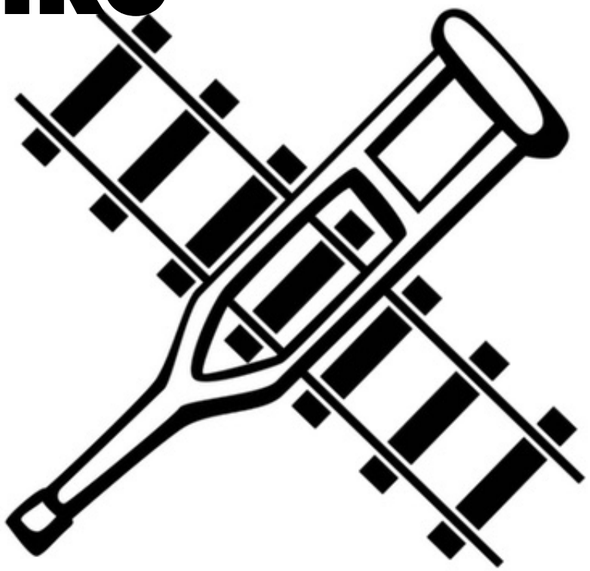
How to stop being Rails-developer

Ivan Nemytchenko

inem.at

@inemation

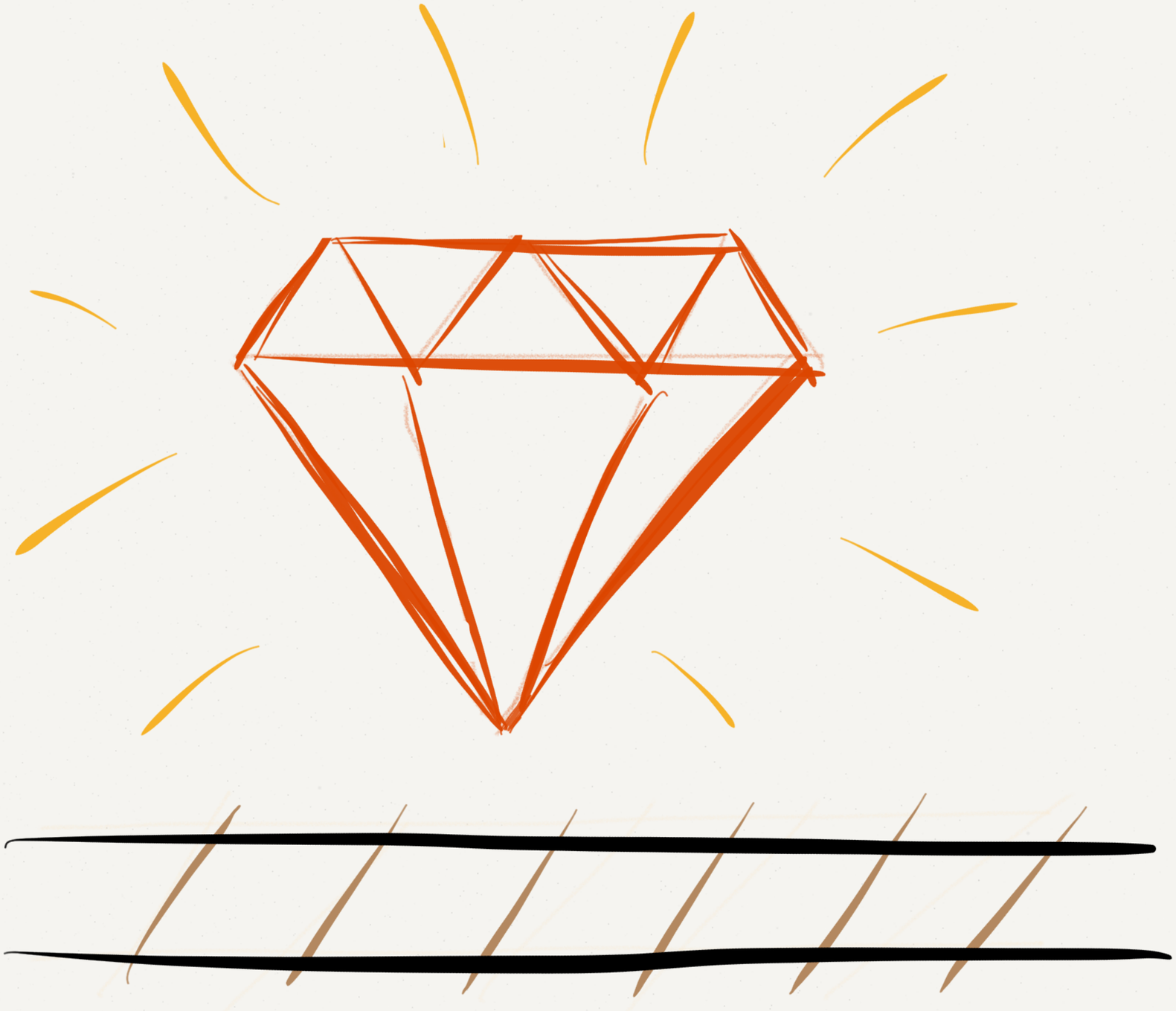


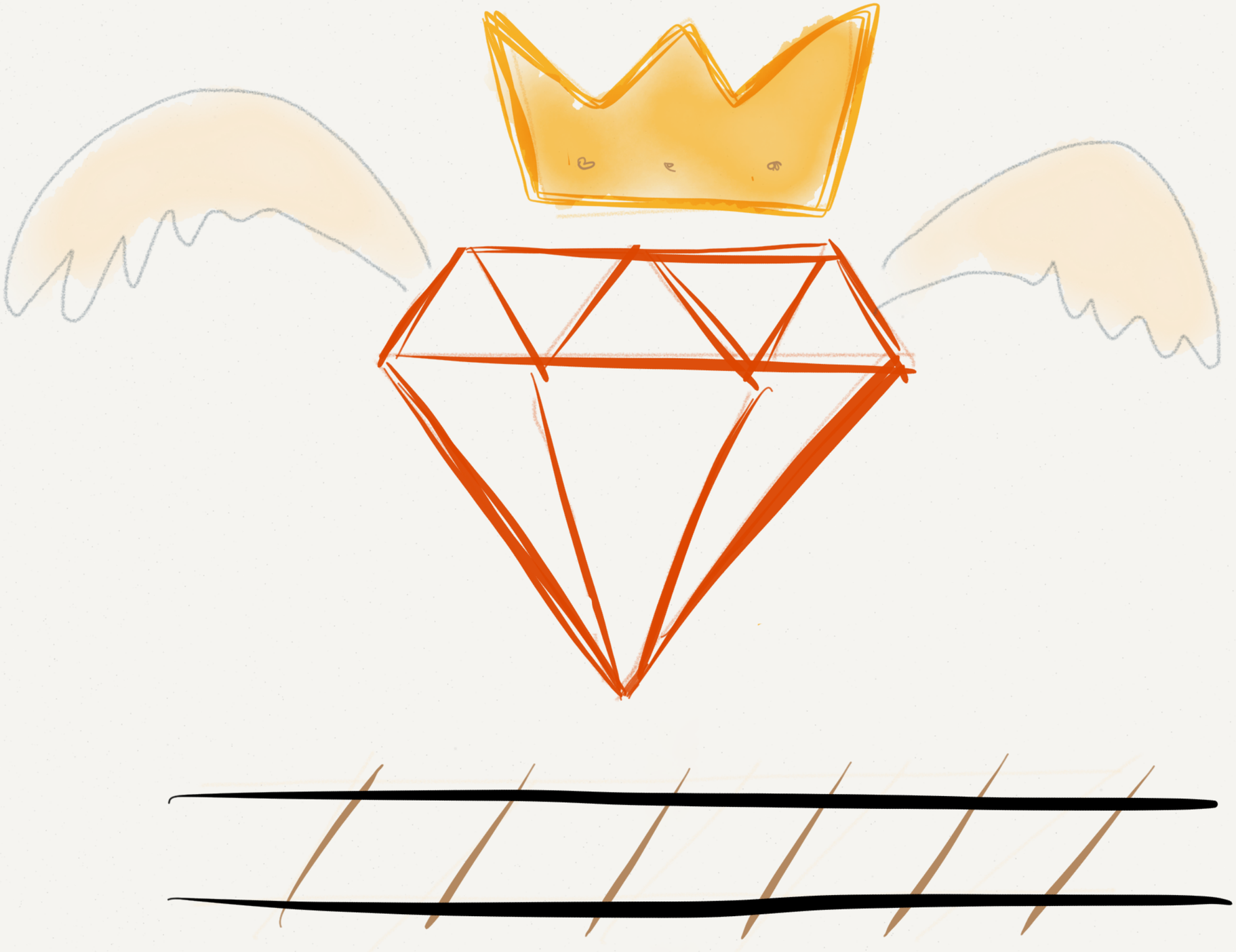
