Why code quality matters

April 2020

Pol Dellaiera
Analysis, development, research
PHPCC – DIGIT.B.4.003

Clean code

The what, the why, and the how.

What is clean code ?

When objectivity > subjectivity

Clean code

- Good code structure (file structure, design patterns)
- Good code formatting (coding standards)
- (Good tests coverage)

Why clean code ?

Clean code - code structure

PHP is a vast language that allows coders of all levels the ability to produce code not only quickly, but efficiently.

However, while advancing through the language, we often forget the basics that we first learnt in favor of shortcuts and, or bad habits.

Source: https://phptherightway.com/

How to get clean code ?

Clean code - code structure

- Use Design Patterns
- Use UTF-8
- PSR-4 code structure
- DRY (**D**on’t **R**epeat **Y**ourself)
- SOLID principles

Source: https://designpatternsphp.readthedocs.io/

Clean code - code structure

- Naming
  - Classes: Noun
  - Methods: Verb
  - Use one word per concept and stick with it (*get, fetch, retrieve, request*)
- Functions/methods
  - As small as possible
  - Pure
  - Keep arguments under 4
- Comments
  - Only if necessary
  - Code should be self-explanatory

Source: https://medium.com/mindorks/how-to-write-clean-code-lessons-learnt-from-the-clean-code-robert-c-martin-9ffc7aef870c

Clean code - code formatting

PHP is an easygoing level language. Unlike very strict language like python, PHP allows you to write code in many different ways.

This has pro and cons.

Along the years, the trends is to have some kind of rules to write PHP code.

One of the most commonly used is PSR-12 from the PHP Framework Interop Group (PHP FIG).

Source: https://designpatternsphp.readthedocs.io/

Clean code - code formatting

At European Commission, discussing coding style may be very time consuming and can sometimes lead to some disagreements.

Therefore, it’s important to agree on a code style before starting a project.

ECPHP packages are following the PSR-12 rules and a set of other custom rules on the top.

Having a good code formatting help people to read code faster and also often helps to find issues in a glimpse of an eye.

Clean code - test coverage

When writing code, tests is one of the best way to assess its reliability across changes and upgrades.

This become even more important when working with many people.

There are multiple testing frameworks, it’s up to the developer to choose the best fit.

Clean code is then the sum of multiple recipes.

Definition proposal of << clean code >>

- Elegant, good code and file structure, good formatting
- Focused,
- Ordered,
- Run all the tests,
- No code duplication (DRY),
- Minimize the amount of line of code

Source: https://medium.com/mindorks/how-to-write-clean-code-lessons-learnt-from-the-clean-code-robert-c-martin-9ffc7aef870c

Do I need to know all the rules by heart ??

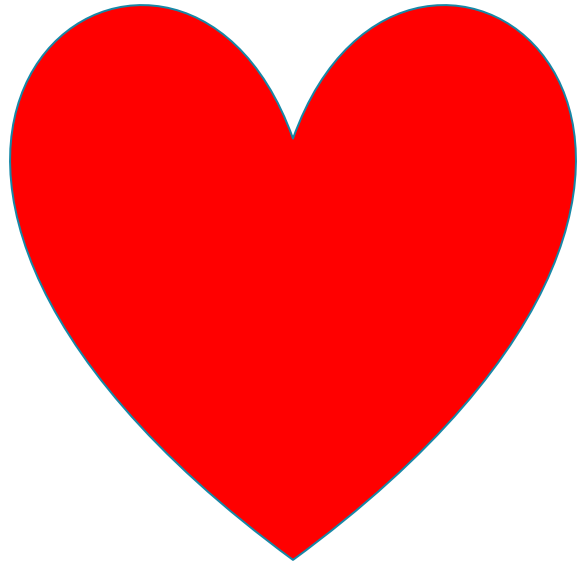
No.

