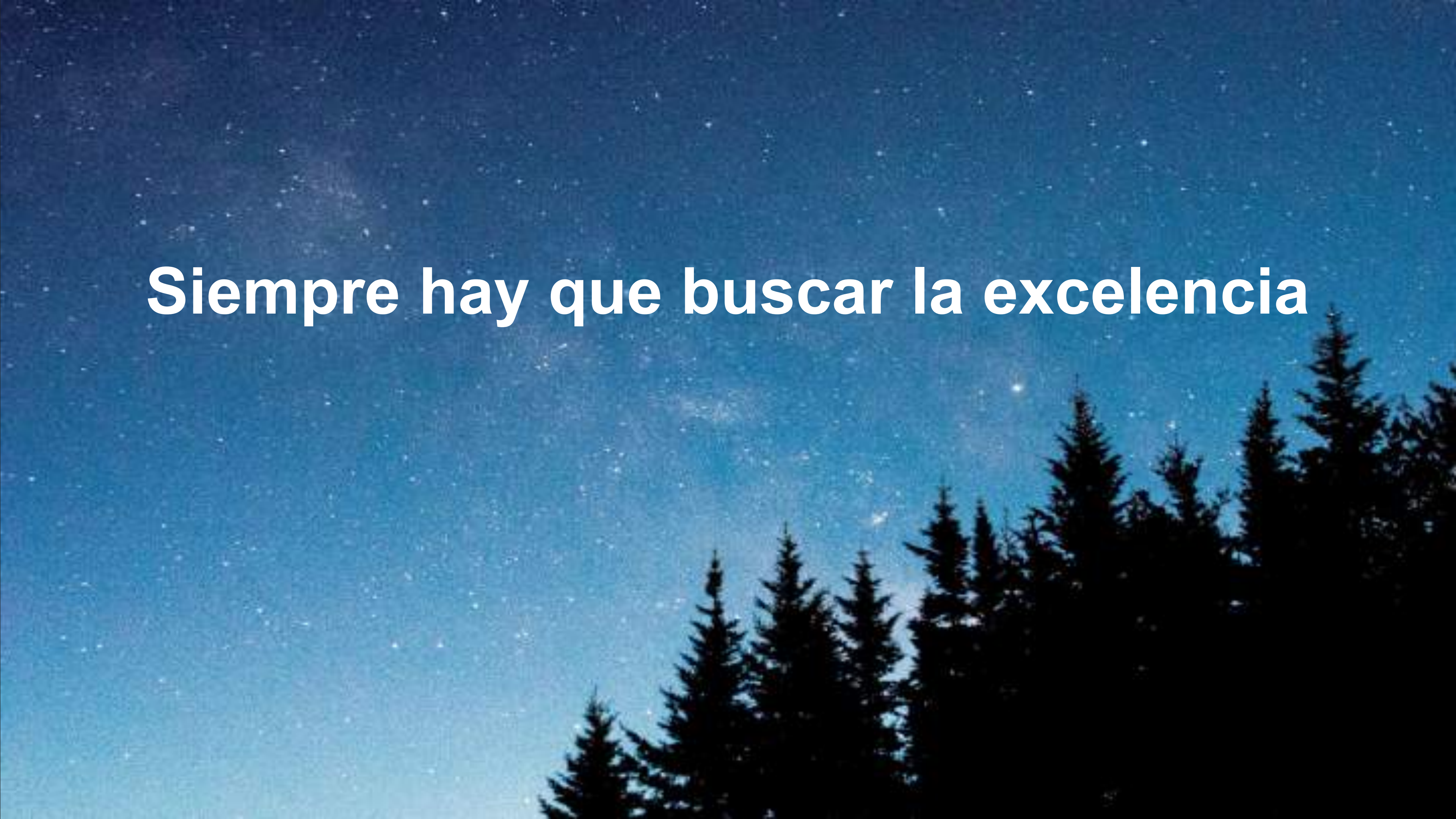
La Idealización del Código

Francisco Quintero

- Ing. de Software
- 8 +/- de experiencia
- Vean películas de Denis Villeneuve



Siempre hay que buscar la excelencia

En todos lados hay código maluco.

¡Uy, Echeverrí!

- Empanadas
- Código espaguete
- Cero documentación
- Clases muy complejas
- Métodos muy largos
- Etc

¿Por qué pasa?

