Design Pattens

In software engineering, a design pattern is a general repeatable solution to a commonly occurring problem in software design. A design pattern isn't a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

Uses of Design Patterns:

Design patterns can speed up the development process by providing tested, proven development paradigms. Effective software design requires considering issues that may not become visible until later in the implementation. Reusing design patterns helps to prevent subtle issues that can cause major problems and improves code readability for coders and architects familiar with the patterns.

Creational Design Patterns:

These design patterns are all about class instantiation. This pattern can be further divided into class-creation patterns and object-creational patterns. While class-creation patterns use inheritance effectively in the instantiation process, object-creation patterns use delegation effectively to get the job done.

Of its types:

1. **Abstract Factory:** Creates an instance of several families of classes
2. **Builder:** Separates object construction from its representation
3. **Factory Method:** Creates an instance of several derived classes
4. **Object Pool:** Avoid expensive acquisition and release of resources by recycling objects that are no longer in use
5. **Prototype:** A fully initialized instance to be copied or cloned
6. **Singleton:** A class of which only a single instance can exist

Structural design patterns:

These design patterns are all about Class and Object composition. Structural class-creation patterns use inheritance to compose interfaces. Structural object-patterns define ways to compose objects to obtain new functionality.

Of its types:

1. **Adapter:** Match interfaces of different classes
2. **Bridge:** Separates an object’s interface from its implementation
3. **Composite:** A tree structure of simple and composite objects
4. **Decorator:** Add responsibilities to objects dynamically
5. **Façade:** A single class that represents an entire subsystem
6. **Flyweight:** A fine-grained instance used for efficient sharing
7. **Private Class Data:** Restricts accessor/mutator access
8. **Proxy:** An object representing another object

Behavioral design patterns:

These design patterns are all about Class's objects communication. Behavioral patterns are those patterns that are most specifically concerned with communication between objects.

Of its types:

1. **Chain of responsibility:** A way of passing a request between a chain of objects
2. **Command:** Encapsulate a command request as an object
3. **Interpreter:** A way to include language elements in a program
4. **Iterator:** Sequentially access the elements of a collection
5. **Mediator:** Defines simplified communication between classes
6. **Memento:** Capture and restore an object's internal state
7. **Null Object:** Designed to act as a default value of an object
8. **Observer:** A way of notifying change to a number of classes
9. **State:** Alter an object's behavior when its state changes
10. **Strategy:** Encapsulates an algorithm inside a class
11. **Template method:** Defer the exact steps of an algorithm to a subclass
12. **Visitor:** Defines a new operation to a class without change

Architecture Patterns

Software Architecture:

Software architecture is the blueprint of building software. It shows the overall structure of the software, the collection of components in it, and how they interact with one another while hiding the implementation.

This helps the software development team to clearly communicate how the software is going to be built as per the requirements of customers.

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 Software Architecture Patterns:

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

Layered Pattern:

As the name suggests, components(code) in this pattern are separated into layers of subtasks and they are arranged one above another.

Each layer has unique tasks to do and all the layers are independent of one another. Since each layer is independent, one can modify the code inside a layer without affecting others.

It is the most commonly used pattern for designing the majority of software. This layer is also known as 'N-tier architecture'. Basically, this pattern has 4 layers:

1. Presentation layer (The user interface layer where we see and enter data into an application.)
2. Business layer (this layer is responsible for executing business logic as per the request.)
3. Application layer (this layer acts as a medium for communication between the 'presentation layer' and 'data layer'.

4. Data layer (this layer has a database for managing data.)

Ideal for: E-commerce web applications development like Amazon.

Client-Server Pattern:

The client-server pattern has two major entities. They are a server and multiple clients, Here the server has resources (data, files or services) and a client requests the server for a particular resource. Then the server processes the request and responds back accordingly.

Examples of software developed in this pattern:

1. Email
2. WWW
3. File sharing apps
4. Banking, etc…

Event-Driven Pattern:

Event-Driven Architecture is an agile approach in which services (operations) of the software are triggered by events.

What does an event mean?

When a user takes action in the application built using the **EDA** approach, a state change happens and a reaction is generated that is called an event.

Ideal for: Building websites with JavaScript and e-commerce websites in general.

Microkernel Pattern:

Microkernel pattern has two major components. They are a core system and plug-in modules. The core system handles the fundamental and minimal operations of the application. The plug-in modules handle the extended functionalities (like extra features) and customized processing.

Microkernel pattern is ideal for: Product-based applications and scheduling applications. We love new features that keep giving dopamine boost to our brain. So this pattern is mostly preferred for app development.

Microservices Pattern:

The collection of small services that are combined to form the actual application is the concept of microservices pattern. Instead of building a bigger application, small programs are built for every service (function) of an application independently. And those small programs are bundled together to be a full-fledged application.

So adding new features and modifying existing microservices without affecting other microservices are no longer a challenge when an application is built in a microservices pattern.

Modules in the application of microservices patterns are loosely coupled. So they are easily understandable, modifiable and scalable.

Example Netflix is one of the most popular examples of software built-in microservices architecture. This pattern is most suitable for websites and web apps having small components.