Notable examples

Clean code - code example

```
$result = $serializer->serialize($data, 448);
```

Clean code - code example

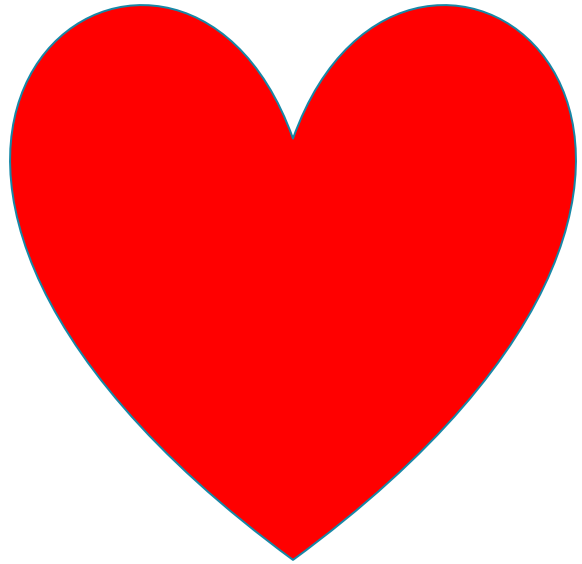


```
$json = $serializer->serialize(
    $data,
    JSON_UNESCAPED_SLASHES | JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE
);
```

Clean code - code example

```
$moment = new \DateTimeImmutable();

$ymdstr = $moment->format('y-m-d');
```

Clean code - code example

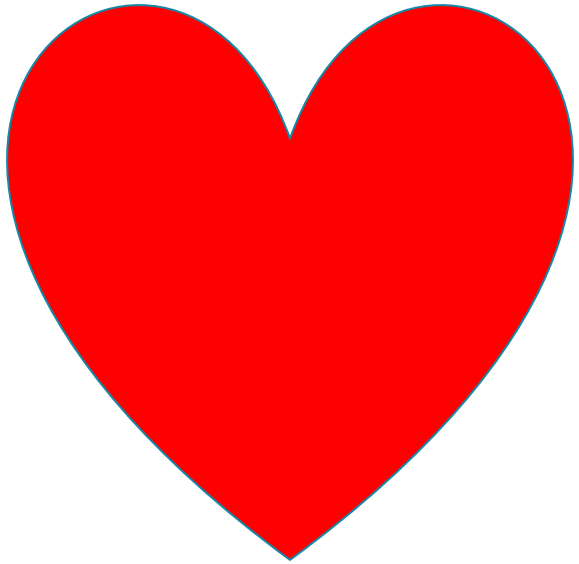


```
$moment = new \DateTimeImmutable();

$currentDate = $moment->format('y-m-d');
```

Clean code - code example

```
class Car
{
    public $carMake;
    public $carModel;
    public $carColor;

    //...
}
```

Clean code - code example

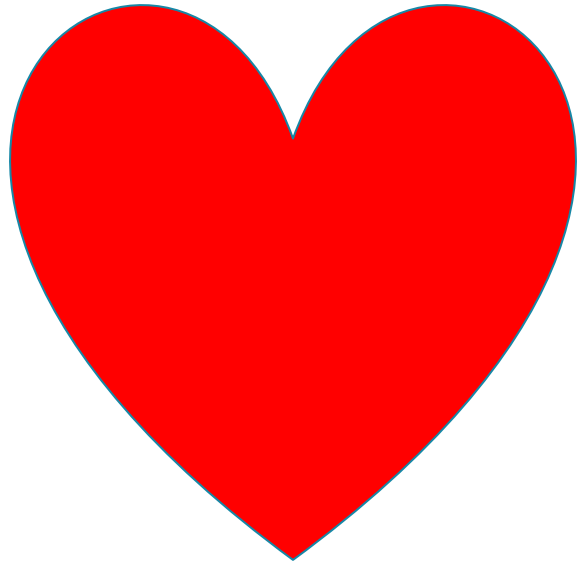


```
class Car
{
    public $make;
    public $model;
    public $color;

    //...
}
```

Clean code - code example

```
class User
{
    // What the heck is 7 for?
    public $access = 7;
}

// What the heck is 4 for?
if ($user->access & 4) {
    // ...
}

// What's going on here?
$user->access ^= 2;
```

