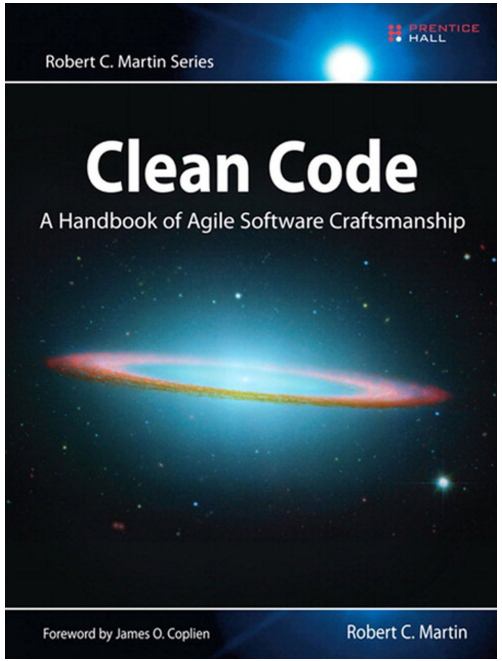
#WCZGZ

LIMPIA TU CÓDIGO

PRINCIPIOS BÁSICOS PARA NO ODIARTE A TI MISMO

Fernando Paredes Murillo
WordPress Dev - MVF Global

@f_paredesmur | fparedes.com



Esta charla no es (solamente) un resumen de este libro

0

EL CÓDIGO LIMPIO NO ES UN CAPRICHO

> **No es cosa menor. Dicho de otra manera, es cosa mayor**

– Mariano Rajoy

>1

Es legible

>1 Es legible

>2 Está bien estructurado

>1 Es legible

>2 Está bien estructurado

>3 Hace lo que parece que hace

La deuda técnica crece de manera exponencial

```
//When I wrote this, only God and I
understood what I was doing
//Now, God only knows

//Peter wrote this, nobody knows what it
does, don't change it!

// They made me write it, against my will.
```

Fuente: https://loudprogrammer.net/best-comments-in-source-code-i-ever-encountered/

El código limpio siempre es más rentable a medio plazo.

>1

Consejos para escribir buen código

>1 Consejos para escribir buen código

>2 Estándares a seguir

1. Consejos para escribir buen código
2. Estándares a seguir
3. Herramientas que ayudan

1 NOMENCLATURA

>1

Usa nombres descriptivos

```
1  $p; // product price
2  $product_price;
3
4  $list;
5  $products;
```

>1

Usa nombres descriptivos

>1 Usa nombres descriptivos

>2 Sé coherente

```
1  get_album_name();
2  fetch_song_title();
3
4  class Catalogue_Manager{}
5  class Library_Controller{}
6
7  $lista_de_productos;
8  $products_in_category;
```

>1 Usa nombres descriptivos

>2 Sé coherente

1. Usa nombres descriptivos
2. Sé coherente
3. Sigue los estándares

```
1  .btn{}
2  .alert.alert-primary{}
3  .com-6.col-md-4.col-lg-3{}
```

```
1  get_post();
2  get_product();
3  get_film();
```

1. Usa nombres descriptivos
2. Sé coherente
3. Sigue los estándares

>1 Usa nombres descriptivos

>2 Sé coherente

>3 Sigue los estándares

>4 Déjate de bromas

```
1  se_lia_parda();
2  $lo_del_miguelito;
3  Boom::kaboom();
```

Usa el sentido común para nombrar cosas y se coherente

2

COMENTARIOS: MENOS ES MÁS

```
1  /**
2   * Returns the day of the month.
3   *
4   * @return int the day of the month.
5   */
6  function get_day_of_month() : int {
7      return (int) date('d');
8  }
```

wp-includes/class-wp-term.php

```
10  /**
11   * Core class used to implement the WP_Term object.
16   */
17  final class WP_Term {}
```

