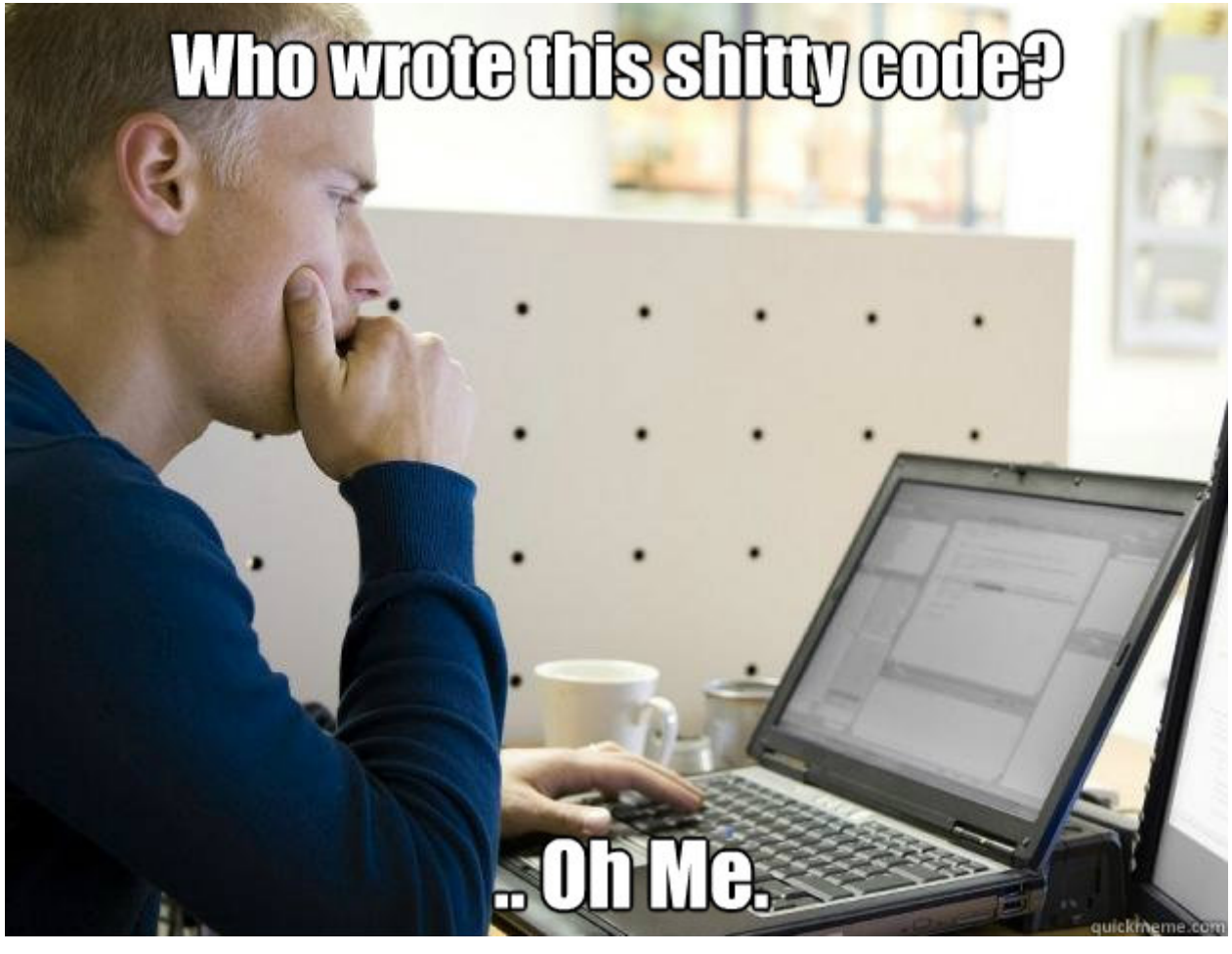
CODE REVIEW PROCESS

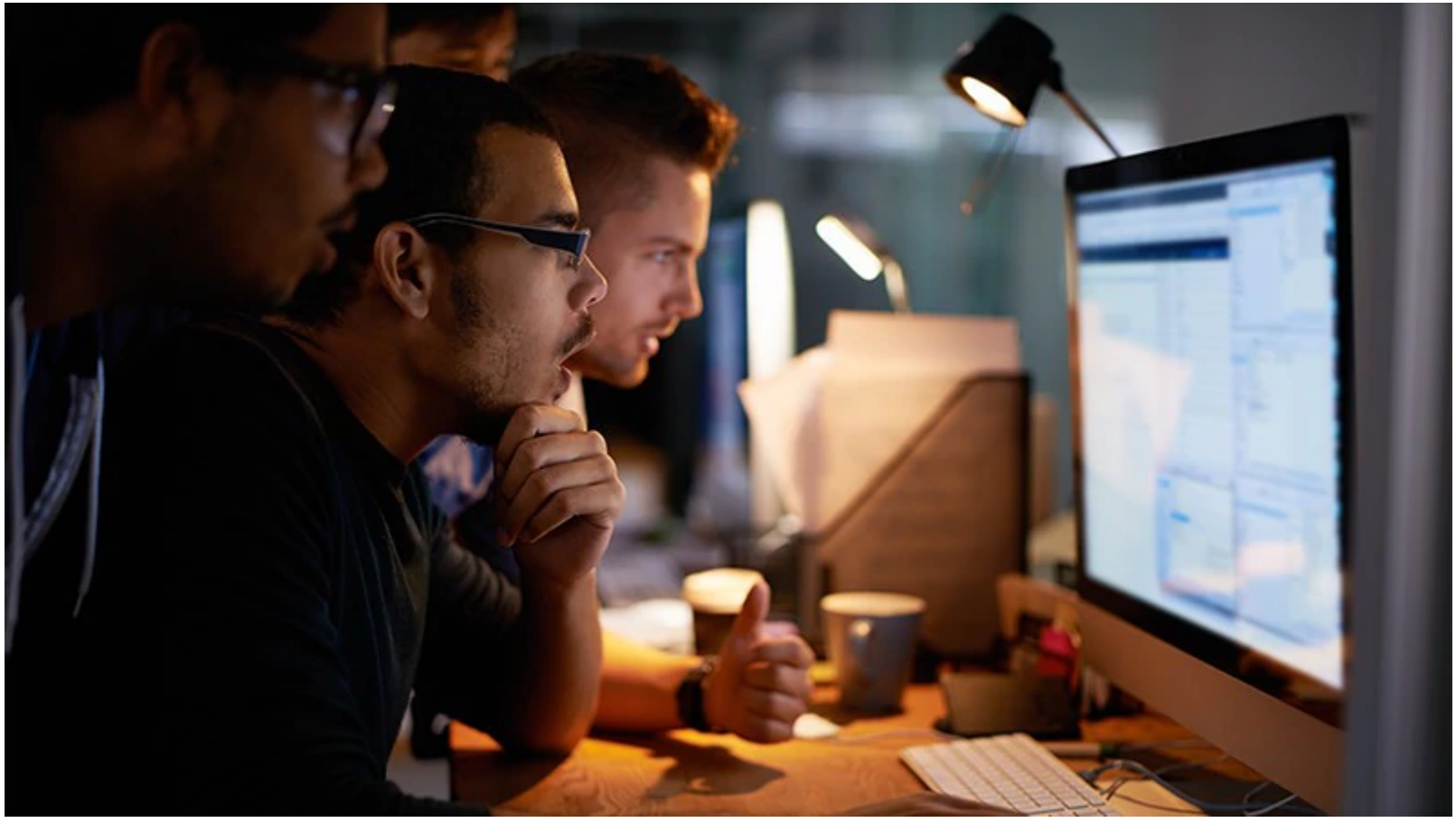
BOGDAN GUSIEV

CODE REVIEW TENDS TO BE NATURAL



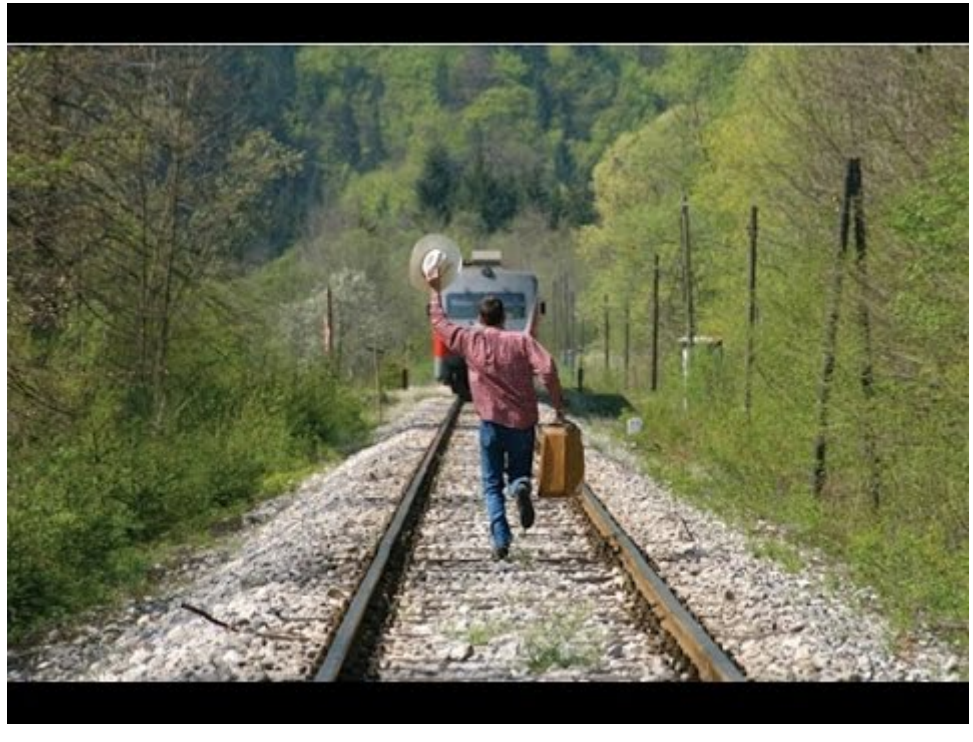
CODE REVIEW YOURSELF

ASKING FOR HELP



SPONDANUOUS CODE REVIEW PROBLEM:

TOO LATE



EVERYTHING HAD CHANGED BY

GITHUB

DO I NEED A CODE REVIEW?

IMHO: NEW PROJECTS

DON'T NEED A FORMAL CODE REVIEW AT ALL

