PGS SOFTWARE

Code quality in Python

A reasonable approach to measuring code quality in your projects

Radosław Ganczarek - 2019

Hi! It's me again!

Radosław Ganczarek (Rad)

EuroPython 2015 (Bilbao)

Reasonable approach?

This talk will be

ABOUT

- Python 3
- Up to date tools
- Tools only for Python
- Code quality checking
- A bit about testing

NOT ABOUT

- Python 2
- IDEs
- Framework-specific tools
- Tools not connected to code itself (e.g. pyaroma)
- Packages not published in PYPI (e.g. pyright)

Meet my friends!

Meet the Hobgoblin!

PEP-0008
“hobgoblins of little minds”

What would a Hobgoblin do?

- narrow minded
- lacks business perspective
- extreme
- rules above value

Meet Timmy!

- Another Python developer
- Your colleague in a project
- Very skilled
- Afraid of changes

Meet the Zen of Python

PEP-0020

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess.

There should be one and preferably only one obvious way to do it.

Although that way may not be obvious at first unless you're Dutch.

Now is better than never.

Although never is often better than *right* now.

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

Namespaces are one honking great idea let's do more of those!

Zen of Python

Beautiful is better than ugly

- Code quality starts from good-looking code
- Elegant line breaks
- Space

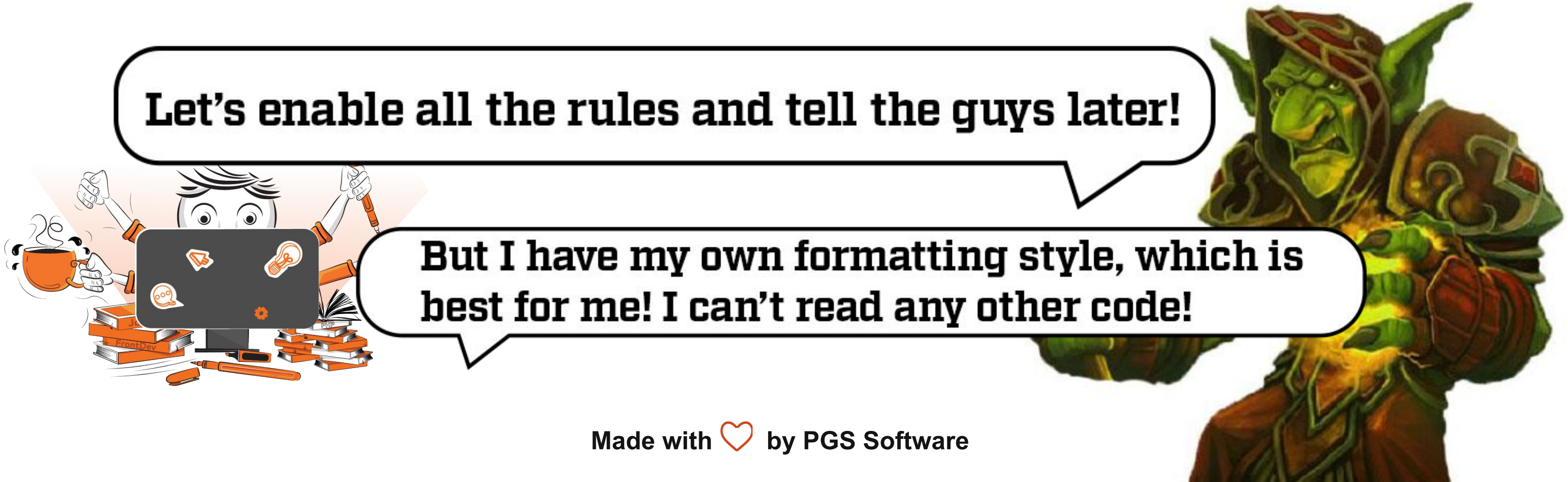
Formatters

Black

- pre-picked formatting rules
- isort support
- no configuration

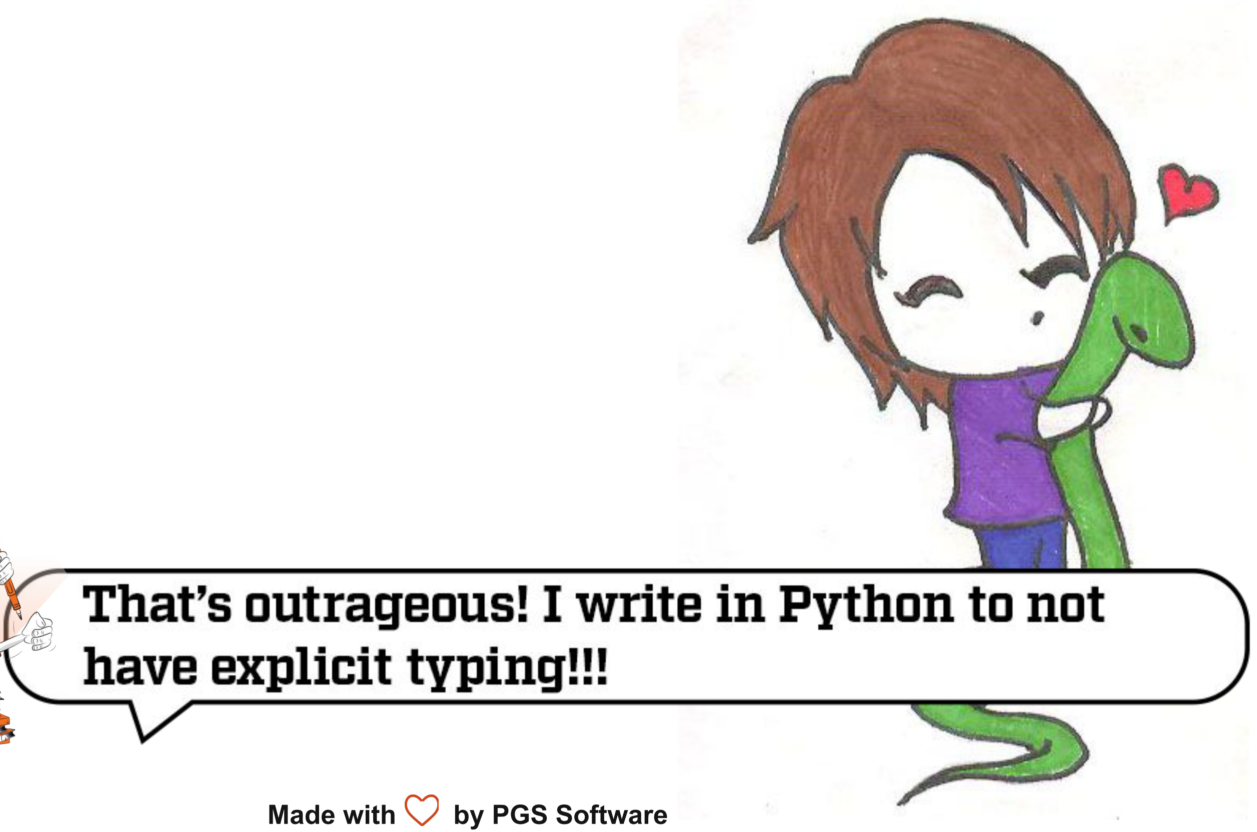
Yapf

- formatting styles
- many configuration options


Let's enable all the rules and tell the guys later!
But I have my own formatting style, which is best for me! I can't read any other code!

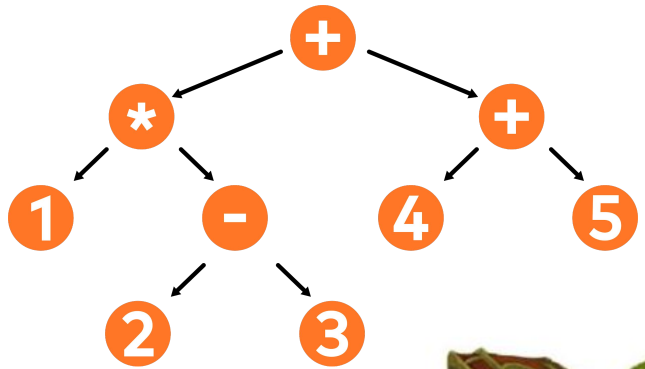
Explicit is better than implicit

- PEP-0484
- PEP-0526
- mypy vs pyre


That's outrageous! I write in Python to not have explicit typing!!!