Clean code - code example

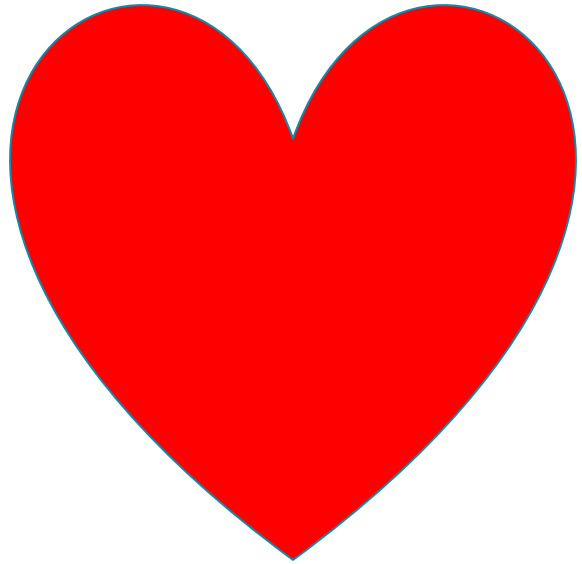


```
class User
{
    public const ACCESS_READ = 1;
    public const ACCESS_CREATE = 2;
    public const ACCESS_UPDATE = 4;
    public const ACCESS_DELETE = 8;

    // User as default can read, create and update something
    public $access = self::ACCESS_READ | self::ACCESS_CREATE | self::ACCESS_UPDATE;
}

if ($user->access & User::ACCESS_UPDATE) {
    // do edit ...
}

// Deny access rights to create something
$user->access ^= User::ACCESS_CREATE;
```

Clean code - code example

```
$address = 'One Infinite Loop, Cupertino 95014';
$cityZipCodeRegex = '/^[^,]+,\s*(.+?)\s*(\d{5})$/';
preg_match($cityZipCodeRegex, $address, $matches);

saveCityZipCode($matches[1], $matches[2]);
```

Clean code - code example

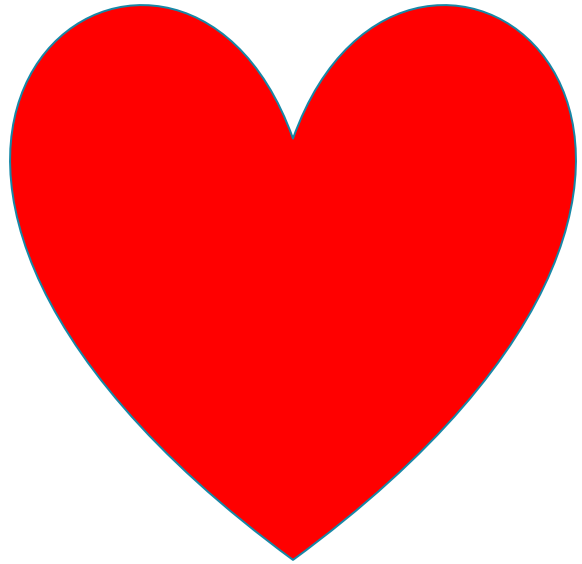


```
$address = 'One Infinite Loop, Cupertino 95014';
$cityZipCodeRegex = '/^[^,]+,\s*(?<city>.+?)\s*(?<zipCode>\d{5})$/';
preg_match($cityZipCodeRegex, $address, $matches);

saveCityZipCode($matches['city'], $matches['zipCode']);
```

Clean code - code example

```
function fibonacci(int $n)
{
    if ($n < 50) {
        if ($n !== 0) {
            if ($n !== 1) {
                return fibonacci($n - 1) + fibonacci($n - 2);
            } else {
                return 1;
            }
        } else {
            return 0;
        }
    } else {
        return 'Not supported';
    }
}
```