CODE REVIEW EVOLUTION

1. Spontanuous Code Reviews
2. "When I am not sure" code reviews
3. Formal Code Review Process
4. Required code review for certain changes
5. Required code reviews for everything

Fails Driven Process

CODE REVIEW PROPERTIES

NECESSITY

- Optional
- Required

PROCESS

- Informal
- Formal

PEOPLE

- Dedicated
- Distributed

POSITION IN A PROCESS

1. Planning
2. Coding
3. Code Review
4. QA
5. Release

QA <-> Code Review

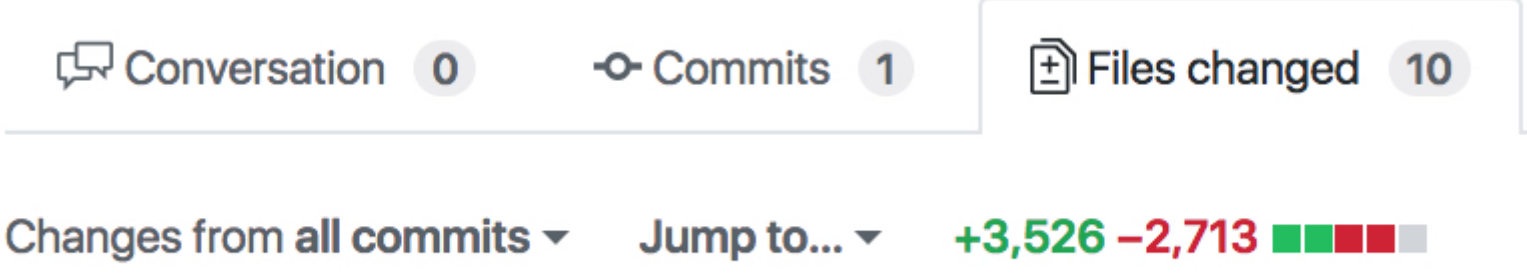
Formally QA after Code Review

1. Make a fork (optional)
2. Create a Branch
3. Open a Pull Request
4. Wait for review
5. Discuss & Fix
6. Merge

CODE REVIEW IS

A PROCESS OF REVIEWING AND APPROVING CODE CHANGES
BEFORE THEY GET ACCEPTED TO THE MAINSTREAM

HOW TO REVIEW?



