

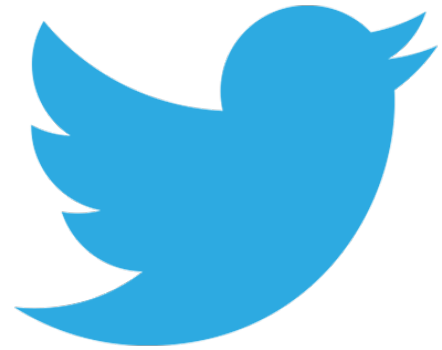


NINA ZAKHARENKO

CODE REVIEW SKILLS FOR PYTHONISTAS

 @NNJA

[bit.ly/
codereviewpy](https://bit.ly/codereviewpy)



LIVETWEET

USE #EUROPYTHON

@NNJA

WHAT WE'LL LEARN TODAY [1/2]

- > What are the proven benefits of review?
- > Setting standards
- > Time saving tools and automation options for Python

WHAT WE'LL LEARN TODAY [2/2]

- > How to review code helpfully
- > How to submit PRs for maximum impact
- > Use code review to build a stronger team

WHAT WILL **YOU** TAKE AWAY FROM THIS TALK?

- > **Novice** - Comprehensive overview of code review best practices
- > **Intermediate** - Tooling & automation
- > **Advanced** - Hardest part - the people factor

NOT ONE SIZE FITS ALL!

- > team size

 - > 2 vs 10

- > product type

 - > agency vs in-house vs open source

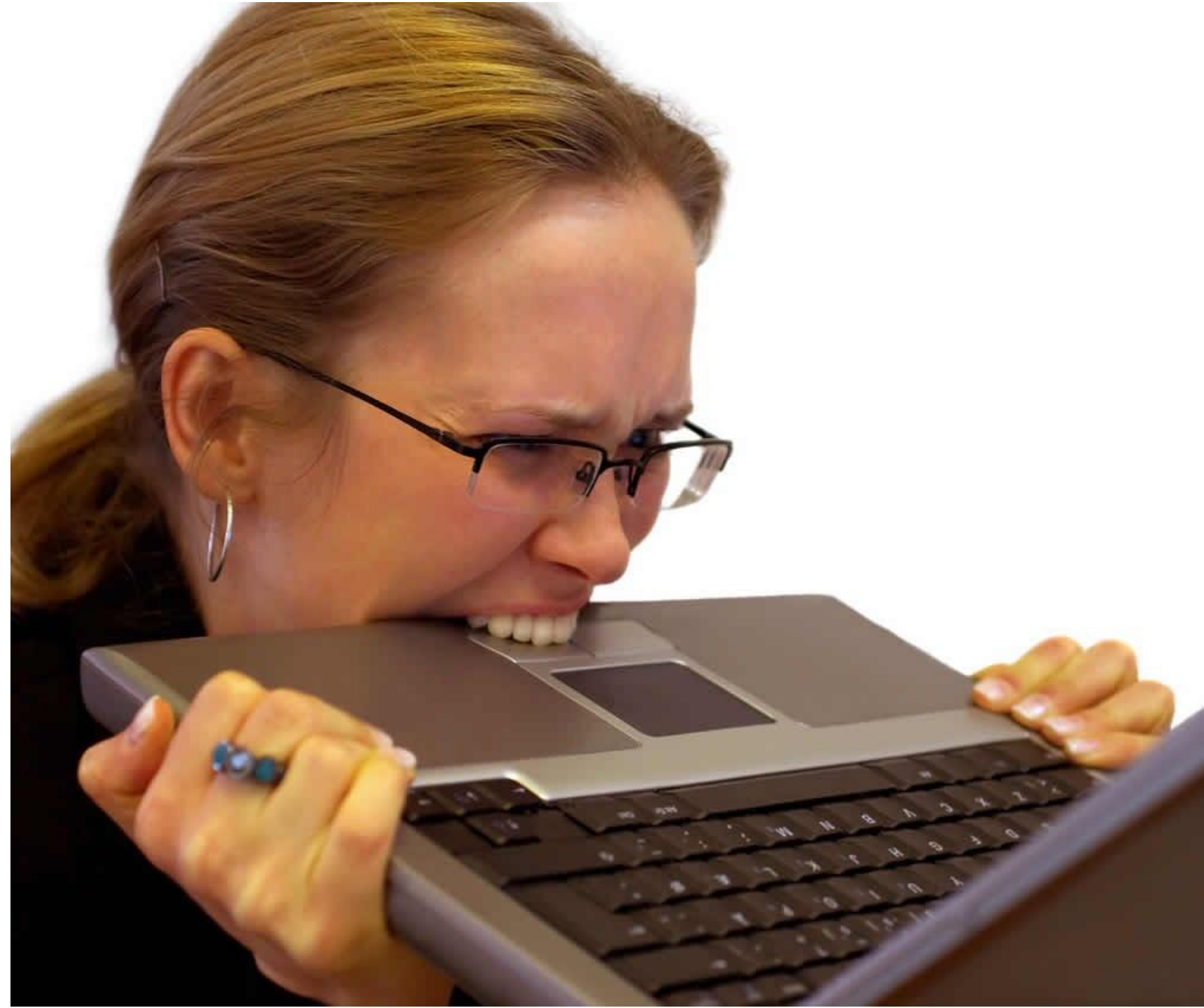
NOT ONE SIZE FITS ALL!

> defect tolerance

> jet engines vs mobile games

WHY CODE REVIEW?

CODE REVIEWS CAN BE FRUSTRATING



 @nnja

APPARENT CODE REVIEW FRUSTRATIONS

- > Adds time demand
- > Adds process
- > Can bring up team tensions
- > “smart” devs think they don’t need it 🙄

CODE REVIEW BENEFITS

FIND BUGS & DESIGN FLAWS

- > Design flaws & bugs can be identified and remedied before the code is complete
- > Case Studies on Review⁵:
- > ↓ bug rate by 80%
- > ↑ productivity by 15%

⁵ blog.codinghorror.com/code-reviews-just-do-it/

THE GOAL IS TO FIND
BUGS BEFORE YOUR
CUSTOMERS DO

SHARED OWNERSHIP & KNOWLEDGE

- > We're in this together
- > No developer is the only expert

LOTTERY FACTOR



Jeff Stein ✓
@JStein_WaPo

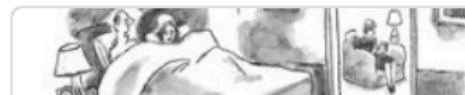
Follow



When the NYC subway vending machines go down, there's apparently only one guy who knows how to fix them.

His name is Miguel, he lives in Port Jarvis (3 hrs from NYC), & apparently he likes to turn his cell phone off on the way home.

Via William Finnegan
[newyorker.com/magazine/2018/ ...](https://www.newyorker.com/magazine/2018/...)



Byford called I.T. and put the tech person on speaker. How quickly could they reboot

New Yorker: Can Andy Byford Save the Subways?

CODE REVIEW BENEFITS?

- > Find Bugs
- > Shared Ownership
- > Shared Knowledge
- > Reduce "Lottery Factor"

HOW?

