Código Limpio

Porque ayer escribiste “legacy code”

Francisco M. González

Bienvenidos

Francisco M. González

Malagueño

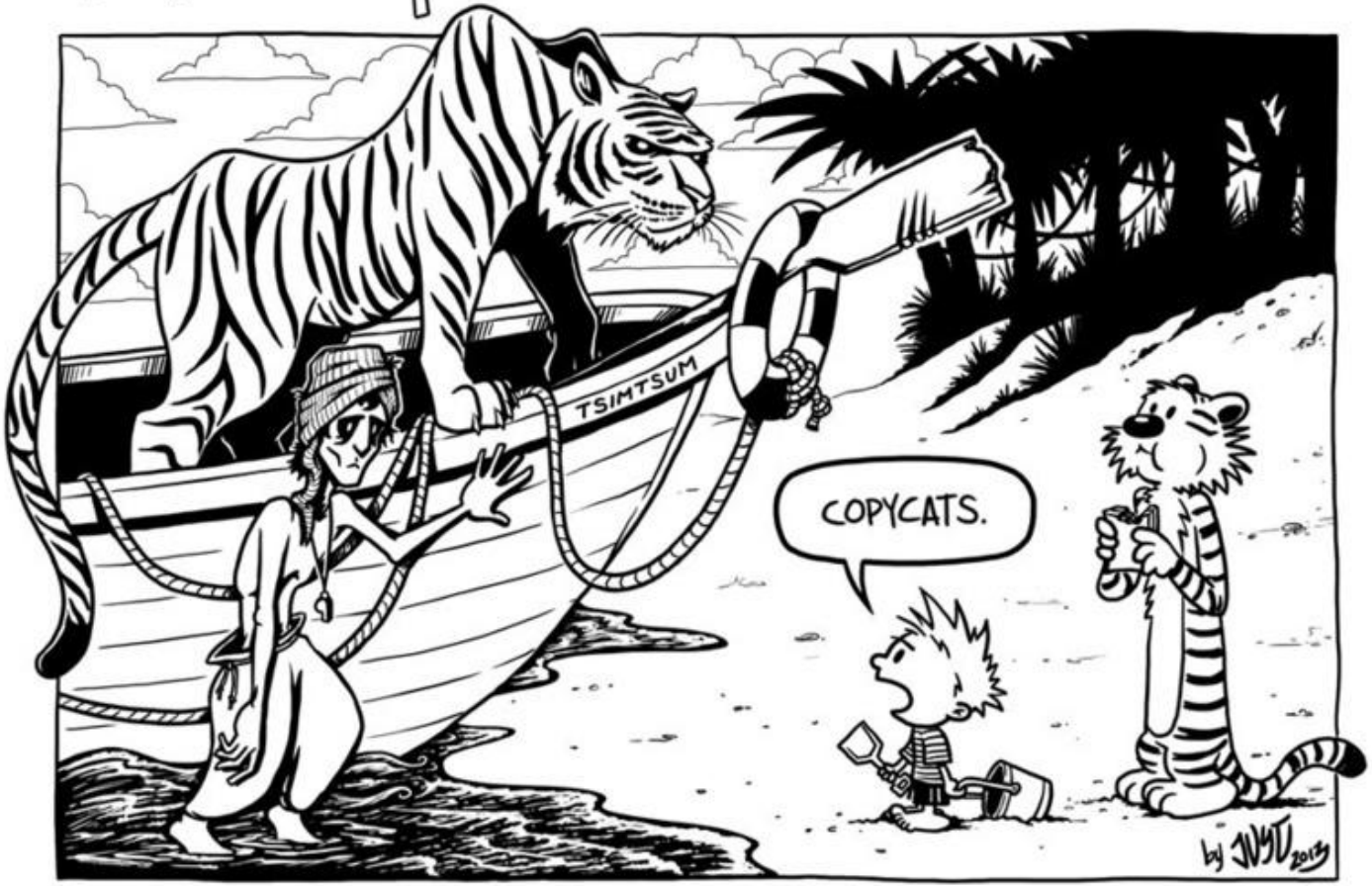
Construyendo cosas en Ciklum

Coleccionista de cómics