FIX COMMON THINGS FIRST?

- Typos
- Whitespace
- Code Style

Wrong Direction!

WHERE TO LOOK?

Things that are most significant should be reviewed first

which are ones that are the most hard to change.

TOP TO BOTTOM CODE REVIEW

1. Architecture
    1. Problem Solution
    2. Public APIs
    3. Database Schema
    4. Object Oriented Design
    5. Public Method Signatures
2. Implementation
    1. Classes & Methods
    2. Views
    3. Tests
    4. Code Style, Typos, Whitespace

PROBLEM SOLUTION

1. Problem makes sense
2. Problem Solved
3. High Level Security
4. High Level Performance

ONLY SHOW CUSTOM PROPERTIES WITH AT LEAST ONE VALUE IN SELECT

```
def selectable_product_categories
+  ProductCategory.with_at_least_one_product
-  ProductCategory.all
end
```

```
ProductCategory.
  where("not exists(select * from products where ...)").
  count # => 0
```

HIGH LEVEL SECURITY

EX. FEATURE:

LOGIN USER AUTOMATICALLY WHEN IT CLICKS ON THE LINK IN THE EMAIL

THIS IS NOT VERY SECURE

HIGH LEVEL PERFORMANCE

CHECK IF THE CHANGE

- Touches Performance sensitive code
- Will be slow for particular data
  - No pagination when there are 1000+ records to display

PUBLIC APIS

USING HTTP API AS EXAMPLE

1. Efficiency
2. Logical Endpoints
3. Request Parameters
4. Response Format

WHATEVER THAT IS DOCUMENTED

API INEFFICIENCY EXAMPLE

```
Purchase.has_one :referral
```

```
GET /purchases/:order_number
  {
    id: 1,
    order_number: '1838382',
    referral_id: 17
  }

POST  /referrals/:id/approve
```

EFFICIENT WAY?

```
POST /purchases/:order_number/referral/approve
```

RESPONSE FORMAT

```
# Easier
render json: @campaign.view_setups.to_json
# Extensible
render json: {view_setups: @campaign.view_setups.to_json}
```

BAD API EXAMPLE

```
Talkable.publish('talkable_offer_close', null, true);
```

```
Talkable.publish('offer_close', null, true);
```

ANALYZE USAGE FIRST

BUT NOT IMPLEMENTATION

```
setCustomProperty: function (person, properties) {
  ...
}
setCustomProperty('advocate', {key: value})
setCustomProperty('friend',   {key: value})
```

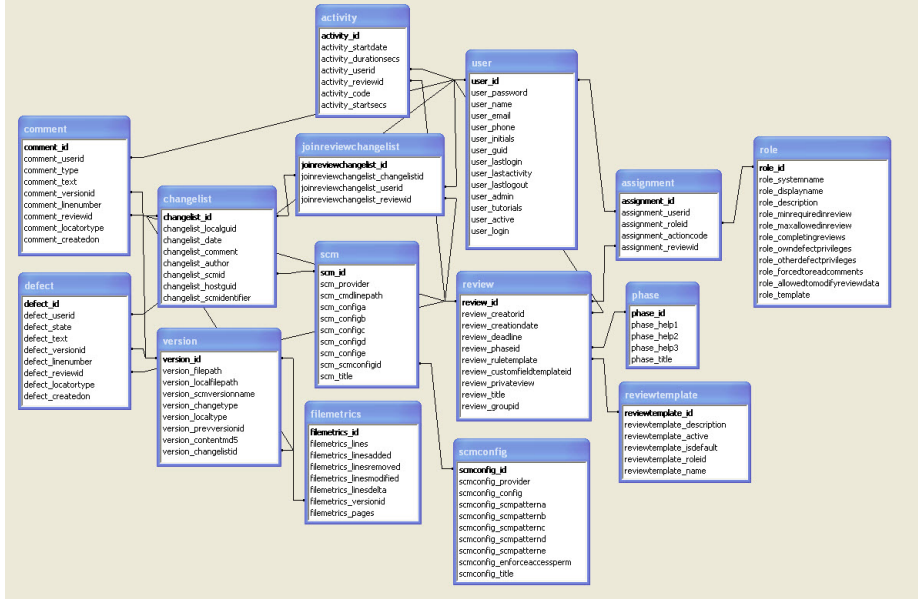
```
setAdvocateCustomProperty: function(properties) {
  private_stuff.setCustomProperty("advocate", properties);
},

setFriendCustomProperty: function(properties) {
  private_stuff.setCustomProperty("friend", properties);
},
```

