Aquí tienes un resumen del texto en formato Markdown:

# Programación Web 2 - UNLaM - 2020 2do Cuatrimestre

## Conociendo Principios y Autores del Desarrollo de Software

El documento tiene como objetivo principal introducir conceptos y autores clave en la industria del software para fomentar la investigación y el aprendizaje profundo. Los principios presentados están alineados con el **Manifiesto Ágil**.

### Autores Referentes y Manifiesto Ágil

Se mencionan autores como **Kent Beck** (TDD), **Robert C. Martin** (SOLID, Clean Code), **Jeff Sutherland** (SCRUM) y **Martin Fowler**. Todos estos autores se alinean y referencian mutuamente, buscando los mismos objetivos que el 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 Clave y Metodologías

### 1. CLEAN CODE (Robert C. Martin)

Consejos fundamentales para escribir código limpio:

- **Objetivo:** Código limpio y funcional.
- **Legibilidad:** El código debe leerse como un diario o un libro.
- **Code Smells:** El código de mala calidad genera "olor a podrido", y existen principios para detectarlos y corregirlos.
- **Flexibilidad:** El software debe ser fácil de cambiar.
- **Regla del Boy Scout:** Dejar el código más limpio de lo que se encontró.
- **Nombres:** Variables y funciones deben ser representativos; objetos sustantivos, métodos verbos.
- **Funciones:**
    - Deben hacer una sola cosa y hacerla bien.
    - No deberían recibir más de 2 parámetros (0 es mejor, 3 no es deseable).
- **Comentarios:** El código bueno no necesita comentarios.
- **Seguridad:** No tener miedo de cambiar el código.
- **Refactorización:** La forma más efectiva es con una buena batería de tests, idealmente escritos con **TDD**.

### 2. DRY (Don't Repeat Yourself)

- **Principio:** Reducir la duplicación de información.
- **Objetivo:** Facilita los cambios, la evolución y mejora la claridad, evitando inconsistencias.
- **Alcance:** Aplica a datos, código fuente y documentación.
- **Beneficio:** Los cambios se realizan en un único lugar.

## 3. KISS (Keep It Simple, Stupid!)

- **Principio:** Los sistemas funcionan mejor si se mantienen simples.
- **Objetivo:** La simplicidad debe ser una meta clave en el diseño, evitando la complejidad innecesaria.

## 4. YAGNI (You Aren?t Gonna Need It)

- **Principio:** No añadir funcionalidades innecesarias o no solicitadas.
- **Razón:** La funcionalidad no esencial consume tiempo que podría usarse en lo básico, y requiere depuración, documentación y soporte.

## 5. SLAP (Single Level of Abstraction Principle)

- **Principio:** Dividir el programa en funciones o métodos que tengan una única responsabilidad, pocas líneas de código y un solo nivel de abstracción.
- **Objetivo:** Cada función debe hacer una cosa y hacerla bien, ocultando los detalles de bajo nivel.
- **Implementación:** Usar funciones o métodos compuestos con nombres claros que identifiquen su responsabilidad.

## 6. CQS (Command and Query Separation) - Bertrand Meyer

- **Principio:** Cada método debe ser un **comando** (realiza una acción) o una **consulta** (devuelve datos), pero no ambos.
- **Regla:** Hacer una pregunta (consulta) no debe cambiar la respuesta. Las consultas deben ser referencialmente transparentes y sin efectos colaterales.

## 7. TDD (Test-Driven Development) - Kent Beck

- **Práctica:** Desarrollo guiado por pruebas que involucra:
  1. **Escribir la prueba primero:** Se escribe una prueba y se verifica que falla.
  2. **Implementar el código:** Se escribe el código mínimo para que la prueba pase.
  3. **Refactorización:** Se refactoriza el código.
- **Objetivo:** Lograr un **código limpio que funcione** y asegurar que el software cumpla con los requisitos establecidos, ya que estos se traducen en pruebas.

## 8. SOLID (Robert C. Martin)

Acrónimo de cinco principios fundamentales de la programación orientada a objetos (POO) y el diseño, cuyo objetivo es crear sistemas fáciles de mantener y ampliar. Son guías para eliminar malos diseños y son parte de las estrategias **ágiles** y **TDD**.

- **S - Single-responsibility Principle (Principio de responsabilidad única):**
  - Cada módulo o clase debe tener una única responsabilidad, la cual debe estar completamente encapsulada (un solo motivo de cambio).
- **O - Open-closed Principle (Principio de abierto/cerrado):**
  - Las entidades de software (clases, módulos, funciones) deben estar **abiertas a la extensión** pero **cerradas a la modificación**.
- **L - Liskov Substitution Principle (Principio de sustitución de Liskov):**

    - Una clase base debe poder ser sustituida por sus clases derivadas sin alterar la corrección del programa. La subclase debe complementar, no reemplazar, el comportamiento de la clase base.

- **I - Interface Segregation Principle (Principio de segregación de la interfaz):**

    - Ningún cliente debe ser forzado a depender de interfaces que no utiliza. Es preferible tener muchas interfaces pequeñas y específicas a una interfaz grande y monolítica.

- **D - Dependency Inversion Principle (Principio de inversión de la dependencia):**

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