SOLID

- **SOLID** -> Principios base a seguir antes de proponer una arquitectura de software. Código escalable a un futuro. Alta cohesión y bajo acoplamiento (menor dependencia, mejor especificación propósitos sistema)
  - Alta cohesión: Información de una clase coherente, estar relacionada con la clase
  - Bajo acoplamiento: Clases menos ligadas entre si. En caso de modificación, menor repercusión posible en el resto. + Reutilización - Dependencia

- **S**: Single Responsability Principle (SRP)
  - "Cada módulo de software debe tener una sola razón para cambiar." Cada clase debe tener un único trabajo/proposito.

- **O**: Open/Closed Principle (OCP)
  - Un módulo/clase está abierto para extensión y cerrado para modificación. Si queremos añadir funcionalidades nuevo lo ideal sería poder construir sobre lo que ya existe (sin modificaciones grandes). No alterar a menos que haya errores.
    Extensiones --> Interfaces/Clases abstractas -> Posibles arreglos para clases que las implementen

- **L**: Liskov substitution Principle (LSP)
  - "Si S es un subtipo de T, entonces los objetos de tipo T en un programa de computadora pueden ser sustituidos por objetos de tipo S (es decir, los objetos de tipo S pueden sustituir objetos de tipo T), sin alterar ninguna de las propiedades deseables de ese programa (la corrección, la tarea que realiza, etc.)" Wikipedia
  - Establece que debería poder usarse una clase derivada en lugar de la clase ppal y con mismo comportamiento sin modificaciones.
    Derivada no afecte comportamiento de la ppal. Clase derivada debe ser sustituible por su clase base.
  - Extensión de ppio OCP: Clases derivadas amplíen clases base sin modificar su comportamiento.

- **I**: Interface Segregation Principle (ISP)
  - "Los clientes no deben ser forzados a implementar interfaces que no usan. En lugar de una sola interfaz, se prefieren muchas interfaces pequeñas basadas en grupos de métodos (muy relacionados, alta cohesión), cada uno de los cuales sirve a un submódulo."
    Una interfaz debe estar más relacionada con el código que la usa (del cliente) que con el código que la implementa (la clase).
    Cada interfaz debe de tener un propósito/responsabilidad único (ppio SRP)

- **D**: Dependency Inversion Principle (DIP)
  - Los módulos de alto nivel no deben depender de módulos de bajo nivel (+ op. detalladas), si no de abstracciones.
  - Las abstracciones no deben depender de los detalles. Los detalles deben depender de las abstracciones.

- Patrones que ayuden a abstraer nuestros módulos.
- Inyeccion de dependencia (inversión de control).
- Evitar acoplamiento entre alto-bajo nivel(cambios de una clase pueden romper otra) --> Hacer que dependan de abstracciones

Objetivo ppios SOLID: Código ordenado, legible y fácil de mantener. El código puede ser que aumente de tamaño, pero será de mejor calidad.

Respetando estos principios se han desarrollado los patrones creacionales, estructurales y de comportamiento.