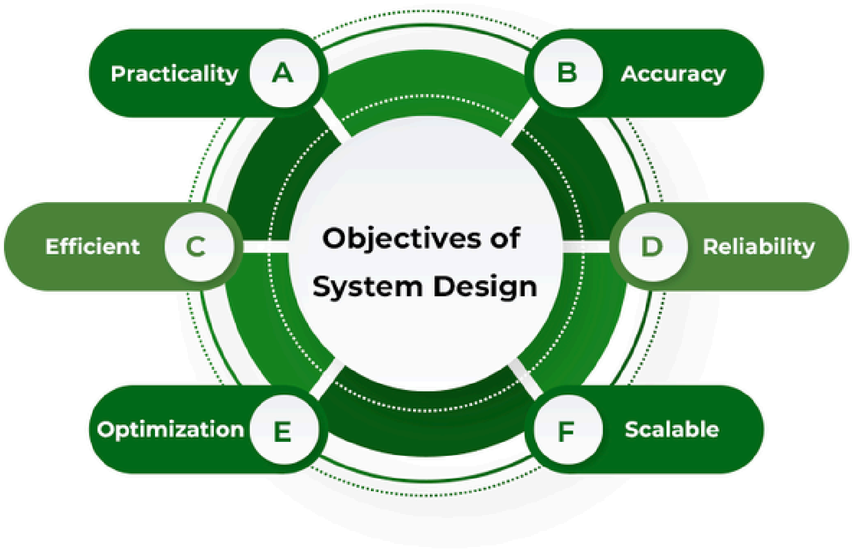
- System Design is crucial in FAANG interviews.
- You need to have good expertise in System Design to be hired for Senior positions.
- System Design proficiency enhances job security.
- Understanding System Design will help you to have good communication.
- Learning System Design improves decision-making.



The importance of System Design phase in SDLC

Objectives of Systems Design

1. **Practicality** : We need a system that should be targetting the set of audiences(users) corresponding to which they are designing.
2. **Accuracy** : Above system design should be designed in such a way it fulfills nearly all requirements around which it is designed be it functional o non-functional requirements.
3. **Completeness** : System design should meet all user requirements
4. **Efficient** : The system design should be such that it should not overuse surpassing the cost of resources nor under use as it will by now we know will result in low thorough put (output) and less response time(latency).
5. **Reliability** : The system designed should be in proximity to a failure-free environment for a certain period of time.
6. **Optimization** : Time and space are just likely what we do for code chunks for individual components to work in a system.
7. **Scalable(flexibility)** : System design should be adaptable with time as per different user needs of customers which we know will keep on changing on time. The best example here out is the well-known firm: Nokia. It is the most important aspect while designing systems and is the result of why 1 of 100 startups succeed over the long run, the best example here out is GeeksforGeeks.



Objectives of System Design

> Note: System Design also helps us to achieve fault tolerence which is ability of a software to continue working where even its 1 or 2 component fails.