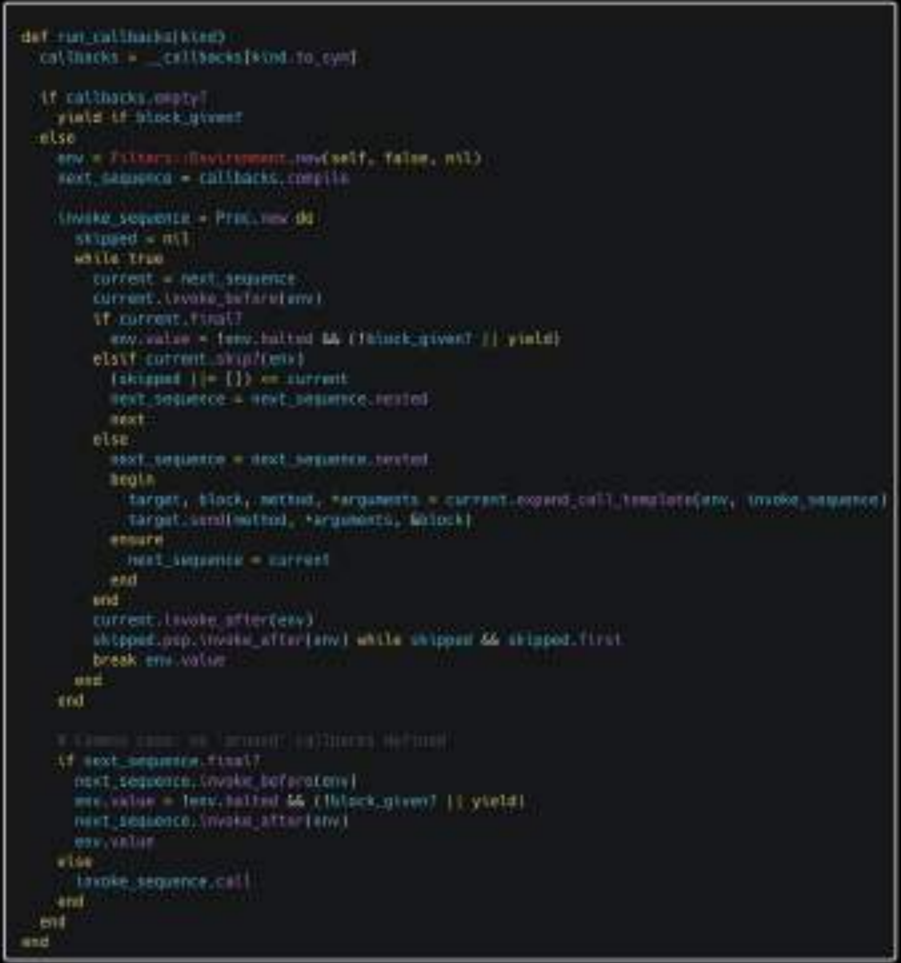
“Big things can only get bigger”

- Sandy Metz

“Sin el contexto de la decisión tomada, es muy fácil decir que ese código está mal hecho”