DATABASE SCHEMA

1. Relations between tables
2. Data Columns
3. Naming

EFFICIENT RELATIONS



1. What are the variants?
2. Was the best one selected?

EFFICIENT SCHEMA

```
add_column :images, :dimension, :string

class Image < AR::Base
  def width
    dimension.split("x").first
  end
  def height
    dimension.split("x").last
  end
end
```

```
add_column :images, :width, :integer
add_column :images, :height, :integer
```

DATA SHOULD BE EASY TO READ

EVEN IF IT MAKES IT HARDER TO WRITE

OBJECT ORIENTED DESIGN

1. Reflects Real World
2. Inheritance
3. Constructors

REVIEWING CONSTRUCTORS

CIRCULAR DEPENDENCY

```
class Field
  def initialize
    @cells = Array.new(10) do
      Array.new(10) { Cell.new(self) }
    end
  end
end
class Cell
  def initialize(field)
    @field = field
  end
end
```

OBJECT CONSTRUCTORS ARE IMPORTANT:

- Which things are required to use an object?
- What are the object responsibilities?
  - Theoretical limit
- Which methods can be implemented in the object?
- Which things would need to be passed to methods as arguments?
  - Constructor defines object API

REVIEWING CONSTRUCTORS

UNDEFINED CONTEXT

```
class ApplicationController
  before_action do
    WhateverRequestAnalyzer.new(request).analyze
  end
end
```

```
CrawlerRequestAnalyzer.new(
  request.user_agent, request.url, request.content_type
).analyze
```

PUBLIC METHOD SIGNATURES

1. Placement
2. Arguments
3. Name

METHOD PLACEMENT

```
class User
  def approve!(referral)
end
# OR
class Referral
  def approve!(user)
end
```

METHOD ARGUMENTS

```
this.extractUserData(this.data)
```

IMPLEMENTATION

CLASSES & METHOD BODIES

Each method or class separately

Check one by one:

1. Approach & Algorithm
2. Performance
3. Security
4. Minimalism
5. Local Variable Names

PERFORMANCE & VULNERABILITIES

In the ideal world performance and vulnerabilities
should not change the code structure.

In practice it can but we should try hard
to fix performance problems only at the implementation level

GOOD PERFORMANCE PATCH

```
 def core_options
-  { header: name, description: description, group: group }
+  @core_options ||= { header: name, description: description, grou
 end
```

```
- @campaign.locale_entries.each do
+ @campaign.locale_entries.preload(:variants).each do
```

GOOD VULNERABILITY PATCH

```
-{{ advocate_info.first_name }}
+{{ advocate_info.first_name | escape }}
```

SECURITY

- Approach Security (Step 1)
  - Is it secure to have this feature?
  - Is it secure to implement the feature this way?
- Vulnerabilities
  - XSS
  - Allowed Parameters
  - Backend Authorisation check
  - Authorisation for UI elements

TESTS

1. Use Cases Coverage
2. Formal Code Coverage
3. Tests Implementation

TOP TO BOTTOM IDEA

IT IS BAD TO:

- Discuss the method name before the method existence as a fact
- Discuss Code Style before implementation itself

BOSS CONTROL APPROACH

Ensure top points from the list are always performed

Choose X things on top of the list to double-check and delegate the rest completely

CODE REVIEW CULTURE

- Be Polite
- Admit good things
- First things first
  - Reduce number of cycles
  - Save Author's time
  - Save Your time

TOP TO BOTTOM CODE REVIEW

1. Architecture
    1. Problem Solution
    2. Public APIs
    3. Database Schema
    4. Object Oriented Design
    5. Public Method Signatures
2. Implementation
    1. Classes & Methods
    2. Views
    3. Test Coverage
    4. Code Style, Typos, Whitespace