CLEAN CODE

Tech Lead at Compufácil compufacil.com.br

Github: jeanCarlomachado

Contributor of:

- Doctrine
- Zend Framework
- PHPmd

Twitter: JeanCarloMachad

Researching optimization based on nature inspirations

WHAT CLEAN CODE STANDS FOR?

Reader-focused development style

Produces software that's easy to:

- write
- read
- maintain

> *Programming is the art of telling another human what one wants the computer to do Donald Knuth*

Broken windows theory

"All the rest of this code is crap, I'll just follow suit."

Software entropy, software rot.

WHY IT MATTERS?

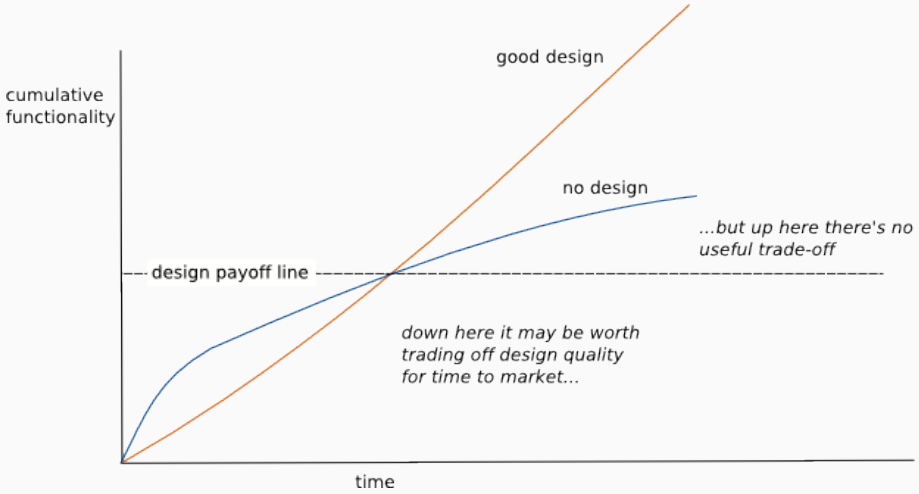

Figure 1: Features over time considering design quality

THE FOUR CHARACTERISTICS OF ROTTING SOFTWARE

- Rigidity
- Fragility
- Immobility
- Viscosity

It’s hard to solve simple problems.

Estimating is hard.

ROTTING SOFTWARE 2 - FRAGILITY

The software breaks too often.

A change in one unrelated part breaks others.

Changes must be echoed in many places.

It’s the inability of reusing software from other places.

It's easier to go to the *hacking mode* than to the *design preservation* mode.

IMPLICATIONS

- Software rot implies in frustrated developers
- Frustrated developers implies in more rotting
- Too much rooting implies in system rewrite
- Loss of money
- Loss of users

SOLUTION?

- Good practices
- Software craftsmanship
- Clean code skills

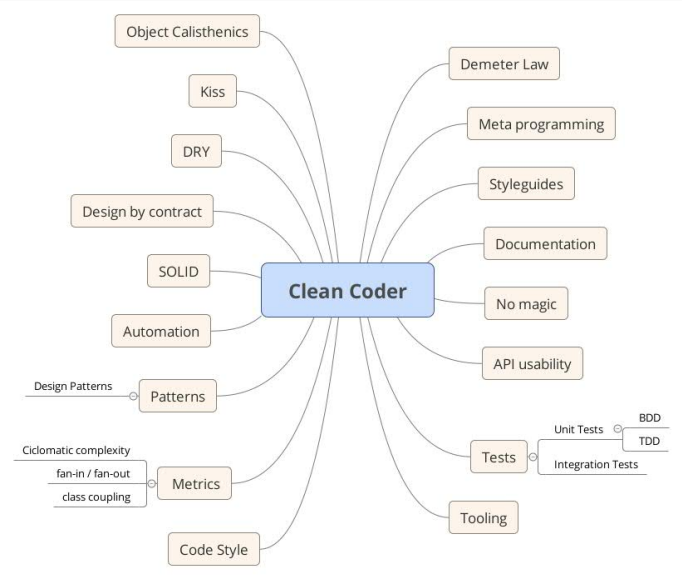

Figure 2: clean-coder-skills.jpg

Follow the ones what suites you most.