Ivan Nemytchenko

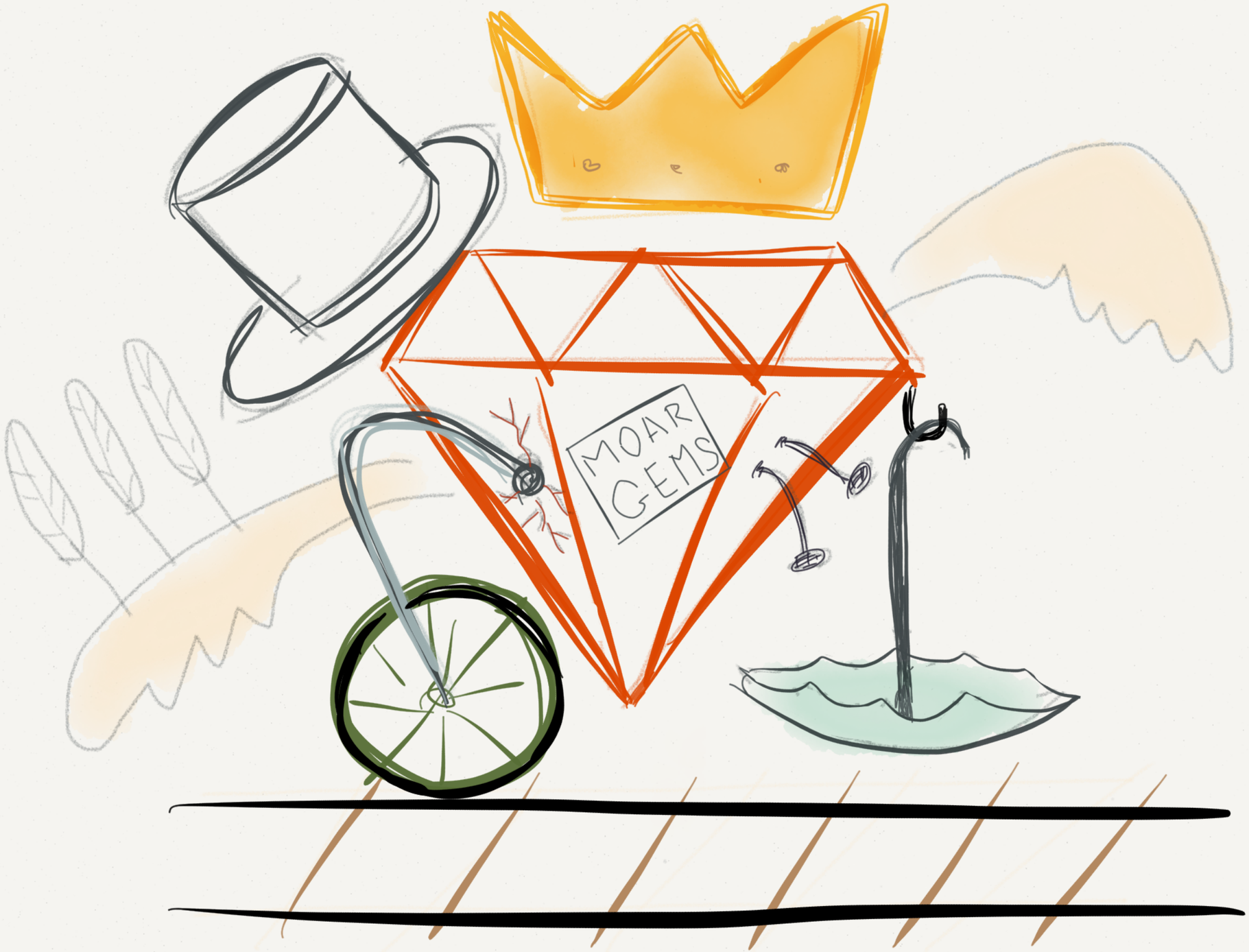
- 14 years in IT
- cofounded 2 agencies
- occasional speaker
- coorganized 2 IT conferences
- worked in a startup
- worked as project-manager
- working with Rails since 2006
- author of **RailsHurts.com**
- started remote internship for ruby junior developers
- twitter: **@inemation**