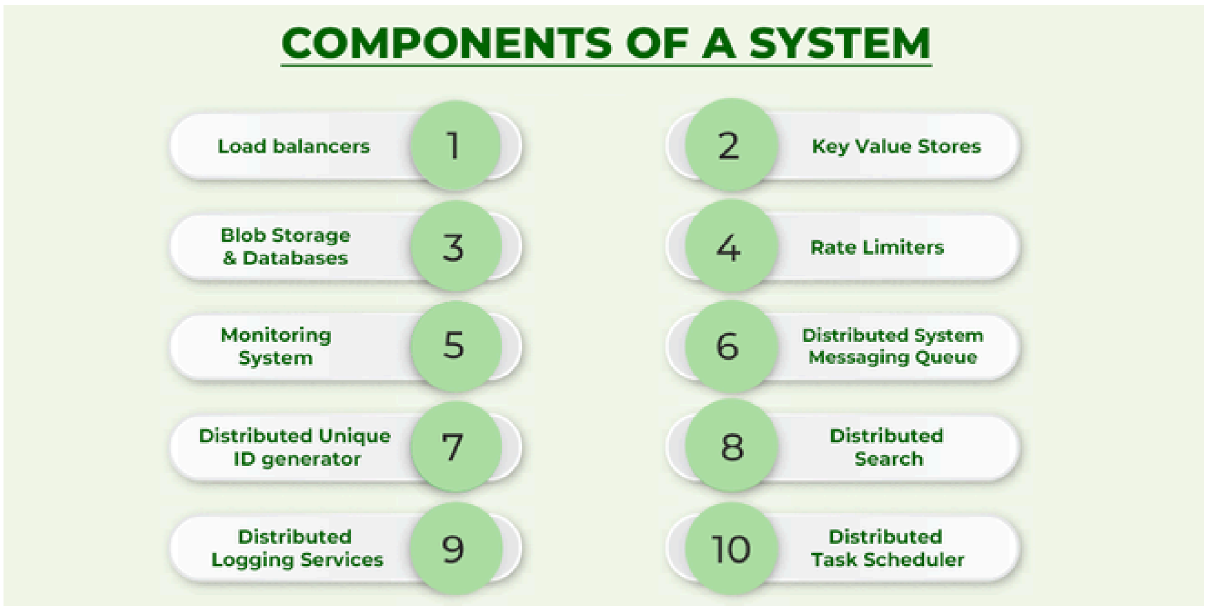
Now after glancing and going through the above objectives let us now discuss the advantages of system design to understand it better as the below advantages get our understanding even closer to real-life.

Components of Systems Design

Below are some of the major components of the System Design. discussed in brief. The detailed version of this will be discussed in different posts:

1. **Load balancers:** Most crucial component for scalability, availability, and performance measures for systems.
2. **Key Value Stores:** It is a storage system similar to hashtables where key-value stores are distributed hash tables.
3. **Blob Storage:** Blob stands for binary large objects, as the name suggests is storage for unstructured data such as YouTube, and Netflix.
4. **Databases:** It is an organized collection of data so that they can be easily accessed and modified.
5. **Rate Limiters:** These sets the maximum number of requests a service can fulfill.
6. **Monitoring System:** These are basically software where system administrator monitor infrastructures such as bandwidth, CPU, routers, switches, etc.
7. **Distributed System Messaging Queue:** Transaction medium between producers and consumers.
8. **Distributed Unique ID generator:** In the case of large distributed systems, every moment multiple tasks are occurring so in order to distinguish it assign a tag corresponding to every event.

9. **Distributed Search:** Over every website, crucial information that visitors will seek is put into the search bar.
10. **Distributed Logging Services:** Tracing sequences of events from end to end.
11. **Distributed Task Scheduler:** Computational resources such as CPU, memory, storage, etc.


COMPONENTS OF A SYSTEM
1 Load balancers
2 Key Value Stores
3 Blob Storage & Databases
4 Rate Limiters
5 Monitoring System
6 Distributed System Messaging Queue
7 Distributed Unique ID generator
8 Distributed Search
9 Distributed Logging Services
10 Distributed Task Scheduler

Components of System Design

System Design Life Cycle (SDLC)

The System Design Life Cycle (SDLC) is a comprehensive process that outlines the steps involved in designing and developing a system, be it a software application, hardware solution, or an integrated system combining both. It encompasses a series of phases that guide engineers through the creation of a system that aligns with the user's needs and organizational goals. The SDLC aims to ensure that the end product is reliable, scalable, and maintainable.

The Phases (Stages) of the System Design Life Cycle are:

1. Planning
2. Feasibility Study
3. System Design
4. Implementation
5. Testing
6. Deployment
7. Maintenance and Support

System Architecture

> Software architecture is a way in which we define **how the components of a design are depicted design and deployment of software** .

