CLEANING UP A BIG BALL OF MUD

… or how to tackle legacy apps

Krzysztof Kucharski @ Qualaroo

QUALAROO

Customer feedback platform

WARNING!

opinionated and not code oriented

BUT BASED ON WORK OF

- ruby community
- Martin Fowler
- Sandi Metz
- Robert Martin aka Uncle Bob
- Michael Feathers

WHAT IS A LEGACY (RAILS) APP?

Has anyone ever seen this video?

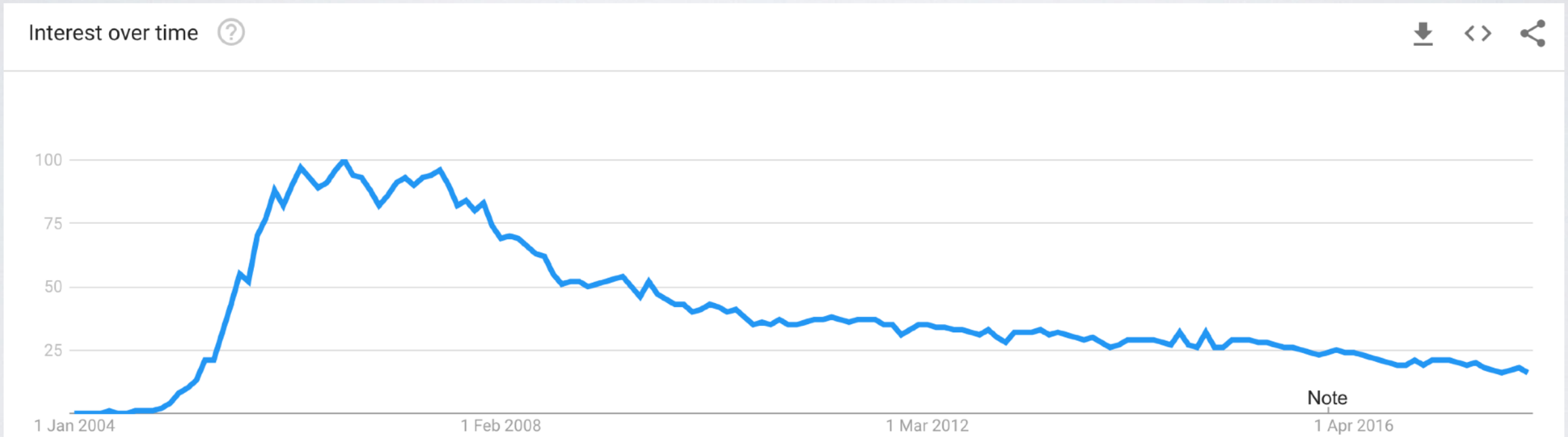
How to build a blog engine
in 15 minutes with Ruby on Rails

http://www.rubyonrails.org

By David Heinemeier Hansson,
originally prepared for the FISL 6.0 conference in Brazil 2005

MVC IS THE ANSWER!