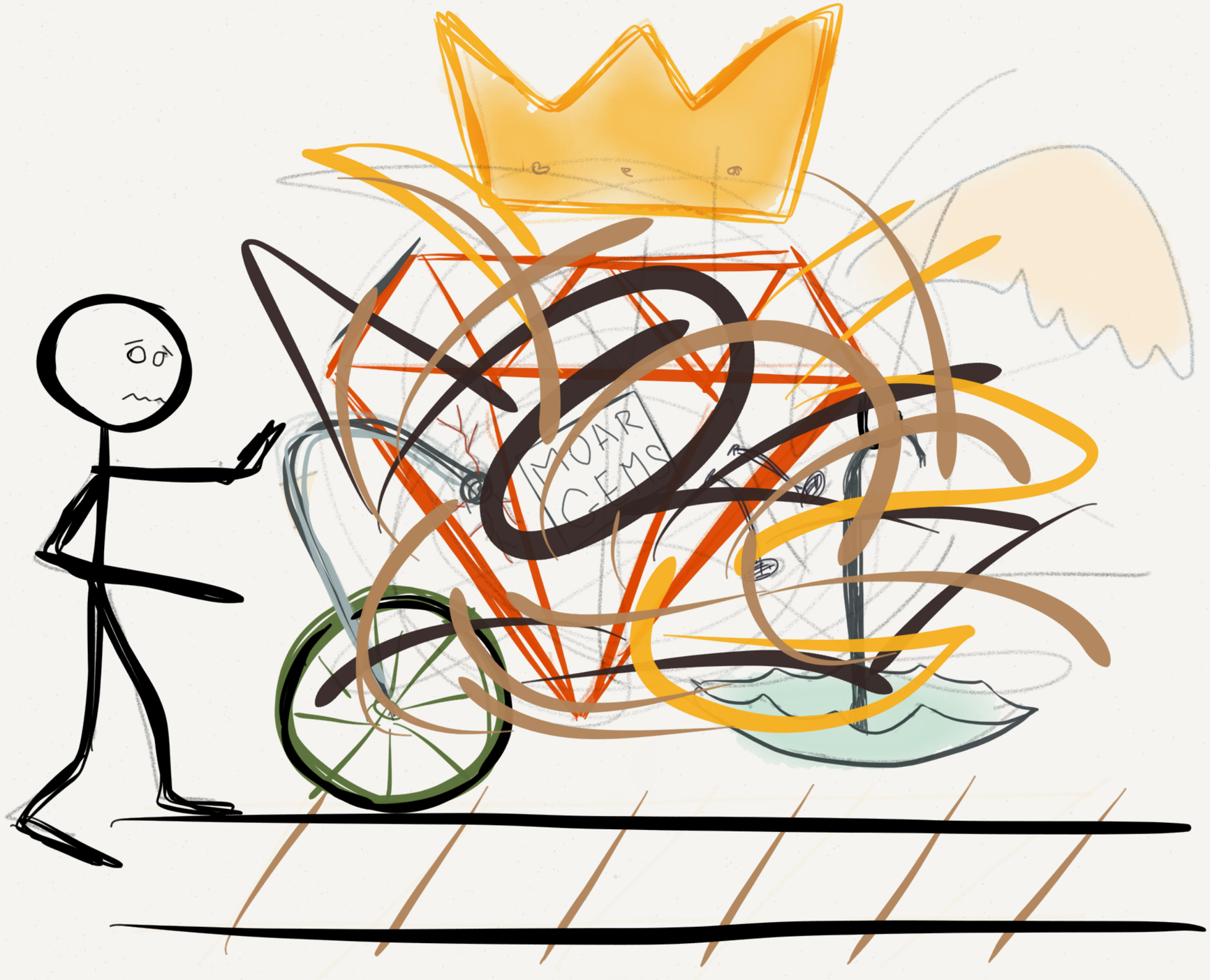
What is wrong with Rails?