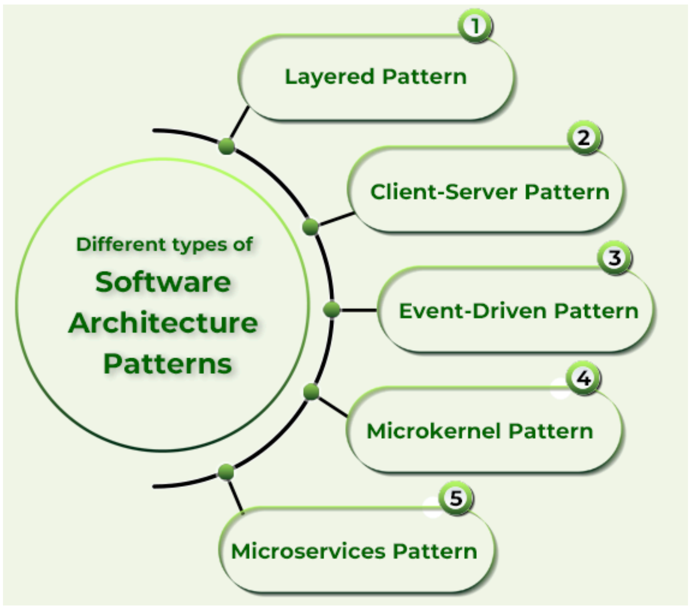
It is basically the skeleton design of a software system depicting components, abstraction levels, and other aspects of a software system. In order to understand it in a layman's language, it is the aim or logic of a business should be crystal clear and laid out on a single sheet of paper. Here goals of big projects and further guides to scaling up are there for the existing system and upcoming systems to be scaled up.

System Architecture Patterns

There are various ways to organize the components in software architecture. And the different predefined organization of components in software architectures are known as software architecture patterns. A lot of patterns were tried and tested. Most of them have successfully solved various problems. In each pattern, the components are organized differently for solving a specific problem in software architectures.

Different types of Software Architecture Patterns include:

1. Layered Pattern
2. Client-Server Pattern
3. Event-Driven Pattern
4. Microkernel Pattern
5. Microservices Pattern



System Architecture Patterns

Modularity and Interfaces In Systems Design

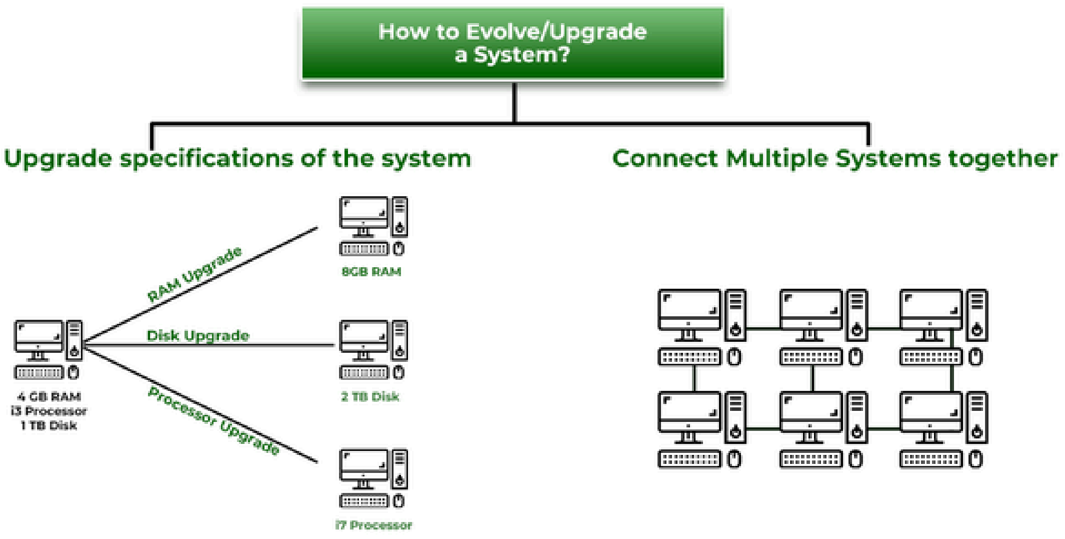
- **Modular design** refers to a method/procedure for product design involving integrating or combining smaller, independent elements to create a finished product. A large product (like a car) can be separated into smaller, simpler components that are separately developed and produced using the modular design approach. The ultimate product is created by integrating (or assembling) each of these component parts.
- **Interfaces In System Design** is the area where users interact. It consists of the screen displays that facilitate system navigation, the screens and forms that gather data, and the system's reports.