Simple is better than complex

```
bellybutton
DeprecatedFnCall:
  description: `deprecated_fn` will
be deprecated in v9.1.2. Please use
`new_fn` instead.
  expr:
//Call[func/Name/@id='deprecated_fn']
  example: "deprecated_fn(*values)"
  instead: "new_fn(values)"
```



$$\begin{tikzcd}
 & & & + \arrow[dll] \arrow[drr] & & & \\
 & * \arrow[dl] \arrow[dr] & & & & + \arrow[dl] \arrow[dr] & \\
1 & & - \arrow[dl] \arrow[dr] & & 4 & & 5 \\
 & 2 & & 3 & & &
\end{tikzcd}$$

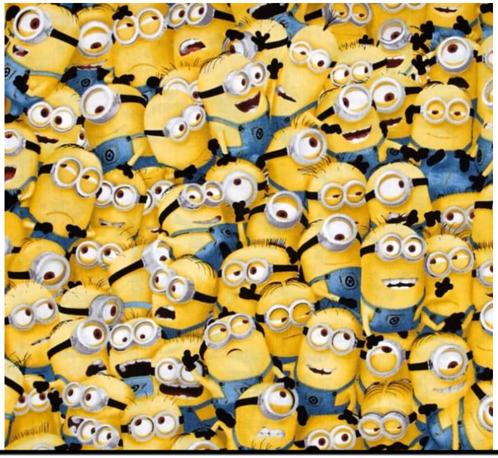
What a waste of time! Tools should provide ready to use rules

