|  |  |  |
| --- | --- | --- |
| Control de cambios: | | |
| Revisión | Fecha | Observaciones |
| 00 | 09/04/2025 | Documento inicial |

ÍNDICE

[1 Objeto 3](#__RefHeading___Toc1514_3823752326)

[2 Campo de aplicación 3](#__RefHeading___Toc1516_3823752326)

[3 Definiciones 3](#__RefHeading___Toc1637_3616825104)

[4 Responsabilidades 3](#__RefHeading___Toc1639_3616825104)

[4.1 Equipo de desarrollo 3](#__RefHeading___Toc1522_3823752326)

[5 Instrucciones 4](#__RefHeading___Toc1671_3823752326)

[5.1 Nombres significativos 4](#__RefHeading___Toc1673_3823752326)

[5.2 Funciones limpias 4](#__RefHeading___Toc1673_3823752326_Copia_)

[5.3 Comentarios útiles (y escasos) 5](#__RefHeading___Toc1673_3823752326_Copia1)

[5.4 Código sin duplicaciones 5](#__RefHeading___Toc1673_3823752326_Copia2)

[5.5 Código expresivo 5](#__RefHeading___Toc1673_3823752326_Copia3)

[5.6 Clases cohesionadas 6](#__RefHeading___Toc1616_3616825104)

[5.7 Formato y estilo coherente 6](#__RefHeading___Toc1673_3823752326_Copia4)

[5.8 Código testable 6](#__RefHeading___Toc1673_3823752326_Copia5)

[Anexo 1: Reglas prácticas de código limpio 8](#__RefHeading___Toc1562_3823752326)

[Anexo 2. Recursos 9](#__RefHeading___Toc1603_3823752326)

# Objeto

Establecer las buenas prácticas de desarrollo recomendadas por el libro “Clean Code: A Handbook of Agile Software Craftsmanship” de Robert C. Martin (Uncle Bob), para que el equipo de desarrollo escriba código profesional, mantenible, legible y escalable.

# Campo de aplicación

Este documento se aplica a todo el equipo de desarrollo de software, en cualquier proyecto que use cualquier lenguaje orientado a objetos.

# Definiciones

* **Código limpio (Clean Code):** Código legible, claro, simple y elegante. Está estructurado para facilitar su comprensión, modificación y mantenimiento.
* **Refactorización:** Proceso de modificar el código para mejorar su estructura sin cambiar su funcionalidad.
* **Responsabilidad única:** Principio según el cual cada clase debe tener una única funcionalidad .
* **Testabilidad:** Facilidad con la que el código puede ser probado de forma automática.

# Responsabilidades

## Equipo de desarrollo

* **Aplicar buenas prácticas en el código fuente**
* **Revisar el código de compañeros asegurando su limpieza y coherencia**
* **Refactorizar** código sucio cuando se detecte durante el desarrollo o revisión

# Instrucciones

## Nombres significativos

* Usa nombres claros y descriptivos para variables, funciones, clases y archivos.
* Evita abreviaturas, siglas ambiguas o palabras genéricas como data, info, temp...

### Ejemplo

### Regla

// ❌ Mal  
$a = calculate($d);

// ✅ Bien  
$invoiceTotal = calculateTotalWithDiscount($invoiceData);

*Si tienes que explicarlo, no está bien nombrado.*

## Funciones limpias

* Deben ser pequeñas y hacer una sola cosa.
* El nombre debe explicar lo que hace.
* Deben tener pocos parámetros (idealmente 0-2).

### Ejemplo

### Regla

// ✅ Función clara y única responsabilidad  
function sendWelcomeEmail(User $user): void {  
 Mail::to($user->email)->send(new WelcomeEmail());  
}

*Si necesitas comentarios divídela en funciones más pequeñas.*

## Comentarios útiles (y escasos)

* El código bien escrito no necesita comentarios.
* Usar comentarios para:
  + Explicar reglas de negocio complejas
  + Advertencias o limitaciones

### Ejemplo

// ✅ Comentario útil  
// Esta fórmula se ajusta a la legislación fiscal de 2024  
$iva = $importe \* 0.21;

### Regla

*Si tienes que añadir comentarios piensa en refactorizar*

## Código sin duplicaciones

* Extraer lógica común en funciones o clases.
* Usar constantes o helpers si repites valores.

### Ejemplo

## Código expresivo

// ❌ Repetición  
if ($user->role === 'admin') { ... }  
if ($user->role === 'admin') { ... }

// ✅ Centralizado  
function isAdmin(User $user): bool {  
 return $user->role === 'admin';  
}

Usar estructuras claras, nombres bien elegidos y evitar construcciones confusas.

### Ejemplo

## Clases cohesionadas

// ❌ Mal  
if (count($a) > 0) { ... }

// ✅ Bien  
if ($hasPendingOrders) { ... }

Las clases deben tener un único propósito. (Single Responsability Principle)

### Ejemplo

## Formato y estilo coherente

// ❌ Clase con varias responsabilidades  
class UserManager {  
 public function createUser() {}  
 public function sendEmail() {}  
 public function logActivity() {}  
}

// ✅ Separadas  
class UserCreator { ... }  
class EmailService { ... }  
class ActivityLogger { ... }

Usar sangrías claras, agrupación lógica, espacios adecuados y orden de métodos consistente (propiedades, constructor, métodos públicos, privados...).

## Código testable

El código debe facilitar los tests. Separar lógica de infraestructura, evitar instancias directas (*new*) y usar inyección de dependencias.

### Ejemplo

// ✅ Correcto  
public function \_\_construct(InvoiceRepository $repo) {  
 $this->repo = $repo;  
}

# ****Anexo 1: Reglas prácticas de código limpio****

| Regla | Aplicación práctica |
| --- | --- |
| Nombres claros | getCustomerAge() mejor que getData() |
| Funciones pequeñas | Idealmente < 10 líneas |
| Sin duplicación | Extraer funciones comunes |
| Comentarios mínimos | Solo si son estrictamente necesarios |
| Estructura uniforme | Mismo estilo en todo el equipo |
| Responsabilidad única | Cada clase/método hace una sola cosa |
| Código fácilmente testeable | Usa inyección de dependencias, evita new innecesarios |

# Anexo 2. Recursos

* **Libro: Clean Code, Robert C. Martin**
* **Estándar PSR-12: https://www.php-fig.org/psr/psr-12/**