> *Relatively simple things can tolerate a certain level of disorganization. However, as complexity increases, disorganization becomes suicidal. Robert Martin*

Need to see the source for to know what a function does? Work on names!

Longer names are generally better than smaller one's.

No `And` or `Or` in names.

Usage scenarios

- Authors name
- Explain trade-offs of implementations

Avoid scenarios

- To express code

Inaccurate comments are way worse than no comments at all.

Don’t Repeat Yourself

DRY on comments

```
/**
*
* @param $title The title of the CD
* @param $author The author of the CD
* @param $tracks The number of tracks of the CD
*
*/
public addCd($title, $author, int $tracks);
```

DRY ON DOCUMENTATION

Code and documentation are different views of the same model

Two places to edit models? DRY violation

Generate code from diagrams or diagrams from code

zendframework/ZendDeveloperTools

The ideal number of arguments of a method is ZERO.

More than tree is unacceptable.

Flags

```
public function useResultCache($bool, $lifetime = null/**...**
{
    if ($bool) {
        $this->setResultCacheLifetime($lifetime);
        $this->setResultCacheId($resultCacheId);

        return $this;
    }
    //...
}
```

Classes should be like journal articles.

In the header you get an general overview.

You are able to decide if you go further or not.

As you read down details increases.

OBJECTS VS DATA STRUCTURES

Objects: hide data, expose operations over it

Data structures: expose data, no meaningful operation

KISS - KEEP IT SIMPLE STUPID

Related with few parts, few responsabilities, decoupling

Simplicity is different of easy

Easy is more related with your current understandings (relative)

There are systems which are easy but not simple

Over time simplicity pays off easiness

> *UNIX is very simple, it just needs a genius to understand it's simplicity.*
> *Dennis Ritchie*

NO MAGIC

Before you commit to a framework, make sure you could write it

Simpler version

Some assembly

REMOVE IS BETTER THAN ADDING

Don’t let existing code dictate future code

Be ready to refactor

Build a *state of art* namespace

TDD is a design process

Prove the parts to prove the hole

Invert dependencies

Use gateways

What make testing harder is a bad in itself

Tests are the best documentation

```
test$method_$expectedResult_$conditions
```

Don’t play with your toy’s toys

If you need to change an object’s state, get the object to do it for you

Any method of an object should call only methods belonging to:

- itself;
- any parameters received;
- any objects it creates and any directly held component objects

One should build upon interfaces

> *OO languages replace function pointers with convenient polymorphism Robert C. Martin*

Closure protocols and haskell type classe

Benefits

- Easier to maintain (no unexpected behaviors)
- Performance gain
- The inversion of source code and run time dependencies

MANY LITTLE CLASSES VS FEW BIG ONES

Any regular system will contain a vast quantity of logic

The same amount of business logic to care of

You prefer your tools being organized in boxes with little compartments and good names?

Or only a compartment and all inside?

Or the “first five principles” by Michael Feathers.

To make systems that are: is easy to maintain and extend over time

SINGLE RESPONSIBILITY PRINCIPLE (SRP)

If you can think of more than one reason for changing a class, then that class has more than one responsibility

```php
<?php
interface UserInterface
{
    public function setId($id);
    public function getId();
    public function setName($name);
    public function getName();
    public function findById($id);
    public function insert();
    public function update();
    public function delete();
}
```

OPEN CLOSE PRINCIPLE (OC)

The interface is closed to modification and new implementation implement that interface

Rectangle, Square

Relates to composite reuse

LISKOV SUBSTITUTION (LS)

It's possible to change subclasses without breaking the program

Usually `instanceof` is a sign of code smell

Interface segregation principle (ISP)

It's better more interfaces than less

Clients should not be forced to depend upon interfaces that they don't use

```
abstract class AbstractPostgreSQLDriver implements
Driver,
ExceptionConverterDriver,
VersionAwarePlatformDriver
```

One should depend only on abstractions.