>1

El código debe poder entenderse por sí mismo

```php
//Check if the product has a valid discount
if (
    $product->is_in_stock()
    && $product->is_published()
    && 0 != $product->discounted_price
    && $product->discounted_price < $product->price
){}
```

```
1  if ($product->has_valid_discount()){}
```

>1

El código debe poder entenderse por sí mismo

>1 **El código debe poder entenderse por sí mismo**

>2 **Nadie mantiene los comentarios**

>1 El código debe poder entenderse por sí mismo

>2 Nadie mantiene los comentarios

>3 Los comentarios necesarios deben ser fáciles de ver

El código que siempre necesita comentarios no es buen código

3

FUNCIONES

>1

Funciones cortas y concretas

```php
class Order {
    public function process( Payment_Method $payment_method ) : string {
        $total = 0;

        foreach ($this->row as $row) {
            $product = $row->product;
            $qty = $row->qty;
            if (
                $qty < $product->stock_available();
                && $product->is_published();
            ){
                if (
                    0 != $product->discounted_price
                    && $product->discounted_price < $product->price
                ) {
                    $total += $qty * $product->discounted_price;
                } else {
                    $total += $qty * $product->regular_price;
                }
            }
        }
```

```
22            if ($total <= 0) {
23                 return 'Invalid order';
24            }
25            switch ($payment_method->name) {
26                 case 'card':
27                      $payment = $payment_method->process($total);
28                      return $payment ? 'Payment Successful' : 'There is a problem';
29                 case 'cheque':
30                      if ($total < 200) {
31                           return 'Order total is not enough for cheque';
32                      }
33                      $payment = $payment_method->process($total);
34                      return $payment ? 'Payment Successful' : 'There is a problem';
35
36                 case 'cash':
37                      if ('store_collection' != $order->delivery_method) {
38                           return 'You need to pay by cash in store';
39                      }
40                      return 'Your order is waiting for you in our shop';
41            }
42       }
43  }
```

```php
class Order {
    public function process(Payment_Method $payment_method) : string {
        if (! $this->is_valid()) {
            return 'Invalid Order';
        }

        $total = $this->get_total_with_discounts():

        return $payment_method->process($total);
    }
}
```

>1

Funciones cortas y concretas

>1 Funciones cortas y concretas

>2 Evita las variables globales

>1 Funciones cortas y concretas

>2 Evita las variables globales

>3 Cuidado con los efectos secundarios

```php
$name = 'Ryan McDermott';

function split_into_first_and_last_name() : void {
    global $name;

    $name = explode(' ', $name);
}

split_into_first_and_last_name();

var_dump($name); // ['Ryan', 'McDermott'];
```

Fuente: https://github.com/jupeter/clean-code-php#avoid-side-effects

```php
function split_into_first_and_last_name(string $name) : array {
    return explode(' ', $name);
}

$name = 'Ryan McDermott';
$new_name = split_into_first_and_last_name($name);

var_dump($name); // 'Ryan McDermott';
var_dump($new_name); // ['Ryan', 'McDermott'];
```

Fuente: https://github.com/jupeter/clean-code-php#avoid-side-effects

Una buena función es corta y hace sólo una cosa.

4

CLASES:
SOLID

S Single Responsibility Principle

SINGLE RESPONSIBILITY PRINCIPLE

Una clase debería cambiar sólo por un motivo.
Esto quiere decir que una clase hace sólo una cosa.

```php
class Product {
    protected string $name;
    protected string $photo;

    public function __construct(string $name, string $photo = '') {
        $this->name = $name;
        $this->photo = $photo;
    }

    public function get_name() : string {
        return $this->name;
    }

    public function print_photo() : void {
        if ($this->$photo) {
            echo '<img src="' . $this->photo . '">';
        }
    }
}
```

```php
class Product {
    protected string $name;
    protected string $photo;

    public function __construct(string $name, string $photo = '') {
        $this->name = $name;
        $this->photo = $photo;
    }

    public function get_name() : string {
        return $this->name;
    }
}
```

```php
class Photo_Printer {
    protected string $photo;

    public function __construct(Product $product) {
        $this->photo = $product->photo;
    }

    public function print() : void {
        if ($this->$photo) {
            echo '<img src="' . $this->photo . '">';
        }
    }
}
```

> El cuchillo que sirve para todo no sirve para nada.

