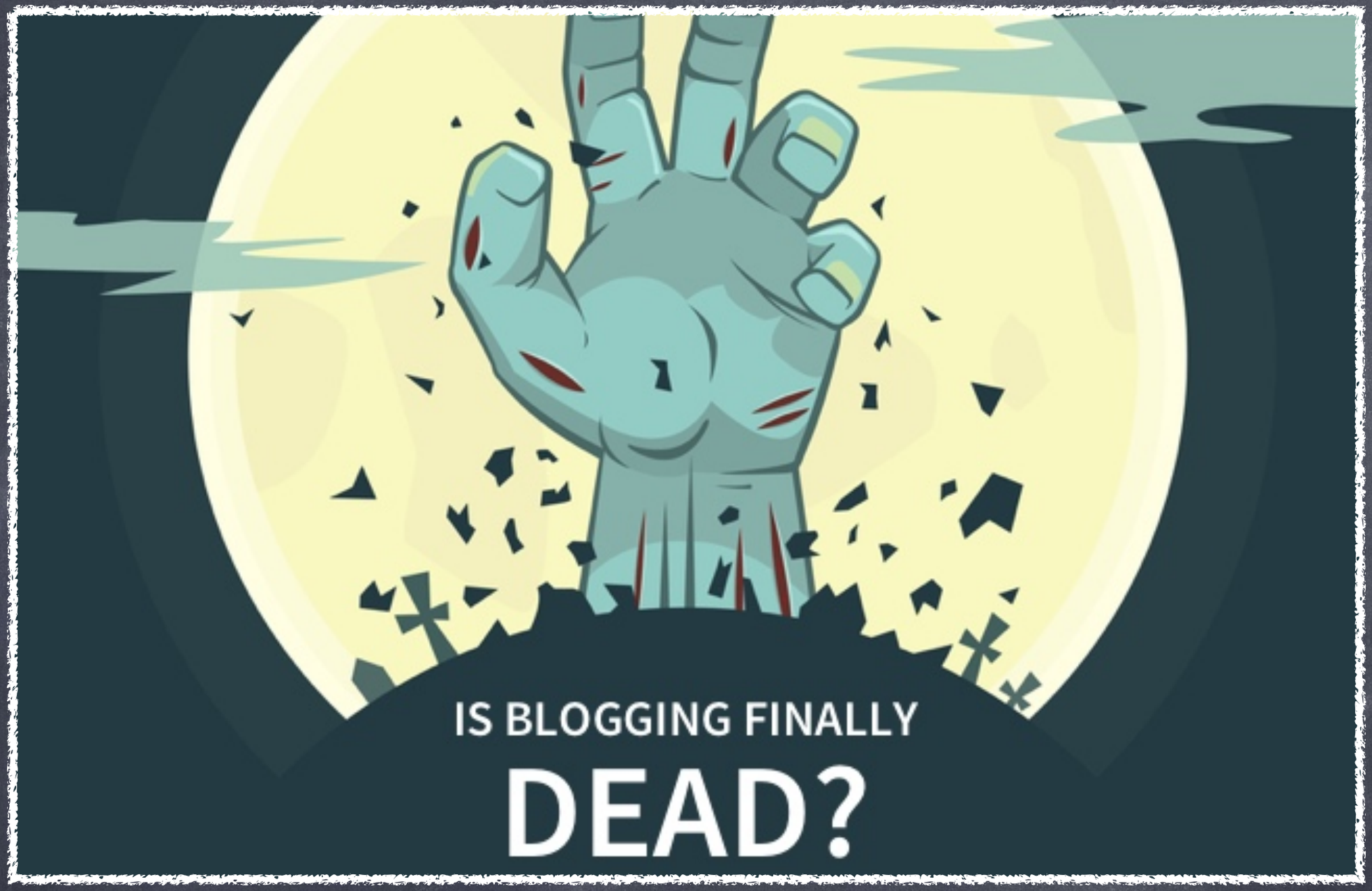
MCV

Por qué me debería importar.
Y cuanto pagan por esto?



Hoy NO vamos a programar un blog.

"Porque ya está bien de Posts y Comments. Leche!"

Vamos a programar un Reactor Nuclear.

Para calentar...