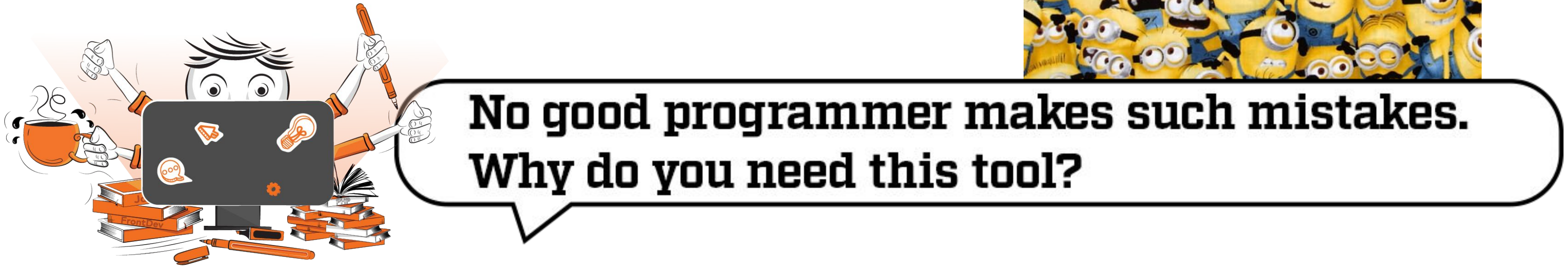
Complex is better than complicated

pylint

vs

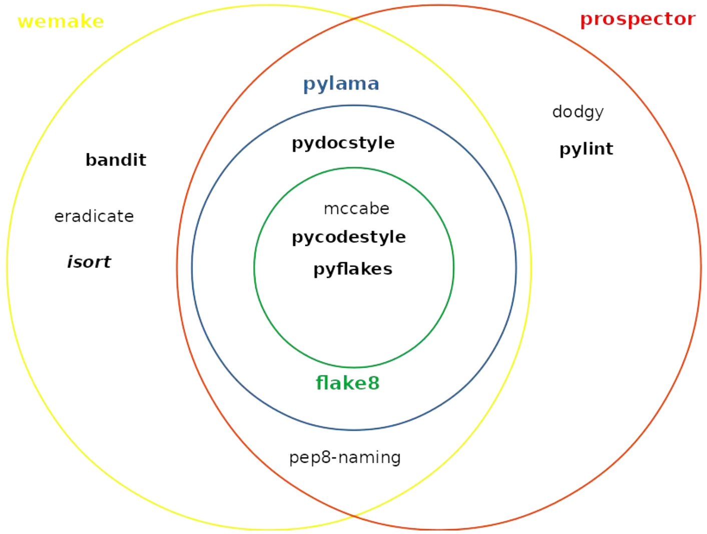
100 flake8 plugins




No good programmer makes such mistakes.
Why do you need this tool?

Flat is better than nested

- Tools and plugins
- Outdated tools
- Deprecated/unsupported tools
- Runtime errors
- Pre-set toolset
- One configuration file



Made with ♡ by PGS Software

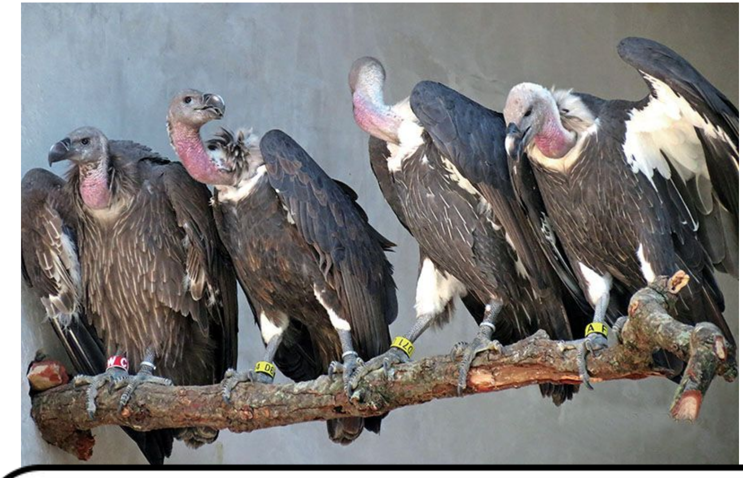
Sparse is better than dense

- diff-cover
- diff-quality
- See only what you broke
- Test coverage trap

It's unfair! What if I delete code?

Readability counts

- vulture



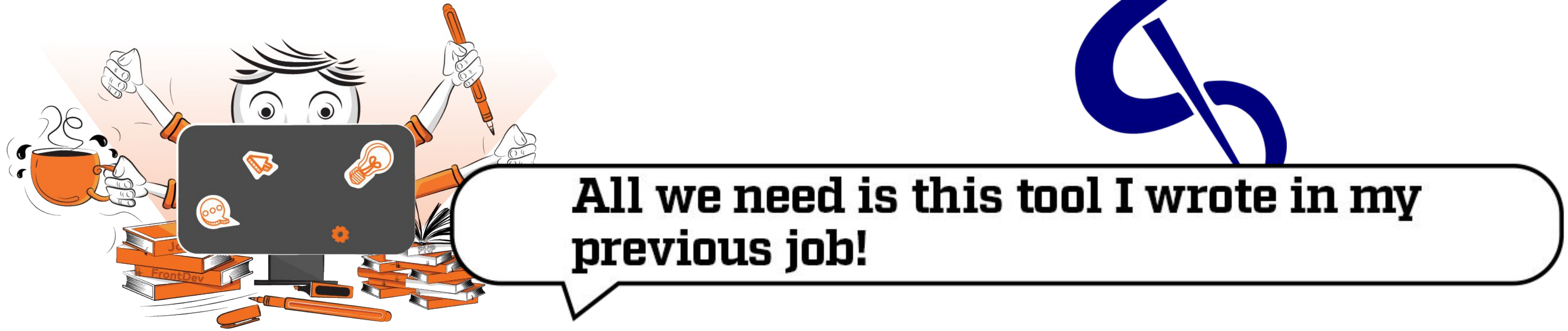
Remove all found things. We'll see if they were used when code crashes.

