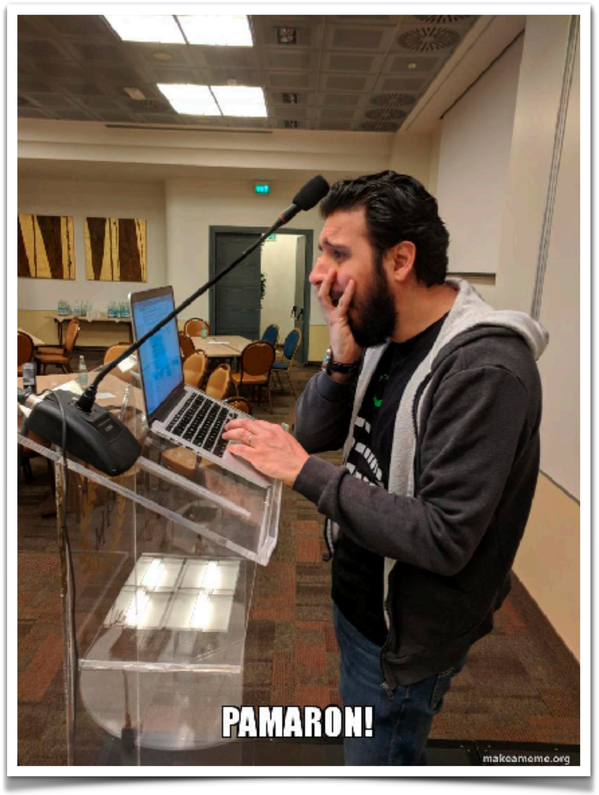
Best practices for good(ish) and clean(ish) code