Ruby on Rails popularity by Google Trends



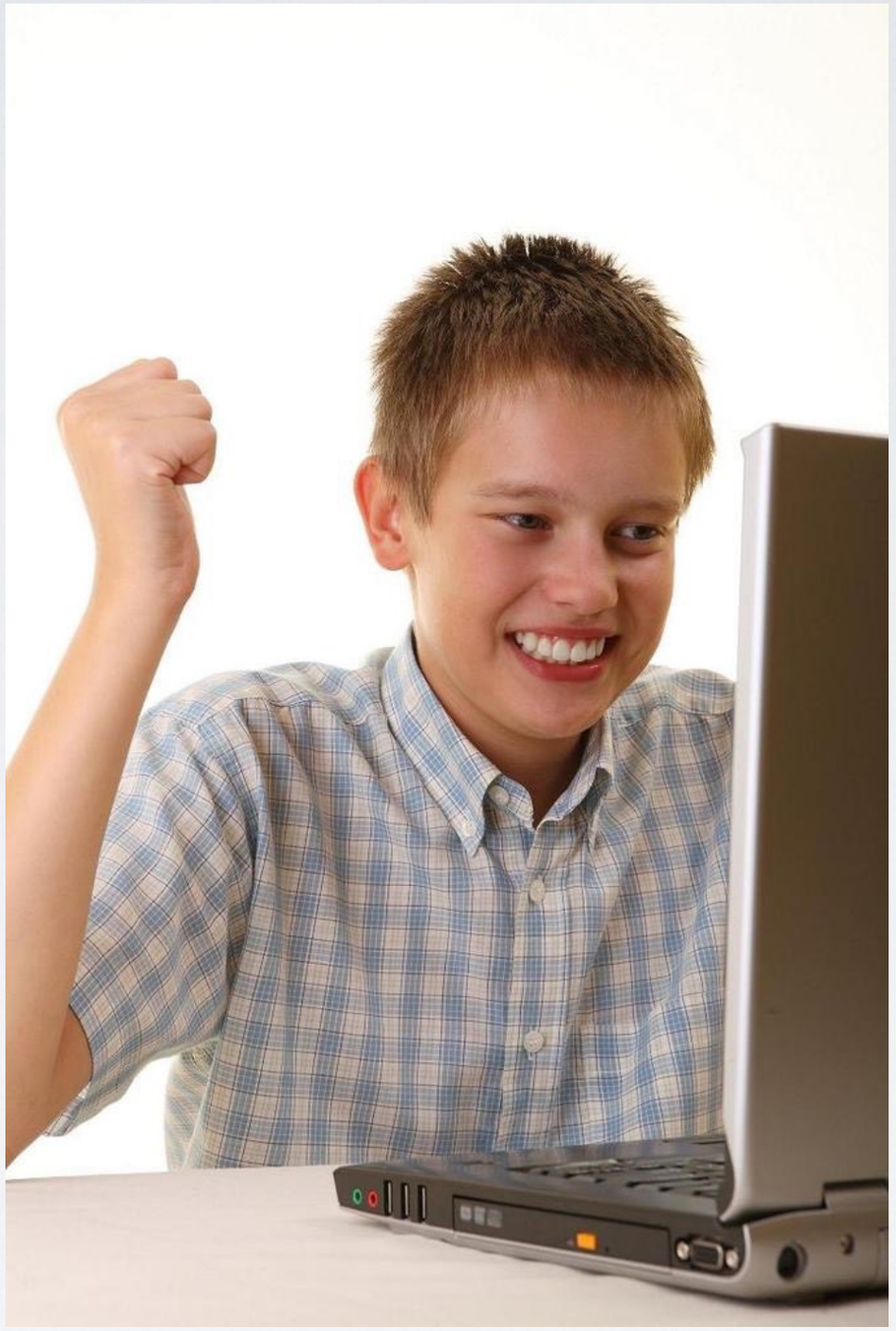
We started simple

```ruby
def new
  @post = Post.new
end

def edit
end

def create
  @post = Post.new(post_params)

  respond_to do |format|
    if @post.save
      format.html { redirect_to @post, notice: 'Post was successfully created.' }
      format.json { render :show, status: :created, location: @post }
    else
      format.html { render :new }
      format.json { render json: @post.errors, status: :unprocessable_entity }
    end
  end
end
```



new requirements came

ended up with highly coupled spaghetti

IS SPAGHETTI LEGACY?

LET'S BE MORE SCIENTIFIC

- “Code without tests” Michael Feathers, Working Effectively With Legacy Code

- “Code is legacy code as soon as it's written.” Pragmatic Programmer

- Code with too many dependencies

- Code I didn’t write or I don’t understand

LEGACY COSTS & RISKS

- hard to estimate
- apparently simple changes take a long time
- unexpected side effects

ARE WE DOOMED?

“LET’S REWRITE!”

we can use elixir in Docker containers on Kubernetes
with tomorrow’s ES9 framework

rewrite is valid in some extreme cases

- doesn't work at all
- requirements bigger than existing functionality coming

BUT MOST OF THE TIME IT'S NOT

- app serves it's purpose
- it brings business value
- it was profitable enough so far
- code is mean to an end

WE CAN PREVENT SOME OF THE TECHNICAL DEBT

PHASE ONE - ASSESMENT

Automatic tools

Run SimpleCov for test cover and save results somewhere safe for later comparisons.

Get initial code quality score.
My personal favourite here is **RubyCritic** gem.

