# Clase 7/5 - Tema Diseño

1- Observen el video que describe los aspectos generales del Diseño:

<https://www.youtube.com/watch?v=tKJeEXwZ1QA>

2- Lean el Flujo de Diseño del PUDS consideren Actividades, Artefactos y Trabajadores:

Apunte RUP.pdf (que está en el aula virtual- Unidad de Metodología)

3- Lean el material de la unidad 6 del Aula Virtual: Conceptos del Diseño 1, Conceptos del Diseño 2. Ingeniería de software Libro Un enfoque práctico Pressman Capitulo 8. <https://drive.google.com/drive/u/3/folders/12evhwMdcMXczyEPgqzpN9OUBe4vaGSoK>

**Actividad 1:**

Elija dos factores de calidad (Última filmina del materia “Conceptos del diseño 2”) y mencione un ejemplo de cada uno:

Por ej: ***Ejemplo de abstracción:*** *Cuando se desarrollan niveles de abstracción distintos, se trabaja para crear abstracciones tanto de procedimiento como de datos.*

***Una abstracción de procedimiento es una secuencia de instrucciones que tienen una función específica y limitada****. El nombre de la abstracción de procedimiento implica estas funciones, pero se omiten detalles específicos.* ***Un ejemplo de esto sería la palabra abrir, en el caso de una puerta. Abrir implica una secuencia larga de pasos del procedimiento (caminar hacia la puerta, llegar y tomar el picaporte, girar éste y jalar la puerta, retroceder para que la puerta se abra, etcétera).***

***Una abstracción de datos es un conjunto de éstos con nombre que describe a un objeto de datos.*** *Siguiendo con el ejemplo anterior puede definirse* ***una abstracción de datos llamada puerta.*** *Como cualquier objeto de datos,* ***la abstracción de datos para puerta agruparía un conjunto de atributos que describirían la puerta (tipo, dirección de apertura, mecanismo de apertura, peso, dimensiones, etc.).***

**Actividad 2:**

Principios SOLID:

Observen el video y realicen un resumen de los principios SOLID mencionando su utilidad. <https://www.youtube.com/watch?v=xrZ5KYw5_Zw>

**Actividad 3:**

Realicen un informe que compare lo que sugiere el PUDS, las consideraciones generales del Diseño teniendo en cuenta los atributos de calidad y los principios SOLID.

Fecha de entrega de las actividades 1, 2 y 3: 7 de Mayo 18:00 hs.

## Actividad 1

**Ejemplo de Modularidad:**

La modularidad en el desarrollo de software se refiere a la división de un sistema en partes o módulos que son más manejables, menos complejos y de manera independiente, lo que facilita tanto el desarrollo como el mantenimiento del software.

Un ejemplo claro de modularidad puede ser una aplicación de comercio electrónico. Esta aplicación se puede dividir en varios módulos como autenticación de usuarios, gestión de inventarios, procesamiento de pagos y generación de informes. Cada módulo maneja una parte específica de la aplicación: el módulo de autenticación gestiona el inicio de sesión y registro de usuarios, mientras que el módulo de procesamiento de pagos se encarga exclusivamente de todas las transacciones financieras. Esta división permite que diferentes equipos trabajen en módulos diferentes simultáneamente sin interferir entre ellos, y facilita la actualización de un módulo sin afectar a los demás.

**Ejemplo de Ocultamiento de la Información:**

El ocultamiento de la información es un principio de diseño de software que implica restringir el acceso a los detalles internos de los componentes del software, de modo que solo se expongan las interfaces necesarias para el uso de esos componentes. Esto ayuda a prevenir que el resto del sistema se vea afectado por cambios internos y mejora la seguridad.

Considerando el mismo ejemplo de la aplicación de comercio electrónico, el módulo de procesamiento de pagos podría implementar el ocultamiento de la información ocultando detalles específicos sobre cómo se procesan las transacciones financieras. Externo a este módulo, otros componentes de la aplicación solo interactúan con una interfaz bien definida para enviar y recibir pagos. Así, si se decide cambiar el proveedor de servicios de pago o modificar la lógica interna de procesamiento, estos cambios no afectarán a los otros módulos que dependen de él, siempre y cuando la interfaz se mantenga consistente.