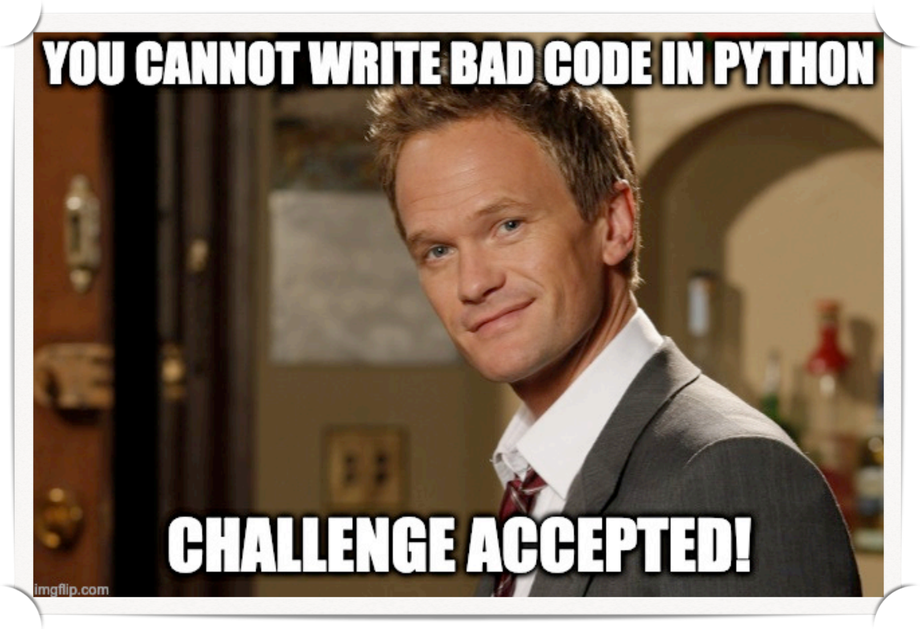
PyConZa 2022

Who am I

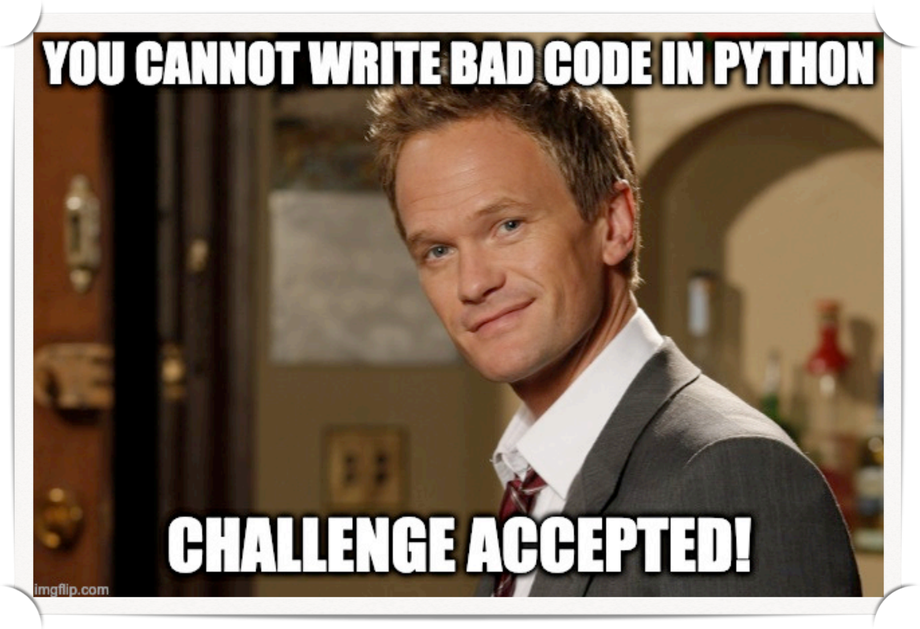


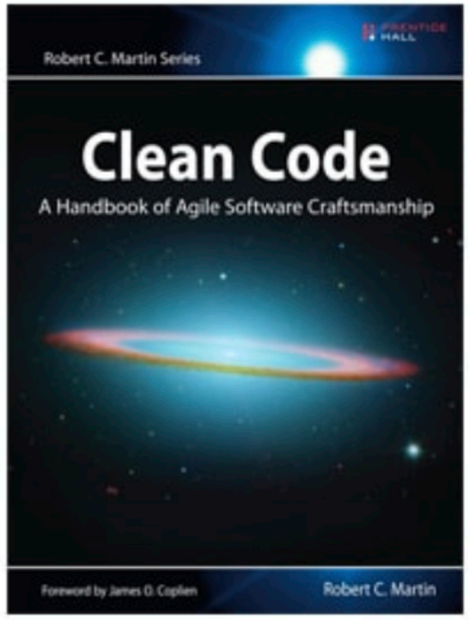
YouGov®

Why this talk



Why this talk





What’s Clean Code

- Readable
- Maintainable
- Elegant
- Testable

Clean code is simple
and direct

Clean code reads like
well-written prose

Grady Booch

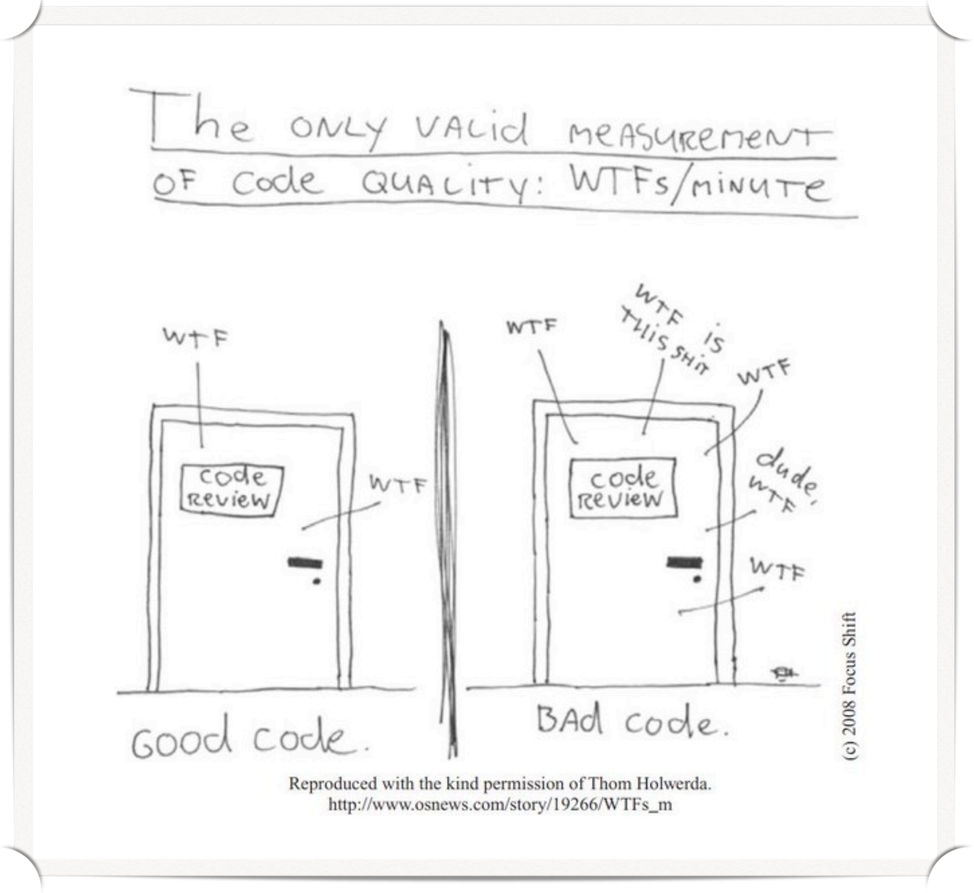
Clean code always
looks like it was written
by someone who cares