Evolution/Upgrade/Scale of an Existing System

With the increase in tech usage, be it offline or online, it is now a must for every developer to design and create a **scalable system** . If the system is not scalable, with the increase in users, it is very likely that the system will crash. Hence the concept of scaling comes into play.

Suppose there is a system with configurations of specific disk and RAM which was handling tasks. Now if we need to evolve our system or scale up, we have two options with us.

1. **Upgrade Specifications of existing system:** We are simply improving the processor by upgrading the RAM and disk size and many other components. Note that here we are not caring about the scalability and availability of network bandwidth. Here as per evolution we are working over the availability factor only considering scalability will be maintained. This is known as vertical scaling.
2. **Create a Distributed System by connecting multiple systems together:** We see above that if scalability is not up to mark then we need multiple systems for this measure as availability measures do have a limitation. In order to scale up, we need more systems (more chunks of blocks) and this is known as horizontal scaling.


How to Evolve/Upgrade a System?
Upgrade specifications of the system
Connect Multiple Systems together
RAM Upgrade
Disk Upgrade
Processor Upgrade
4 GB RAM
i3 Processor
1 TB Disk
8GB RAM
2 TB Disk
i7 Processor

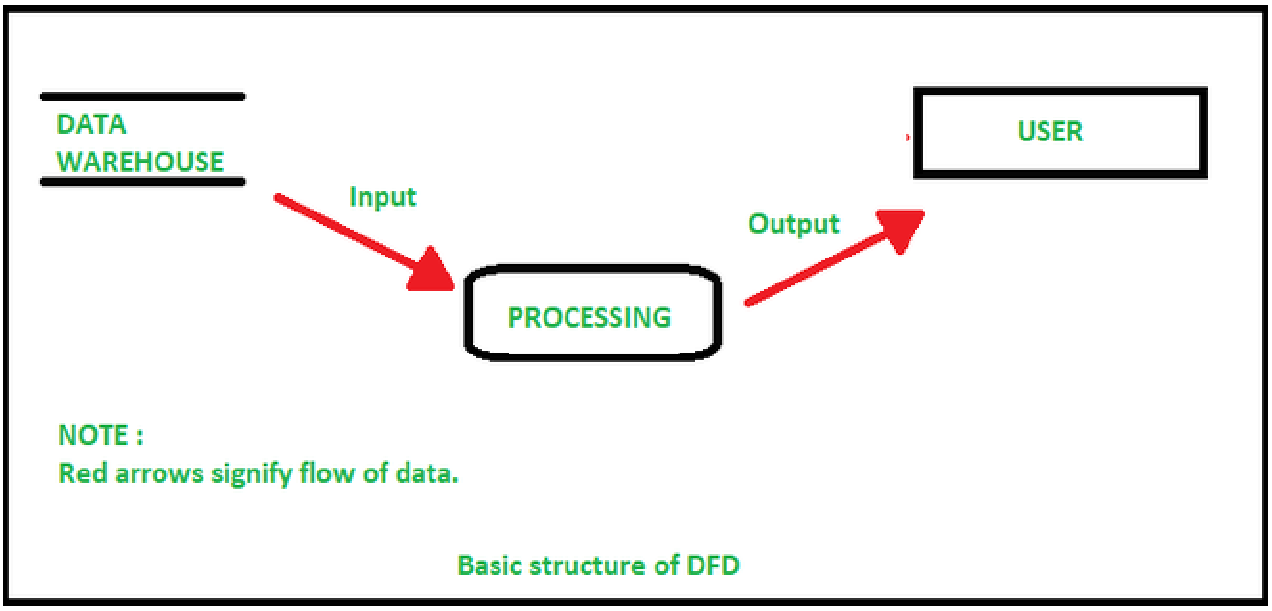
Evolution/Upgrade/Scale of an Existing System

How Data Flows Between System

Data flows between systems through **Data Flow Diagrams or DFDs** .

