

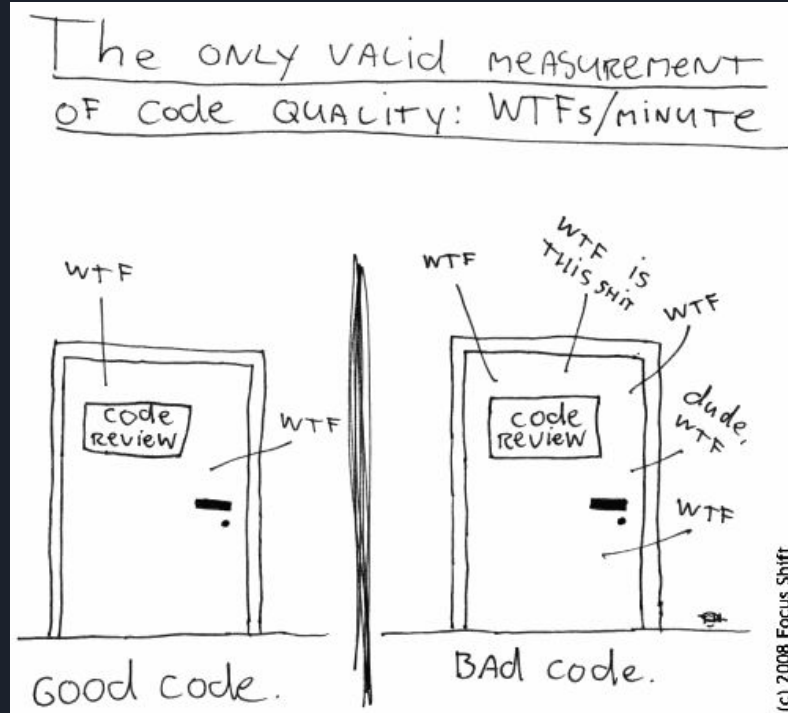


Código Limpio

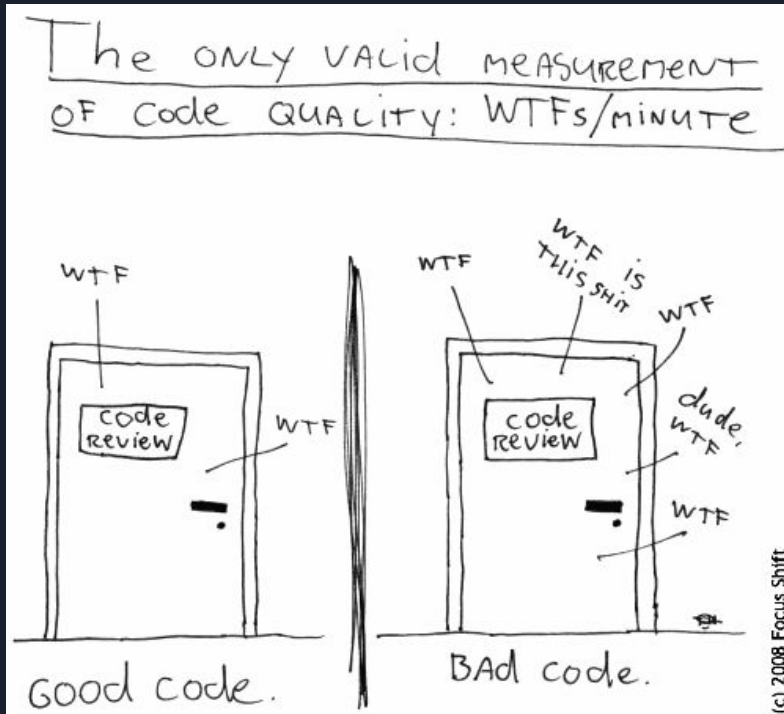
Buenas prácticas de programación

Leticia Cruz

¿Por qué tener un código limpio?



¿Por qué tener un código limpio?



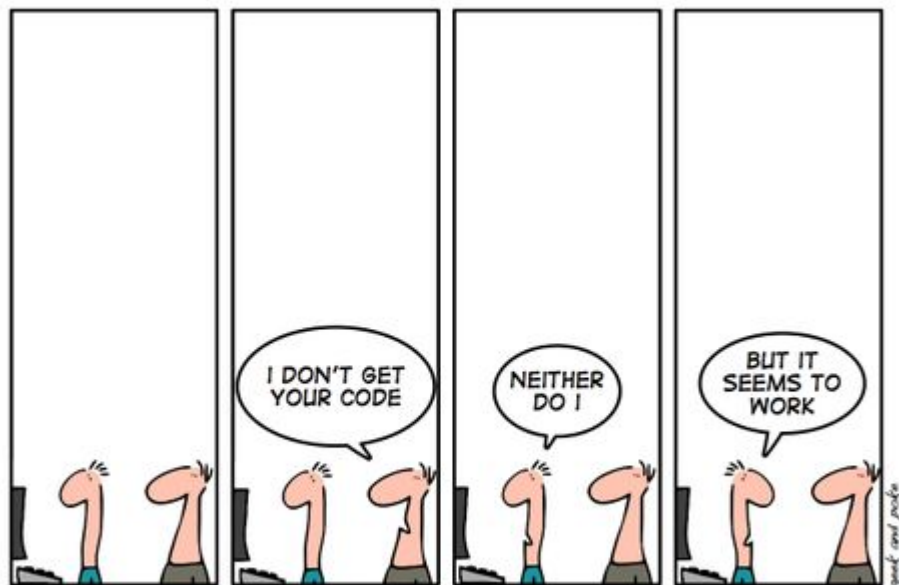
¿Qué puerta representa tu código?