Same pattern






MOAR
GEMS

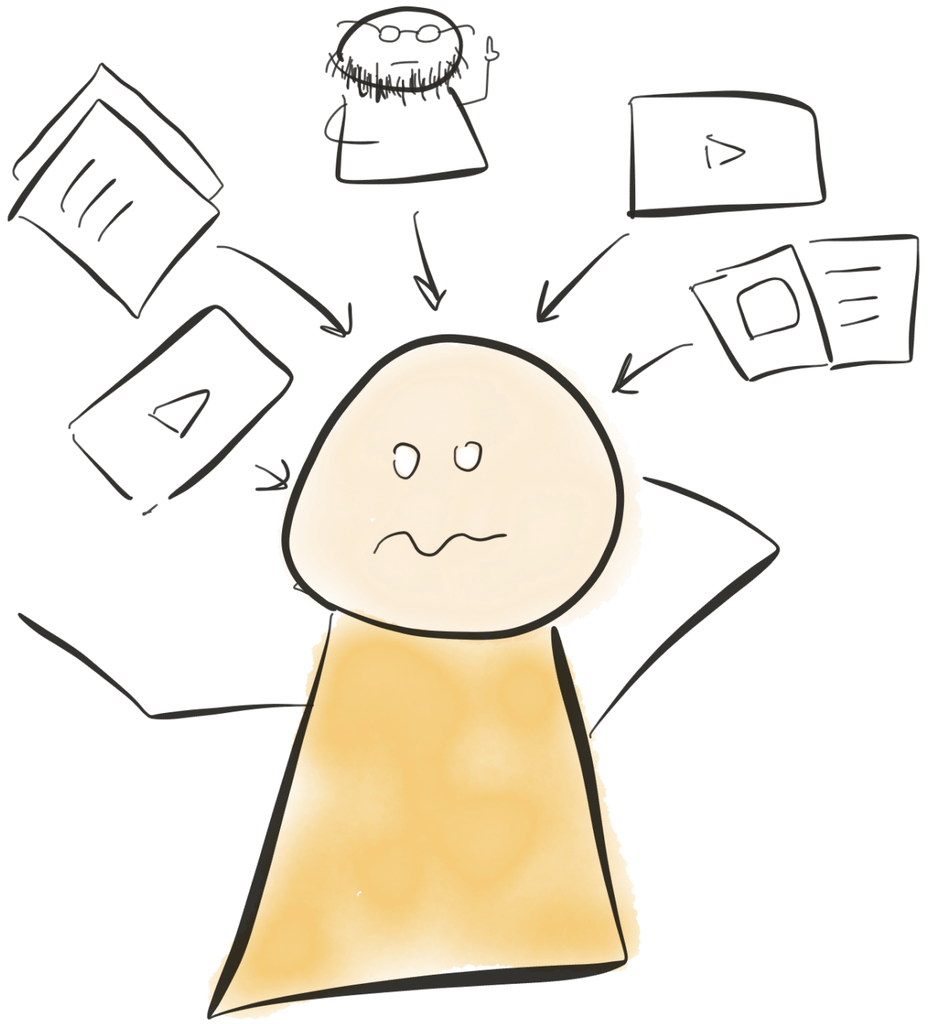

MOAR
GEMS

$$\text{Mess} = \frac{\text{App size}}{\text{Ability to manage complexity}}$$

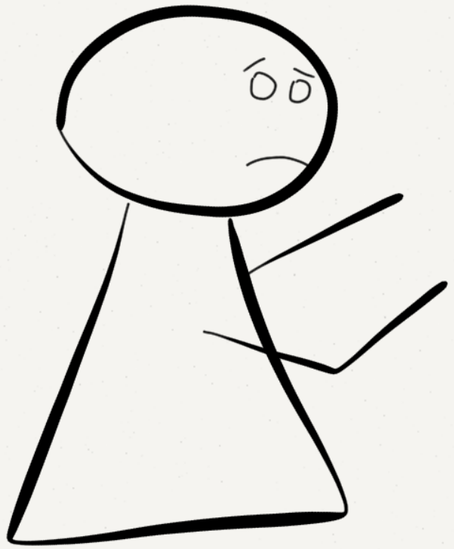
Rails-way is not enough