CONSISTENT CODE

- > Your code isn't yours, it belongs to your company
- > Code should fit your company's expectations and style (**not your own**)
- > Reviews should encourage consistency for code longevity

CODE REVIEWS NEED TO BE UNIVERSAL & FOLLOW GUIDELINES

- > Doesn't matter how senior / junior you are
- > Only senior devs reviewing == bottleneck
- > Inequality breeds dissatisfaction

STYLE GUIDE

- > Distinguishes personal taste from opinion
- > Should be agreed upon beforehand
- > Go beyond PEP8
- > See: Google's pyguide.md or plone styleguide

FORMATTERS

- autopep8

- Black

- YAPF



DEMO @: black.now.sh

```
-if very_long_variable_name is not None and \  
-    very_long_variable_name.field > 0 or \  
-    very_long_variable_name.is_debug:  
- z = 'hello '+'world'  
+if (  
+    very_long_variable_name is not None  
+    and very_long_variable_name.field > 0  
+    or very_long_variable_name.is_debug  
+):  
+    z = "hello " + "world"
```

github.com/jpadilla/black-online



VS Code

> settings:

> "editor.formatOnSave": true

> "python.formatting.provider": "black"

> "python.formatting.blackArgs": ["--line-length", "100"]

> VS Code Black support [docs](#)

**CONSISTENT CODE IS EASIER TO MAINTAIN
BY A TEAM**

**CODE REVIEW IS DONE BY
YOUR PEERS & NOT MANAGEMENT**

DON'T POINT FINGERS!

WHEN CODE REVIEWS ARE POSITIVE.

DEVELOPERS

**DON'T EXPECT THEIR
CHANGES TO BE REVIEWED.**

**THEY WANT THEIR
CHANGES TO BE REVIEWED.**

LET'S REVIEW: CODE REVIEW FUNDAMENTALS

- > Universal code review
- > Performed by Peers
- > Style guides & formatters for consistency
- > No blame culture

**HOW SHOULD
WE CODE
REVIEW?**

**BE A GREAT
SUBMITTER**

**No need
to double check this
change list, if some
problems remain the
reviewer will catch
them.**



**No need to
look at this change list
too closely, I'm sure the
author knows what he's
doing.**



DON'T GET RUBBER-STAMPED.



DON'T BE CLEVER.
READABILITY COUNTS!

**GOOD CODE IS LIKE A GOOD JOKE.
IT NEEDS NO EXPLANATION.
– RUSS OLSEN**

STAGES OF REVIEW

- > 0: before PR submission
- > 1: PR submitted
- > 2: (optional) work in progress PR (30-50%)
- > 3: review almost done (90-100%)
- > 4: review completed

STAGE 0:
BEFORE SUBMISSION

PROVIDE CONTEXT (THE WHY)

- > What was the motivation for submitting this code?
- > Link to the underlying ticket/issue
- > Document why the change was needed
- > For larger PRs, provide a changelog
- > Point out any side-effects

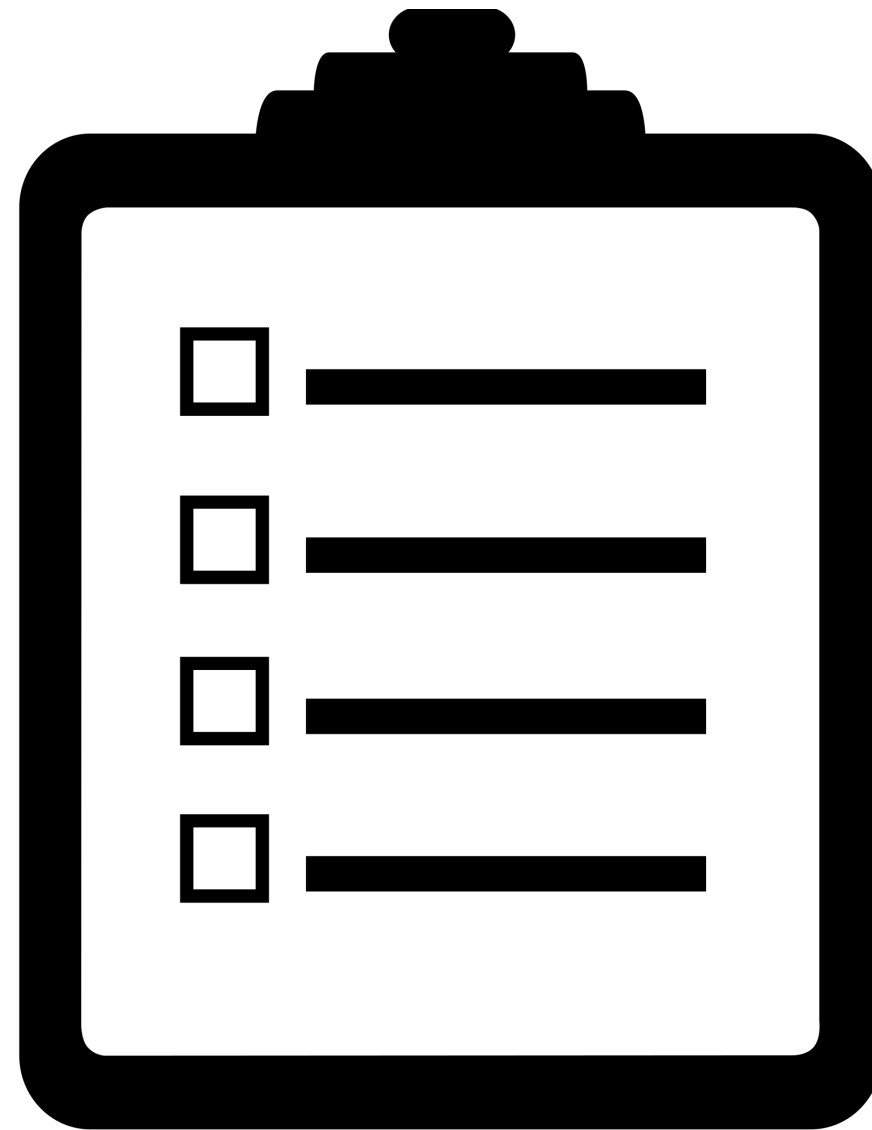
YOU ARE THE PRIMARY REVIEWER

- > Review your code with the same level of detail you would give giving reviews.
- > Anticipate problem areas.

THE PRIMARY REVIEWER

- > Make sure your code works, and is thoroughly tested.
- > Don't rely on others to catch your mistakes.

BEFORE SUBMITTING, TRY A CHECKLIST





SMALL STUFF

- > Did you check for reusable code or utility methods?
- > Did I remove debugger statements?
- > Are there clear commit messages?



BIG STUFF

- > Is my code secure?
- > Will it scale?
- > Is it maintainable?
- > Is it resilient against outages?

Tip: Read [The Checklist Manifesto](#)

**STAGE 1:
SUBMITTED**

YOU'RE STARTING A CONVERSATION

- > Don't get too attached to your code before the review process
- > Anticipate comments and feedback
- > Acknowledge you will make mistakes

STAGE 2:
(OPTIONAL)
WORK IN PROGRESS

SUBMIT **WORK IN PROGRESS** PULL REQUESTS

- > Open them your code is 30-50% done
- > Good idea for bigger features
- > Don't be afraid of showing incomplete, incremental work

WHEN CODE IS WORK IN PROGRESS

- > Feedback to expect:
 - > Architectural issues
 - > Problems with overall design
 - > Design pattern suggestions

STAGE 3:
ALMOST DONE

WHEN CODE IS ALMOST DONE

- > Feedback to expect:
 - > Nit Picks
 - > Variable Names
 - > Documentation & Comments
 - > Small Optimizations

ONE REVIEW PER PR

- > Solving multiple problems? Break them up into multiple PRs for ease of review.
- > Solved an unrelated problem? Make a new PR with a separate diff

PREVENT REVIEWER BURNOUT

- > Reviews lose value when they're more than 500 lines of code¹
- > Keep them small & relevant
- > If a big PR is unavoidable, give the reviewer extra time