> **Data Flow Diagrams or DFDs** is defined as a graphical representation of the flow of data through information. DFD is designed to show how a system is divided into smaller portions and to highlight the flow of data between these parts.

Here's an example to demonstrate the Data Flow Diagram's basic structure:



Data Flow Diagram's basic structure

Components of a DFD:

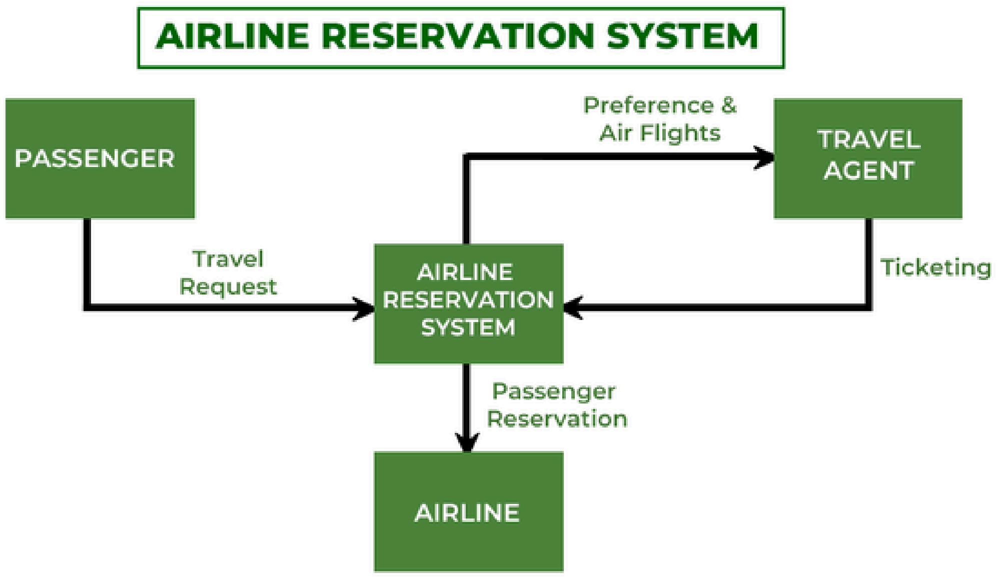
Representation	Action performed
Square	Defines the source of destination of data
Arrow	Identifies data flow and acts as a pipeline throughwhich information flows
Circle/Bubble	Represents a process that transforms incoming data flow into outgoing data
Open Rectangle	It is a data store or data at rest/temporary repository of data

> **Note:** Sender and Receiver should be written in uppercase always. Rather it is good practrice to use uppercaswe letter what so ever is placed in square box as per DFD conventions.

System Design Example: Airline Reservation System

Now since we have discussed about the basics of System Design so far, let us now understand System Design through a basic example – Airline Reservation System.

To understand better about the components and design of Airline Reservation System, let us first review its context-level flow diagram:



System Design Example: Airline Reservation System

Let us now understand the DFD of the Airline Reservation System:

- In the above flow diagram, **Passenger** , **Travel Agent** , **Airline** are the sources across which data is migrating.
- Here data is transmitted from **Passenger to book an Airline ticket** as shown with the DFD arrow sign where the travel request is placed.

- Now, this data is transmitted across two sources, as shown above, namely ‘ **Travel Agent** ‘ and ‘ **Airline** ‘ where if the seat is available **Preferences** and **Air Flight** request is placed to the source.
- Travel Agent and corresponding Ticketing are placed as requested.
- If no ticket is available, then a request for Passenger Reservation is placed to the source – Airline.

Advantages of System Design

Upon detailed discussion of the introduction to system design, it is a must now to discuss its merits and demerits.

> The greatest advantage of system design is inculcating awareness and creativity in full-stack developers’ via synergic bonding of API protocols gateways, networking and databases.

Some of the major advantages of System Design include:

- Reduces the design cost of a product.
- Speedy software development process
- Saves overall time in SDLC
- Increases efficiency and consistency of a programmer.
- Saves resources