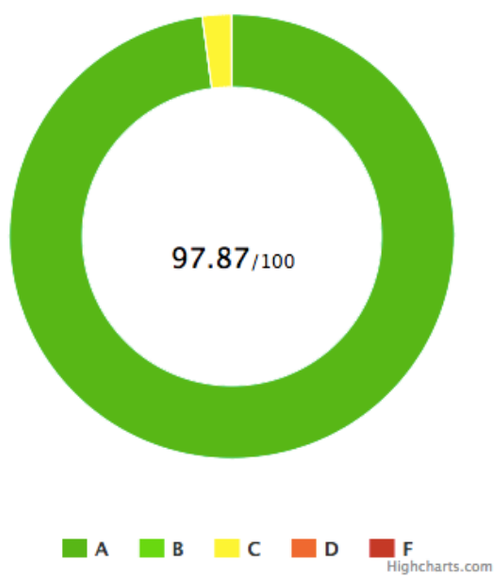
RUBYCRITIC

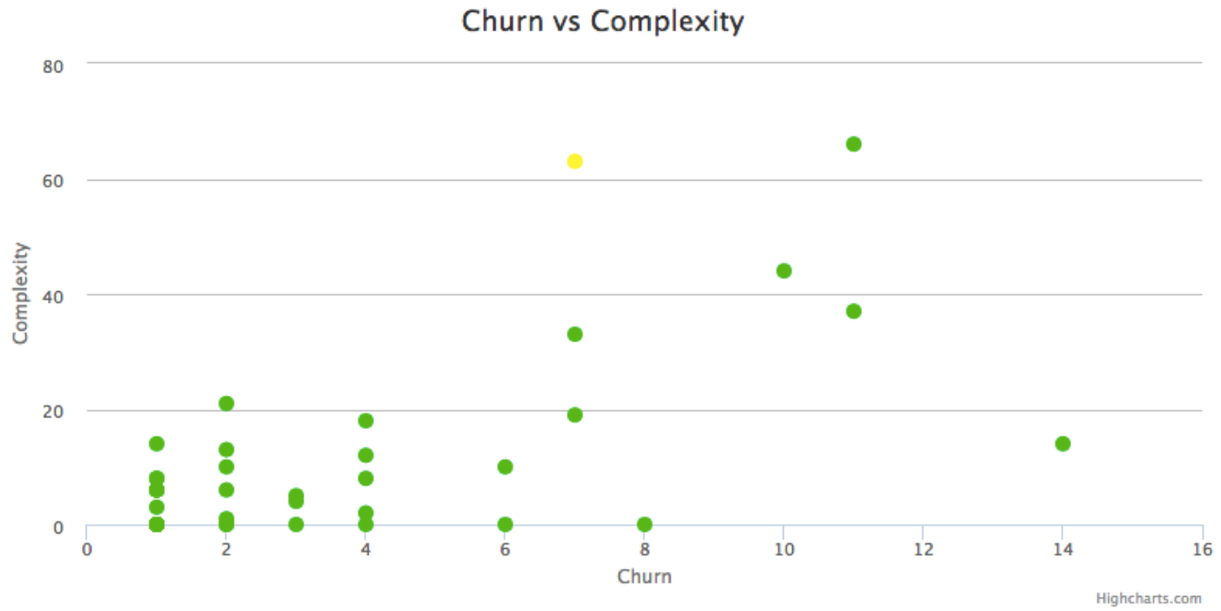
Overview

Code

Smells

Overview





Summary

A	46 files	145 churns	54 smells	B	0 files	0 churns	0 smells	C	1 files	7 churns	2 smells
D	0	0	0	F	0	0	0				

It combines power of 3 tools

Reek

Ruby code smells like unused parameters, control couple

Flay

code duplication (can also process JS or HAML)

Flog

code score as single number

Don’t treat it undeniably.

Most of the advices will consume time and effort.

Hook it up to your CI.

- Rubocop
- rails_best_practices
- Traceroute rake task

getting your hands (a little) dirty

MANUAL ASSESMENT

REVIEW THE GEMFILE

Focus on:

- no longer used gems
- duplicated responsibilities gems
- JS libs packed as gems

YOUR DB AND SCHEMA.RB

- Dead tables. Look for *updated_at* and *created_at*

- God models, first candidate for refactoring

- Slow queries. Consider adding indexes

PHASE ONE SUMMARY

- Dead code found
- Areas to refactor identified

PHASE TWO - TECHNIQUES

BREAKING DEPENDENCIES

"Dependency Management is an issue that most of us have faced. Whenever we bring up on our screens a nasty batch of tangled legacy code, we are experiencing the results of **poor dependency management.**

Poor dependency management leads to code that is hard to change, fragile, and non-reusable." *Uncle bob*

DEPENDENCY INJECTION

before

```ruby
class Car
  def start
    Engine.new.start
  end
end

# usage
car = Car.new
car.start
```

after

```ruby
class Car
  attr_writer :engine

  def start
    engine.start
  end

  def engine
    @engine ||= Engine.new
  end
end

# usage
car = Car.new
car.engine = FakeEngine

car.start
```

from *Practical Object-Oriented Design in Ruby* by *Sandi Metz*

SOLID

- **S**ingle Responsibility
- **O**pen/Closed
- **L**iskov Substitution
- **I**nterface Segregation
- **D**ependency Inversion

http://rubyblog.pro/2017/05/solid-single-responsibility-principle-by-example

SERVICE OBJECTS

to rule them all

Encapsulate complicated business logic in simple PORO.