Special cases aren’t special enough to break the rules

- File discovery regex
- Bash find
- Line regex
- Is special case so special?

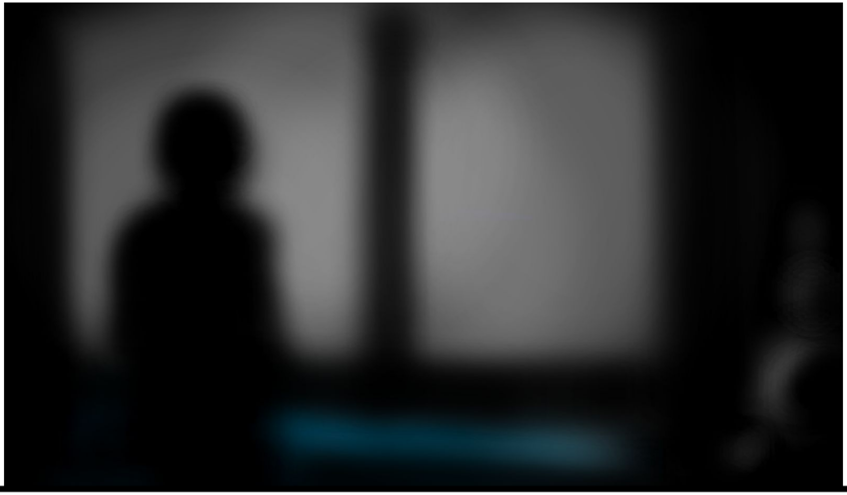
Although practicality **beats** purity

- pydiatra



Errors should never pass silently

- Code-checking CI
- tox
- Jenkins
- CI output



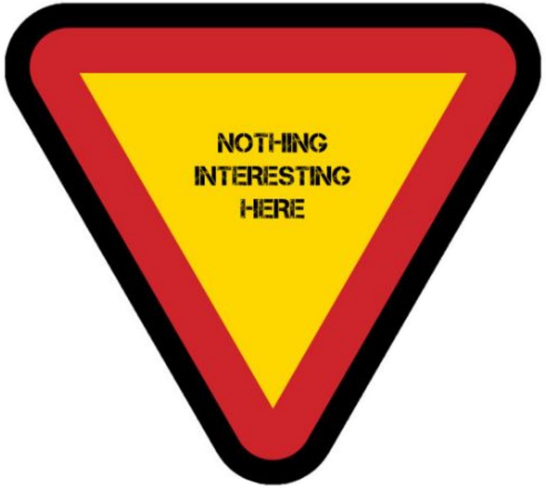
To give your people better feedback about their code, set up a dart launcher that will target the employee that broke the build!

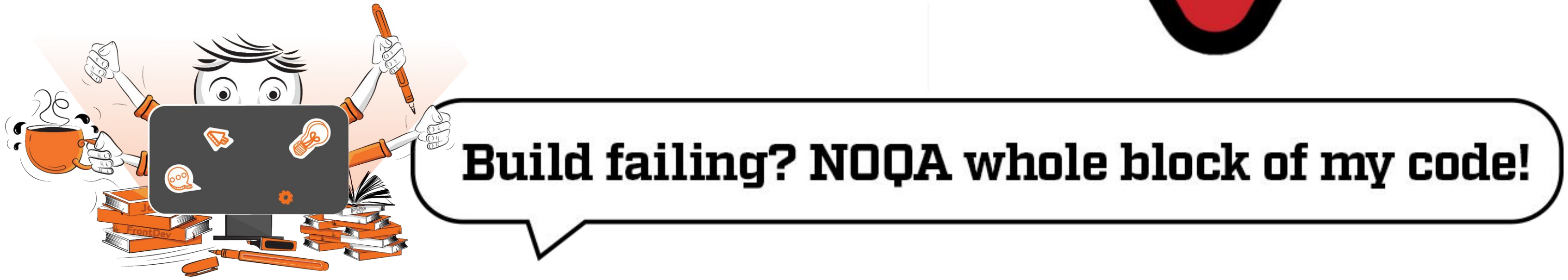
Unless explicity silenced

# noqa

# pylint: disable=missing-docstring

# nofmt





In the face of ambiguity, refuse the temptation to guess

- pytype

I write code without types, pytype adds types and the lead developer is happy. Win-win!