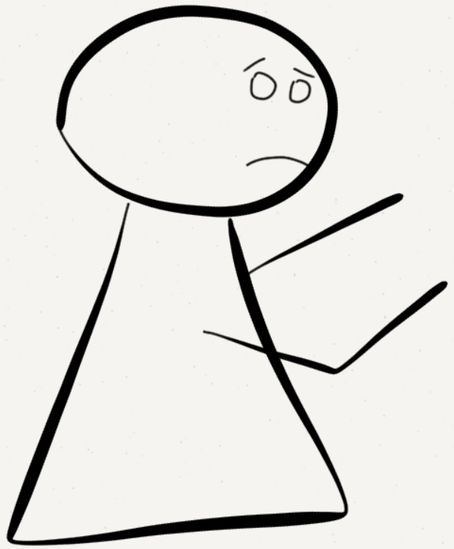
These things might help you

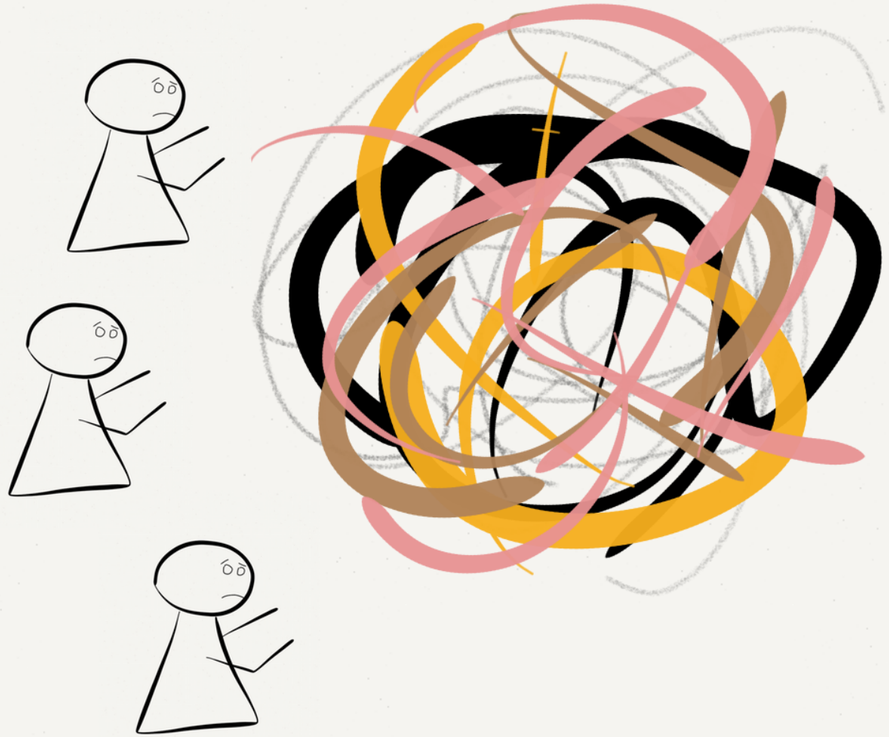
- SOLID principles
- Design Patterns
- Refactoring techniques
- Architecture types
- Code smells identification
- Testing best practices