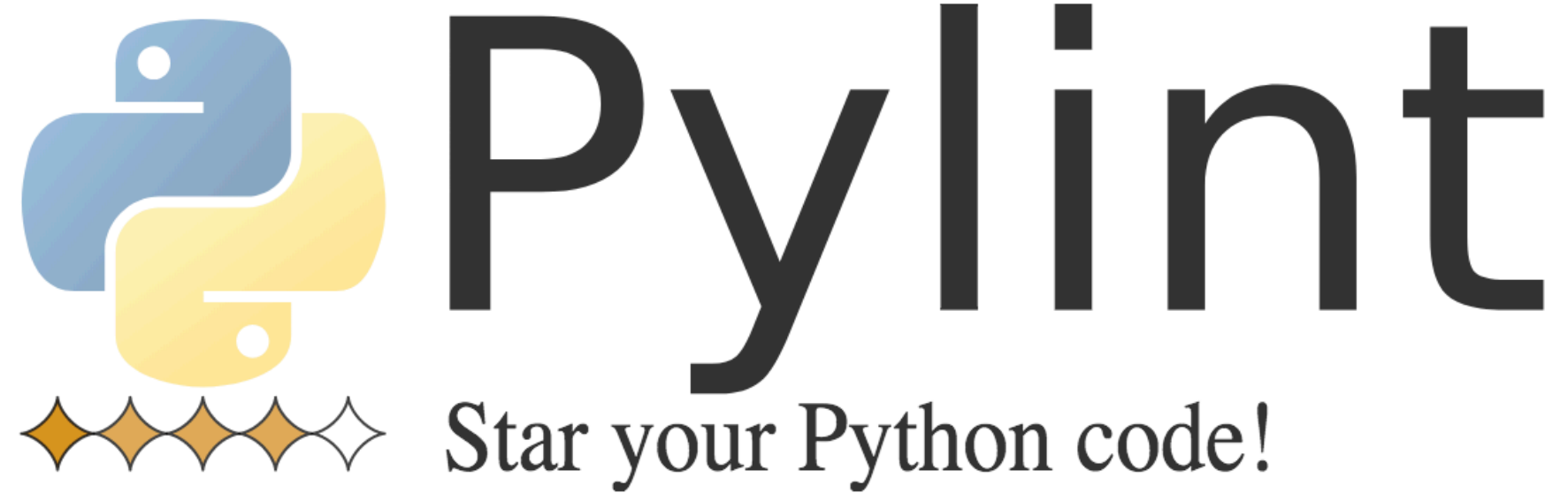
¹ <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/bosu2015useful.pdf>

✓ CHECK CODE
WITH AUTOMATED TOOLS

✓ LINTER

```
1 • def product(x, y):  
2     return x * y  
3  
4 • y = product(a, b)  
5
```

flake8 F821 undefined name 'a'
flake8 F821 undefined name 'b'



- > Coding standard
- > Error detection
- > Refactoring help
- > IDE & editor integration



PYLINT RULE: trailing-comma-tuple

```
foo = (1,  
       3,  
       4,  
       )
```

```
bar = 2,
```

USE VULTURE.PY TO FIND DEAD OR UNREACHABLE CODE

```
$ pip install vulture  
$ vulture script.py package/
```

or

```
$ python -m vulture script.py package/
```

github.com/jendrikseipp/vulture

Sample code

```
def foo():  
    print("foo")
```

```
def bar():  
    print("bar")
```

```
def baz():  
    print("baz")
```

```
foo()  
bar()
```

Run vulture.py

```
> python -m vulture foo.py  
foo.py:7: unused function 'baz' (60% confidence)
```

GIT PRE-COMMIT HOOKS

- > run linter
- > check syntax
- > check for TODOs, debugger statements, unused imports
- > enforce styling (autopep8, black formatter, sort imports, etc)
- > **option to reject commit if conditions don't pass**

pre-commit.com

pre-commit

A framework for managing and maintaining multi-language pre-commit hooks.

build passing coverage 100% build passing

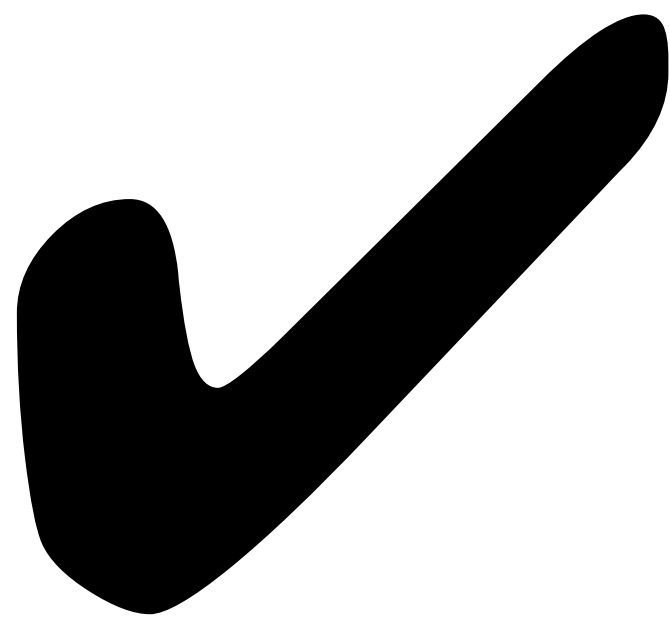
Star 2,109

Tweet

 @nnja

pre-commit.com

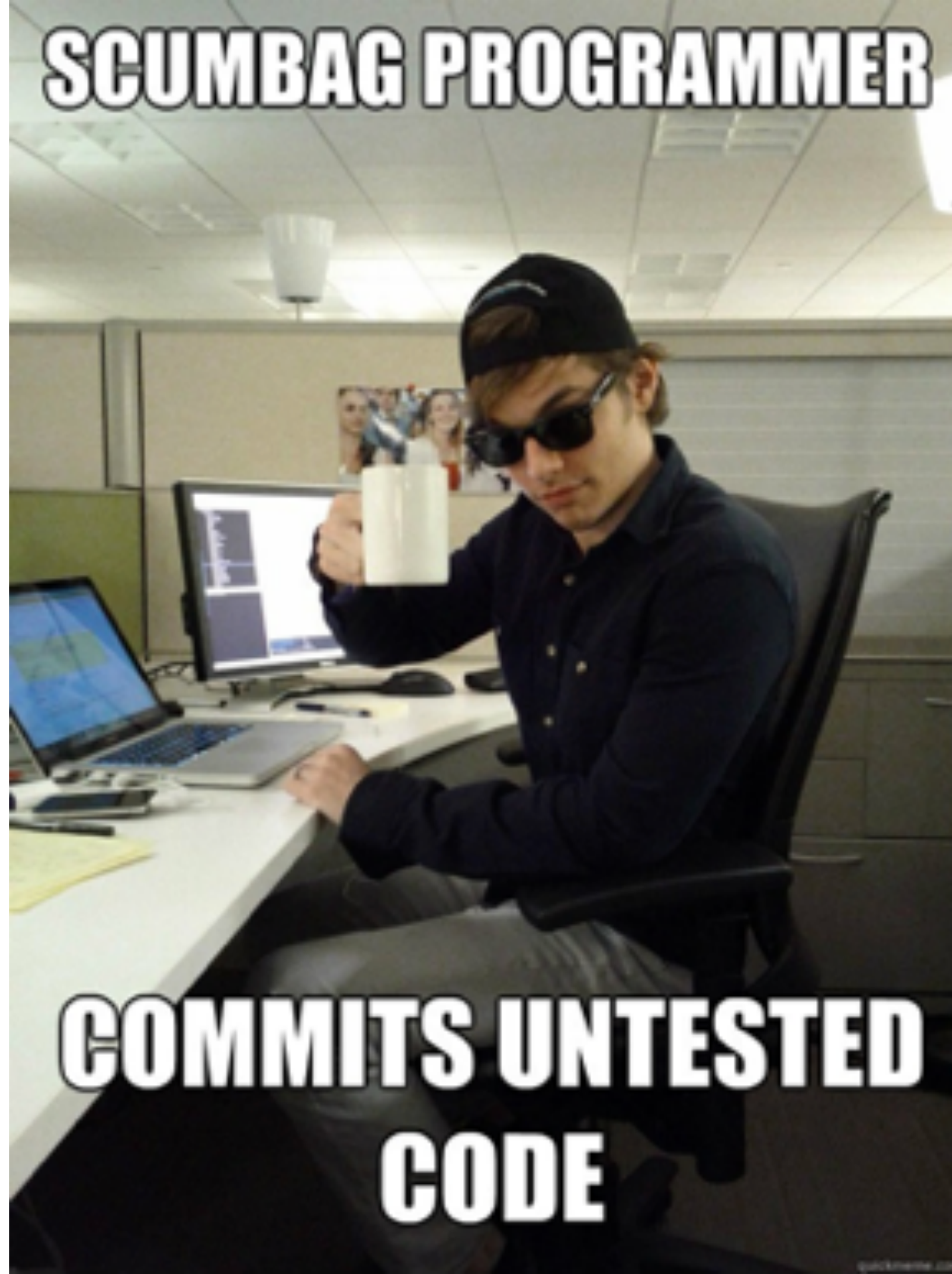
- > `autopep8-wrapper` - Runs `autopep8` over source
- > `flake8` and `pyflakes` - Run `flake8` or `pyflakes` on source
- > `check-ast` - Check whether files contain valid python
- > `debug-statements` - Check for debugger imports and `breakpoint()` calls