There should be one - and preferably only one - obvious way to do it

- bandit (no dodgy)
- mypy OR pyre OR pytype
- pycodestyle
- pydocstyle OR sphinx docstring check
- pyflakes+mccabe OR pylint
- isort
- black OR yapf

Optional:

- pydiatra
- vulture
- bellybutton
- autoflake
- autopep8

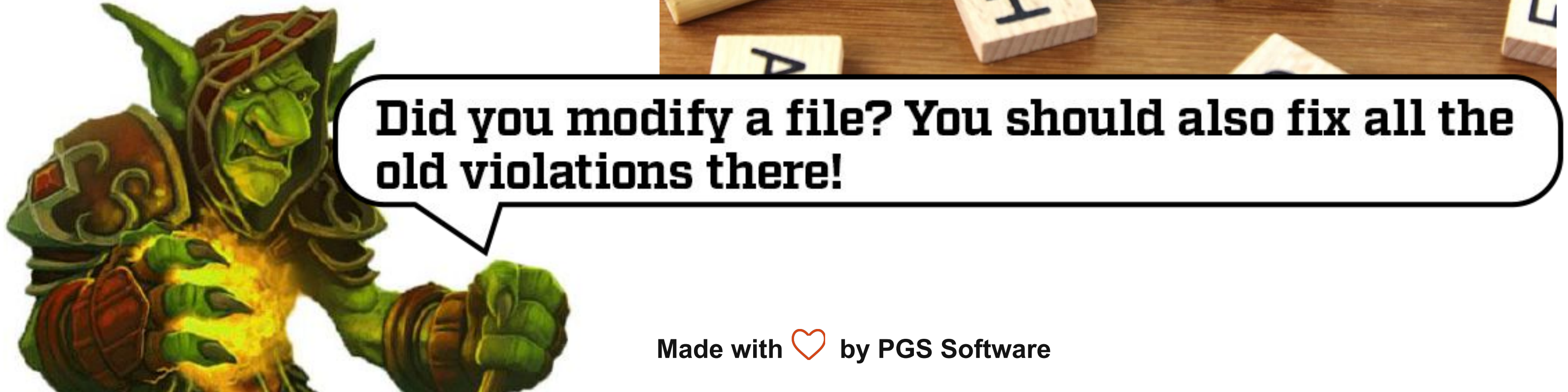
The more tools we use, the better our codebase is!

Although that way may not be obvious at first - unless you're Dutch

Now is better than never

How to start?

- autopep8
- autoflake
- isort
- pycodestyle
- pylint


Did you modify a file? You should also fix all the old violations there!