Padre

@fmgonzalez76

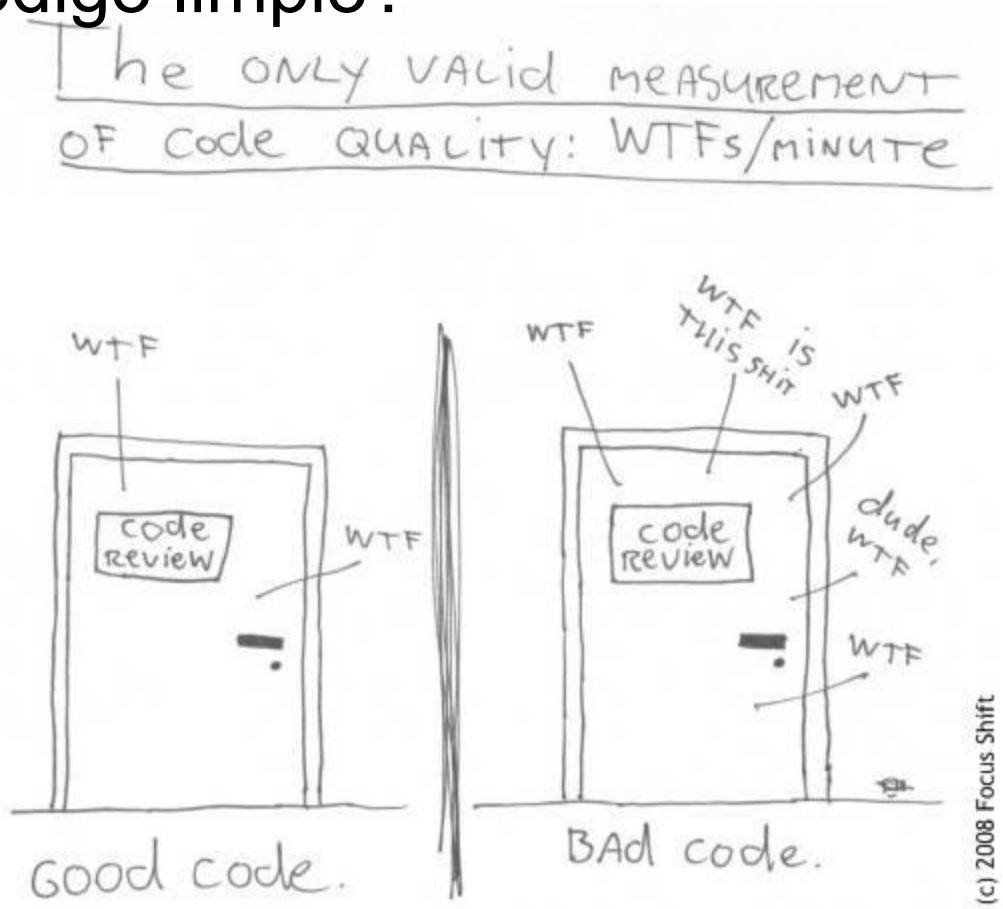
life of pi (alternate ending)




CLEAN CODE
CLEAN CODE
CLEAN CODE

¿Qué es código limpio?

¿Qué es código limpio?



¿Qué NO es código limpio?