TESTS

- > Write them!
- > Don't know code health if tests are failing
- > Tests identify problems immediately


SCUMBAG PROGRAMMER




**COMMITTS UNTESTED
CODE**


✓ CONTINUOUS INTEGRATION


CPYTHON USES  VSTS





✓ **All checks have passed**
9 successful checks [Hide all checks](#)

✓  **VSTS: Linux-PR** — Linux-PR_20180725.27 succeeded [Details](#)


✓  **VSTS: Linux-PR-Coverage** — Linux-PR-Coverage_20180725.32 succeeded [Details](#)

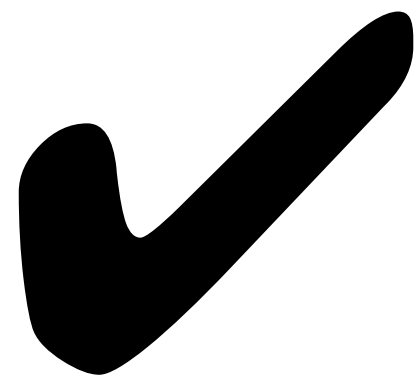
✓  **VSTS: Windows-PR** — Windows-PR_20180725.27 succeeded [Details](#)

✓  **VSTS: docs** — docs_20180725.35 succeeded [Details](#)

✓  **VSTS: macOS-PR** — macOS-PR_20180725.27 succeeded [Details](#)

✓ **This branch has no conflicts with the base branch**
Only those with [write access](#) to this repository can merge pull requests.

 @nnja



coverage.py

% OF CODE EXECUTED WHEN RUNNING A TEST SUITE

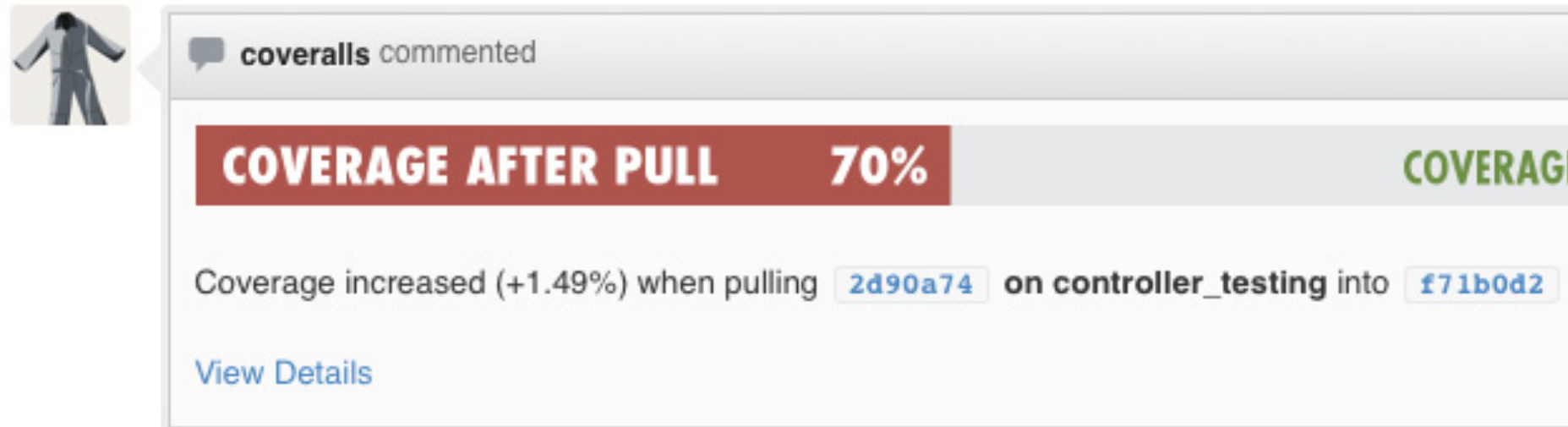
Coverage report: 87%

<i>Module</i>	<i>statements</i>	<i>missing</i>	<i>excluded</i>	<i>coverage</i>
mymath.py	9	3	0	67%
test_mymath.py	14	0	0	100%
Total	23	3	0	87%

coverage.py v4.1, created at 2016-07-18 15:04

✓ COVERAGE

- > Coverage tools integrate into GitHub
- > coverage.py
- > coveralls.io



STAGE 4:
REVIEW COMPLETE

BE RESPONSIVE

- > Reply to every comment
- > Common Responses:
 - > Resolved
 - > Won't Fix
 - > If you won't fix, make sure you've come to a mutual understanding with the reviewer

IT'S STILL A CONVERSATION

If there were comments, let your reviewer know when you've pushed changes and are ready to be re-reviewed.

DON'T BIKE-SHED

- > bikeshed.com
- > back & forth > 3 times? step away from the keyboard
- > talk instead!
- > record the results of the conversation in the PR

VS CODE LIVE SHARE

```
17 app.get('/', async function (req, res) {
18   · const data = await sentimentService();
19   · let sentimentWithLevel = [];
20   Amanda Silver
21   · for (let s in data.tweets) {
22     · let newTweet = {
23       · sentiment: s.sentiment,
24       · level: util.getHappinessLevel(s.sentiment)
25     · };
26     · sentimentWithLevel.push(newTweet);
27   · }
```

Real-time sharing in tools you love.

Share the full context of your code.

Collaboratively edit. Navigate independently.

Collaboratively debug. Inspect on your own.

[Download Extension](#)

**FIGHT FOR WHAT YOU BELIEVE. BUT
GRACEFULLY ACCEPT DEFEAT.**



**DON'T TAKE FEEDBACK PERSONALLY.
IT'S AN OPPORTUNITY FOR GROWTH.**

HOW TO BE A GREAT SUBMITTER?

- > Provide the why (context!)
- > Review your own code
- > Expect conversation
- > Submit in progress work

HOW TO BE A GREAT SUBMITTER?

- > Use automated tools
- > Be responsive
- > Accept defeat

#1: BE A GREAT REVIEWER



Why do you
think you're so
hostile in code
reviews?

If only I had
been
more popular in
High School.

Have Empathy

BE OBJECTIVE

'**THIS** METHOD IS MISSING A DOCSTRING'

INSTEAD OF

'**YOU** FORGOT TO WRITE A DOCSTRING'

ASK QUESTIONS DON'T GIVE ANSWERS

- > “Would it make more sense if... ?”
- > “Did you think about... ? ”

OFFER SUGGESTIONS

- > “It might be easier to...”
- > “We tend to do it this way...”