Michael Feathers

Ask yourself if the code is “clean” enough

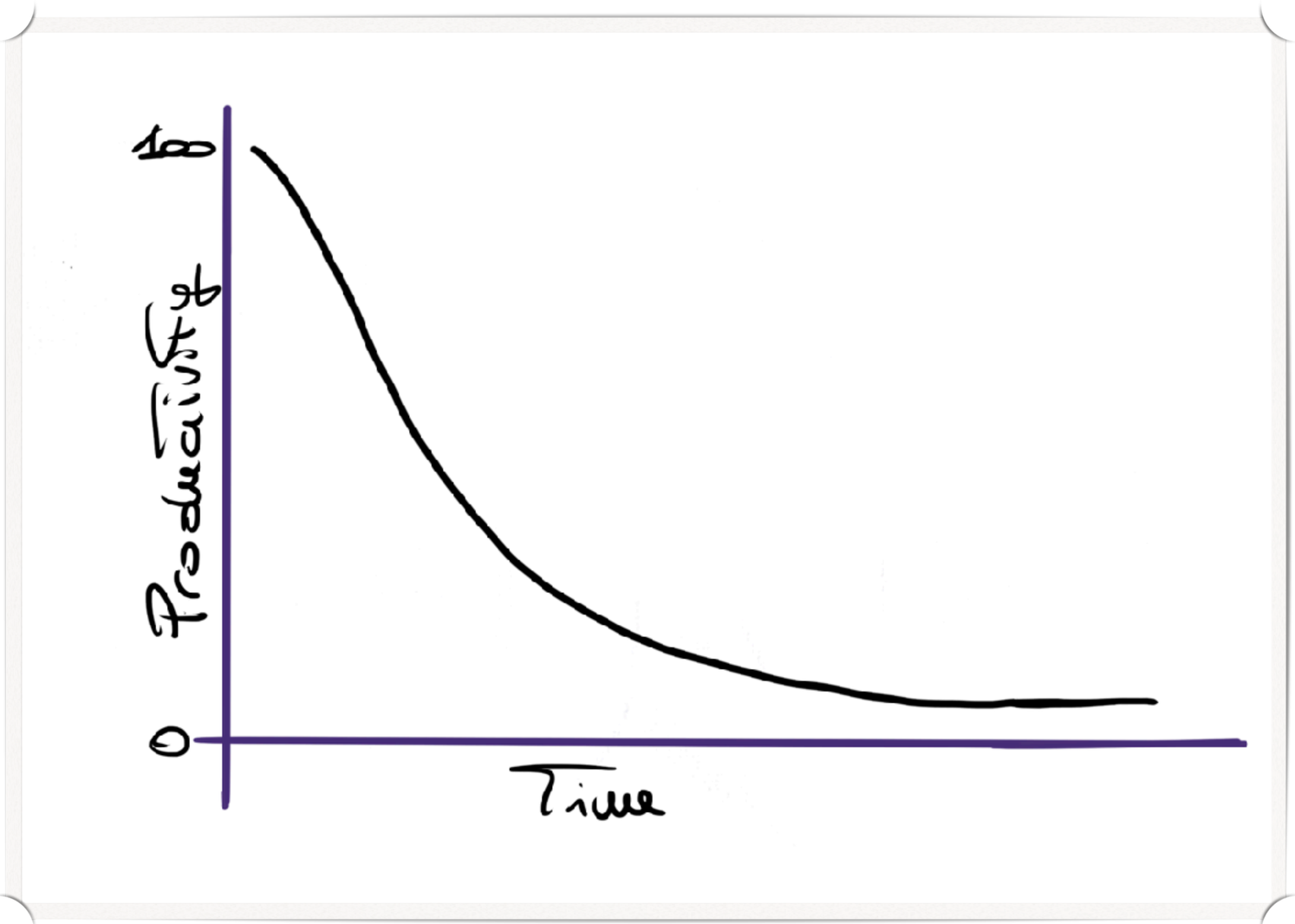
Can be inherited and enhanced by other developers other than the original author

Clean Code Measurement



Reproduced with the kind permission of Thom Holwerda.
http://www.osnews.com/story/19266/WTFs_m

Owning a mess, is it worth?



What prevents clean code?

Rule of the thumb

I’ll do the refactor later

Spoiler alert: “Later” never comes

Owning a mess, is it worth?

Owning a mess, is it worth?

```python
def get_duplicated_trips(trip):
    # calculate signature for trip
    signature = ''
    for leg in trip.legs:
        for segment in leg.segments:
            signature += get_signature(segment)

    # get trips candidate for being duplicate
    company = trip.company
    candidate_trips = Trip.objects.filter(company=company)

    # calculate signature for each trip
    candidate_signatures = []
    for candidate_trip in candidate_trips:
        candidate_signature = ''
        for leg in candidate_trip.legs:
            for segment in leg.segments:
                candidate_signature += get_signature(segment)
        candidate_signatures.append(candidate_signature)

    # get the trips for which the signature matches
    duplicated_trips = []
    for candidate_trip, candidate_signature in zip(candidate_trips, candidate_signatures):
        if candidate_signature == signature
            duplicated_trips.append(candidate_trip)
```

Owning a mess, is it worth?

```python
def get_duplicated_trips(trip):
    signature = get_trip_signature(trip)
    candidate_trips = get_candidate_trips(trip)
    return [trip for trip in candidate_trips if get_trip_signature(trip) == signature]


def get_trip_signature(trip):
    signature = ''
    for leg in trip.legs:
        for segment in leg.segments:
            signature += get_signature(segment)
    return signature


def get_candidate_trips(trip):
    company = trip.company
    return Trip.objects.filter(company=company)
```

Descriptive names

```python
# bad
d = datetime.date(2019, 4, 13)

#better
days_since_creation = datetime.date(2019, 4, 13)
```

The name should always represent the developer's idea

Meaningful names

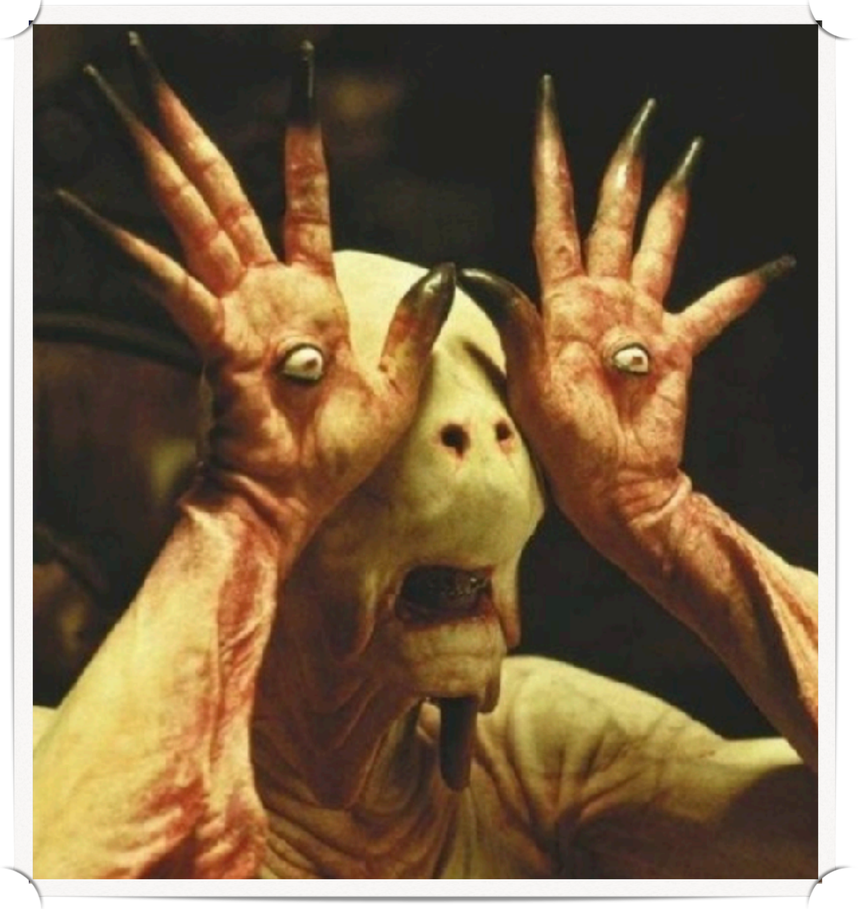
```python
# bad
moddYmd = datetime.date(2019, 4, 13)
psquint = "102"

#better
modification_date = datetime.date(2019, 4, 13)
record_id = "102"
```