For PHP:

- https://github.com/container-interop/container-interop
- https://github.com/auraphp/Aura.Di
- https://github.com/zendframework/zend-servicemanager

Seven code qualities premisses:

- Cohesion
- Loose coupling
- No redundancy
- Encapsulation
- Testability
- Readability
- Focus

Benefits

Finding bugs is much easier

If you have more than one indentation level you have more than one abstraction level

2 - don't use else keyword

Else's encourages the inclusion of more intermediate if's

When possible avoid even if's

Use polymorphism instead

3 - WRAP ALL PRIMITIVES AND STRINGS

Small objects make programs more maintainable

They serves as a container for logic that otherwise would be sparse

Any class with a collection shouldn't contain other member variables

5 - ONE DOT PER LINE

Bad:

```
this->myMemberObject
    ->myMemberObjectMemberObject
    ->doFoo();
```

Good:

```
this->myMemberObjectMemberObject
    ->functionThatDoFooToo();
```

Relates to Law of Demeter

6 - don't abbreviate

Abbreviation because of exhaustive use?

DRY violation

Too long names?

Maybe a SRP problem

7 - KEEP ALL ENTITIES SMALL

No classes over 50 lines and no packages over 10 files

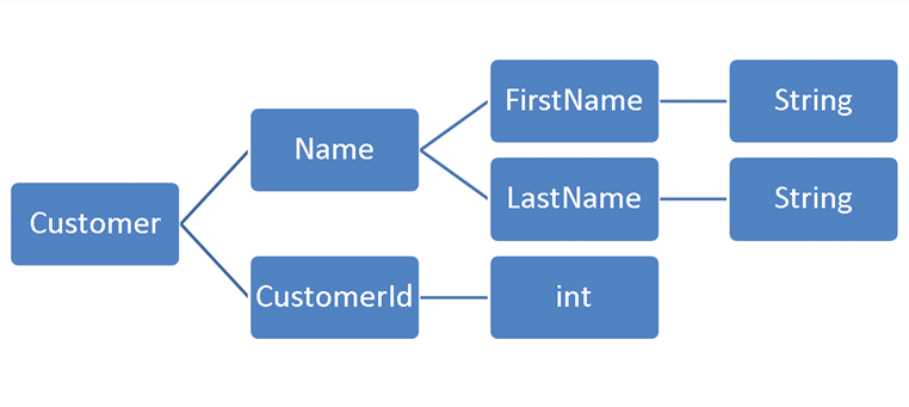

Figure 3: Instance variables

9 - no getters/setters/properties

Encapsulate everything

Make objects change dependencies by themselves

Invert Dependencies

- Codacy: https://www.codacy.com/
- https://github.com/adoy/vim-php-refactoring-toolbox
- `https://github.com/object-calisthenics/phpcs-calisthenics-rules`

Style Guides

- PSR2
- Zend
- Symphony

MORE?

- Orthogonality
- Don’t return nulls
- Metrics
- Code Review
- Error Handling

Much more.

1. Clean code: A hand book of Agile Software craftsmanship; Robert C. Martin
2. The pragmatical programmer; Andrew Hunt
3. Code Complete; Steve McConnell
4. Refactoring: Improving the Design of Existing Code;
5. Release It!: Design and Deploy Production-Ready Software; Michael T. Nygard

Those who do not remember the past are condemned to repeat it.
Jorge Agustin Nicolas Ruiz de Santayana y Borras

Quality is a team issue. Andy hunt.

Teams as a hole should not tolerate broken windows.

Clean code is not about perfection.. It's about honesty.

We made our best to leave the camp cleaner than we find it?

But if we aim for the 80% where code needs the most. We are good enough.

Parts not critical to performance must be clean - not optimized.

> *The best programmers are 28 times best than the worst ones. Robert Glass, Facts and Fallacies of Software Engineering*

There's always room for improvement.

QUESTIONS?

Contact info

E-mail: contato@jeancarlomachado.com.br

IRC: JeanCarloMachado

THANKS