AVOID THESE TERMS

- > Simply
- > Easily
- > Just
- > Obviously
- > Well, actually...

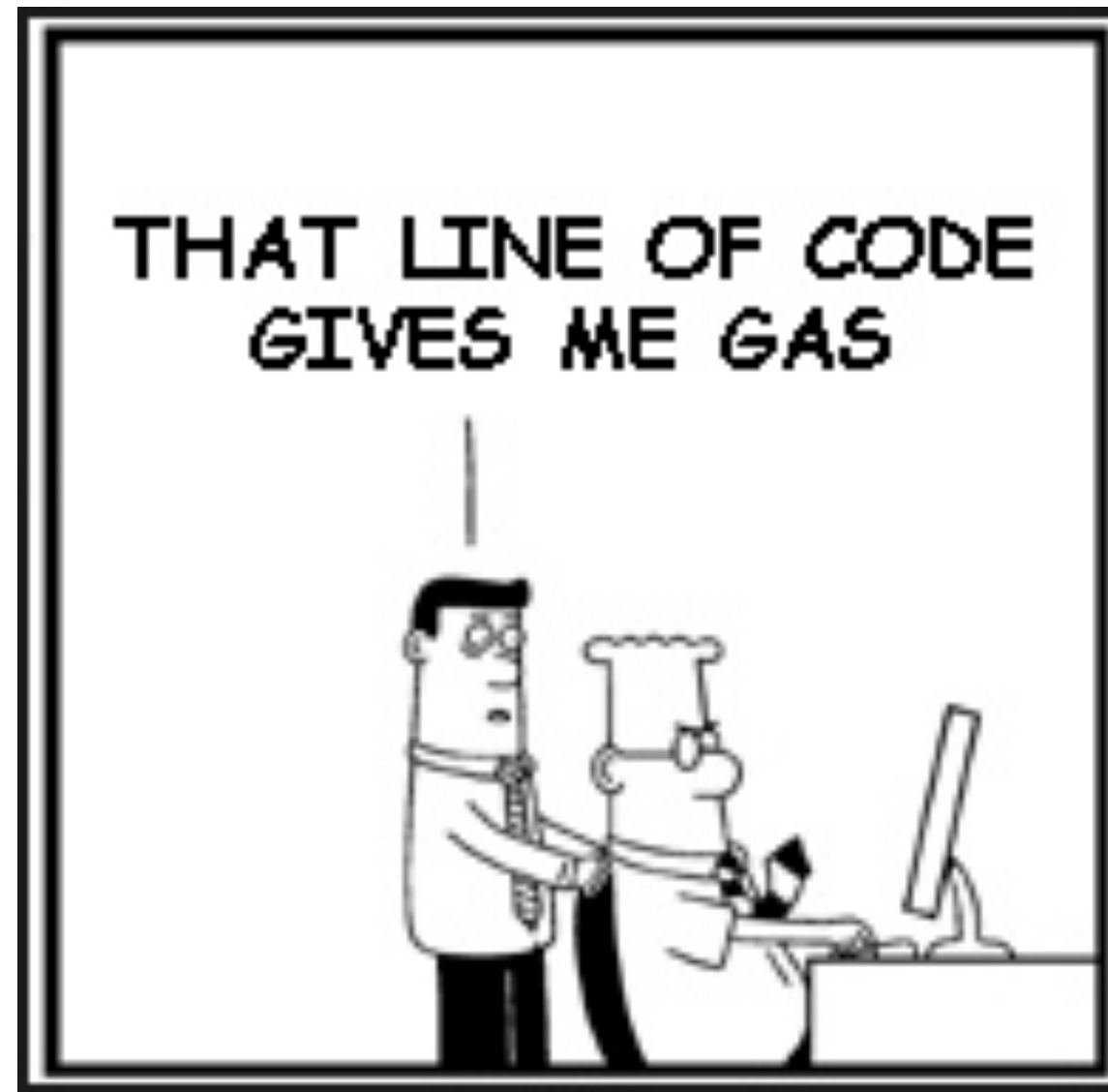
... **NOW. SIMPLY**



HAVE CLEAR FEEDBACK

- > Strongly support your opinions
- > Share **How & Why**
- > Link to supporting documentation, blog post, or stackoverflow examples

THIS IS **NOT** CLEAR FEEDBACK



COMPLIMENT GOOD WORK AND GREAT IDEAS



nnja commented 2 minutes ago



DON'T BE A PERFECTIONIST



Rebecca Turner

@ReBeccaOrg

Follow



4 The goal is better code, not "exactly the code you would have written"

11:31 AM - 22 Jul 2015

14 Retweets **29** Likes



DON'T BE A PERFECTIONIST

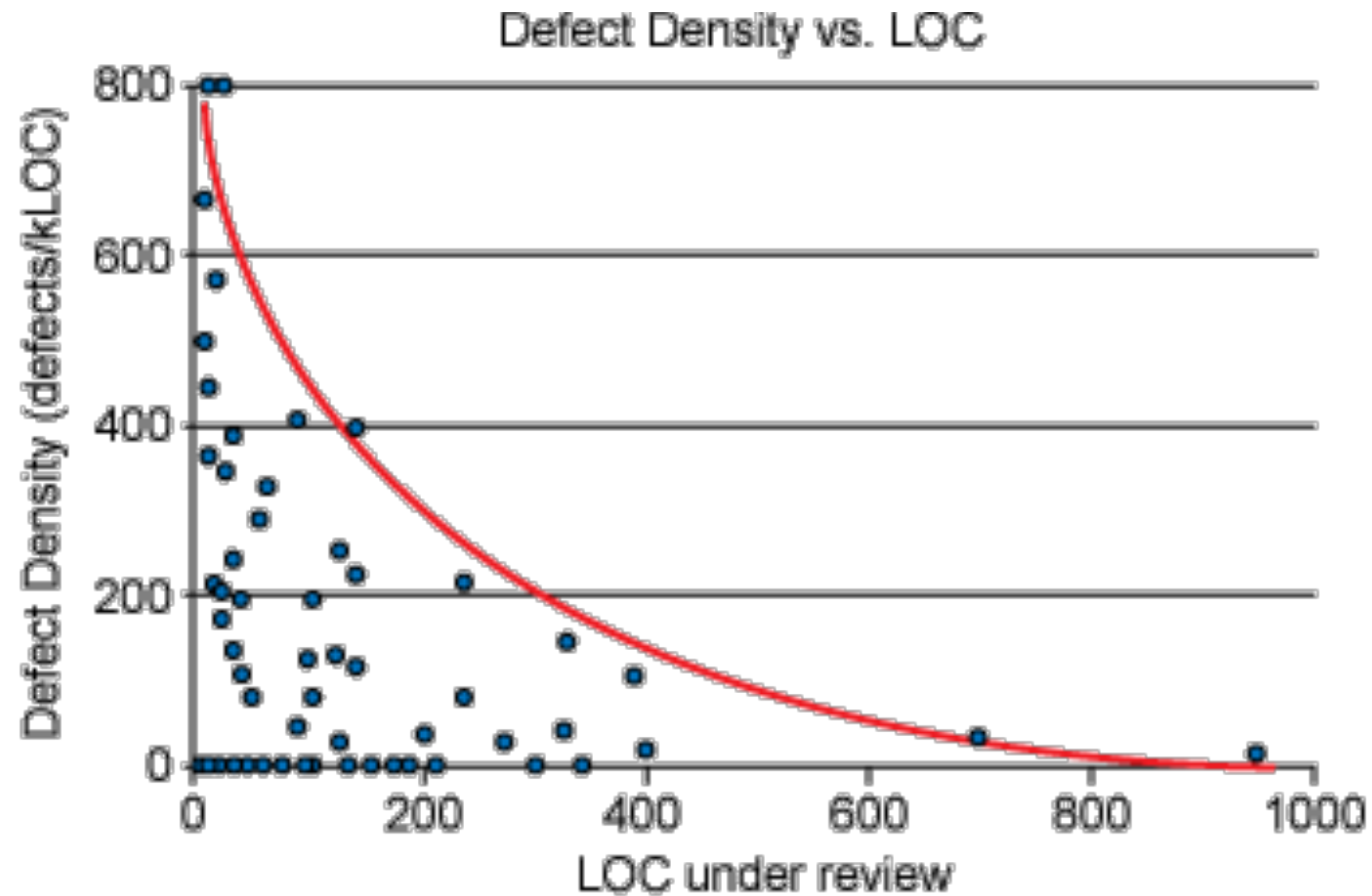
- > For big issues, don't let perfect get in the way of perfectly acceptable.
- > Prioritize what's important to you.
- > Usually 90% there is good enough.