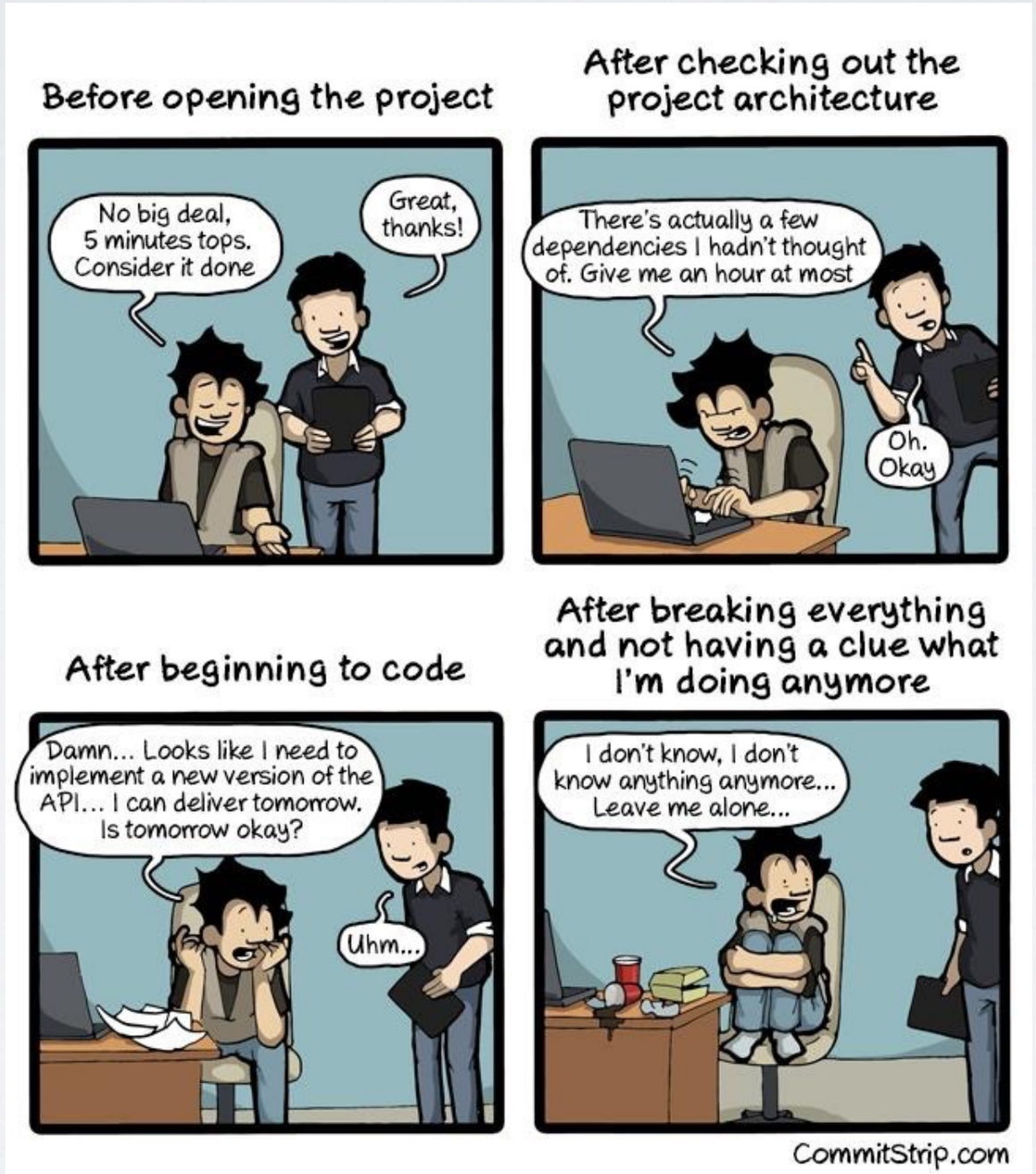
1) Just one public method
2) Return meaningful error

PHASE THREE - APPLY

REFACTOR

ALL THE THINGS

STOP.