- No describe lo que hace
- Es difícil de entender
- Es difícil de modificar

Hack horrible

Hack horrible

que no es mío

Solución temporal

Hack horrible

que es mío

Mal organizado

Código mal organizado

que no es mío

Arquitectura compleja

Código mal organizado

que es mío

Necesita un refactor

Funciona,

pero no lo entiendo

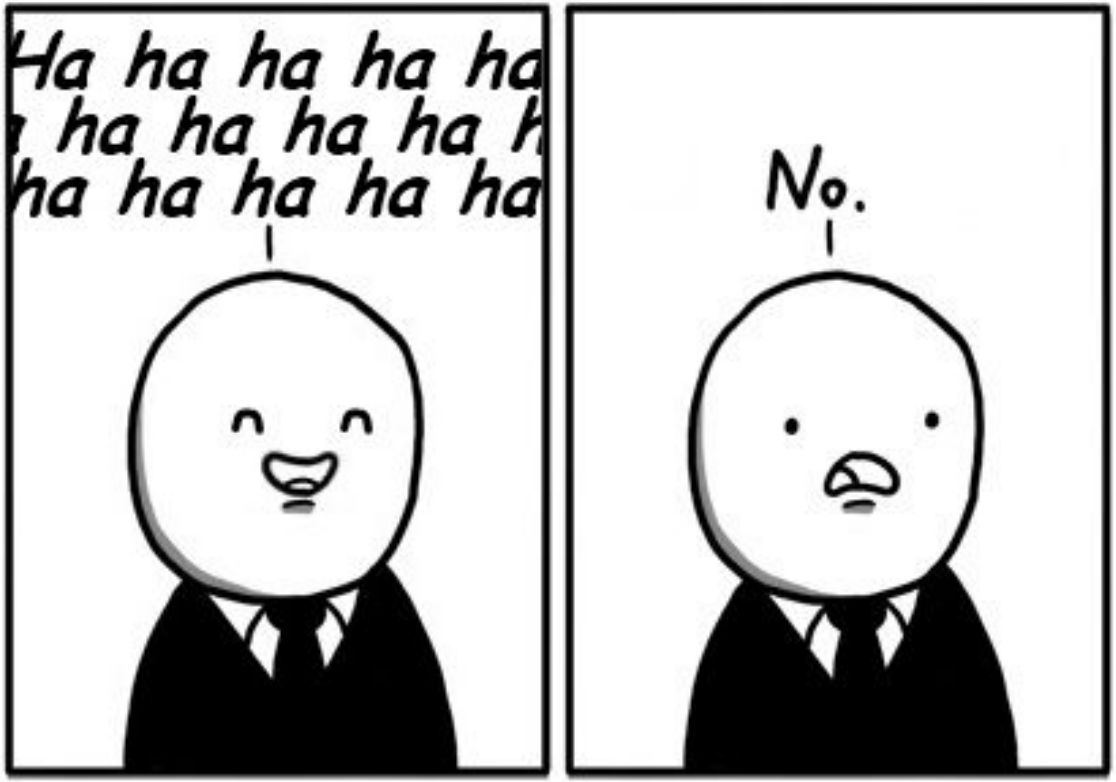
Solución limpia

Funciona,

y lo entiendo

Excusas

- Hay que ir rápido
- Planificaciones vertiginosas
- Gestores intransigentes
- Presión de los clientes


Ha ha ha ha ha
ha ha ha ha h
ha ha ha ha ha
No.

¿Qué es código limpio?

- Elegante
- Eficiente
- Simple y Directo
- Legible como prosa
- Cuidado
- Pequeño

Naming

Todo son nombres

- Clases
- Namespaces
- Funciones
- Variables
- Ficheros
- Directorios
- ...

Declara intención

```
int d; // elapsed time in days
```

```
int elapsedTimeInDays;

int fileAgeInDays;

int daysSinceCreation;
```

Declara intención

```
/** Useful range constant */

CONST INCLUDE_NONE = 0;

CONST INCLUDE_FIRST = 1;

CONST INCLUDE_SECOND = 2;

CONST INCLUDE_BOTH = 3;
```

Evita la desinformación

```
public function getCountry();
```

```
public function getCountryName();
```

Dice lo que hace y hace lo que dice.

Nombres pronunciables y que se puedan buscar

getXYZ();

genDDMMYY();

m_qdox;

qty_pass_m;

qty_pass_s;

Evita codificaciones y sufijos de tipo

IAccount (interface)

pWriter (pointer)

bActive (boolean)

eStatus (enum)

m_size (member attribute)

Mejor usa solo nombres

Nombres de clase

- Nombres, no verbos
- Evitar sufijos Data o Info

- Métodos:
  - Verbos
  - Accessors: get
  - Mutators: set
  - Predicates: is / has

La longitud de las variables acorde al ámbito

```
foreach ($userItems as $item) {
    $item->doSomething();
}
```

La longitud de las variables acorde al ámbito

```
try {

    ...

} catch (Exception $e) {

    log($e->getMessage());

}
```

La longitud de los métodos acorde a la visibilidad

Private:

- tryProcessInstructions()
- closeServiceInSeparateThread()

Public:

- open()
- save()

Nombres de clases

La herencia tiende a alargar los nombres, añadiendo adjetivos

Account ← SavingAccount

Naming. Resumen

- Elige tus nombres cuidadosamente
- Comunican intención
- Evita la desinformación
- Pronunciables
- Evita codificaciones
- Busca la prosa
- Recuerda la regla del ámbito

Funciones

Dos reglas básicas para las funciones

1. Hazlas pequeñas

Dos reglas básicas para las funciones

1. Hazlas pequeñas
2. Hazlas **más** pequeñas

Do one thing

- Solo una cosa
- Hacerla bien
- Sólo esa cosa

Las Funciones deben

- Hacer algo (cambiar el estado de un objeto). Una orden.
- Devolver algo. Una consulta.