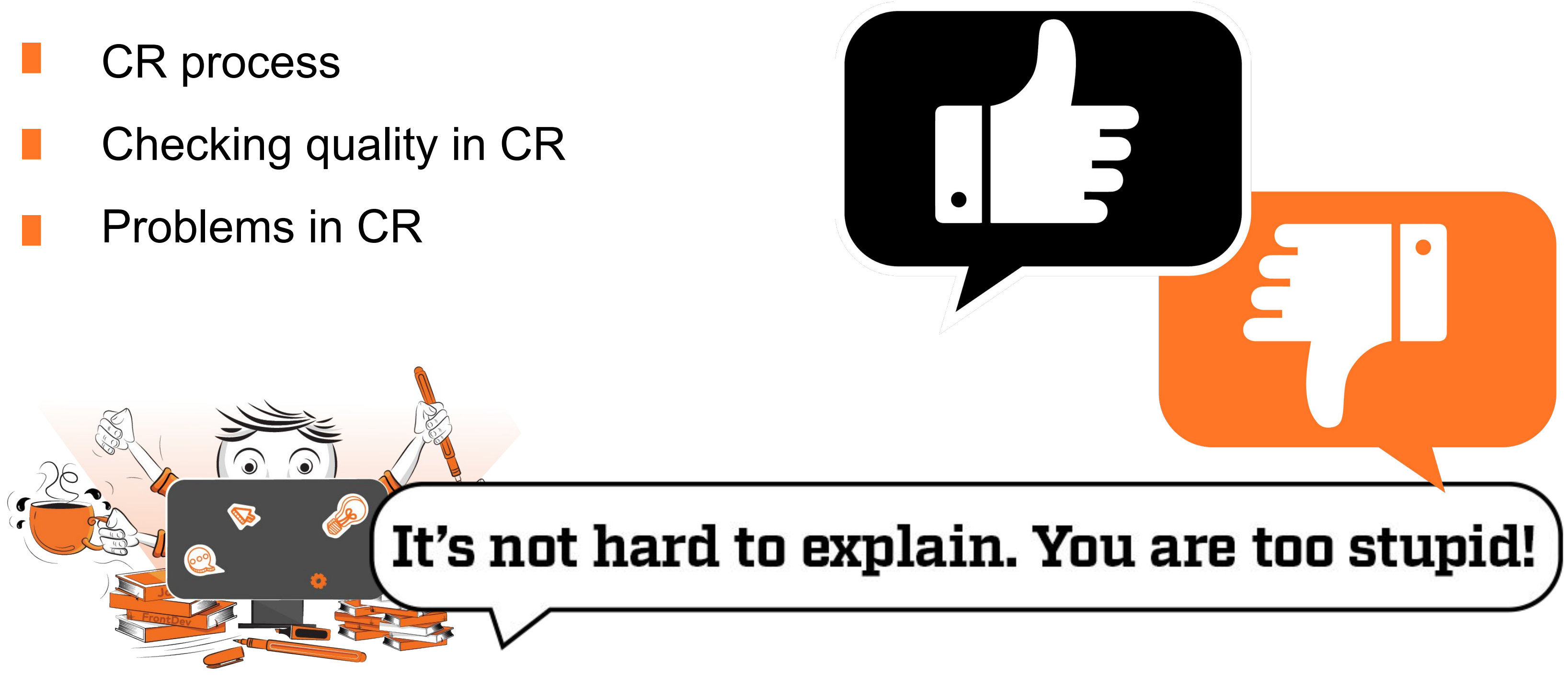
Although never is often better than *right* now

- You vs the team
- Business perspective
- Agreeing on rules

We've got sooo many more things to do!

If the implementation is hard to explain, it's a bad idea

- CR process
- Checking quality in CR
- Problems in CR



If the implementation is easy to explain, it may be a good idea

Advantages:

- getting rid of checkable mistakes
- maintaining uniform style
- following best practices
- readability
- maintainability
- enhanced refactoring

Disadvantages:

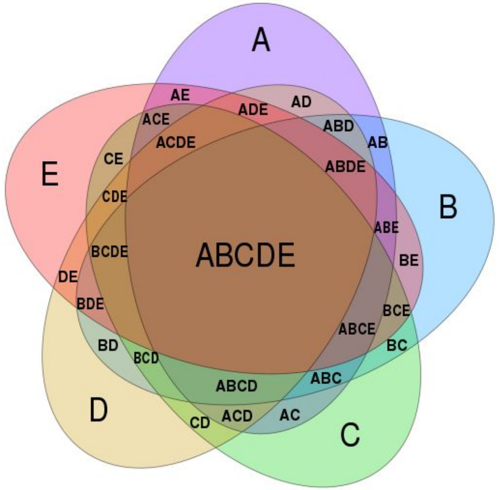
- additional time
- false positives
- converting people and project

Namespaces are one honking great idea - let's do more of those!

Error groups

github.com/PyCQA

github.com/mre/awesome-static-analysis#python



It's just the beginning of your road to perfect code! Hahahahaha!

Next steps

After all, you want to become this

Not this

Thank you!

Radosław Ganczarek
Software Developer & Team Leader
@ **PGS Software**
pgs-soft.com

Any questions?
Want to send feedback?
radoslaw@ganczarek.in