Clean code - code example

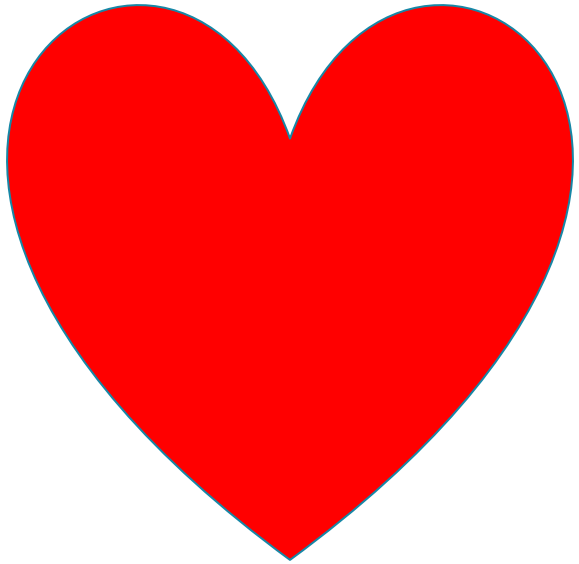


```
function fibonacci(int $n): int
{
    if ($n === 0 || $n === 1) {
        return $n;
    }

    if ($n >= 50) {
        throw new \Exception('Not supported');
    }

    return fibonacci($n - 1) + fibonacci($n - 2);
}
```

Clean code - code example

```php
<?php

$a = '42'; // string
$b = 42; // integer

var_dump($a == $b); // true
```

Clean code - code example

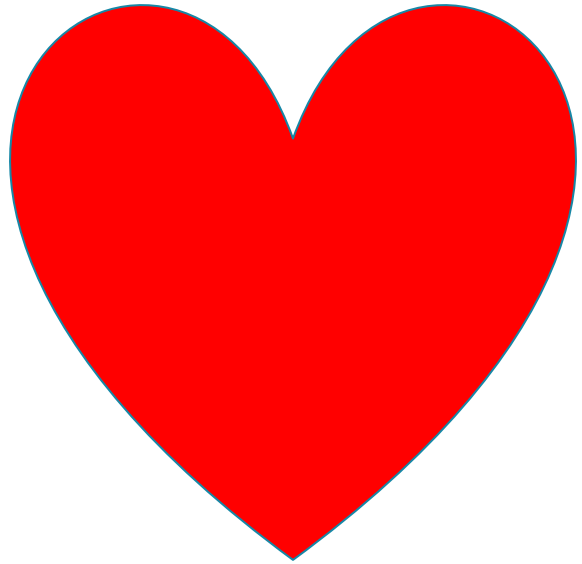


```php
<?php

$a = '42'; // string
$b = 42; // integer

var_dump($a === $b); // false
```

Clean code - code example

```
function travelToTexas($vehicle): void
{
   if ($vehicle instanceof Bicycle) {
      $vehicle->pedalTo(new Location('texas'));
   } elseif ($vehicle instanceof Car) {
      $vehicle->driveTo(new Location('texas'));
   }
}
```

Clean code - code example

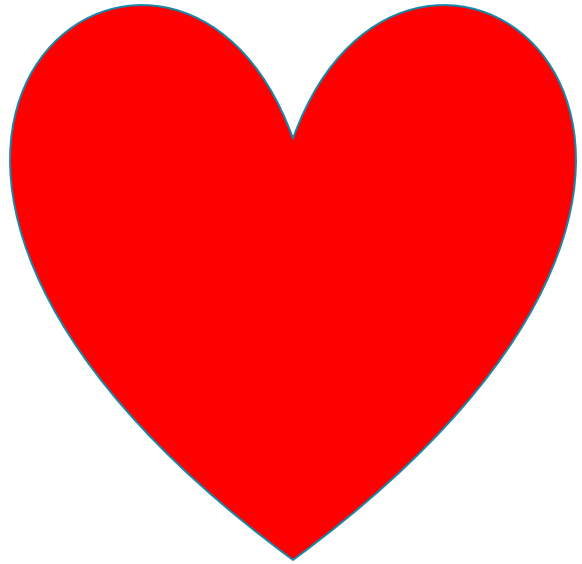


```
function travelToTexas(Vehicle $vehicle): void
{
    $vehicle->travelTo(new Location('texas'));
}
```