- Toda aplicación en cualquier lenguaje se puede escribir en conjunto (en línea, monolítica...)
- Es como se empezó a programar.

```ruby
#!/usr/bin/env ruby

# require 'security' # TODO: fixme
require 'rubygems'
require 'date'
require 'graph-ruby'

STATUSES = %w( OK WARNING NOT_OK DISASTER DEATH )
@reactor = { name: "Chernobyl", date: Date.new(1986, 4, 26), threshhold: 500, current_status: "OK" }
@core = { energy_out: 500, buffer: 1000 }

while @core[:buffer] > 0

  # Energy calculations
  @core[:energy_in] = rand(1000)
  @core[:buffer] = @core[:buffer] - @core[:energy_out] + @core[:energy_in]
  buffer = @core[:buffer] - @core[:energy_out]

  #status
  @core.delete(@reactor[:current_status])
  status = "OK"
  status = "WARNING" if buffer < @reactor[:threshhold]
  status = "NOT_OK" if buffer < (@reactor[:threshhold] / 2)
  status = "DISASTER" if buffer < 0
  @core[status] = buffer
  @reactor[:current_status] = status

  GraphRuby.histogram! data: @core.dup, axis: 100
  sleep(0.5)


end

puts "DEATH"
```

Bueno

- Es rápido. Muy, muy rápido.
- Es rapidísimo
- Es muy, muy rápido.
- he dicho que es rápido?

Malo

- Mas difícil:
  - de escribir
  - de mantener
  - de entender
  - de enseñar
  - de SEPARAR

Y a mi qué me importa separar?

- Para que no te griten en los foros.
- Porque si no programas orientado a objectos, que haces aquí?
- Todas las ventajas de la programación orientada a objetos viene de separaciones.
  - Por funcionalidad -> Clases
  - Por... sincronicidad? -> Workers
  - Por acceso -> Roles
  - Por lenguaje -> locales
  - Porque "Esto debería estar aquí?" -> La eterna discusión.
    - Monolithic vs APIs vs Engines
  - Por responsabilidades -> .... PATRONES

Pequeña nota importante:

- Los patrones de programación y las arquitecturas de programación a veces se explican en conjunto, otras por separado, y muchas veces causan confusión.

- La definición de ambos es "una solución a un problema común, escrita de una forma reutilizable y genérica."

- En realidad da igual como los llamemos si sabemos para qué se utilizan, y cómo se utilizan llegado el momento.

Juego

Patrones por todos lados!!! Ruby || Rails || None

Para gustos se hicieron los patrones. (En Inglés)

- Creational
  - Factories -> FactoryGirl, Generators
  - Prototype -> Form Builders, Mocks
  - Singleton -> Date, Constants
- Structural
  - Adapters -> Postgres, SQL Lite, MySQL...
  - Decorators -> View decorators, Drapper
- Behavioural
  - Iterator -> Enumerable, ActiveRecord::Relation
  - Null Object -> Nil
  - Servant -> STI
- Concurrency
  - Active Object -> Unicorn
  - Lock -> SQL, threads...
- Architectural....

Model - Controller - View

- No, no tiene nada que ver con REST
- Como patrón (o arquitectura) no esta ligado a ningún lenguaje. Mis ejemplos son en Ruby, pero se aplica a PHP (CakePHP), Java (Spring, Javalite)...
- Es una separación lógica de Responsabilidades. Y una manera de aplicar un estándar para que nos entendamos entre programadores.
  - Piensa que tal vez, (ojalá) tu código lo mantenga otra persona.

Todo es muy bonito hoy que funciona pero, y el día de mañana??

- Que especificaciones tengo?
  - Diferentes usuarios -> Estadistas, técnicos, business...
  - Seguridad -> Injections, Escaping, Man in middle attack
- Cual es la solución más fácil que funciona?

AL Lio

Como separarías nuestra central nuclear?

Config

Model

Model

Model / Controller

Controller/View

View

```ruby
#!/usr/bin/env ruby

# require 'security' # TODO: fixme
require 'rubygems'
require 'date'
require 'graph-ruby'

STATUSES = %w( OK WARNING NOT_OK DISASTER DEATH )
@reactor = { name: "Chernobyl", date: Date.new(1986, 4, 26), threshhold: 500, current_status: "OK" }
@core = { energy_out: 500, buffer: 1000 }

while @core[:buffer] > 0

  # Energy calculations
  @core[:energy_in] = rand(1000)
  @core[:buffer] = @core[:buffer] - @core[:energy_out] + @core[:energy_in]
  buffer = @core[:buffer] - @core[:energy_out]

  #status
  @core.delete(@reactor[:current_status])
  status = "OK"
  status = "WARNING" if buffer < @reactor[:threshhold]
  status = "NOT_OK" if buffer < (@reactor[:threshhold] / 2)
  status = "DISASTER" if buffer < 0
  @core[status] = buffer
  @reactor[:current_status] = status

  GraphRuby.histogram! data: @core.dup, axis: 100
  sleep(0.5)


end

puts "DEATH"

```

Cheat Sheet - Model

- Calculations
- Validations
- conversions
- Syntactic Sugar
- Basically, any WORK

```
36
37 ▼ def release_energy
38     update(buffer: buffer - energy_out)
39 ▲ end
40
41 ▼ def recharge
42     update(buffer: buffer + energy_in)
43 ▲ end
```

```
44
45 ▼ def formatted_status
46     "<div class='#{name}'> #{status} </div>"
47 ▲ end
48
```

Cheat Sheet - View

- HTML
- CSS
- JAVASCRIPT

```
1  - content_for(:head) do
2    = javascript_include_tag 'nuclify'
3
4  %h1= @reactor.name
5
6  = form_for @reactor do |f|
7    = f.field :name
8    = f.field :threshhold
9    = f.submit
10
```

- AND THATS IT.