- X

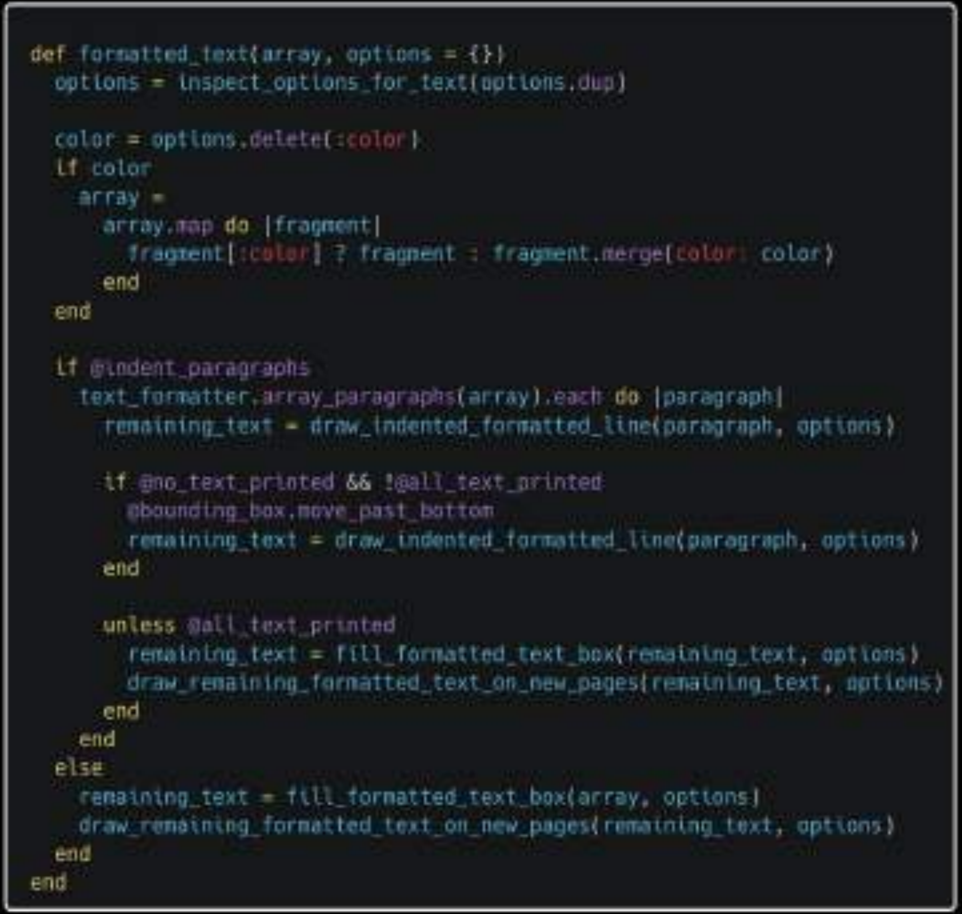
- GitLab
- Empresa conocida de Gases
- Ruby on Rails
- Prawn PDF



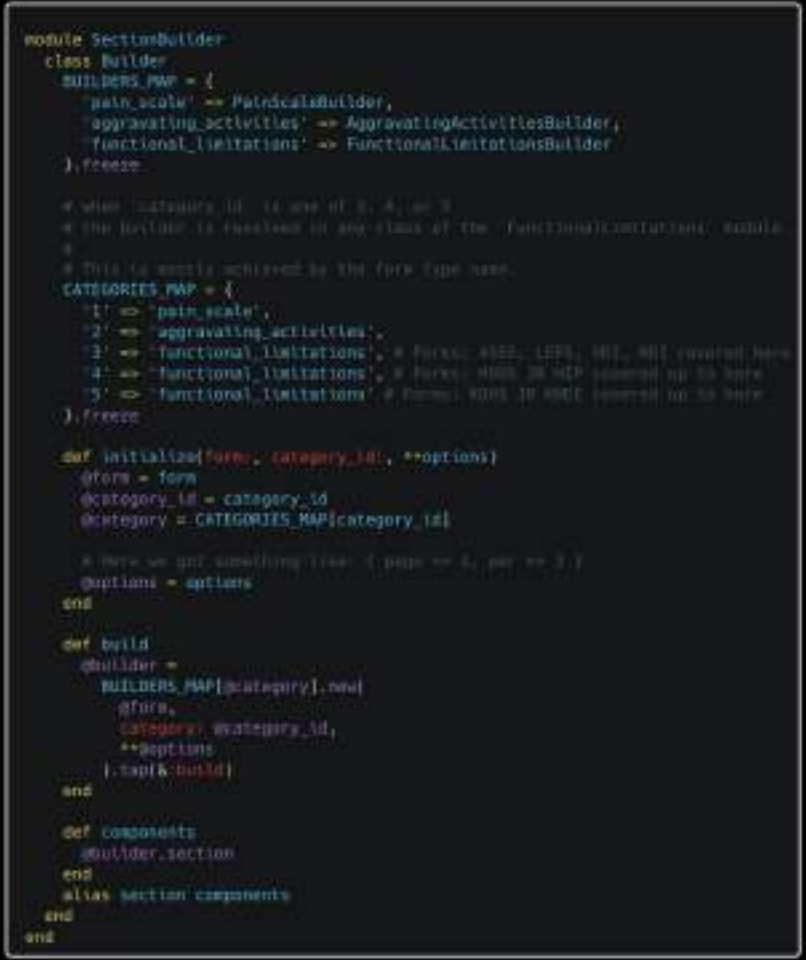
```ruby
def run_callbacks(kind)
  callbacks = __callbacks[kind.to_sym]

  if callbacks.empty?
    yield if block_given?
  else
    env = Filters::Environment.new(self, false, nil)
    next_sequence = callbacks.compile

    invoke_sequence = Proc.new do
      skipped = nil
      while true
        current = next_sequence
        current.invoke_before(env)
        if current.final?
          env.value = !env.halted && (!block_given? || yield)
        elsif current.skip?(env)
          (skipped ||= []) << current
          next_sequence = next_sequence.nested
          next
        else
          next_sequence = next_sequence.nested
          begin
            target, block, method, *arguments = current.expand_call_template(env, invoke_sequence)
            target.send(method, *arguments, &block)
          ensure
            next_sequence = current
          end
        end
        current.invoke_after(env)
        skipped.pop.invoke_after(env) while skipped && skipped.first
        break env.value
      end
    end

    # Common case: no 'around' callbacks defined
    if next_sequence.final?
      next_sequence.invoke_before(env)
      env.value = !env.halted && (!block_given? || yield)
      next_sequence.invoke_after(env)
      env.value
    else
      invoke_sequence.call
    end
  end
end
```