¿Qué puerta representa a tu equipo o empresa?


¿Por qué estamos en ese cuarto?

¿Es solamente una revisión normal de código o encontramos una y otra vez terribles problemas inmediatamente después de que salimos a producción?

¿Depuramos nuestro código en estado de pánico estudiando detenidamente el código que juramos funcionaba bien?



THE ART OF PROGRAMING



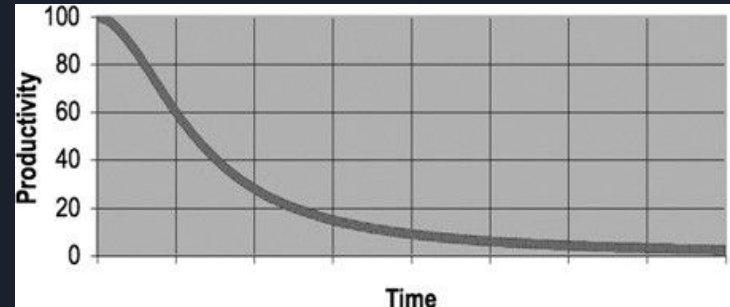
¿Por qué preocuparnos por tener un código limpio?

Aún cuando ya se estén diseñando lenguajes de programación que escriban código, siempre vamos a tener la necesidad de darle instrucciones a una máquina y eso... es código.

Bad Code

- It was the bad code that brought the company down.
- We've all looked at the mess we've just made and then have chosen to leave it for another day.
- We've felt the relief of seeing our messy program work and deciding that a working mess is better than nothing.
- LeBlanc's law: Later equals never.

The Total Cost of Owning a Mess





Attitude


Have you ever waded through a mess so grave that it took weeks to do what should have taken hours?

Have you seen what should have been a one-line change, made instead in hundreds of different modules?

Why does good code rot so quickly into bad code?

Explanations:

- Complaining about the schedules that were too tight to do things right.
- We blather about stupid managers and intolerant customers.
- We are unprofessionals.

- 
- How this mess be *our* fault?
 - What about the requirements?
 - What about the schedule?
 - What about the stupid managers and the useless marketing types?
 - Don't they bear some of the blame?

No.

- ***Managers & marketers:*** need from us promises & commitments.
- ***Users:*** Validate with us the way the requirements will fit into the system.
- ***Project Managers:*** look to us to help work out the schedule.

In general, we share a great deal of the responsibility for any failures; especially if those failures have to do with bad code.



"But wait!" you say. "If I don't do what my manager says, I'll be fired."

Probably not. Most managers want the truth, even when they don't act like it.

Most managers want good code, even when they are obsessed with the schedule. They may defend the schedule and requirements with passion; but that's their job. It's *your* job to defend the code with equal passion.

Ignaz Semmelweis in 1847 recommended to physicians hand-washing.

It was rejected because the doctors were too busy and wouldn't have time to wash their hands between patient visits.



What is clean code?



Bjarne Stroustrup, inventor of C++

- Code elegant and efficient.
- Make it hard for bugs to hide.
- Dependencies minimal to ease maintenance.
- Error handling complete according to an strategy.
- **Clean code does on thing well.**





Grady Booch, author of Object Oriented Analysis and Design with Applications

- Simple and direct.
- Clean code is like well-written prose.
- Straightforward lines of control

Dave Thomas, godfather of the Eclipse strategy

- Clean code can be read and enhanced by a developer other than its original author.
- Relationship between cleanliness to tests.
- Code without tests, is not clean.



The Boy Scout Rule

Leave the campground cleaner than you found it.

Can you image working on a project where the code *simply got better* as time passed?

Do you believe that any other option is professional?

Indeed, isn't continuous improvement an intrinsic part of professionalism?



Meaningful Names

Use Intention-Revealing Names

The name of a variable, function, or class, should answer all the big questions:

- Why it exists?
- What it does?
- How it is used?

BAD CODE

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

If a name requires a comment, then the name does not reveal its intent.