– Jordi Cruz

S Single Responsibility Principle

S Single Responsibility Principle

O Open/Closed Principle

OPEN / CLOSED PRINCIPLE

Las clases deben estar abiertas a ser extendidas, pero cerradas a ser modificadas.

```php
class Product {
    protected string $name;
    protected string $photo;
    protected array $sizes;
}
```

```php
class Product {
    protected string $name;
    protected string $photo;
}

class Tshirt extends Product {
    protected array $sizes;
}

class Fridge_Magnets extends Product {
    protected string $picture_content;
}
```

S Single Responsibility Principle

O Open/Closed Principle

S Single Responsibility Principle

O Open/Closed Principle

L Liskov Substitution Principle

LISKOV SUBSTITUTION PRINCIPLE

Si hay una clase que implementa a otra, ambas deben poder intercambiarse sin producir errores.

```php
class Animal {
    public function make_noise() {
        echo 'I am making a generic noise';
    }
}

class Cat extends Animal{
    public function make_noise() {
        echo 'meow meow';
    }
}

$animal = new Animal();
$animal->make_noise(); // I am making a generic noise

$animal = new Cat();
$animal->make_noise(); // meow meow
```

```php
class Animal {
    public function make_noise() {
        echo 'I am making a generic noise';
    }
}

class Fox extends Animal{
    public function make_noise() {
        throw new Exception('What does the fox say?');
    }
}

$animal = new Animal();
$animal->make_noise(); // I am making a generic noise

$animal = new Fox();
$animal->make_noise(); // ⚠️ Fatal error
```

S Single Responsibility Principle

O Open/Closed Principle

L Liskov Substitution Principle

S Single Responsibility Principle

O Open/Closed Principle

L Liskov Substitution Principle

I Interface segregation principle

INTERFACE SEGREGATION PRINCIPLE

Una clase no debe depender de interfaces que no usa para nada.

```php
interface Employee{
    public function work() : void;
    public function eat() : void;
}

class Human_Employee implements Employee {
    public function work() : void {
        // ... working
    }

    public function eat() : void {
        // ... eating in lunch break
    }
}

class Robot_Employee implements Employee {
    public function work() : void {
        // ... working much more
    }

    public function eat() : void {
        // ... robot can't eat, but it must implement this method
    }
}
```

Fuente: https://github.com/jupeter/clean-code-php#interface-segregation-principle-isp

```php
interface Workable {
    public function work() : void;
}

interface Feedable {
    public function eat() : void;
}

interface Living_Employee extends Feedable, Workable {}

class Human_Employee implements Living_Employee {
    public function work() : void {
        // ... working
    }

    public function eat() : void {
        // ... eating in lunch break
    }
}

class Robot_Employee implements Workable {
    public function work() : void {
        // ... working. Robot can only work 😰
    }
}
```

Fuente: https://github.com/jupeter/clean-code-php#interface-segregation-principle-isp

S Single Responsibility Principle

O Open/Closed Principle

L Liskov Substitution Principle

I Interface segregation principle

S Single Responsibility Principle

O Open/Closed Principle

L Liskov Substitution Principle

I Interface segregation principle

D Dependency Inversion Principle

DEPENDENCY INVERSION PRINCIPLE

Las clases de alto nivel no debe depender directamente de un módulo de bajo nivel, sino de abstracciones.

```php
class Password_Reminder {
    private $dbConnection;

    public function __construct(MySQL_Connection $db_connection) {
        $this->db_connection = $db_connection;
    }
}
```