Searchable names

```
# bad
CTX_RSP_DICT = {}
sum = 90

# better
CONTEXT_REPONSE = {}
total_of_people = 90
```

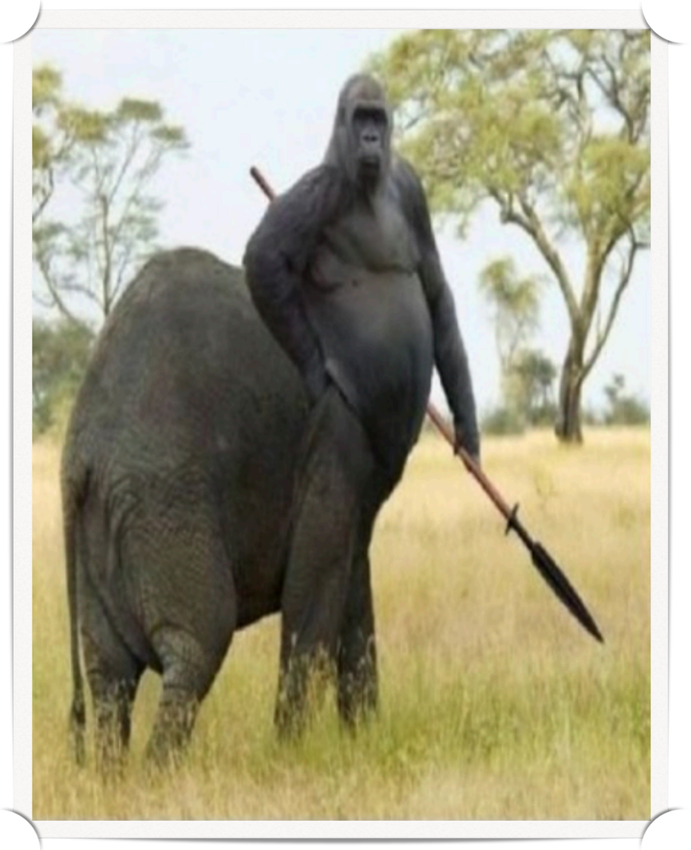


Class names

Classes should have noun or noun phrase names like Customer, WikiPage, Account, AddressParser.

Avoid words like Manager, Processor, Data … in the name of a class.

A class name should not be a verb.



Method names

Methods should have verb of verb phrase names like post_payment, delete_page, save.

Method names should say what they do!

Comments sometimes lie

Code never lies

```python
def is_billed(self, bill_id:str) -> bool:
    # Check from the stock repository if there is bill ID
    # that is not expired.
    return self.billing_repository.exists(bill_id)
```

```python
if element > 0:
  # return sqrt of the element if it's not None
  return math.sqrt(element)
```

Code never lies

```python
def is_billed(self, bill_id:str) -> bool:
    # Check from the stock repository if there is bill ID
    # that is not expired.
    return self.billing_repository.exists(bill_id)
```

```python
if element > 0:
  # return sqrt of the element if it's not None
  return math.sqrt(element)
```

If they don’t lie, they can be lazy


```
# Calculate side length using the Pythagorean Theorem
# and put the value into variable `r2`
delta = h*h-r1*r1
r2 = math.sqrt(delta)
```

