**Análisis y diseño de software**

* Analysis:

Estudio y separación de un problema para su entendimiento.

En software hace mención a la toma de requerimientos, no a la solución del problema ni a su implementación.

* Diseño:

Solución conceptual que satisface los requerimientos.

“Definición de la estructura, consistente en módulos que conjuntamente trabajan para proveer la conducta y satisfacer los requerimientos de rendimiento.”

En el diseño orientado a la programación orientada a objetos lo más común es el uso de UML.

Para mejorar el diseño, se debe de tomar en consideración:

1. Principios de diseño.
2. Heurística.
3. Considerar el rendimiento.

Los malos diseños de software tiene alguna de las siguientes 4 características:

* **Fragilidad**: Un cambio en el software desencadena cambios en cadena (se cae por el mínimo cambio).
* **Rigidez**: Dificultad para aceptar cambios.
* **Inmovilidad**: incapacidad de la reutilización del software.
* **Viscosidad**: Cambios en el software resultan muy complicados.

Con tal de evitar un mal diseño en un producto de software aplique lo siguiente:

1. Inicio del diseño.
   1. Implementar los principios de diseño.
      1. Principios fundamentales
      2. Principios prácticos.
2. En su diseño.
   1. Uso de patrones.
3. En la implementación.
   1. **Testing**: Probar el código mientras lo desarrolla(TDD).
4. En producción (released).
   1. **Refactoring**: mejorar el código ya existente.

**Principios de diseño**

1. **Principios fundamentales**: Teoría basada en la orientación a objetos.
2. **Principios prácticos**: Buenas prácticas.

* **Open close Principle**: Todo módulo debe de estar abierto para ser extendido, pero cerrado para ser modificado.
* **Ley demeter**: Todo método de objeto debe llamar solo métodos que:

1. Son propios.
2. Pertenecen a algún objeto que le fue enviado como parámetro.
3. Pertenece a un objeto creado dentro de él.
4. Pertenece a una de las instancias que pertenecen al método.

* **Sustitución de liskov**: Cualquier instancia de la subclase puede sustituir a una instancia de la clase.
* **Principio de inversión de la dependencia**: Dependa de las abstracciones, no de las implementaciones de ellas.
* **Principio de segregación de interfaces**: Tener muchas interfaces funcionales específicas es mejor que tener una sola interfaz para todos.

**Principios pragmáticos**

Serie de acrónimos que sintetizan los principios fundamentales en ideas prácticas.

* Dry: Don't repeat yourself
  + Estás perdiendo el tiempo, repitiendo código en vez de crear un proceso que lo haga automático.
* KISS: Keep it simple stupid
  + Concentrarse en codificar solo lo necesario para cumplir un requerimiento con un buen rendimiento, no realizar una implementación viscosa y rígida.
* SOLID:
  + Single Responsibility: Cada clase debe de tener una única responsabilidad al igual que sus métodos.
  + Open close responsibility: A las clases o módulos existentes, se les pueden agregar cosas, pero no modificar las ya existentes.
  + Liskov substitution principle: Donde quiera que una clase A pueda ser contenida o referenciada, una subclase de A también puede ser referenciada y contenida.
  + Interface segregation: Factorizar conductas en tantas interfaces como sea posible.
  + Dependency Injection: evitar amarrarse directamente a instancias específicas a través del operador NEW.
* YAGNI: You ain’t gonna need it
  + No crear cosas innecesarias, pensar solo en lo que se ocupa para cumplir el requerimiento.

**Introducción a patrones**

* **Historia**:

Christopher Alexander a finales de los 60's observó que en varias partes del mundo en donde había edificios exitosos y hermosos habían ciertas soluciones estructurales que se repetían para cumplir un objetivo. Debido a esto él definió los patrones, los cuales deben de definir lo siguiente:

1. Un nombre.
2. Propósito del patrón (qué problema resuelve).
3. Como conseguir aplicar el patrón (descripción de lo que se ocupa hacer).
4. Limitaciones y fuerzas a tomar en consideración.

Debido al éxito del libro publicado por Alexander, un grupo de programadores, cuestionaron si esto pasaba a la hora de crear software, por lo cual en el año de 1995, nace

el famoso libro GOF "Design patterns: Elements of reusable OO Software", el cual aplica y enseña 23 patrones basados en la orientación a objetos.

* **Patrón de diseño**:

Un patrón es una solución ya creada para resolver un problema existente. Los patrones describen el problema y la esencia de una solución a este.

Para especificar un patrón de diseño se debe de detallar:

1. **Nombre**: Nombre del patrón.
2. **Propósito (Intent)**: Problema que resuelve.
3. **Problema**: Definición del problema a resolver.
4. **Solución**: Como el patrón provee una solución.
5. **Participantes y colaboradores**: Entidades que participan en el patrón.
6. **Consecuencias**: relaciones de causa efecto al usar el patrón.
7. **Implementación**: Guia de como implementar el patrón.