Clean code - code example

```
function addition($val1, $val2): int
{
    if (!is_numeric($val1) || !is_numeric($val2)) {
        throw new \Exception('Must be of type Number');
    }

    return $val1 + $val2;
}
```

Clean code - code example

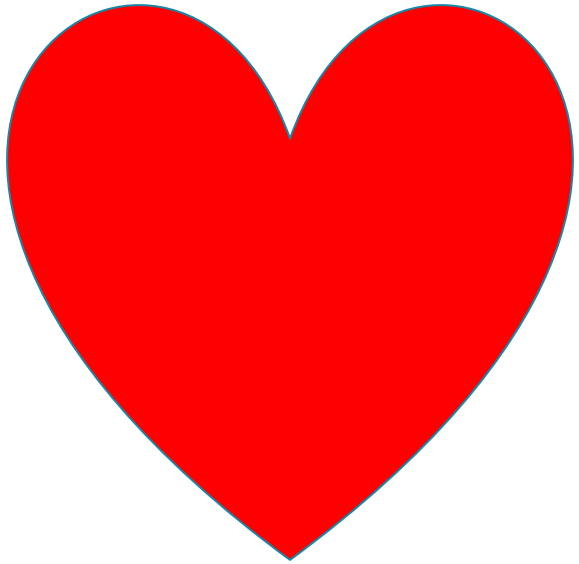


```
function addition(int $val1, int $val2): int
{
    return $val1 + $val2;
}
```

Clean code - code example

```
class BankAccount
{
    public $balance = 1000;
}

$bankAccount = new BankAccount();

// Buy hand sanitizer...
$bankAccount->balance -= 100;
```

Clean code - code example

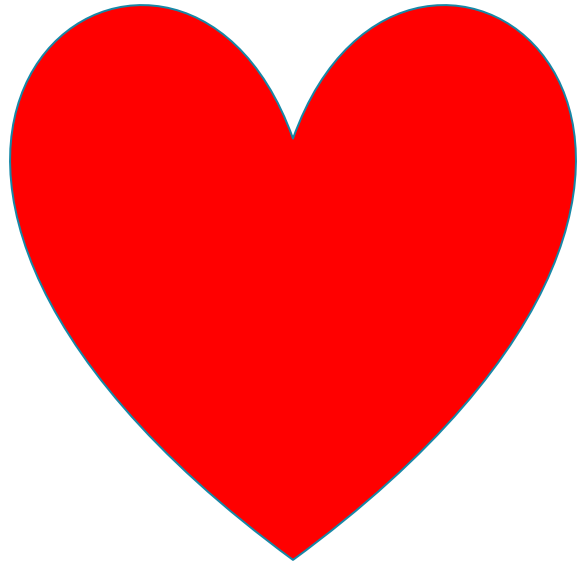


```
class BankAccount
{
    private $balance;

    public function __construct(int $balance = 1000)
    {
      $this->balance = $balance;
    }

    public function withdraw(int $amount): void
    {
        $this->balance -= $amount;
    }
}

$bankAccount = new BankAccount();

// Buy hand sanitizer...
$bankAccount->withdraw(100);
```

Clean code - code example

```
$a = my_function();

if ($a !== null) {
   return $a;
}
```

Clean code - code example

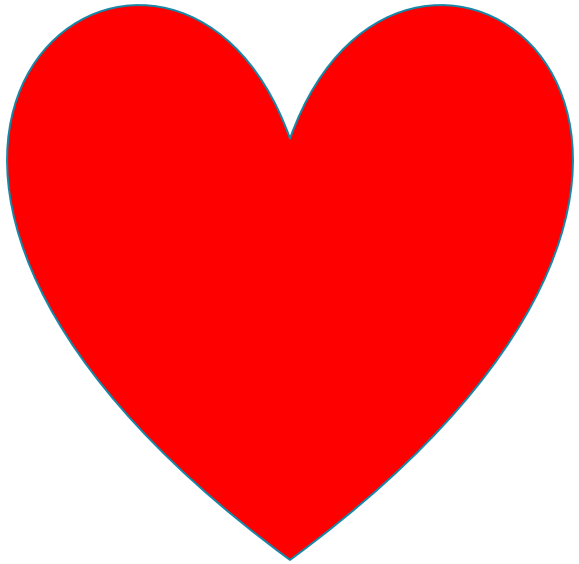


```
if (null !== $a = my_function()) {
   return $a;
}
```