Fuente: https://scotch.io/bar-talk/s-o-l-i-d-the-first-five-principles-of-object-oriented-design

```php
interface DB_Connection {
    public function connect();
}

class MySQL_Connection implements DB_Connection {
    public function connect() {
        // Database connection
    }
}

class Password_Reminder {
    private $db_connection;

    public function __construct(DB_Connection $db_connection) {
        $this->db_connection = $db_connection;
    }
}
```

Fuente: https://scotch.io/bar-talk/s-o-l-i-d-the-first-five-principles-of-object-oriented-design

SOLID es el primer gran paso para hacer OOP de verdad

SOLID EN PROFUNDIDAD

https://scotch.io/bar-talk/s-o-l-i-d-the-first-five-principles-of-object-oriented-design

https://github.com/jupeter/clean-code-php#solid

https://medium.com/mindorks/solid-principles-explained-with-examples-79d1ce114ace

https://www.baeldung.com/solid-principles

4

ESTANDARES VS WORDPRESS

```php
function foo_bar() {
    echo 'Hello';
}

//...

function heroWeGoAgain()
{
    echo 'It\'s me';
}
```

PSR-1
PSR-12

VS

WP CODING STANDARDS

Escoge un estándar para tu código y cíñete a él siempre.

5

HERRAMIENTAS

>1

Linters: ESLint, StyleLint, PHPMD

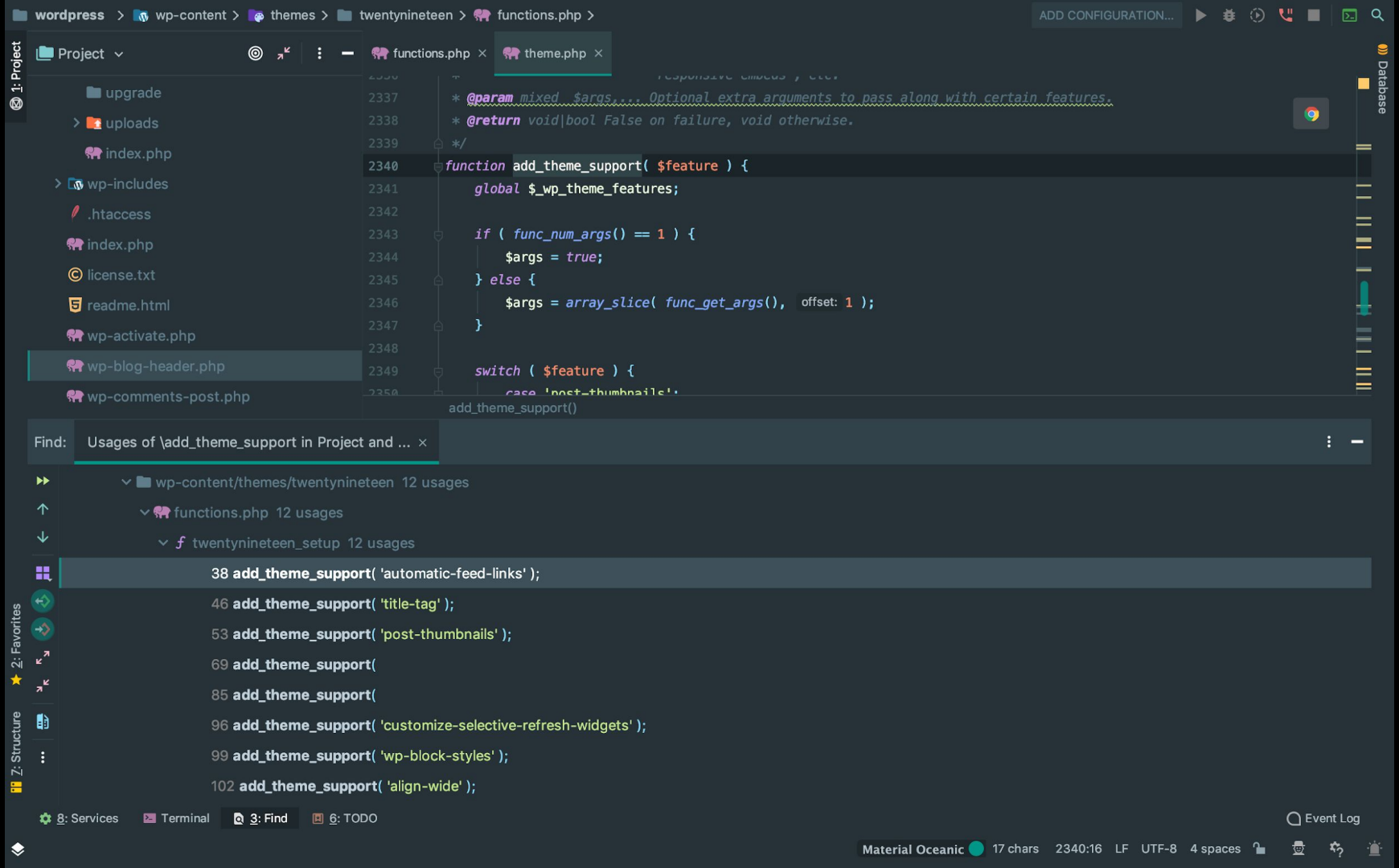
```
rules: {
    //Use 4 spaces for indentation
    "indentation": 4,

    //Use kebab-case for selectors
    "selector-class-pattern": "^([a-z][a-z0-9]*)(-[a-z0-9]+)*$",

    //One selector per line
    "selector-list-comma-newline-after": "always-multi-line",

    //Only one declaration allowed in a single line (`.selector-1 { width: 10%; }`)
    "declaration-block-single-line-max-declarations": 1,

    //There must always be a newline after the semicolon in multi-line rules.
    "declaration-block-semicolon-newline-after": "always-multi-line",
}
```