  (No, ni en ese caso especial porque tu app es tan complicada que bla bla bla...)

```
10
11 %table
12   - @reactor.reactor_cores.each do |core|
13     %tr
14       %td= core.name
15       %td
16         - enough_buffer = core.buffer - core.energy_out
17         = "You are " + engough_buffer > 0 ? "Alive" : "Dead"
18
```

Cheat Sheet - Controllers (La difícil)

- Assignments
- Error processing
  (Que no validations)
- Control the model
- Create Jobs
  - "Send Email"
  - Queue Workers
- INVOCATION

```
37
38 ▼ class NuclearController
39 ▼   def perform_emergency_shutdown
40       @reactor = Reactor.find(params[:id])
41
42 ▼     if @reactor.status == "DEATH"
43         EmergencyMailer.inform_workers(@reactor).deliver
44         EmergencyMailer.inform_families(@reactor).deliver
45         @reactor.emergency_shutdown
46         @reactor.emergency_evacucation
47         notice = "You are all dead now, thank you for using our system."
48       else
49         @reactor.emergency_shutdown
50         notice = "That was close. You are not so dead yet."
51 ▲     end
52       redirecto_to @reactor, notice: notice
53 ▲   end
54 ▲ end
55
```

```
54
55 ▼   def generate_report
56       @reactors = Reactor.all
57       str = "<xml reactors=#{@reactors.count}>"
58       str += @reactors.map{ |reactor|  "<reactor name=#{reactor.name}>....</reactor>" }.join
59       str += "</xml>"
60 ▲   end
61
```

```
61
62 ▼   def show_energy
63       @energy1 = @current_reactor.buffer - @current_reactor.energy_out
64       @energy2 = @current_reactor.buffer - @current_reactor.energy_in
65 ▲   end
66
```

Por qué parar ahí??

Cheat Sheet - Jobs

- Asynchronous work
  - Email
  - Importing big files
  - Uploading big files
- En realidad cualquier cosa que tenga que ver con archivos grandes.
- api work. (concurrency)

```
89
90 ▼ class ReactorCoreWorker
91 ▼   def perform(core_id)
92       core = Core.find(id)
93       core.energy_in = rand(1000)
94       core.energy_out = core.output
95       core.recharge
96       core.release_energy
97 ▲   end
98 ▲ end
99
```

Cheat Sheet - Decorators

- Cualquier función del modelo que se encuentre escribiendo código de la vista (Statuses!!)

```
81
82 ▼ class ReactorCoreDecorator
83     STATUSES = %w( OK WARNING NOT_OK DISASTER DEATH )
84
85 ▼   def formatted_status
86       content_tag :div, status.humanize
87 ▲   end
88 ▲ end
89
```

Cheat Sheet - General

- Si crees que necesitas algo que no tiene sitio, y has mirado, y no encuentras el lugar adecuado: Mira otra vez.
- Utiliza rake tasks, lib, config, deploy... bien (para lo que están diseñados) y ahorrarás tiempo, humillación en foros, y tu aplicación tendrá mas posibilidades de perdurar.
- Si tienes que incluir ActionView::Helpers en el modelo o controlador: lo estas haciendo mal. Piensa en tu diseño.
- Si tienes que poner - en vez de = en tus vistas: Exacto! lo estas haciendo mal. Piensa en tu diseño.
- Si te ves escribiendo código así:

```
38 ▼ class NuclearController
39
40 ▼   def emc2
41       @core = ReactorCore.find(params[:id])
42       @core.energy_in = rand(100)
43       @core.energy = @core.mass * ReactorCore::SPEED_OF_LIGHT * ReactorCore::SPEED_OF_LIGHT
44 ▲   end
45
46 ▲ end
47
```

- Bingo.
- Y la prueba esta en los caracteres:

```
35
36 ▼ class ReactorCore
37
38 ▼   def emc2
39       energy_in = rand(100)
40       energy = mass * SPEED_OF_LIGHT * SPEED_OF_LIGHT
41 ▲   end
42
43 ▲ end
```

Ya se acabó

Jacobo Carbó - jcpenche@gmail.com - 630031143