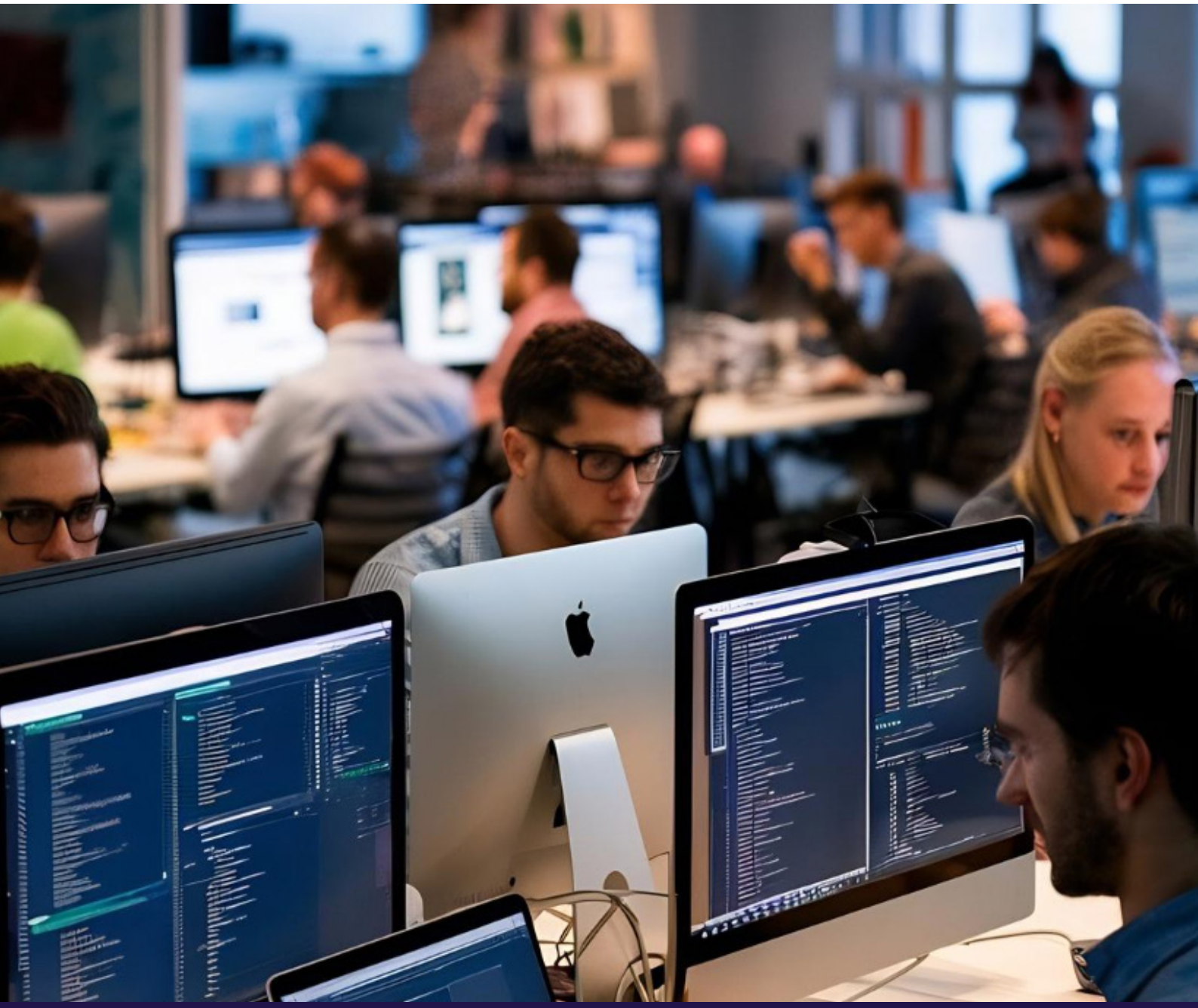
TECNOLÓGICA DEL ORIENTE
INSTITUCIÓN DE EDUCACIÓN SUPERIOR



FUNDAMENTOS DE *SOFTWARE*

PRINCIPIO DE ACOPLAMIENTO EN EL DISEÑO DE *SOFTWARE*

PRINCIPIO DE ACOPLAMIENTO EN EL DISEÑO DE *SOFTWARE*

El principio de acoplamiento en el diseño de ***software***, se refiere al grado de dependencia entre los módulos o componentes de un sistema. Un acoplamiento bien gestionado facilita el mantenimiento, la escalabilidad y la adaptación del sistema, a nuevos requerimientos. El objetivo es minimizar el acoplamiento, promoviendo un diseño modular y flexible.