>1

Linters: ESLint, StyleLint, PHPMD

>1 Linters: ESLint, StyleLint, PHPMD

>2 IDEs


wordpress > wp-content > themes > twentynineteen > functions.php >
ADD CONFIGURATION...
Project
functions.php
theme.php
upgrade
uploads
index.php
wp-includes
.htaccess
index.php
license.txt
readme.html
wp-activate.php
wp-blog-header.php
wp-comments-post.php
2337 * @param mixed $args,... Optional extra arguments to pass along with certain features.
2338 * @return void|bool False on failure, void otherwise.
2339 */
2340 function add_theme_support($feature) {
2341 global $_wp_theme_features;
2342
2343 if (func_num_args() == 1) {
2344 $args = true;
2345 } else {
2346 $args = array_slice(func_get_args(), offset: 1);
2347 }
2348
2349 switch ($feature) {
add_theme_support()
Find:
Usages of \add_theme_support in Project and ...
wp-content/themes/twentynineteen 12 usages
functions.php 12 usages
twentynineteen_setup 12 usages
38 add_theme_support('automatic-feed-links');
46 add_theme_support('title-tag');
53 add_theme_support('post-thumbnails');
69 add_theme_support(
85 add_theme_support(
96 add_theme_support('customize-selective-refresh-widgets');
99 add_theme_support('wp-block-styles');
102 add_theme_support('align-wide');
8: Services
Terminal
3: Find
6: TODO
Event Log
Material Oceanic
17 chars
2340:16
LF
UTF-8
4 spaces

Automatiza lo que puedas y hazte la vida fácil

RECAPITULANDO

5 PRINCIPIOS PARA NO ODIARTE A TI MISMO

1. Nombra con sentido y coherencia
2. Procura que el código se entienda por sí mismo
3. Estructura tu código en unidades que hagan una sola cosa
4. Sigue un estándar de estilo
5. Usa herramientas que vigilen por ti

Deja el campo mejor de lo que lo encontraste

¡GRACIAS!

Limpia tu código.
Principios básicos para no odiarte a ti mismo

WordCamp Zaragoza - 18/01/2020

Fernando Paredes Murillo
> fparedes.com
> @f_paredesmur

Descarga la presentación

https://fparedes.com/wczgz2020