IT'S OK TO NIT-PICK

- > Syntax Issues
- > Spelling Errors
- > Poor Variable Names
- > Missing corner-cases
- > Specify: Are your nitpicks blocking merge?

Save the nit-picks for last, after any pressing architecture, design, or other large scale issues have been addressed.

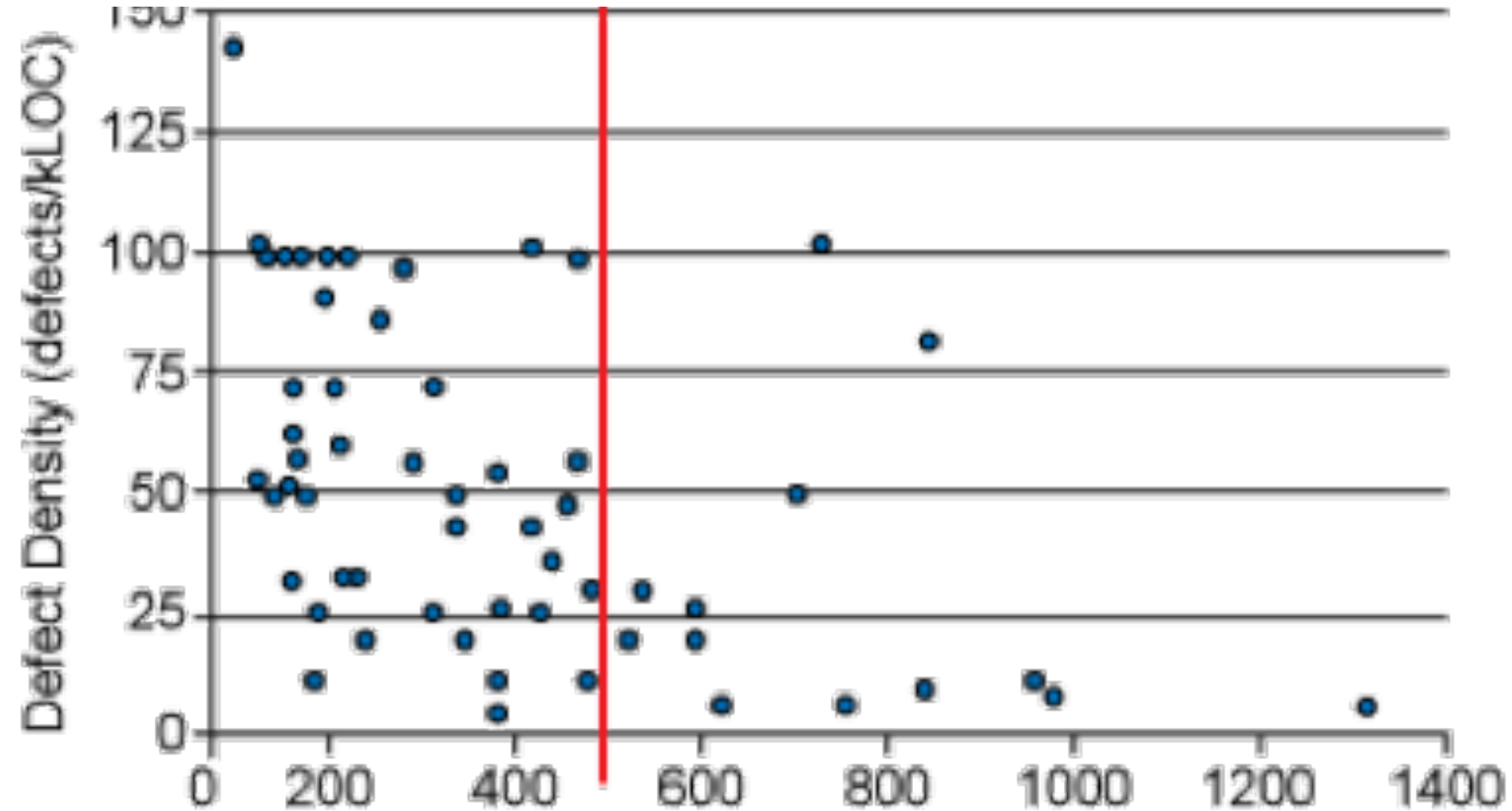
Don't burn out. Studies show reviewer should look at 200-400 lines of code at a time for maximum impact².



²

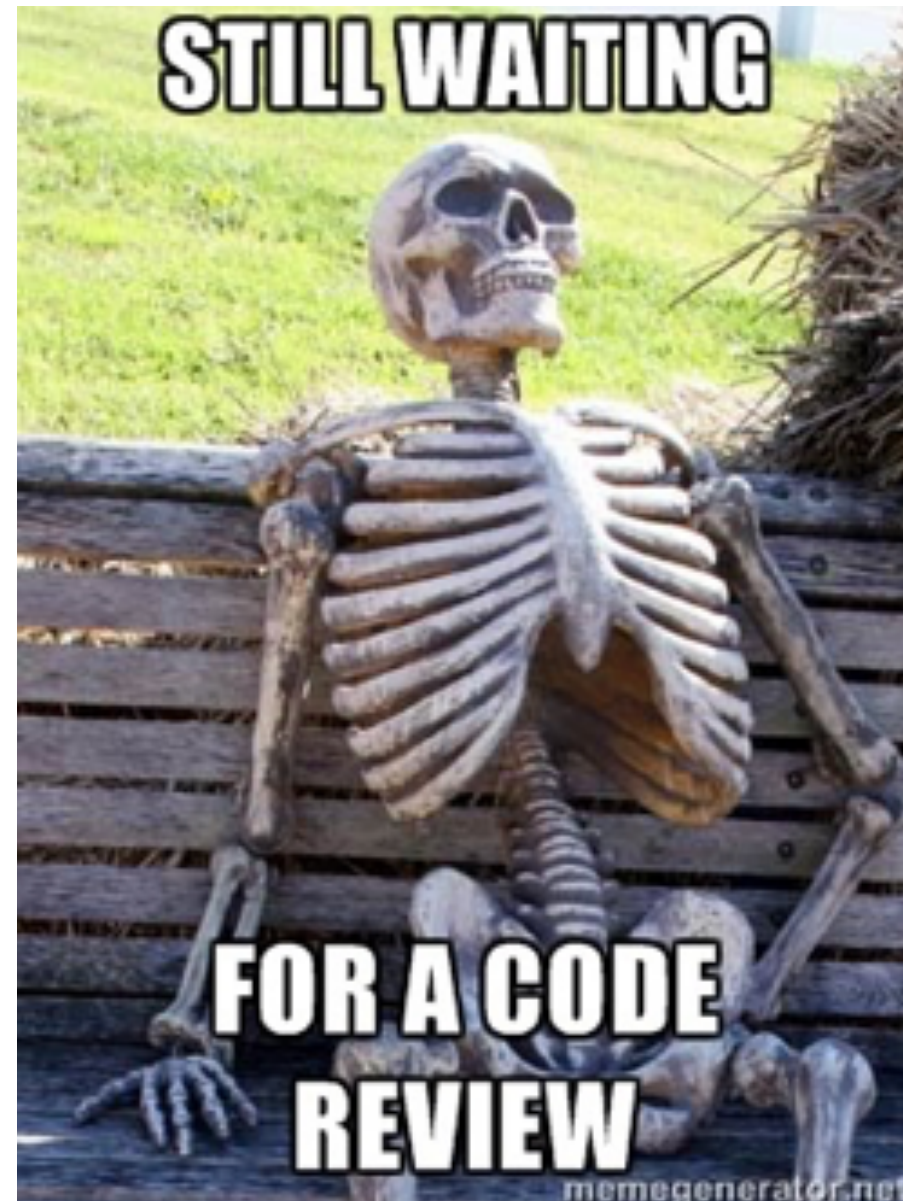
<https://smartbear.com/learn/code-review/best-practices-for-peer-code-review/>

Limit reviews to 400 lines in 60 mins
to maximize effectiveness³.