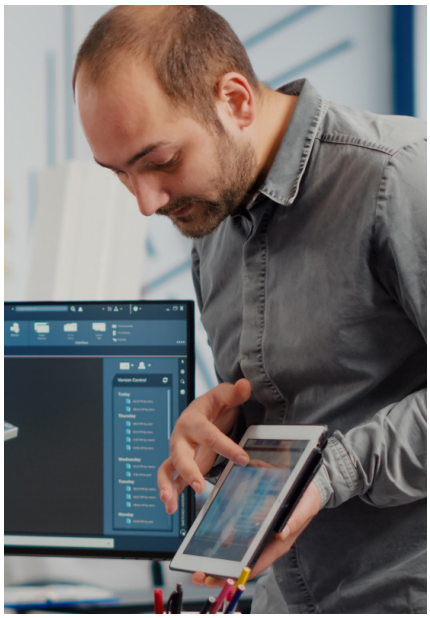
El acoplamiento mide la relación entre los módulos en términos de:

- **Interdependencia:** cuánta dependencia existe entre los módulos para que funcionen.
- Interacción: la cantidad y tipo de información que los módulos necesitan compartir.
- Cambio propagado: cómo los cambios en un módulo afectan a otros.

Un bajo acoplamiento, favorece la independencia funcional y define interacciones mínimas y bien estructuradas entre módulos.

Por qué es importante el acoplamiento en el diseño de *software*

- **Mantenibilidad:** los sistemas con bajo acoplamiento son más fáciles de actualizar y depurar, dado que los cambios en un módulo tienen un impacto limitado en otros.
- **Escalabilidad:** facilita la adición de nuevos módulos o funcionalidades, sin alterar profundamente la estructura del sistema.
- **Reutilización:** los módulos desacoplados son más reutilizables, debido a su menor dependencia de contextos específicos.
- **Pruebas unitarias:** un bajo acoplamiento simplifica las pruebas, permitiendo evaluar cada módulo de manera independiente.



Tipos de acoplamiento

Los niveles de acoplamiento describen diferentes grados de dependencia, ordenados de menor a mayor grado:

Acoplamiento nulo o desacoplamiento completo:

- No existe dependencia entre módulos.

• Ideal, aunque poco común en sistemas prácticos.

Ejemplo: dos módulos independientes que no necesitan interactuar.

Acoplamiento de datos:

• Los módulos comparten solo los datos necesarios, como parámetros específicos.

• Es el nivel más deseable en sistemas reales.

Ejemplo: una función que recibe argumentos de entrada y no depende de variables globales.

Acoplamiento de control:

• Un módulo controla el flujo de otro mediante señales o indicadores.

• Introduce cierta dependencia, pero aún manejable.

Ejemplo: una función que pasa un indicador booleano a otra para determinar una operación.

Acoplamiento externo o de interfaz:

• Los módulos dependen de un formato común para intercambiar información.

• La dependencia está en el nivel de protocolo o estándar de comunicación.

Ejemplo: módulos que se comunican mediante un servicio REST.

Acoplamiento común:

• Varios módulos comparten un recurso global, como una variable global.

• Puede generar errores difíciles de rastrear y es menos deseable.

Ejemplo: una base de datos compartida sin encapsulamiento adecuado.

Acoplamiento de contenido:

• Un módulo accede o modifica directamente el contenido interno de otro.

• Es el nivel más alto y menos deseable de asociación.

Ejemplo: una función que modifica directamente variables privadas de otra clase.

Buenas prácticas para reducir el acoplamiento

Encapsulación:

• Cada módulo debe manejar su propia lógica y datos internos.

• Limitar el acceso externo a través de métodos controlados.

Interfaces bien definidas:

• Diseñar contratos claros para la interacción entre módulos.

• Minimizar la cantidad de información intercambiada.

Descomposición modular:

• Dividir el sistema en módulos pequeños con responsabilidades específicas.

• Aplicar el principio de responsabilidad única.

Inyección de dependencias:

• Utilizar patrones como la inyección de dependencias para desacoplar módulos.

• Facilitar el cambio de implementaciones sin afectar a otros módulos.

Adopción de patrones de diseño:

• Usar patrones como observador, fábrica o inyección de dependencias para minimizar dependencias entre módulos.

Abstracción:

• Diseñar componentes que interactúen a través de abstracciones en lugar de implementaciones concretas.

Ejemplo: usar una interfaz en lugar de depender directamente de una clase específica.

Ejemplo práctico:

Mal diseño con alto acoplamiento: un módulo accede directamente a las variables internas de otro módulo y depende de su implementación específica. Un cambio en uno requiere modificar también al otro.

Buen diseño con bajo acoplamiento: un módulo se comunica con otro a través de una interfaz bien definida, intercambiando solo los datos necesarios. Esto permite modificar un módulo sin impactar al otro.

Beneficios de minimizar el acoplamiento

- Mayor flexibilidad para adaptarse a los cambios.
- Mejor modularidad, favoreciendo el trabajo en equipo y la integración continua.
- Reducción de errores derivados de dependencias ocultas.
- Mejor escalabilidad del sistema.