Clean code - code example

```
<?php

function getString(string $str) {
   return $str . $str;
}

getString(12);
getString('12');
```

Clean code - code example



```
<?php

declare(strict_types=1);

function getString(string $str) {
   return $str . $str;
}

getString(12);
getString('12');
```

Notable tips and tricks

Golden rules to follow

- Do not rewrite what's already existing; re-use instead,
- Take time to think about what you're trying to achieve,
- Try to use existing design pattern to increase maintainability,
- Use proper naming in order to write as less comment as possible,
- Try to write as less code as possible,
- Do not aim for perfection and iterate,
- Reading your code 6 months later and… #facepalm,
- Simplicity is the ultimate complexity

Source: https://dzone.com/articles/how-to-be-a-better-programmer

Take care of yourself

- Reduce the sleep debt,
- Drink a lot of water,
- Be kind with your eyes,
- Stop questioning your skills,
- Timebox your work,
- Remember that another life exist besides coding.

Source: https://dzone.com/articles/how-to-be-a-better-programmer

Notable tools

Tools

- Composer normalizer
- PHPStan
- PSalm
- PHP CS
- PHP CS Fixer (*with custom configuration...*)
- Linters: PhpLint, YamlLint, JsonLint, TwigCS
- PHPUnit
- PHPSpec
- Infection
- GrumPHP
- PHP EA Extended

Tools - Static analysers - PHPStan

PHPStan focuses on finding errors in your code without actually running it. It catches whole classes of bugs even before you write tests for the code.

It moves PHP closer to compiled languages in the sense that the correctness of each line of the code can be checked before you run the actual line.

Tools - Static analysers - PHPStan

- Existence of classes used in instanceof, catch, typehints and other language constructs. PHP does not check this and just stays instead, rendering the surrounded code unused.
- Existence and accessibility of called methods and functions. It also checks the number of passed arguments.
- Whether a method returns the same type it declares to return.
- Existence and visibility of accessed properties. It will also point out if a different type from the declared one is assigned to the property.
- Correct number of parameters passed to sprintf/printf calls based on format strings.
- Existence of variables while respecting scopes of branches and loops.
- Useless casting like (string) ‘foo’ and strict comparisons (=== and !==) with different types as operands which always result in false.

Tools - Static analysers - Psalm

Psalm is a static analysis tool that attempts to dig into your program and find as many type-related bugs as possible.

Tools - Static analysers - Psalm

- Mixed type warnings
- Intelligent logic checks
- Property initialisation checks

Tools - PHP CodeSniffer (PhpCs)

PHP CodeSniffer is a set of two PHP scripts; the main phpcs script that tokenizes PHP, JavaScript and CSS files to detect violations of a defined coding standard, and a second phpcbf script to automatically correct coding standard violations. PHP CodeSniffer is an essential development tool that ensures your code remains clean and consistent.

A coding standard in PHP CodeSniffer is a collection of sniff files. Each sniff file checks one part of the coding standard only. Multiple coding standards can be used within PHP CodeSniffer so that the one installation can be used across multiple projects. The default coding standard used by PHP CodeSniffer is the PEAR coding standard, which is nowadays outdated compared to PSR-12.

Tools - PHP CS Fixer

The PHP Coding Standards Fixer tool fixes your code to follow standards; whether you want to follow PHP coding standards as defined in the PSR-1, PSR-2, PSR-12, etc., or other community driven ones like the Symfony one. You can **also** define your own style through configuration.

