

Aquí tienes un resumen del texto en formato Markdown, estructurado para una mejor legibilidad:

Resumen: Principios y Autores del Desarrollo de Software

Programación Web 2 - UNLaM - 2020 (2do Cuatrimestre)

Este documento busca introducir conceptos y autores clave en la industria del software, con el objetivo de **despertar la curiosidad** y **motivar la investigación** en profundidad. Se presentan técnicas y principios alineados con el **Manifiesto ágil**, forjados por sus firmantes y referentes como **Kent Beck (TDD)**, **Robert C. Martin (SOLID, Clean Code)**, **Jeff Sutherland (SCRUM)** y **Martin Fowler**.

Todos estos principios se alinean buscando los mismos objetivos, reflejando los valores del Manifiesto ágil:

- Individuos e interacciones sobre procesos y herramientas
- Software funcionando sobre documentación extensiva
- Colaboración con el cliente sobre negociación contractual
- Respuesta ante el cambio sobre seguir un plan

Principios y Prácticas Clave

Clean Code (Robert C. Martin)

Este libro enfatiza la importancia de escribir código que sea **limpio y funcional**. Algunos puntos clave incluyen:

- **Legibilidad:** Priorizar código fácil de leer, similar a un diario o libro.
- **Code Smells:** Identificar y corregir código de mala calidad que indica problemas de diseño.
- **Facilidad de Cambio:** El software debe ser adaptable; aplicar la **Regla del Boy Scout** (dejar el código más limpio de lo que se encontró).
- **Nomenclatura:**
 - Variables y funciones: Nombres representativos.
 - Objetos: Sustantivos.
 - Métodos: Verbos.
- **Funciones:** Deben hacer **una sola cosa** y hacerla bien; idealmente 0-2 parámetros (máximo).
- **Comentarios:** Evitar comentarios innecesarios; el código limpio se auto-documenta.
- **Refactorización y Testing:** La **TDD (Desarrollo Guiado por Pruebas)** es esencial para permitir refactorizaciones rápidas y sin miedo a romper el código.

DRY: Don't Repeat Yourself

Este principio aboga por la **reducción de la duplicación** en el desarrollo de software.

- Toda "pieza de información" (datos, código fuente, documentación) no debe duplicarse.

- La duplicaci#n dificulta los cambios, perjudica la claridad y puede causar inconsistencias.
- Aplicar DRY eficientemente significa que los cambios se realizan en un **#nico lugar**.

KISS: Keep It Simple, Stupid!

Establece que los sistemas funcionan mejor si se mantienen **simples**. La simplicidad debe ser un objetivo clave del dise#o, evitando complejidades innecesarias.

YAGNI: You Aren't Gonna Need It

Consiste en **no agregar funcionalidades innecesarias o no solicitadas**.

- Evita la tentaci#n de escribir c#digo para futuros escenarios hipot#ticos.
- Prioriza las funcionalidades b#sicas, ahorrando tiempo en depuraci#n, documentaci#n y soporte de caracter#sticas no esenciales.

SLAP: Single Level of Abstraction Principle

Enfocado en la **abstracci#n**, propone dividir el programa en funciones o m#todos que:

- Tengan una **#nica responsabilidad**.
- Contengan pocas l#neas de c#digo.
- Operen en un **#nico nivel de abstracci#n**.
- Si es necesario, se pueden usar funciones compuestas que llamen a otras funciones privadas con nombres claros.

CQS: Command and Query Separation (Bertrand Meyer)

Este principio de la programaci#n orientada a objetos afirma que cada m#todo debe ser:

- Un **comando** que realiza una acci#n (con efectos secundarios).
- O una **consulta** que devuelve datos (sin efectos secundarios).
- En otras palabras, **hacer una pregunta no debe cambiar la respuesta**; las consultas deben ser referencialmente transparentes.

TDD: Test-Driven Development (Kent Beck)

Es una pr#ctica de ingenier#a de software que integra:

- **Escribir las pruebas primero (Test First Development)**.
- **Refactorizaci#n**.
- **Ciclo de trabajo**: Escribir una prueba que falla -> Implementar c#digo para que pase -> Refactorizar el c#digo.
- **Prop#sito**: Lograr **c#digo limpio y funcional**, asegurando que el software cumple con los requisitos establecidos.

SOLID (Robert C. Martin)

Un acr#nimo que representa **cinco principios b#sicos del dise#o y la programaci#n orientada a objetos**, cuyo objetivo es crear sistemas f#ciles de mantener y ampliar. Son gu#as para eliminar malos dise#os y refactorizar c#digo.

- **S - Single-responsibility Principle (Principio de responsabilidad #nica):**

- Cada m#dulo o clase debe ser responsable de **una #nica cosa** (un #nico motivo de cambio).
- **O - Open-closed Principle (Principio de abierto/cerrado):**
 - Las clases, m#dulos o funciones deben estar **abiertas para su extensi#n, pero cerradas para su modificaci#n**. Se debe poder extender el comportamiento sin alterar el c#digo existente.
- **L - Liskov Substitution Principle (Principio de sustituci#n de Liskov):**
 - Una clase **debe ser sustituible por su clase padre**. La clase heredada debe complementar el comportamiento de la base, no reemplazarlo.
- **I - Interface Segregation Principle (Principio de segregaci#n de la interfaz):**
 - Ning#n cliente debe depender de m#todos que no utiliza. Es preferible tener **muchas interfaces peque#as** a una grande con m#todos no usados.
- **D - Dependency Inversion Principle (Principio de inversi#n de la dependencia):**
 - Las dependencias deben recaer sobre **abstracciones (interfaces)**, no sobre clases concretas (implementaciones).