En Ruby on Rails

En Prawn PDF



```ruby
def formatted_text(array, options = {})
  options = inspect_options_for_text(options.dup)

  color = options.delete(:color)
  if color
    array =
      array.map do |fragment|
        fragment[:color] ? fragment : fragment.merge(color: color)
      end
  end

  if @indent_paragraphs
    text_formatter.array_paragraphs(array).each do |paragraph|
      remaining_text = draw_indented_formatted_line(paragraph, options)

      if @no_text_printed && !@all_text_printed
        @bounding_box.move_past_bottom
        remaining_text = draw_indented_formatted_line(paragraph, options)
      end

      unless @all_text_printed
        remaining_text = fill_formatted_text_box(remaining_text, options)
        draw_remaining_formatted_text_on_new_pages(remaining_text, options)
      end
    end
  else
    remaining_text = fill_formatted_text_box(array, options)
    draw_remaining_formatted_text_on_new_pages(remaining_text, options)
  end
end
```



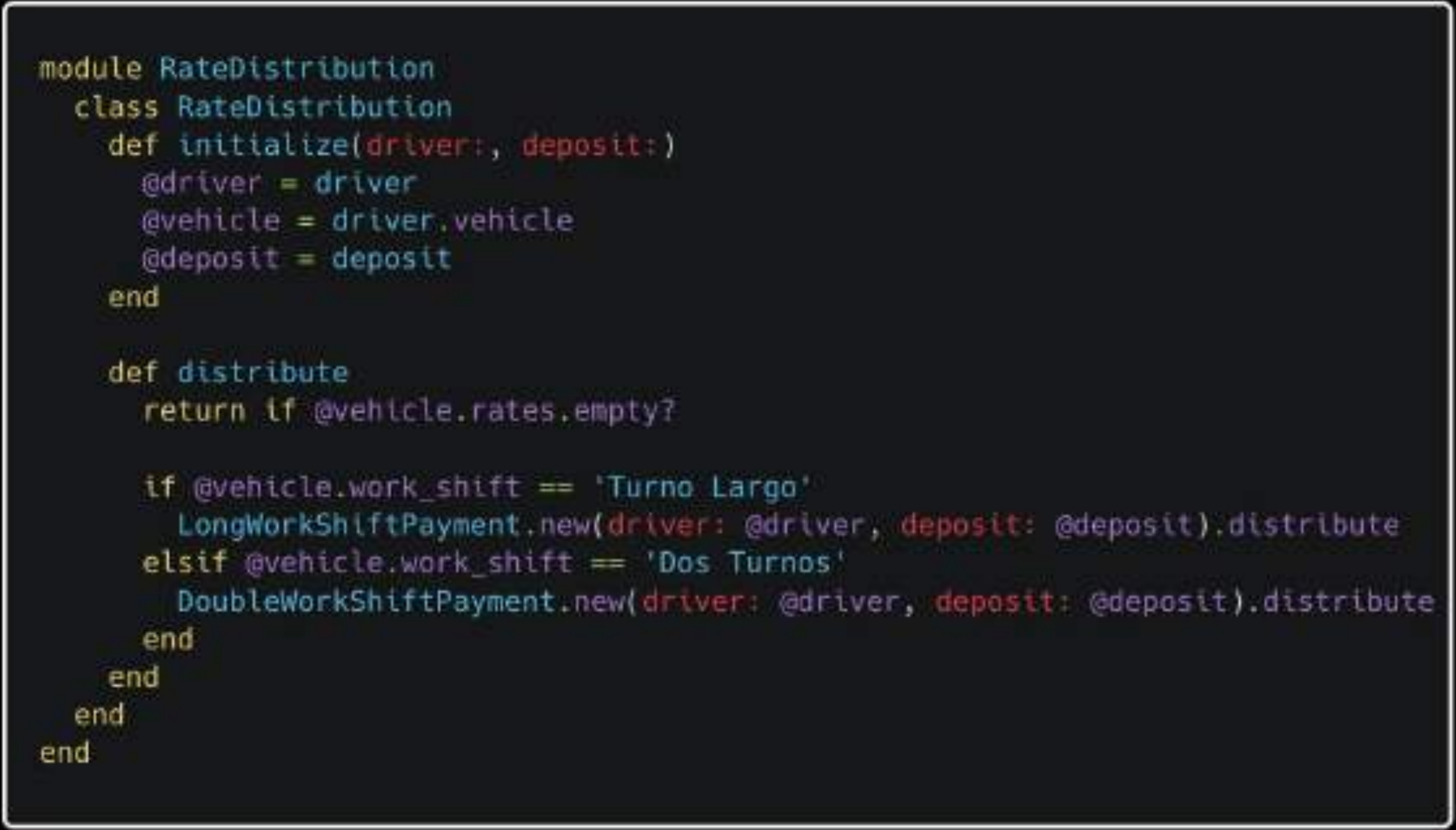
```ruby
module SectionBuilder
  class Builder
    BUILDERS_MAP = {
      'pain_scale' => PainScaleBuilder,
      'aggravating_activities' => AggravatingActivitiesBuilder,
      'functional_limitations' => FunctionalLimitationsBuilder
    }.freeze

    # When `category_id` is one of 3, 4, or 5
    # the builder is resolved in any class of the `FunctionalLimitations` module.
    #
    # This is mostly achieved by the form type name.
    CATEGORIES_MAP = {
      '1' => 'pain_scale',
      '2' => 'aggravating_activities',
      '3' => 'functional_limitations', # Forms: ASES, LEFS, NDI, ODI covered here
      '4' => 'functional_limitations', # Forms: HOOS JR HIP covered up to here
      '5' => 'functional_limitations' # Forms: KOOS JR KNEE covered up to here
    }.freeze

    def initialize(form:, category_id:, **options)
      @form = form
      @category_id = category_id
      @category = CATEGORIES_MAP[category_id]

      # Here we got something like: { page => 1, per => 2 }
      @options = options
    end

    def build
      @builder =
        BUILDERS_MAP[@category].new(
          @form,
          category: @category_id,
          **@options
        ).tap(&:build)
    end

    def components
      @builder.section
    end
    alias section components
  end
end
```