No ambas.

Recordad: Do one thing.

Extrae hasta que no puedas más

- Si puedes extraer parte de tu función a otra función, debes hacerlo
- Por definición está haciendo más de una cosa
- Debes extraer hasta que no puedas más

No mezclar niveles de abstracción

```
// Alto nivel de abstracción

getHtml();

// Nivel intermedio

String htmlPageSection = SectionParser.render(section);

// Nivel bajo

outputHtml.append(“\n”);
```

Stepdown rule

- Leerse como un libro
- Definir conceptos de arriba a abajo
- Evitar los saltos por el fichero para entender la prosa

Usar nombres descriptivos

- No temas usar nombres largos
- Un nombre largo y descriptivo es mejor que uno corto y confuso
- Prueba con varios, no temas renombrarlos hasta que se ajusten a tu diseño

Argumentos

- niladic. Ideal
- monadic
- dyadic
- triadic. Evitar si es posible
- polyadic. Requieren de una clara justificación y después NO usarlos de todos modos.
- Flag arguments

Devolver las transformaciones

```
void transform(StringBuffer out);
```

```
StringBuffer transform(StringBuffer in);
```

Salidas

- No devuelvas varias salidas
- Mejor lanzar excepciones que códigos de error
- Extrae los bloques de try/catch
- El tratamiento de errores ya es *one thing*

Funciones. Resumen

- Pequeñas
- Más pequeñas
- Nombres descriptivos
- Hacen una sola cosa. Órdenes y consultas.
- Extrae hasta que no puedas más

Comentarios

Los comentarios son errores

- Los ignoramos al leer
- Son un signo de un mal naming
- El único comentario bueno es el que no escribes
- Deben ser raros y destacados en el IDE

Buenos comentarios

- Legales (no hay remedio)
- Formato (como una expresión regular)

  // format matched kk:mm:ss EEE, MMM dd, yyyy

  string pattern = “\\d*:\\d*:\\d* \\w*, \\w* \\d*, \\d*”

Algunos ejemplos

/* The logger class */

protected $logger;

/* The version */

public $version;

Algunos ejemplos

```
/**
 * @param $item The item to be added
 * @param $position The position to be added
 */
public function addItem($item, $position = 0);
```

Algunos ejemplos

```
/**
 * 11/12/2010 - Added the colour feature
 * 03/02/2011 - removed deprecated function
 * 07/02/2011 - Fixed bug in addPoints
 * …
```

Algunos ejemplos

```
// ========================

// =  INSTANCE VARIABLES  =

// ========================
```

Algunos ejemplos

```
if (firstName.equals(monthName)) {
    return validMonthName;
} // end if
```

Algunos ejemplos

/** Added by Franci */

Comentarios. Resumen

- No los uses
- Puedes hacerlo mejor

Estilo de código

¿Por qué usar un código de estilo?

- El estilo de código persiste más que el código en sí
- Facilita la modificación y la refactorización
- Es más legible
- Es un acuerdo de equipo
- Es una herramienta de comunicación

¿Por qué un estilo de código?

```
namespace Foo;

public class Bar

{

        public function __construct()

        {

                …

        }

}
```

```
namespace Foo;

public class Bar {

        public function __construct() {

                …

        }

}
```

¿Por qué un estilo de código?

```
namespace Foo;

public class Bar

{

        private string $title    = ‘default title’;

        private int      $code = 0;

        ...

}
```

```
namespace Foo;

public class Bar

{

        private string $title = ‘default title’;

        private int $code = 0;

        ...

}
```

Standards

- PHP: PSR-1, PSR-2, Zend, Symfony, PEAR, Wordpress, Drupal,...
- Javascript: airbnb, es6,...

Herramientas

- phpcs, eslint,...
- editorconfig
- posibilidad de automatizar (IDE, githooks, CI)


THANK YOU
FOR
THE
ATTENTION