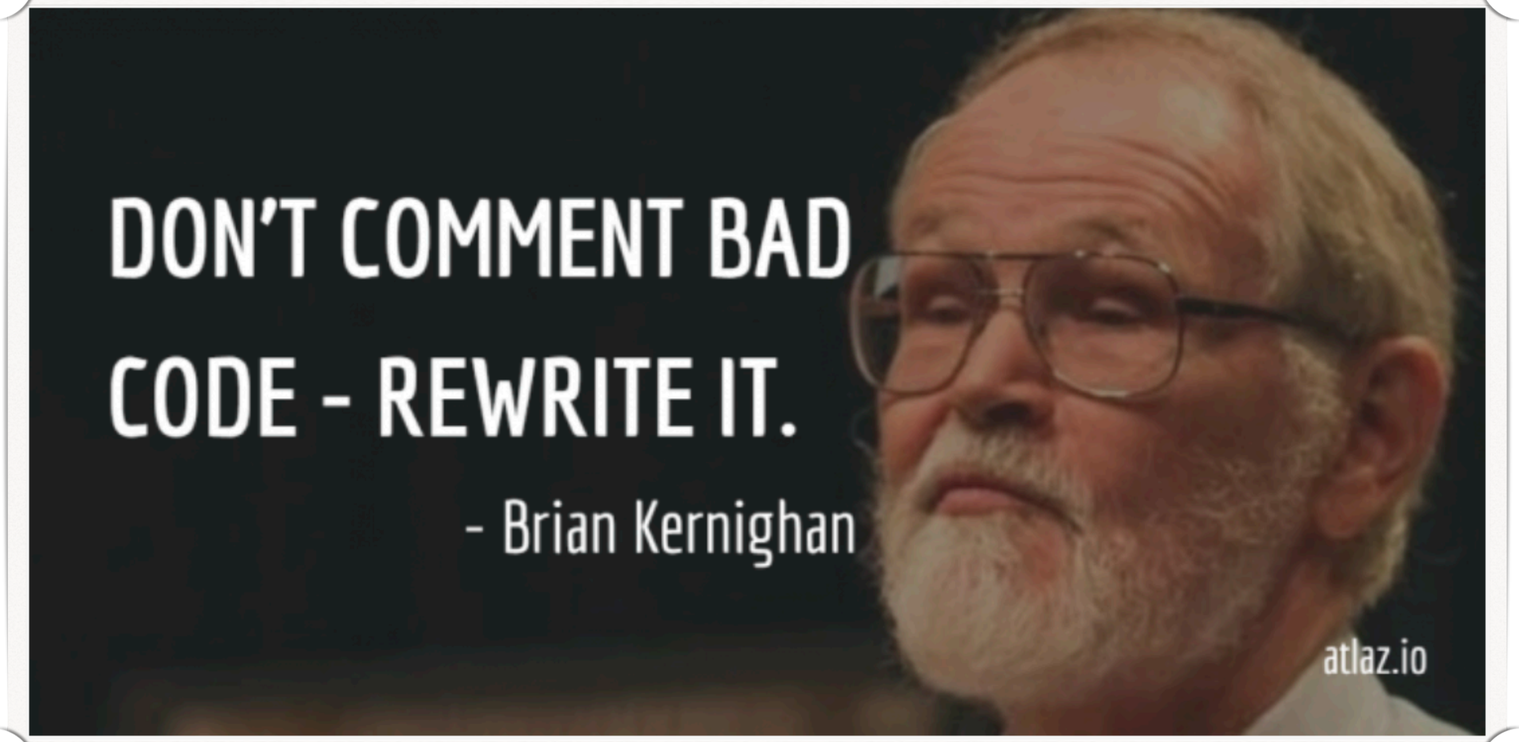
Crime: using a comment to avoid programming effort.

If they don’t lie, they can be lazy

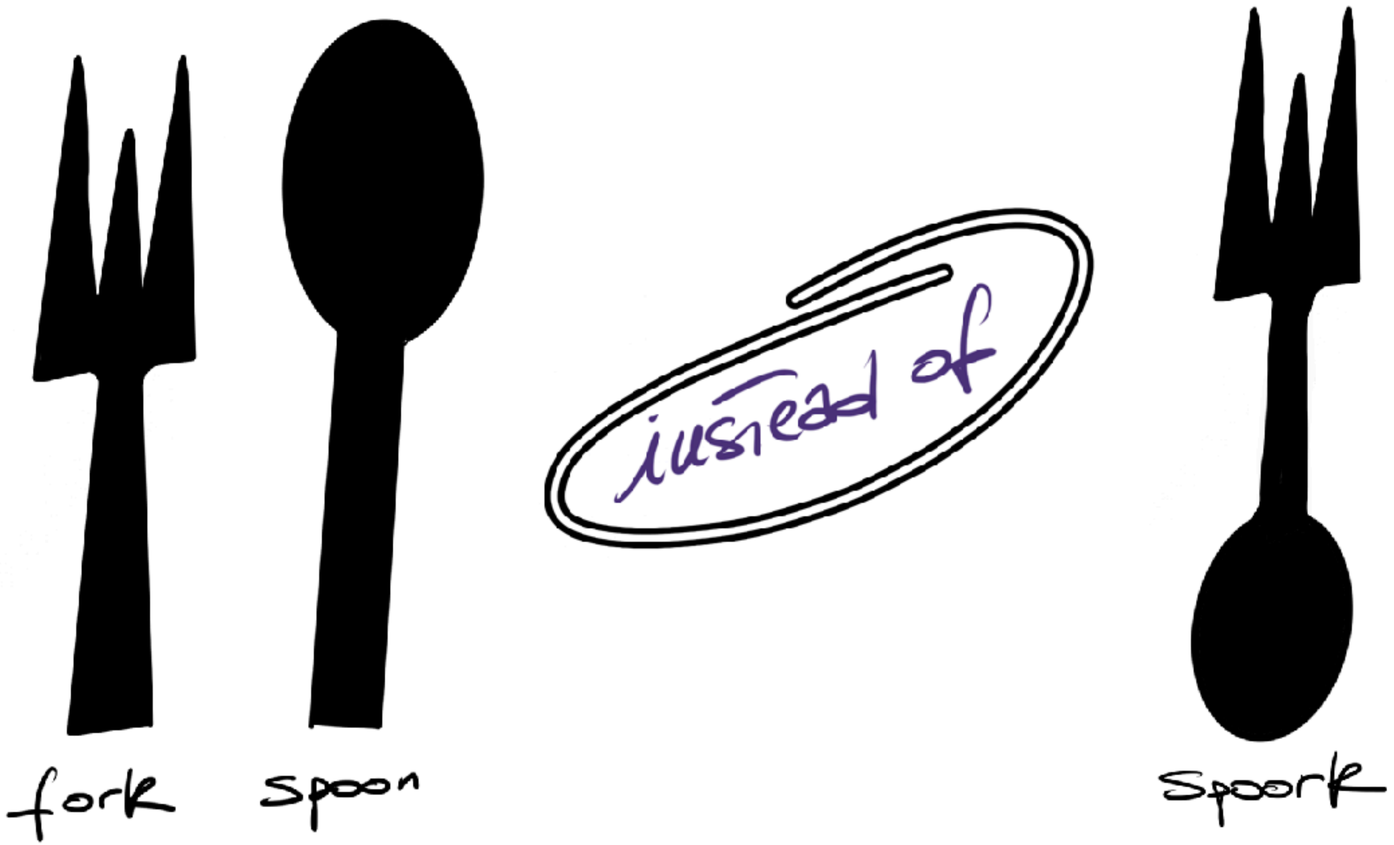

```python
# Calculate side length using the Pythagorean Theorem
# and put the value into variable `r2`
delta = h*h-r1*r1
r2 = math.sqrt(delta)
```

```python
len_side_b = math.sqrt(math.pow(hypotenuse, 2) - math.pow(len_side_a, 2))
```