## 

## Actividad 2

Los principios ***SOLID*** son fundamentales en el desarrollo de software orientado a objetos y ayudan a crear sistemas más comprensibles, flexibles y mantenibles.

* **Principio de Responsabilidad Única (Single Responsibility Principle - SRP):** Este principio afirma que cada módulo o clase debe tener responsabilidad sobre una única parte de la funcionalidad proporcionada por el software. Eso significa que la clase tendrá solo una razón para cambiar. Este principio es útil porque reduce la complejidad del módulo o clase, facilita su comprensión, y mejora su mantenibilidad.
* **Principio de Abierto/Cerrado (Open/Closed Principle - OCP):** Según este principio, las entidades de software (clases, módulos, funciones, etc.) deben estar abiertas para la extensión, pero cerradas para la modificación. Esto se logra utilizando abstracción y herencia, permitiendo que los usuarios añadan nuevas funcionalidades sin alterar el código existente. La utilidad de este principio radica en su capacidad para permitir que el software crezca y cambie con requisitos mínimos de reescritura.
* **Principio de Sustitución de Liskov (Liskov Substitution Principle - LSP):** Este principio establece que los objetos de una superclase deberían ser reemplazables con objetos de sus subclases sin afectar el funcionamiento del programa. LSP es útil para garantizar que una clase derivada mantenga el comportamiento esperado, promoviendo la correcta integración y uso de la herencia.
* **Principio de Segregación de la Interfaz (Interface Segregation Principle - ISP):** Este principio sugiere que varias interfaces específicas de cliente son mejores que una interfaz general. En otras palabras, no se debe obligar a un cliente a depender de interfaces que no utiliza. Esto hace que los módulos sean menos acoplados y más fáciles de reorganizar, modificar, y entender.
* **Principio de Inversión de Dependencias (Dependency Inversion Principle - DIP):** El principio dicta que los módulos de alto nivel no deben depender de los módulos de bajo nivel, sino que ambos deben depender de abstracciones. Además, las abstracciones no deben depender de los detalles; los detalles deben depender de las abstracciones. Este principio es útil porque reduce la dependencia directa entre los componentes del software, lo que facilita la escalabilidad y el mantenimiento.

Estos principios, cuando se implementan correctamente, hacen que el software sea más fácil de entender, mantener y extender, contribuyendo a la calidad y eficiencia del desarrollo de software.

## Actividad 3

**1. Proceso Unificado de Desarrollo de Software (PUDS):** El PUDS es un marco de desarrollo que enfatiza una metodología iterativa y adaptativa. Se centra en la arquitectura del software y en su adaptabilidad a los cambios durante el ciclo de desarrollo. Este proceso busca minimizar los riesgos mediante revisiones y ajustes regulares basados en el feedback y los cambios en los requisitos.

**2. Principios SOLID:** Los principios SOLID son un conjunto de reglas y recomendaciones para escribir software limpio y manejable:

* Single Responsibility Principle (Principio de Responsabilidad Única)
* Open/Closed Principle (Principio de Abierto/Cerrado)
* Liskov Substitution Principle (Principio de Sustitución de Liskov)
* Interface Segregation Principle (Principio de Segregación de la Interfaz)
* Dependency Inversion Principle (Principio de Inversión de Dependencia)

**Comparación y cómo se complementan:**

* **Adaptabilidad y Flexibilidad:** Tanto el PUDS como los principios SOLID y las consideraciones de diseño enfatizan la adaptabilidad y la facilidad para manejar cambios. Por ejemplo, el principio Open/Closed de SOLID propone que los software deben estar abiertos para la extensión, pero cerrados para la modificación.
* **Calidad y Revisión Continua:** El PUDS y las consideraciones generales del diseño sugieren iteraciones y revisiones continuas, lo que se alinea con la evaluación continua de la calidad y con principios como el de Responsabilidad Única de SOLID, que ayuda a mantener el código limpio y más fácil de revisar y mantener.
* **Abstracción y Modularidad:** La abstracción es crucial tanto en el PUDS como en los principios SOLID. La modularidad, un atributo de calidad interno, se refuerza con el principio de Segregación de la Interfaz y la Inversión de Dependencia de SOLID, promoviendo un diseño más limpio y organizado.