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

Este documento introduce conceptos y autores clave en el desarrollo de software, buscando fomentar la curiosidad y la investigaci#n en los estudiantes. Se centra en t#cnicas y principios alineados con el **Manifiesto #gil**.

Introducci#n y Fundamentos del Desarrollo #gil

El texto busca presentar a referentes de la industria del software, motivando a investigar m#s a fondo sus contribuciones. Los principios expuestos est#n alineados al **Manifiesto #gil**, y sus autores principales son figuras destacadas que lo firmaron o son sus promotores:

- **Kent Beck** (TDD)
- **Robert C. Martin** (SOLID, Clean Code)
- **Jeff Sutherland** (SCRUM)
- **Martin Fowler** Estos autores, aunque diversos, comparten el mismo objetivo y se referencian mutuamente, bas#ndose en los valores del Manifiesto #gil:
- **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 de Desarrollo de Software

A continuaci#n, se detallan varios principios fundamentales para un desarrollo de software robusto y mantenible:

CLEAN CODE (Robert C. Martin)

Consejos clave de este principio para un c#digo de calidad:

- El objetivo es un **c#digo limpio y funcional**.
- **Priorizar la legibilidad**, buscando una lectura similar a la de un diario o libro.
- El c#digo de mala calidad genera "olor a podrido" (**Code Smell**), que debe ser detectado y corregido.
- El software debe ser **f#cil de cambiar**.
- Aplicar la **Regla del Boy Scout**: Dejar el c#digo m#s limpio de lo que se encontr#.
- Los **nombres** de variables y funciones deben ser representativos y claros.
- Las **funciones** deben hacer *una sola cosa y hacerla bien*.
- Las funciones no deben recibir m#s de **2 par#metros** (0 es ideal, 3 no es deseable).
- Los **comentarios no son buenos**; el buen c#digo se documenta a s# mismo.
- Los nombres de los **objetos** deben ser sustantivos, y los de los **m#todos** verbos.
- El objetivo es **no tener miedo de cambiar el c#digo**.
- La #nica forma de lograr refactorizaciones r#pidas y seguras es con una **buena bater#a de tests**, idealmente escritos con **TDD**.

DRY: Don't Repeat Yourself

- **Principio:** "No te repitas" o "Una vez y s#lo una".
- **Objetivo:** Reducir la duplicaci#n de cualquier "pieza de informaci#n" (datos, c#digo fuente, documentaci#n).

- **Beneficio:** Facilita cambios y evoluci#n, mejora la claridad y previene inconsistencias. Los cambios deben requerir modificaciones en un #nico lugar.

KISS: Keep It Simple, Stupid!

- **Principio:** "#Mantenlo sencillo, est#pido!".
- **Objetivo:** La simplicidad debe ser una meta clave en el dise#o, evitando cualquier complejidad innecesaria. Los sistemas funcionan mejor si se mantienen simples.

YAGNI: You Aren?t Gonna Need It

- **Principio:** "No vas a necesitarlo".
- **Objetivo:** No agregar funcionalidad innecesaria o no solicitada.
- **Raz#n:** Implementar funcionalidades hipot#ticas sacrifica tiempo, y toda caracter#stica nueva requiere depuraci#n, documentaci#n y soporte.

SLAP: Single Level of Abstraction Principle

- **Principio:** "Principio de Nivel #nico de Abstracci#n".
- **Objetivo:** Escribir c#digo con funciones o m#todos que tengan una **#nica responsabilidad**, pocas l#neas y un solo nivel de abstracci#n.
- **Implementaci#n:** Usar nombres claros para funciones y m#todos, y componerlos a partir de llamadas a otros m#todos m#s espec#ficos si es necesario.

CQS: Command and Query Separation (Bertrand Meyer)

- **Principio:** "Separaci#n de comandos y consultas".
- **Regla:** Cada m#todo debe ser un **comando** (realiza una acci#n) o una **consulta** (devuelve datos), pero *nunca ambos*.
- **Implicaci#n:** Hacer una pregunta (consulta) no debe cambiar la respuesta; los m#todos de consulta no deben tener efectos colaterales.

TDD: Test-driven Development (Kent Beck)

- **Pr#ctica:** "Desarrollo guiado por pruebas de software".
- **Proceso:**
    1. Escribir una prueba y verificar que **falla**.
    2. Implementar el c#digo m#nimo para que la prueba **pase**.
    3. **Refactorizar** el c#digo escrito.
- **Prop#sito:** Lograr un **c#digo limpio que funcione**, garantizando que el software cumple con los requisitos al pasar las pruebas.

SOLID (Robert C. Martin)

SOLID es un acr#nimo mnem#nico para cinco principios b#sicos de la programaci#n orientada a objetos (POO) y el dise#o, introducidos por **Robert C. Martin**. Aplicarlos en conjunto facilita la creaci#n de sistemas mantenibles y extensibles, ayudando a refactorizar el c#digo para que sea legible y escalable, y formando parte de estrategias como el desarrollo #gil.

1. **S - Single-responsibility Principle (SRP): Principio de Responsabilidad #nica**

    - **Concepto:** Cada m#dulo o clase debe ser responsable de **una #nica cosa**. Esto implica que debe tener solo un motivo para cambiar.

2. **O - Open-closed Principle (OCP): Principio de Abierto/Cerrado**

- **Concepto:** Las clases, m#dulos o funciones deben estar **abiertas para su extensi#n**, pero **cerradas para su modificaci#n**. Se busca extender el comportamiento sin alterar el c#digo existente.

3. **L - Liskov Substitution Principle (LSP): Principio de Sustituci#n de Liskov**

    - **Concepto:** Una clase base debe ser **sustituible por su clase heredada**. La clase derivada debe complementar el comportamiento de la base, no reemplazarlo de forma incompatible.

4. **I - Interface Segregation Principle (ISP): Principio de Segregaci#n de la Interfaz**

    - **Concepto:** Ning#n cliente debe verse obligado a depender de m#todos que no utiliza. Es preferible tener **muchas interfaces peque#as y espec#ficas** que una interfaz grande con muchos m#todos no relevantes para todos los clientes.

5. **D - Dependency Inversion Principle (DIP): Principio de Inversi#n de la Dependencia**

    - **Concepto:** Las dependencias deben recaer sobre **abstracciones (interfaces)**, no sobre clases concretas (implementaciones). Esto desacopla los m#dulos, permitiendo mayor flexibilidad.