Crime: using a comment to avoid programming effort.


DON'T COMMENT BAD
CODE - REWRITE IT.
- Brian Kernighan
atlaz.io

Only one thing



Only one thing

```python
# bad
def retrieve_flight_info(flight_code):
    flight = Flight.objects.get(code=flight_code)
    passengers = Passenger.objects.filter(flight=flight)
    return flight, passengers

my_flight, passengers = retrieve_flight_info("A453Z")



# better
def retrieve_flight(flight_code):
  return Flight.objects.get(code=flight_code)

def retrieve_passengers(flight):
  return Passenger.object.filter(flight=flight)

my_flight = retrieve_flight("A453Z")
passengers = retrieve_passengers(my_flight)
```

Extract, extract, extract

```python
def duplicated_trips(trip):
    # calculate signature for trip
    signature = ""
    for leg in trip.legs:
        for segment in leg.segments:
            signature += get_signature(segment)

    # get trips candidate for being duplicate
    company = trip.company
    candidate_trips = Trip.objects.filter(company=company)

    # calculate signature for each trip
    candidate_signatures = []
    for candidate_trip in candidate_trips:
        candidate_signature = ""
        for leg in candidate_trip.legs:
            for segment in leg.segments:
                candidate_signature += get_signature(segment)
        candidate_signatures.append(candidate_signature)

    # get the trips for which the signature matches
    duplicated_trips = []
    for candidate_trip, candidate_signature in zip(
        candidate_trips, candidate_signatures
    ):
        if candidate_signature == signature:
            duplicated_trips.append(candidate_trip)
```

Extract, extract, extract

```
def duplicated_trips(trip):
    signature = trip_signature(trip)
    candidate_trips = candidate_trips(trip)
    return [trip for trip in candidate_trips if trip_signature(trip) == signature]


def trip_signature(trip):
    signature = ""
    for leg in trip.legs:
        for segment in leg.segments:
            signature += retrieve_signature(segment)
    return signature


def candidate_trips(trip):
    return Trip.objects.filter(company=trip.company)
```

Small cognitive/cyclomatic complexity

```python
def post_comment(self):
    if self.type == 'success':
        comment = 'Build succeeded'
    elif self.type == 'warning':
        comment = 'Build had issues'
    elif self.type == 'failed':
        comment = 'Build failed'
    else:
        comment = 'Unknown status'

    if self.type == 'success':
        self.post(comment, type='success')
    else:
        self.post(comment, type='error')
```

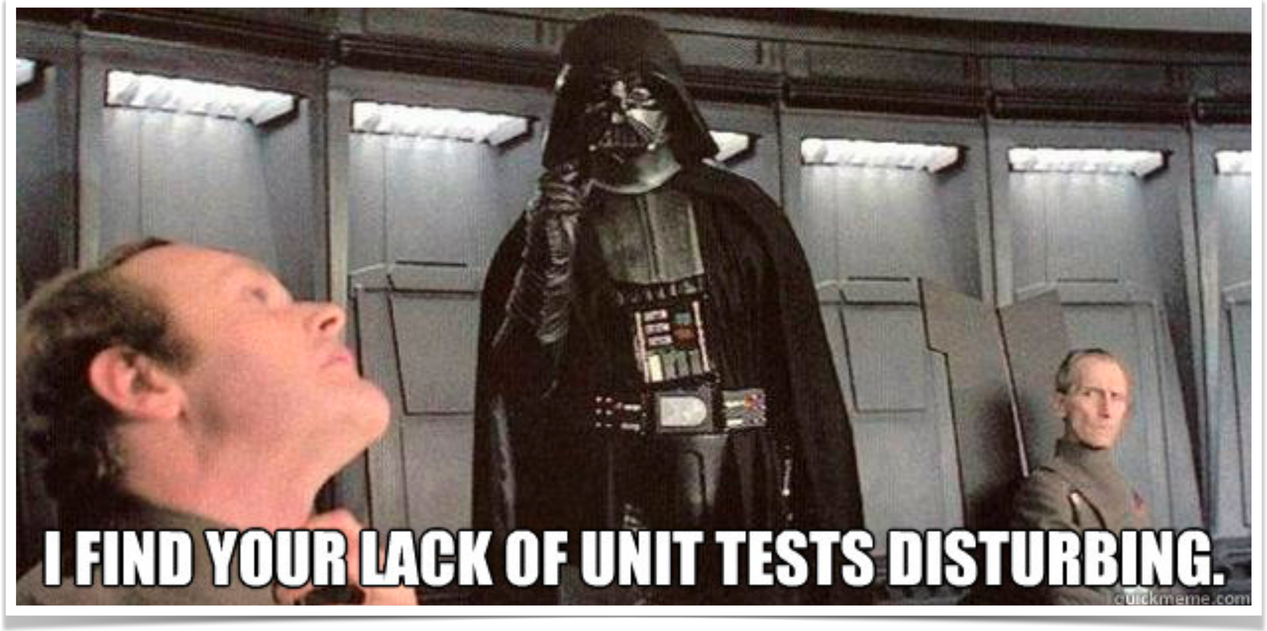
Small cognitive/cyclomatic complexity

```
def get_comment(self):
    comments = {
        'success': 'Build succeeded',
        'warning': 'Build had issues',
        'failed': 'Build failed'
    }
    return comments.get(self.type, 'Unknown status')

def post_comment(self):
    comment = self.get_comment(self)
    self.post(comment, type=self.type)
```