³ <https://smartbear.com/learn/code-review/best-practices-for-peer-code-review/>

TRY TO DO REVIEWS IN 24-48 HOURS

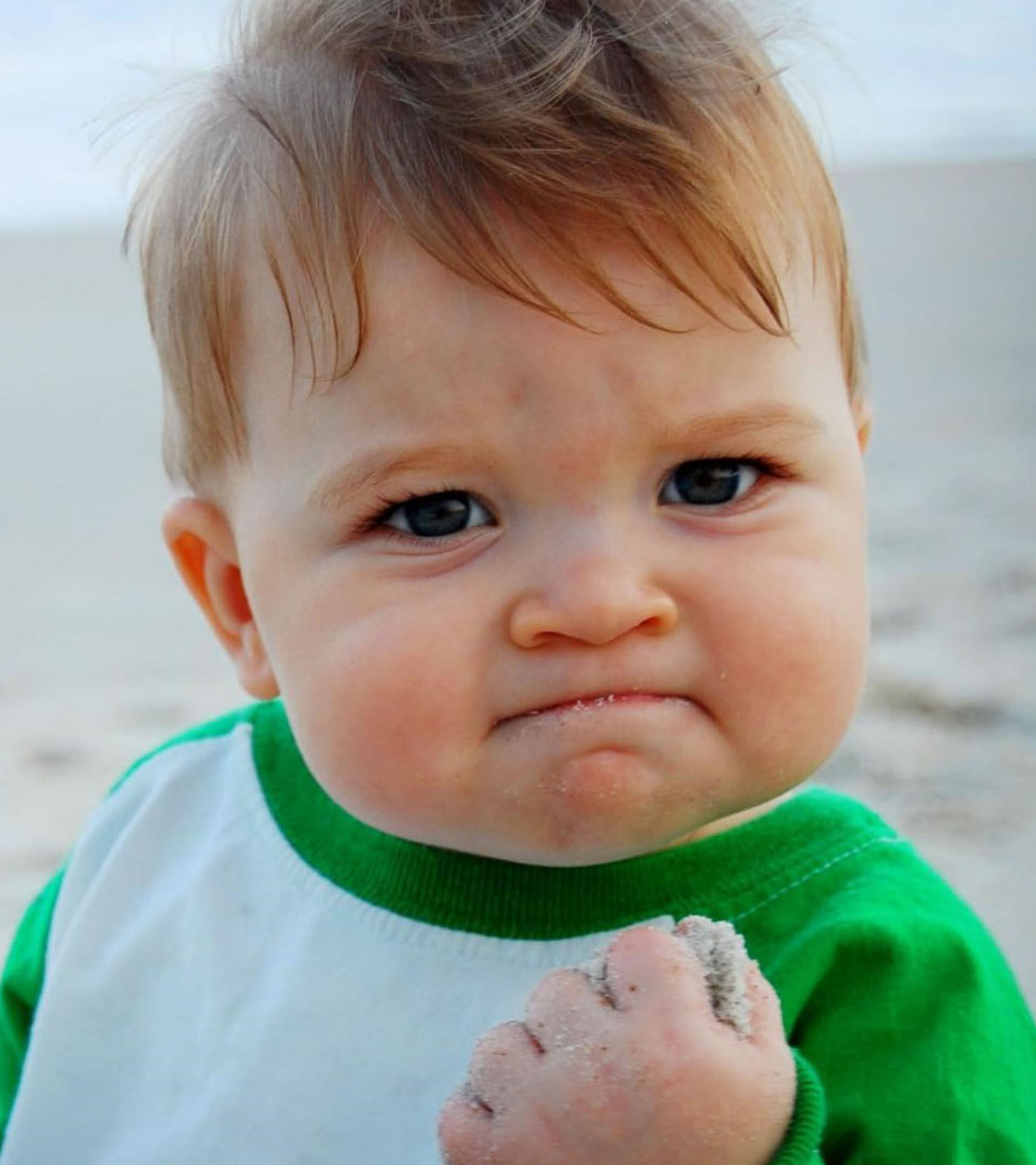


HOW CAN WE BE A GREAT REVIEWER?

- > Have Empathy
- > Watch your Language
- > Have Clear Feedback
- > Give Compliments

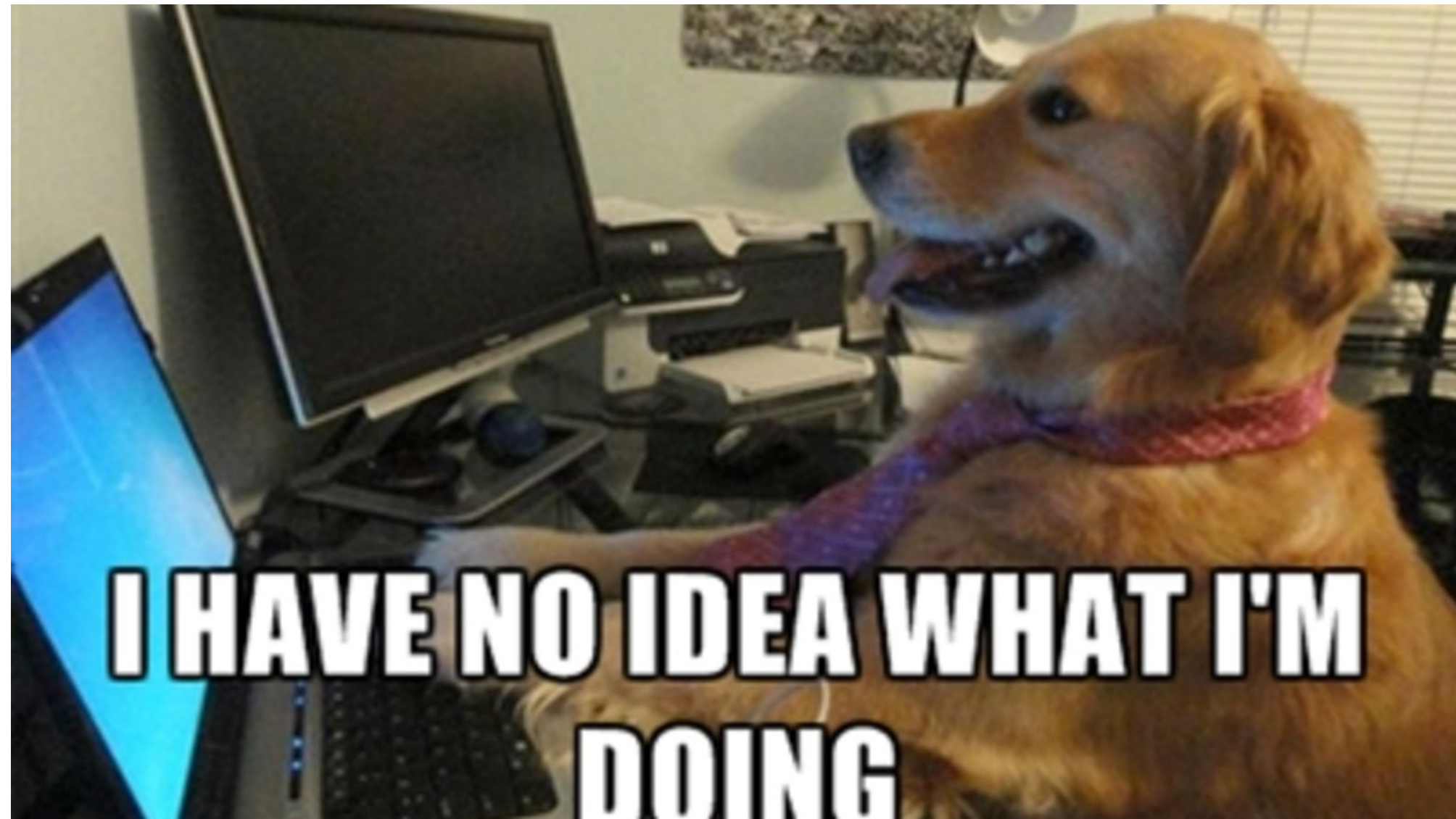
HOW CAN WE BE A GREAT REVIEWER?