CLEAN CODE

```
int elapsedTimeInDays;
```

```
int daysSinceCreation;
```

```
int fileAgeInDays;
```

Meaningful Names

Meaningful Names



- Use Pronounceable Names : Bad

```
class DtaRcrd102 {  
    private Date genymdhms;  
    private Date modymdhms;  
    private final String pszqint = "102";  
    /* ... */  
}
```

- Use Pronounceable Names : Good

```
class Customer {  
    private Date generationTimestamp;  
    private Date modificationTimestamp;  
    private final String recordId = "102";  
    /* ... */  
};
```




Functions

- Small!
 - From 20-30 lines to 3-4 lines.
 - The indent level of a function should not be greater than one or two.
- Do one thing
 - Functions should do one thing. They should do it well. They should do it only.
- Use descriptive names
 - Each routine turns out to be pretty much what you expected.
 - The smaller and more focused a function is, the easier it's to choose a descriptive name.
 - Hunting for a good name sometimes results in a favorable restructuring of the code.
- Function Arguments
 - The ideal number of arguments for a function is zero. Then one or two. Three arguments should be avoided where possible. More than three requires special justification.
 - Arguments result as very hard when testing.
 - DRY: Don't Repeat Yourself.



Comments

"Don't comment bad code, rewrite it." Kernighan and Plaugher.

- Truth can only be found in one place: the code.
- Rather than spend your time writing the comments that explain the mess you've made, spend it cleaning that mess.
- Explain yourself in code.

Good comments:

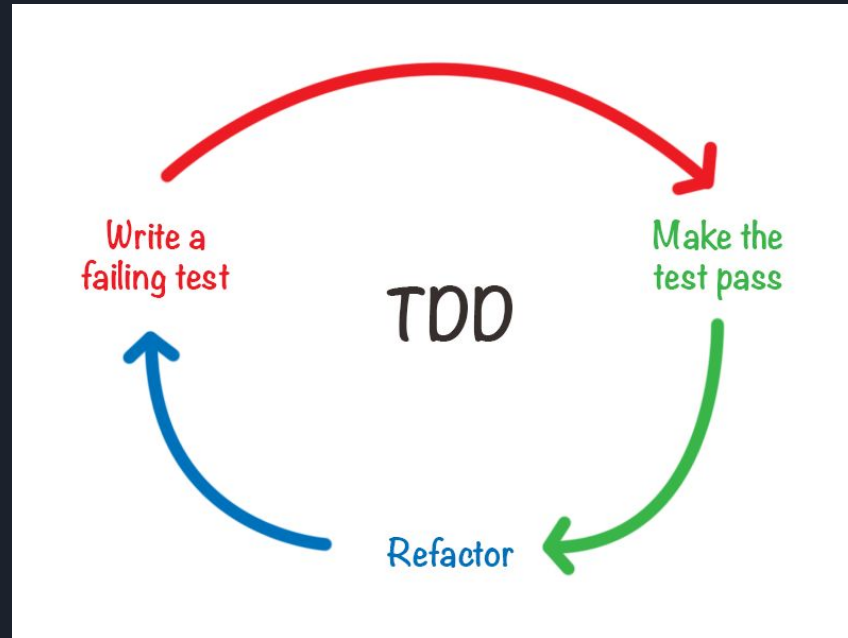
- Informative comments.
- Explanation of Intent.
- Clarification.
- Warning of Consequences.
- TODO Comments.

Bad comments:

- Redundant comments.
- Misleading comments.
- Noise comments.
- Warning of Consequences.
- Commented-out Code.

TDD: Test Driven Development

TDD is a development technique where you must first write a test that proves that the implementation works or fails.





The Three Laws of TDD

1. First Law: You may not write production code until you have written a failing unit test.
2. Second Law: You may not write more of a unit test than is sufficient to fail, and not compiling is failing.
3. You may not write more production code than is sufficient to pass the current failing test.

Keep the Tests Clean

- Tests must change as the production code evolves. The dirtier the tests, the harder they are to change.
- Having dirty tests is equivalent to, if not worse than, having no tests.
- The code is just as important as production code.



F.I.R.S.T.

Fast: They should run quickly.

Independent: They should not depend on each other.

Repeatable: Tests should be repeatable in any environment.

Self-Validating: Tests should have a boolean output.

Timely: Tests should be written just before the production code that makes them pass.
Otherwise, you may find the production code to be hard to test.



Benefits of TDD

- You have robust production code.
- TDD eradicates fear of change.
- Your production code is able to support changes, then it's scalable.
- Your production code is maintainable.
- Your production code is reusable.
- You can improve architecture and design without fear.

Reading recommendation: <https://medium.com/javascript-scene/tdd-changed-my-life-5af0ce099f80>

How do I write Clean Code?

I will not write any more bad code
I will not write any more bad code
I will not write any more bad code
I will not write any more bad code
I will not write any more bad code
I will not write any more bad code
I will not write any more bad code
I will not write any more bad code
I will not write any more bad code
I will not write any more bad code
I will not write any more bad code





Writing good software is a continuous quality-improvement process.

Be updated with design patterns and other best practices.

Review code from other programmers.

A programmer who writes clean code is an artist who can take a blank screen through a series of transformations until it is an elegantly coded system.