Before opening the project
No big deal, 5 minutes tops. Consider it done
Great, thanks!
After checking out the project architecture
There's actually a few dependencies I hadn't thought of. Give me an hour at most
Oh. Okay
After beginning to code
Damn... Looks like I need to implement a new version of the API... I can deliver tomorrow. Is tomorrow okay?
Uhm...
After breaking everything and not having a clue what I'm doing anymore
I don't know, I don't know anything anymore... Leave me alone...
CommitStrip.com

DO IT IN STEPS

Implement changes gradually.

Delete code first - it’s easy.

Deploy them in order and closely monitor.

TEST FIRST, REFACTOR AFTER

Write tests to old code to gain some confidence.
Break good practices and rules if you have to.
(monkey patch, send private methods, mock external interfaces…)

After refactoring **delete ad hoc tests and write new ones**.

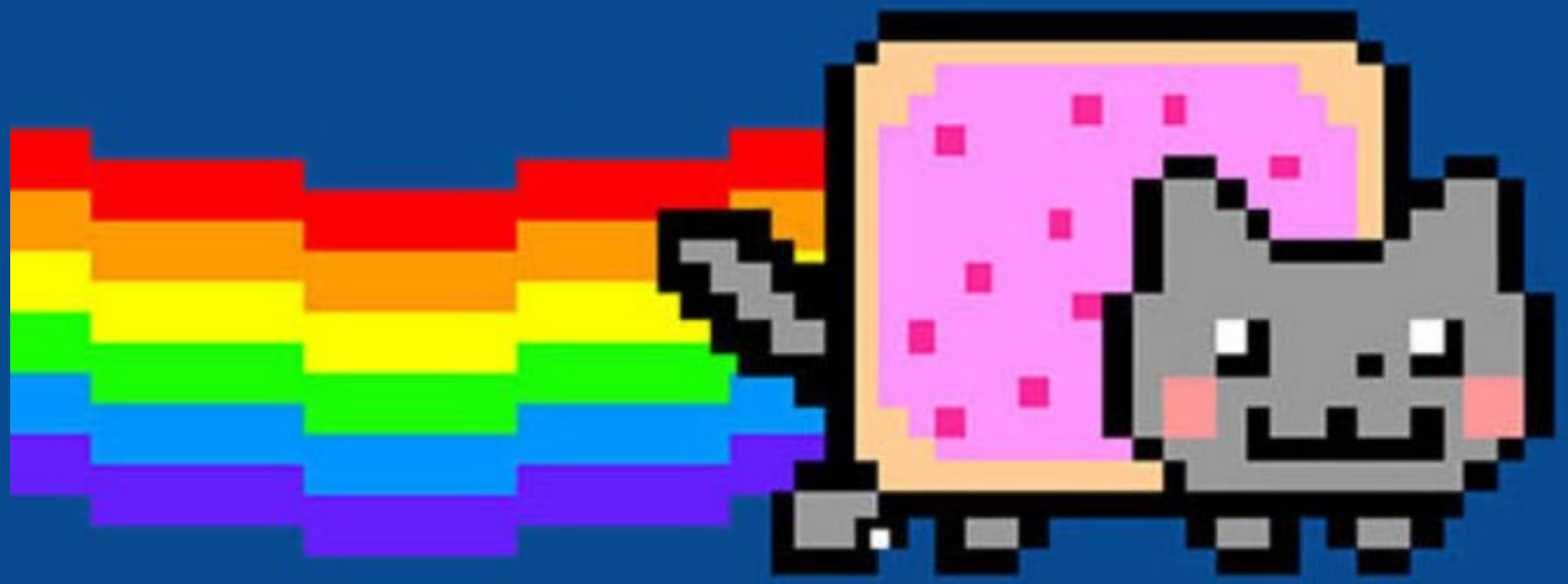
KEEP YOUR STANDARDS HIGH

Don’t fall into “it’s already broken” fallacy.

Code-Scouts rule: always try to leave code cleaner than it was before your changes.

IS MAINTAINING THE PROJECT THAT BAD ?

THERE ARE BRIGHT SIDES



YOU WILL LEARN:

- How to debug the apps
- How to optimise
- How to recognise bad code

EMBRACE NEW TECH

ES6+ is pretty good.

Don’t hate node.js by definition
and give Webpacker gem a shot.

SUMMARY

1) ANALYZE
2) LEARN NEW TECHNIQUES
3) GRADUALLY APPLY CHANGES

THANKS!