Aqu# tienes el resumen del texto en formato Markdown, estructurado y legible:

Programaci#n web 2 - UNLaM - 2020 2do cuatrimestre

Conociendo principios y autores del desarrollo de software

Este documento tiene como objetivo **introducir conceptos y autores clave** en la industria del software, con la finalidad de **despertar la curiosidad** y motivar la investigaci#n en profundidad de los estudiantes. Los principios presentados est#n alineados con el **Manifiesto #gil**.

Autores Referentes y Manifiesto #gil

Los principios discutidos provienen principalmente de las personas que firmaron el manifiesto #gil y que buscan un objetivo com#n, haciendo referencia constante entre ellos. Algunos de los autores destacados son:

- **Kent Beck**: Conocido por TDD.
- **Robert C. Martin (Uncle Bob)**: Referente en SOLID y Clean Code.
- **Jeff Sutherland**: Co-creador de SCRUM.
- **Martin Fowler**: Reconocido experto en desarrollo de software.

El **Manifiesto #gil** enfatiza:

- Individuos e interacciones sobre procesos y herramientas
- Software funcionando sobre documentaci#n extensiva
- Colaboraci#n con el cliente sobre negociaci#n contractual
- Respuesta ante el cambio sobre seguir un plan

Principios y T#cnicas de Desarrollo de Software

A continuaci#n, se detallan los principios y t#cnicas fundamentales para un desarrollo de software robusto y eficiente.

CLEAN CODE (Robert C. Martin)

Este libro ofrece directrices para escribir c#digo de alta calidad:

- **Objetivo**: C#digo limpio y funcional.
- **Legibilidad**: Priorizar la lectura fluida, similar a un peri#dico.
- **Code Smells**: Identificar y corregir "olores" de c#digo de mala calidad (poco legible o escalable).
- **Flexibilidad**: El software debe ser f#cil de cambiar.
- **Regla del Boy Scout**: Dejar el c#digo m#s limpio de como se encontr#.
- **Nombres**: Variables y funciones deben ser representativas; objetos sustantivos, m#todos verbos.
- **Funciones**: Deben hacer una sola cosa y hacerla bien; idealmente 0-2 par#metros (nunca 3 o m#s).
- **Comentarios**: Evitarlos; el buen c#digo no necesita explicaciones.
- **Confianza**: No temer cambiar el c#digo.
- **Refactoring**: Solo es posible con una buena suite de tests, preferiblemente escritos con **TDD**.

DRY: Don't Repeat Yourself

- **Filosof#a**: Reducir la duplicaci#n de informaci#n.

- **Principio**: Toda "pieza de informaci#n" debe existir una #nica vez.
- **Alcance**: Aplica a datos, c#digo fuente y documentaci#n.
- **Beneficio**: Facilita cambios y evoluci#n, mejora la claridad y previene inconsistencias.

KISS: Keep It Simple, Stupid!

- **Premisa**: Los sistemas funcionan mejor si se mantienen simples.
- **Meta**: La simplicidad debe ser un objetivo clave del dise#o, evitando complejidades innecesarias.

YAGNI: You Aren?t Gonna Need It

- **Concepto**: No agregar funcionalidades innecesarias o no solicitadas.
- **Justificaci#n**: Escribir c#digo que "quiz#s se necesite" en el futuro sacrifica tiempo y recursos que podr#an usarse en funcionalidades b#sicas, adem#s de implicar debugging, documentaci#n y soporte extra.

SLAP: Single Level of Abstraction Principle

- **Base**: El c#digo se construye sobre abstracciones, ocultando detalles de bajo nivel.
- **Pr#ctica**: Dividir el programa en funciones o m#todos que tengan:
  - Una #nica responsabilidad.
  - Pocas l#neas de c#digo.
  - Un #nico nivel de abstracci#n.
- **Estructura**: Utilizar funciones compuestas que llamen a otras funciones privadas con nombres claros.

CQS: Command and Query Separation (Bertrand Meyer)

- **Principio**: Cada m#todo debe ser exclusivamente:
  - Un **comando**: Realiza una acci#n y/o modifica el estado.
  - Una **consulta**: Devuelve datos sin producir efectos secundarios (no cambia el estado).
- **Idea clave**: "Hacer una pregunta no debe cambiar la respuesta".

TDD: Test-driven development (Kent Beck)

- **Pr#ctica de Ingenier#a**: Combina "Escribir las pruebas primero" (Test First Development) y "Refactorizaci#n".
- **Ciclo de Desarrollo (Red-Green-Refactor)**:
  1. Escribir una **prueba** y verificar que **falle** (rojo).
  2. Implementar el **c#digo m#nimo** necesario para que la prueba **pase** (verde).
  3. **Refactorizar** el c#digo, manteniendo la funcionalidad garantizada por las pruebas.
- **Prop#sito**: Lograr un c#digo limpio y funcional, asegurando que el software cumpla con los requisitos.

SOLID (Robert C. Martin)

SOLID es un acr#nimo de cinco principios fundamentales de la Programaci#n Orientada a Objetos (POO) que, aplicados en conjunto, promueven un dise#o de software **f#cil de mantener y extender**. Es parte del desarrollo #gil.

- **S - Single-responsibility Principle (Principio de responsabilidad #nica)**
  - Una clase o m#dulo debe tener **una #nica raz#n para cambiar**.
- **O - Open-closed Principle (Principio de abierto/cerrado)**

    - Las clases deben estar **abiertas para su extensi#n**, pero **cerradas para su modificaci#n**. Se debe poder a#adir funcionalidad sin cambiar el c#digo existente.

- **L - Liskov Substitution Principle (Principio de sustituci#n de Liskov)**

    - Un objeto de una clase base debe poder ser **sustituido por un objeto de una de sus clases derivadas** sin alterar el comportamiento correcto del programa. La clase heredada debe complementar, no reemplazar, el comportamiento base.

- **I - Interface Segregation Principle (Principio de segregaci#n de la interfaz)**

    - **Ning#n cliente debe verse obligado a depender de m#todos que no utiliza**. Es mejor tener muchas interfaces peque#as y espec#ficas que una interfaz grande y monol#tica.

- **D - Dependency Inversion Principle (Principio de inversi#n de la dependencia)**

    - Las dependencias deben recaer sobre **abstracciones (interfaces)**, no sobre **clases concretas (implementaciones)**. Los m#dulos de alto nivel no deben depender de m#dulos de bajo nivel, ambos deben depender de abstracciones.