Código propio

En Gitlab: user.rb

Patrones de Diseño, ¿para qué?

The greatest risk with design patterns is over-application. Not every problem can be solved cleanly with an existing design pattern; don't try to force a problem into a design pattern when a custom approach will be cleaner. Using design patterns doesn't automatically improve a software system; it only does so if the design patterns fit. As with many ideas in software design, the notion that design patterns are good doesn't necessarily mean that more design patterns are better.



```ruby
module RateDistribution
  class RateDistribution
    def initialize(driver:, deposit:)
      @driver = driver
      @vehicle = driver.vehicle
      @deposit = deposit
    end

    def distribute
      return if @vehicle.rates.empty?

      if @vehicle.work_shift == 'Turno Largo'
        LongWorkShiftPayment.new(driver: @driver, deposit: @deposit).distribute
      elsif @vehicle.work_shift == 'Dos Turnos'
        DoubleWorkShiftPayment.new(driver: @driver, deposit: @deposit).distribute
      end
    end
  end
end
```

¿Solución?

- Code reviews
- Pair programming
- Líderes interesados por la salud del proyecto
- Espacios para resolver deuda técnica
- Linters automatizados

Hay que aceptarlo y aprender a vivir con ello.

Fuentes: “Mi mamá me dijo”

- La Idealización del Código https://otroespacioblog.wordpress.com/2021/02/25/la-idealizacion-del-codigo/
- Clean, DRY, SOLID Spaghetti https://dev.to/codemouse92/clean-dry-solid-spaghetti-1lgm
- You dont believe in clean code https://dev.to/danlebrero/you-dont-believe-in-clean-code-113n
- Are There Actually Companies out There That Write Good Code? https://dev.to/daedtech/are-there-actually-companies-out-there-that-write-good-code-1pbo
- “A Philosophy of Software Design” - John Osterhout

gracias!/
HONEY