Design pattern

Design patterns are used to represent some of the best practices adapted by experienced object-oriented software developers. A design pattern systematically names, motivates, and explains a general design that addresses a recurring design problem in object-oriented systems. It describes the problem, the solution, when to apply the solution, and its consequences. It also gives implementation hints and examples.

We use design pattern for a lot of reasons :

-avoid reinventing the wheel

-communication between software engineers using common vocabulary

-use the power of object oriented design

We have 3 broad categories that design patterns are divided into, as you can see in the diagram above - **creational patterns, behavioral patterns and structural patterns**. Each one of them is best suited for a specific situation. If they're used in the wrong scenario, they may do more harm than good. Let's take a look at the 3 categories of design patterns.

Behavioral design pattern :

Behavior Design Patterns are responsible for how one class communicates with others. By doing so , these patterns increase flexibility in carrying out communication.

Chain of Responsibility - It representatives the command to a chain of processing object.

Command - It generates the objects which encapsulate actions of parameters.

Interpreter - It implements a specialized language.

Chain of responsibility: An approach to passing a request between a chain of objects.

Command: Data driven behavioral pattern, which encapsulates all the information that is required for performing a "command" or action, or to trigger an event at a later time.

Use in this cases :

-encapsulates a request in an object

-decouple sender from processor or receiver

-undo operation

-when we have too much functionality in class

Pitfalls :

-usually used in combination with other patterns

-high number of classes and objects to achieve the required goal

Interpreter: Using this design pattern we can incorporate language elements in programs.

Iterator: This pattern is utilized to get a method for getting to the components of a collection object in a sequential way with no requirement to know its fundamental representation.

Mediator: Simplifies the communication between objects.

Memento: Restores and captures objects internal states.

Null Object: This design pattern is to create a null object that encapsulates the absence of an object.

Observer: This design pattern acts as a notifier of change to multiple classes.

State: The state design pattern is used when objects change their behavior based on their internal states.

Strategy: Strategy design pattern is also known as the policy pattern, that helps selecting algorithms at runtime.

Template Method: Template method, as the name suggests, defines the framework of an algorithm in the superclass, however it lets the subclasses override certain specific steps in the algorithm without making a change to the fundamental structure.

Visitor: This design pattern, separates an algorithm from the object structure on which it operates. It essentially results in the addition of new operations to the existing classes without any changes.

Design Patterns are concerned **with technical aspects of an implementation**. They have a local impact on specific parts of the implementation of a system.

Architectural Patterns are concerned **with strategic aspects of a system.** They have a global impact on the whole implementation of a system.

1. 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.

-Presentation layer (The user interface layer where we see and enter data into an application.)

-Business layer (this layer is responsible for executing business logic as per the request.)

-Application layer (this layer acts as a medium for communication between the ‘presentation layer’ and ‘data layer’.

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

Ideal for:

E-commerce web applications development like Amazon.

2. 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:

-Email.

-WWW.

-File sharing apps.

-Banking, etc…

So this pattern is suitable for developing the kind of software listed in the examples.

3. Event-Driven Pattern :

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

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.

Eg: A new user fills the signup form and clicks the signup button on Facebook and then a FB account is created for him, which is an event.

Ideal for:

Building websites with JavaScript and e-commerce websites in general.

4. 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.

Let’s imagine, you have successfully built a chat application. And the basic functionality of the app is that you can text with people across the world without an internet connection. After some time, you would like to add a voice messaging feature to the application, then you are adding the feature successfully. You can add that feature to the already developed application because the microkernel pattern facilitates you to add features as plug-ins.

Microkernel pattern is ideal for:

Product-based applications and scheduling applications. We love new features that keep giving dopamine boost to our brain. Such as Instagram reels, YouTube Shorts and a lot more that feasts us digitally. So this pattern is mostly preferred for app development.

5. 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.