- > Don't be a perfectionist
- > Avoid Burn Out
- > Complete in 24-48 hours



**CODE
REVIEWS
BUILD A
STRONGER
TEAM**

FIRST DAY VIBES...



NEWBIES

- > Not everyone has experience being reviewed.
- > Remember what it felt like when you introduced the process.
- > Ease into it!

ONBOARDING

- > The first submitted PR is the hardest
- > The first review done is challenging too
- > Start by reading recently completed reviews
- > First code review should be small
- > Share the style guide

EVERYONE'S A REVIEWER

- > Junior devs start by doing pair-reviews with a more experienced teammate.
- > Use it as a mentorship opportunity.

**HIRING SENIOR ENGINEERS IS HARD.
YOU CAN HIRE JUNIOR ENGINEERS.
AND GROW THEM
INTO FUNCTIONAL PRODUCTIVE PARTS OF
YOUR TEAM.
- SASHA LAUNDY**

IF YOU'RE NOT DOING CODE REVIEWS.
YOU'RE MISSING A BIG OPPORTUNITY.

REMEMBER...

- > Allocate the time
- > Develop, don't force the process
- > Not one size fits all
- > Or a one stop fix
 - > Use in addition to tests, QA, etc for maximum impact

**I'VE BECOME A MUCH BETTER PROGRAMMER
BY PARTICIPATING IN CODE REVIEWS**

**WHAT DID WE
LEARN?**



Sarah Drasner

@sarah_edo

Following



Coworkers who are good at code review are worth their weight in gold.

10:23 am - 11 Jul 2018

220 Retweets **1,132** Likes



26



220

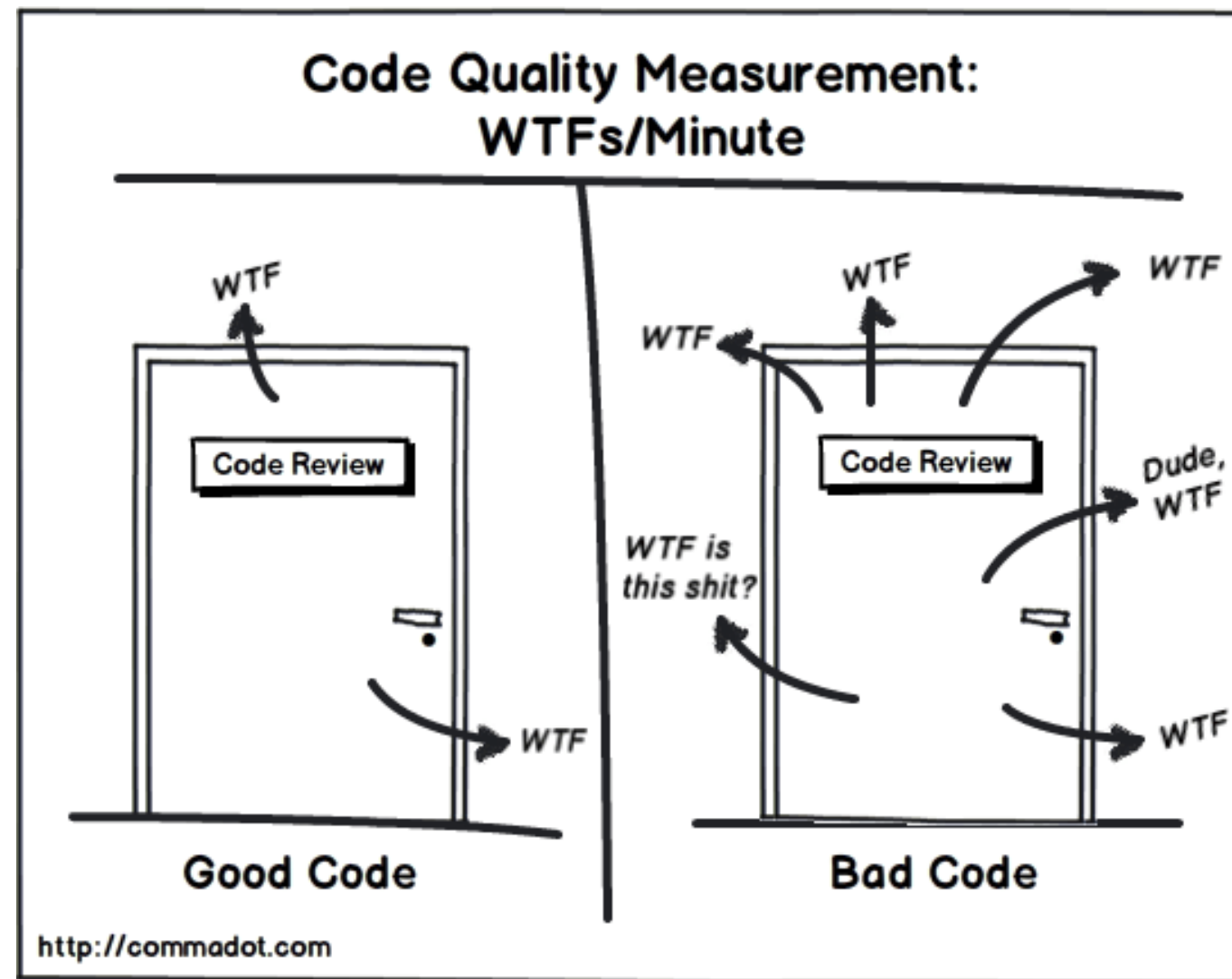


1.1K



@nnja

REVIEWS DECREASE WTFs/M BY INCREASING CODE QUALITY LONG TERM



**LESS WTF'S →
HAPPIER DEVS!**

THANKS!

SLIDES: bit.ly/codereviewpy
aka.ms/python

 [@annja](https://twitter.com/annja)

(Additional resources on
next slides)



RESOURCES & ADDITIONAL READING

- > Microsoft Study on Effective Code Review
- > Code Reviews: Just do it
- > Code Project - Code Review Guidelines
- > Great Example Checklist
- > Best Practices for Code Review
- > Rebecca's Rules for Constructive Code Review
- > My Big Fat Scary Pull Request
- > The Gentle Art of Patch Review - Sage Sharp
- > Watch: Raymond Hettinger - Beyond PEP8

EXAMPLE STYLE GUIDES

> Python

> Plone

Google has many good, but strict style guides
at: <https://github.com/google/styleguide>

Doesn't matter which one you use.
Pick one and stick with it.