It can modernize your code and optimize it.

If you are already using a linter to identify coding standards problems in your code, you know that fixing them by hand is tedious, especially on large projects.

This tool does not only detect them, but also fixes them for you.

Tools - Linters

- PhpLint
- YamlLint
- JsonLint

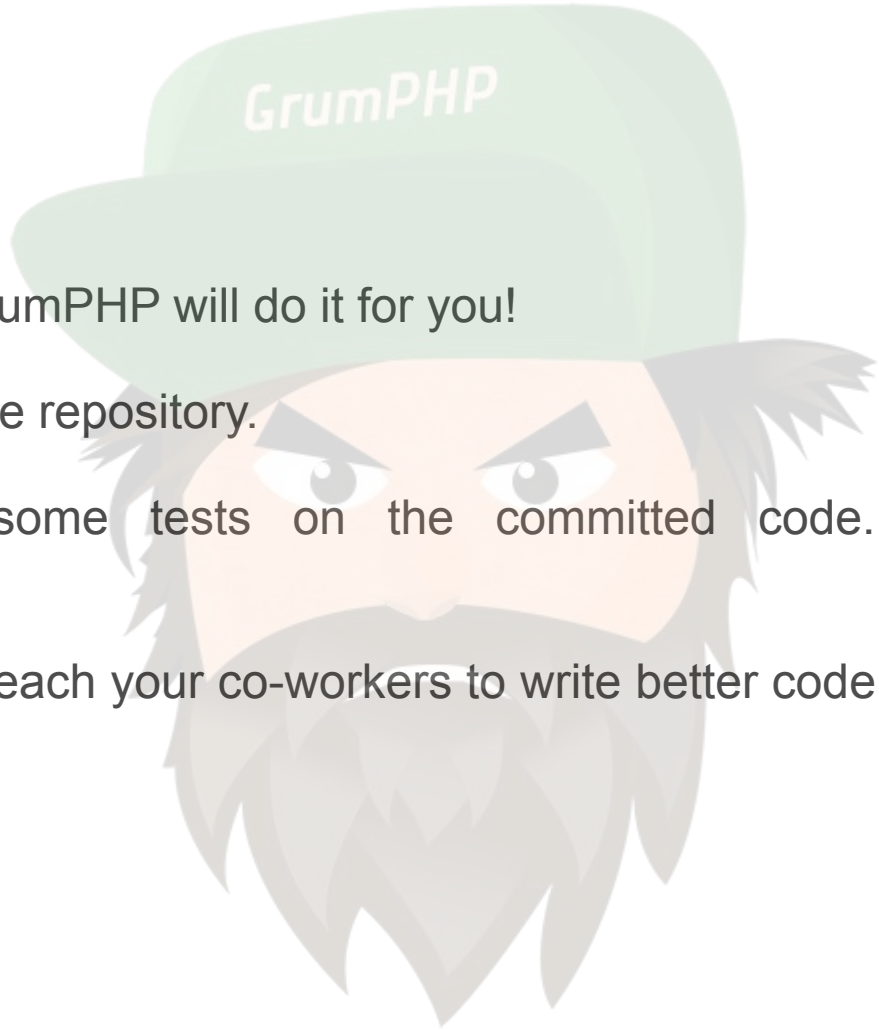
Tools - Testing frameworks

- PhpUnit
- PhpSpec

Tools - Testing frameworks

- Infection

Tools - Orchestrator



Sick and tired of defending code quality over and over again? GrumPHP will do it for you!

This composer plugin will register some git hooks in your package repository.

When somebody commits changes, GrumPHP will run some tests on the committed code. If the tests fail, you won't be able to commit your changes.

This handy tool will not only improve your codebase, it will also teach your co-workers to write better code following the best practices you've determined as a team.

Live demo

Thanks

- European Commission Open Source Program Office
  - For their proofreading and corrections
- You !
  - For coming here to listen to me !

Licences and images

Images used in this presentation are free of use.

More detailed information on Creative Commons can be found here.

European Commission