Principios:

**Agilidad arquitectural**: promueve diseñar sistemas de software de forma que puedan adaptarse fácilmente a los cambios

**Regla de los Boy scouts:** dejar el codigo más limpio de lo que lo encontraste

**Inversión de dependencias**: (la D de SOLID) los modulos de alto nivel no deben depender de modulos de bajo nivel, si no de abstracciones (interfaces). A su vez, las abstracciones no deben depender de detalles, los detalles deben depender de las abstracciones.

**Don’t Repeat Yourself**: evita la duplicación de codigo, lógica o información

**Encapsulación**: ocultar detalles internos y mostrar solamente lo necesario a través de una interfaz publica

**Dependencia explicita**: si un componente necesita algo para funcionar, esto debe ser claramente aclarado dentro de la clase

**Fail Fast**: el sistema debe identificar rápidamente los errores y fallar inmediatamente cuando algo no está bien

**Hollywood:** “no nos llames, nosotros te llamaremos”, está relacionado con la inversión de dependencias. En vez de que los modulos de bajo nivel llamen a los de alto nivel, los de alto nivel deben manejar el flujo y llamar a los pequeños solo cuando sea necesario

**Segregación de interfaces**: es mejor tener varias interfaces especificas y flacas que interfaces gordas y generales (la gordofobia está bien)

**Inversión de control**: en vez de que el codigo gestione cuando y como ejecutar una funcionalidad , un framework o contenedor externo lo hace por él.

**Keep it** simple: eso, mantener todo simple. No tiene sentido hacer algo mas complejo, mayor complejidad significa mayor probabilidad de error.

**Principio del menor asombro**: el software debe comportarse de manera predecible y lógica

**Sustitución de Lizkov**: los conceptos de una clase derivada pueden reemplazar a los de su clase base sin alterar el correcto funcionamiento del programa. Una clase hija debe cumplir las expectativas y comportamientos que definidos por la clase padre

**Una vez y solo una vez:** similar al DRY, cada parte del conocimiento o funcionalidad debe estar definida una sola vez en el código. Evita duplicar lógica, reglas o datos en varios sitios

**Abierto – cerrado:** las entidades del software deben estar abiertas para la extensión, pero cerradas para la modificación.

**Persistencia ignorancia**: un modulo debe conocer solamente lo necesario sobre otro modulo con el que interactúa.

**Separación de intereses:** cada modulo o componente debe encargarse de una sola cosa. Divide el sistema en partes que manejan aspectos específicos, evitando mezclar funcionalidades diferentes

**Responsabilidad** **singular**: Cada clase o componente debe tener una sola razón para cambiar

**SOLID**: es el conjunto de principios compuesto por:

* **SRP**: Single Responsability Principle // RESPONSABILIDAD SINGULAR
* **OCP**: Open/Closed Principle // ABIERTO/CERRADO
* **LSP**: Lizkov Substitution Principle // SUSTITUCION DE LISKOV
* **ISP**: Interface Segregation Principle // SEGREGACION DE INTERFACES
* **DIP**: Dependency Inversion Principle // INVERSION DE DEPENDENCIA

**Dependencias estables:** un modulo debe depender solamente de uno mas estable que él, o sea, componentes menos propensos a cambiar

**Tell, Don’t ask:** En lugar de pedir datos a un objeto y luego tomar decisiones, deberías decirle al objeto que hacer

**Tolerancia a la imperfección:** el sistema debe ser resiliente a errores menores, entradas incompletas o condiciones inesperadas. En lugar de fallar por completo, debe degradarse con gracias, ofrecer soluciones parciales o informar el problema sin detener todo

**YAGNI**: “You Aren’t Gonna Needed”, no lo vas a necesitar, no implementes algo hasta que sea realmente necesario, no agregar algo “por si acaso”.