Every successful application depends on

Architecture – e.g. MVC, WebApi, MVVM etc.

Design Principles –

Design Patterns – Singleton, Factory etc.

**Different Types of Software Design Principles**

1. **SOILD :**

**-** Enable us to manage most of the software design problems.

**- SOLID** is an acronym for five design principles intended to make software designs more understandable, flexible and maintainable.

**It helps us to:**

* Achieve reduction in complexity of code.
* Increase readability, extensibility and maintainability.
* Reduce error and implement reusability.
* Better testability.
* Reduce tight coupling.

**S-** Single Responsibility Principle (SRP)

* + **A class should have only one reason to change.**

Consider an interface which breaks the SRP because LogError and SendEmail has nothing to do with IUser (Same example can be used for the ISP)

interface IUser

{

bool Login();

bool Register();

void LogError();

bool SendEmail();

}

After applying SRP

interface IUser

{

bool Login();

bool Register();

}

interface ILogger

{

void LogError();

}

interface IEmail

{

bool SendEmail();

}

**O-** Open/Closed Principle (OCP)

* + Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification.

**L-** Liskov Substitution Principle (LSP)

* + S is a subtype of T, then objects of type T may be replaced with objects of type S.
  + Derived types must be completely substitutable for their base types.

**Guidelines**

* + 1. No new exception can be thrown by the subtype.
    2. Clients should not know which specific subtype they are calling.
    3. New derived class just extend without replacing the functionality of old class.

**I-** Interface Segregation Principle (ISP)

* + Many client-specific interfaces are better than one general-purpose interface.

**D-** Dependency Inversion Principle (DIP)

* + High level modules should not depend on Low level modules. Both should depends on abstractions.
  + Abstraction should not depend on details. Details should depend on abstractions.

1. **DRY (Don’t Repeat Yourself)**

This principle states that each small pieces of knowledge (code) may only occur exactly once in the entire system. This helps us to write scalable, maintainable and reusable code.

**Example –** Asp.Net MVC framework works on this principle.

1. **KISS (Keep it simple, Stupid!)**

This principle states that try to keep each small piece of software simple and unnecessary complexity should be avoided. This helps us to write easy maintainable code.

1. **YAGNI (You ain't gonna need it)**

This principle states that always implement things when you actually need them never implements things before you need them.