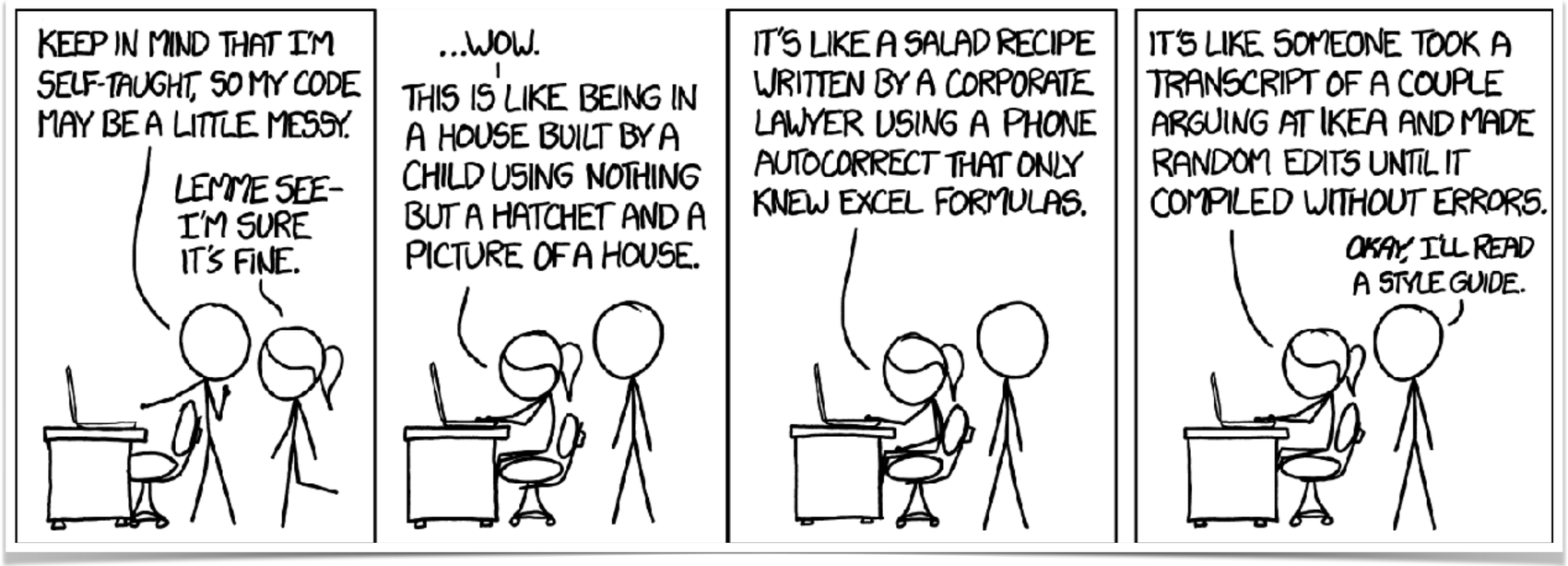
Advantages of measuring software complexity

1. Better Test Coverage



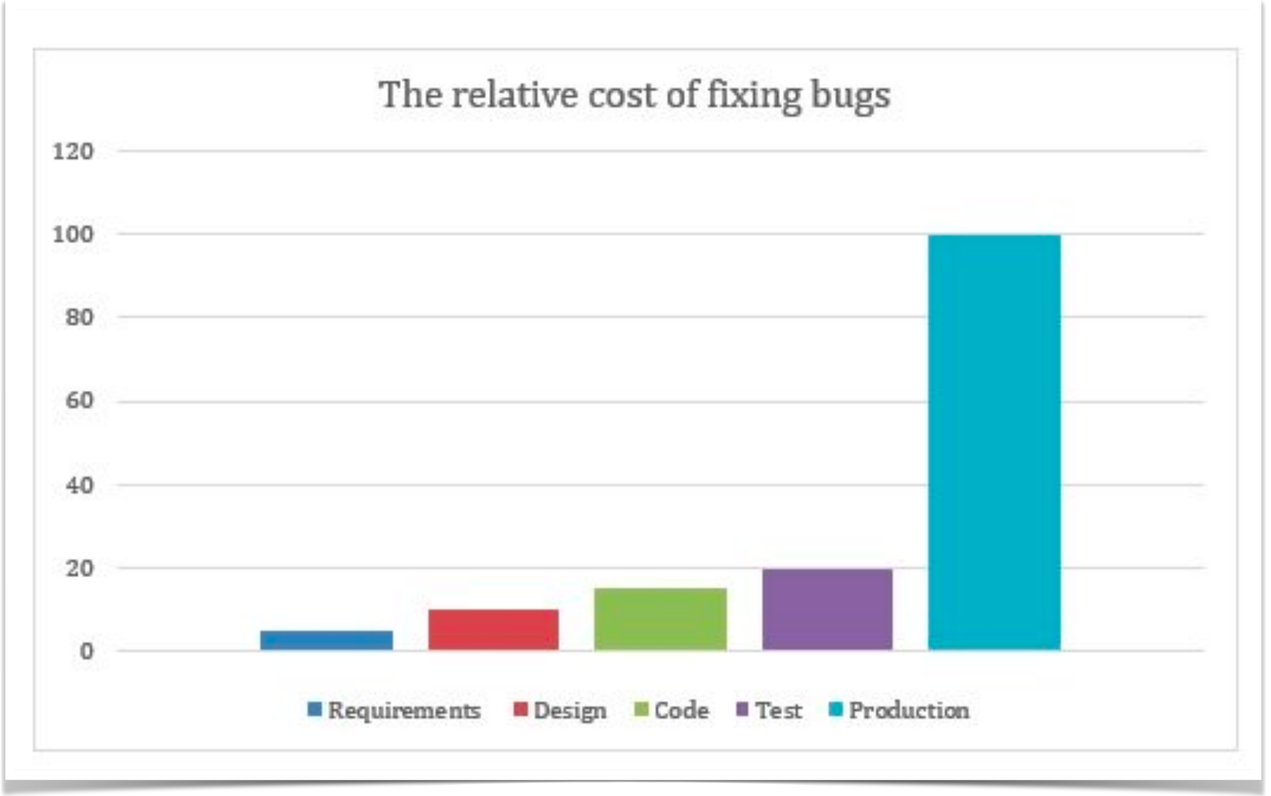
Advantages of measuring software complexity

3. Reduced Risk



Advantages of measuring software complexity

3. Lower Costs



Law of Demeter (“only 1 dot”)

Ask a dog to walk


```python
# callers need to know internals
for leg in dog.legs:
    leg.move(forward=3)


# better
dog.move(forward=3)

class Dog:
    def move(self, forward):
        for leg in self.legs:
            leg.move(forward=3)
```

👨‍🎨 `book.pages().last().text()`

*Lieberherr, K.J.; Holland, I.M. (September 1989). "Assuring good style for object-oriented programs". IEEE Software. **6** (5): 38–48. doi:10.1109/52.35588.*

Law of Demeter: drawbacks

it won't apply to cats

“Premature optimisation **is the root of all evil**”

— **Donald Knuth**

Continuous Refactoring

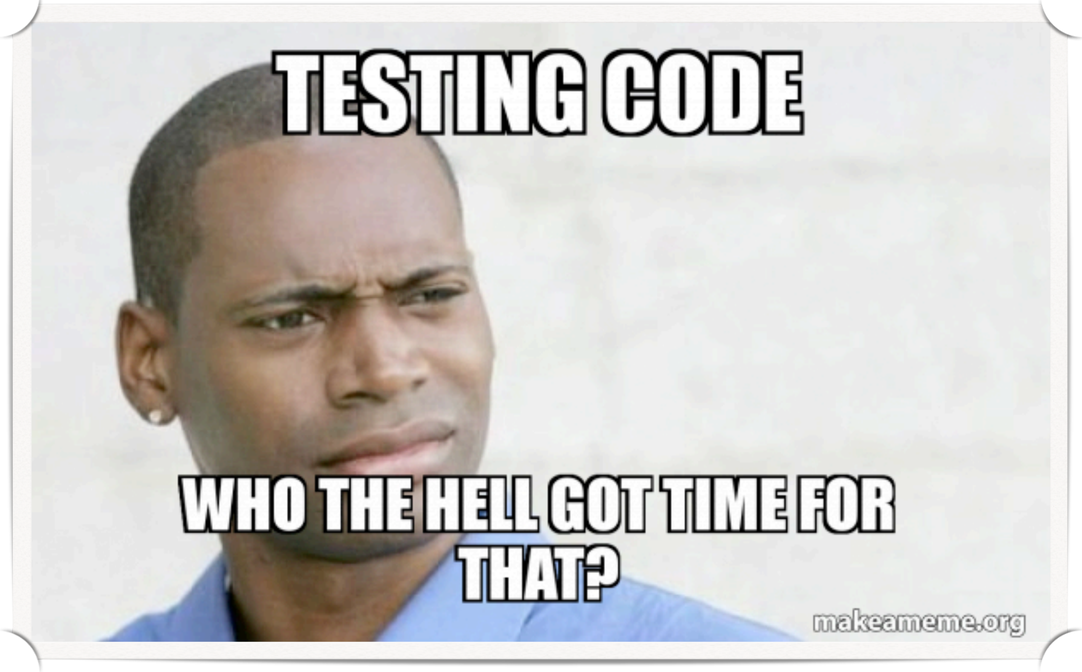
“The most common “presenting” pathology in the hoarded codebases I’ve seen — by far — is that developers don’t feel they have time and/or permission to refactor code.”

— Sarah Mei

[Refactoring is] a very core technique to the whole agile way of thinking because it fits in with the whole way in which we can build software in a way that it can change easily … Refactoring is central to this because refactoring is a disciplined way of making changes. ”

— Martin Fowler

Tests matter



“How can I make sure that if I change this, nothing else breaks?”

… also experience matter

Most popular code smells

Duplication	Improper use of inheritance
Unnecessary complexity	Convoluted Code
Useless/misleading comments	Tight coupling
No Code Reviews	Over abstraction
Huge methods	Design Pattern overuse
Poor naming	No test
Commented code / code that’s not used	Improper usage of private and public objects

Thanks❤️

 **github.com/ernestoarbitrio**

 **twitter.com/__pamaron__**

Questions? 🤔