Knowing about something doesn't mean you know how to apply it





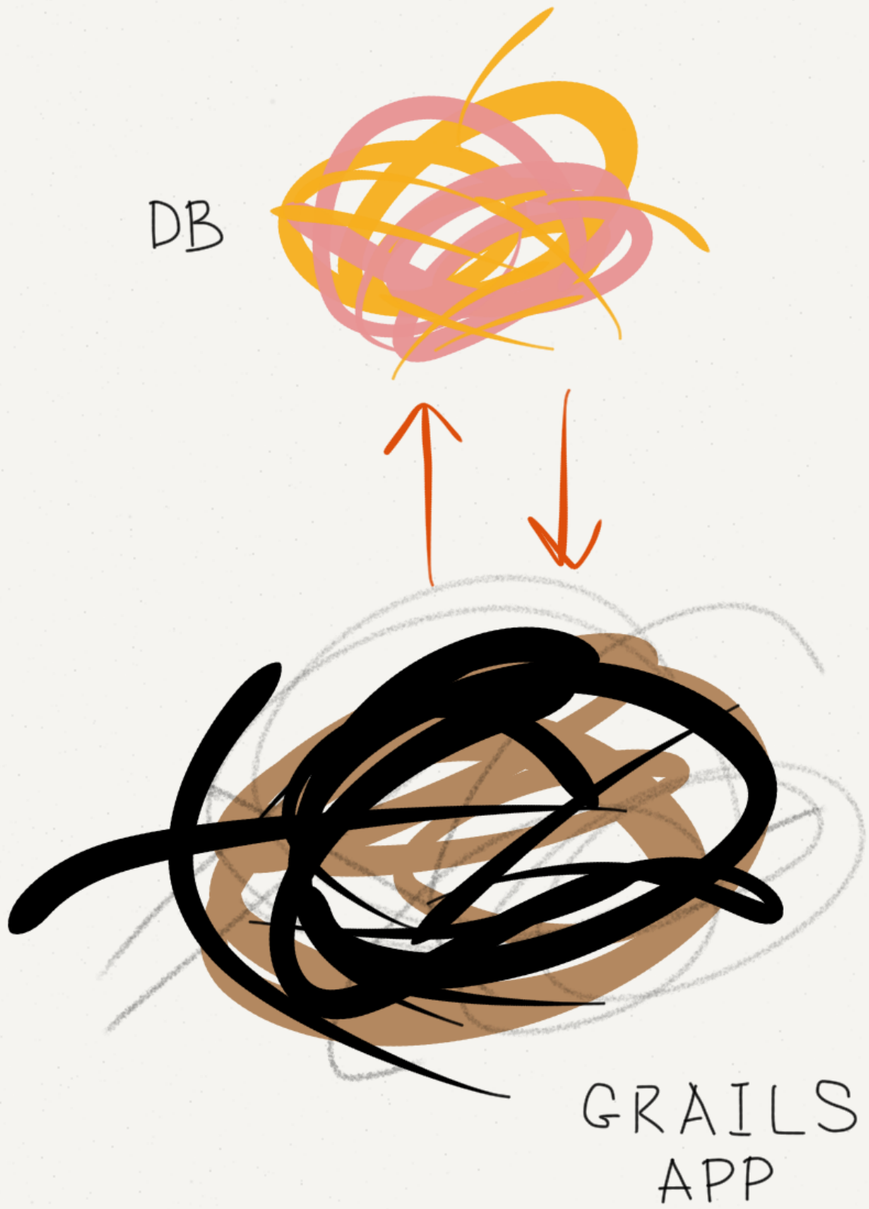


Problem:

- Current team stuck

Project:

- Monolythic Grails app
- Mysql database of ~70 tables

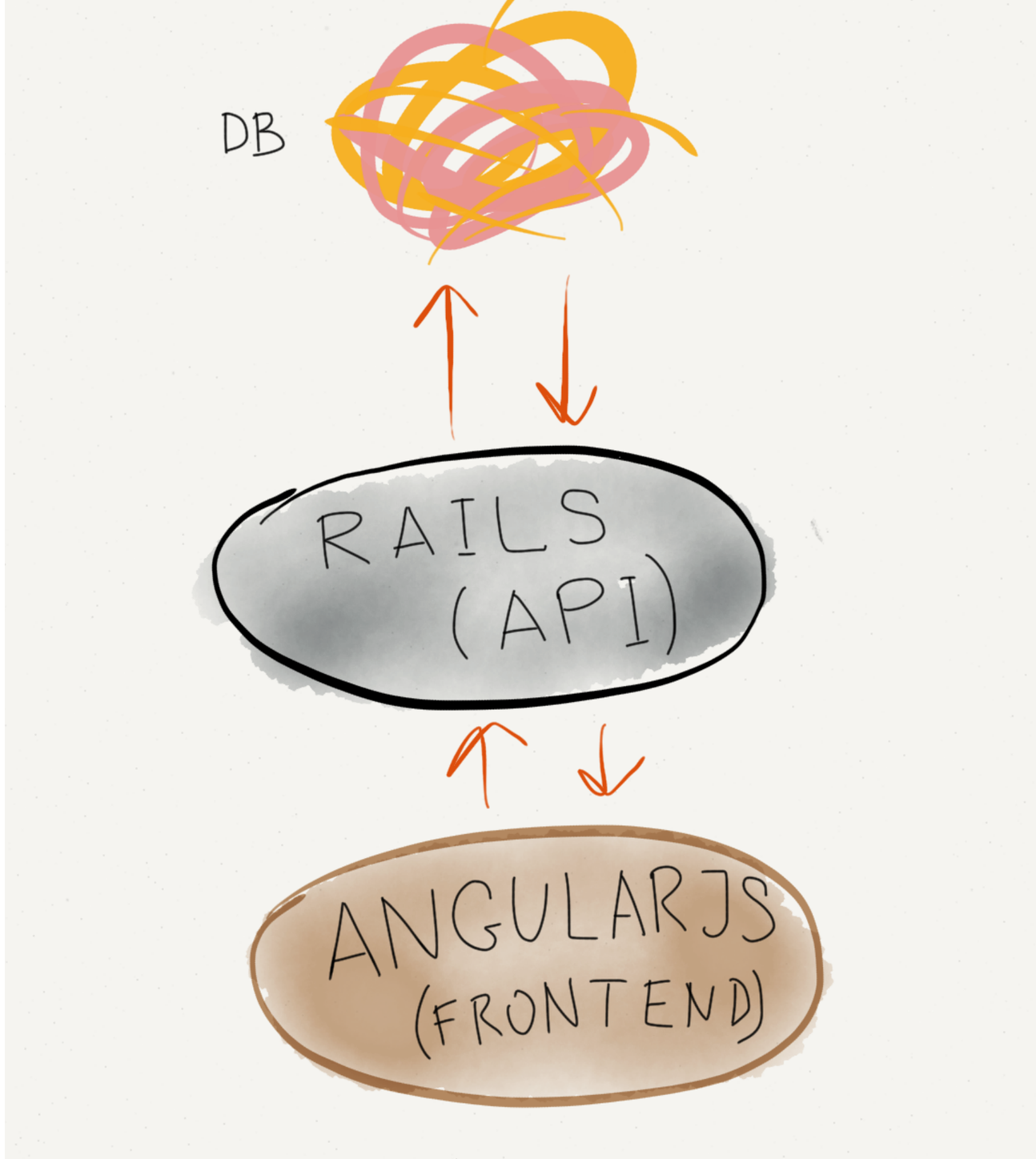


Solution

- Split frontend and backend
- AngularJS for frontend
- Rails for API

Limitations

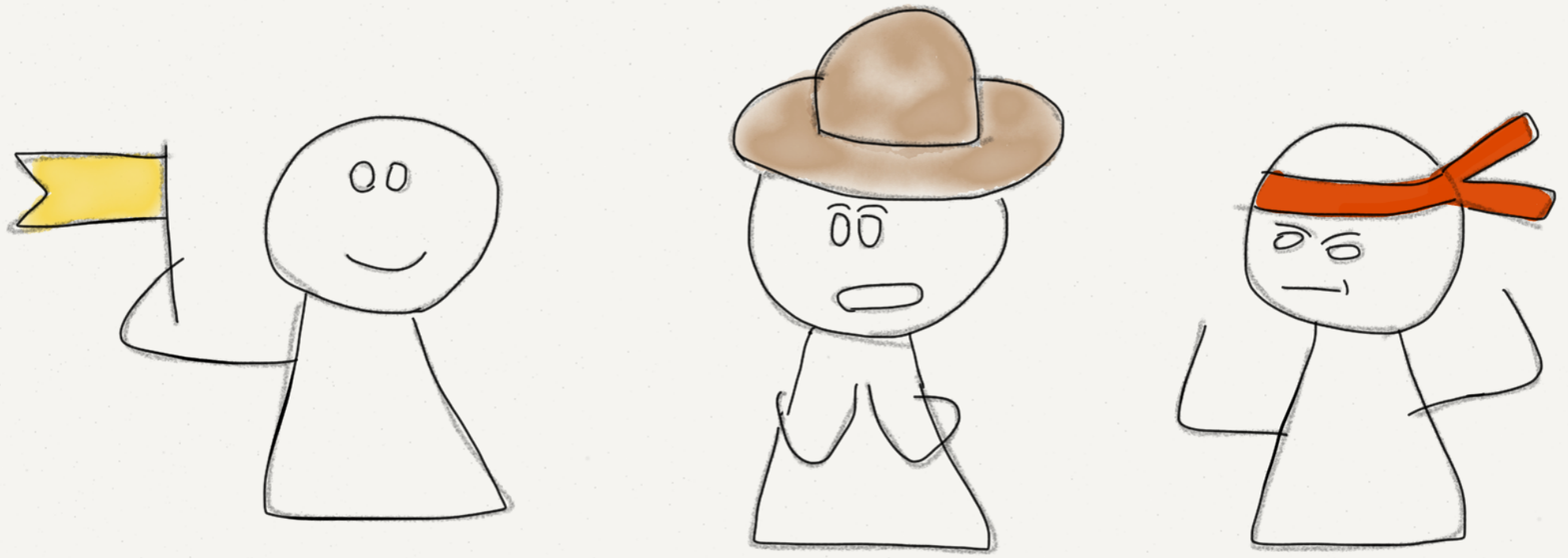
- Keep DB structure
- Features one by one



Ok, lets do it!

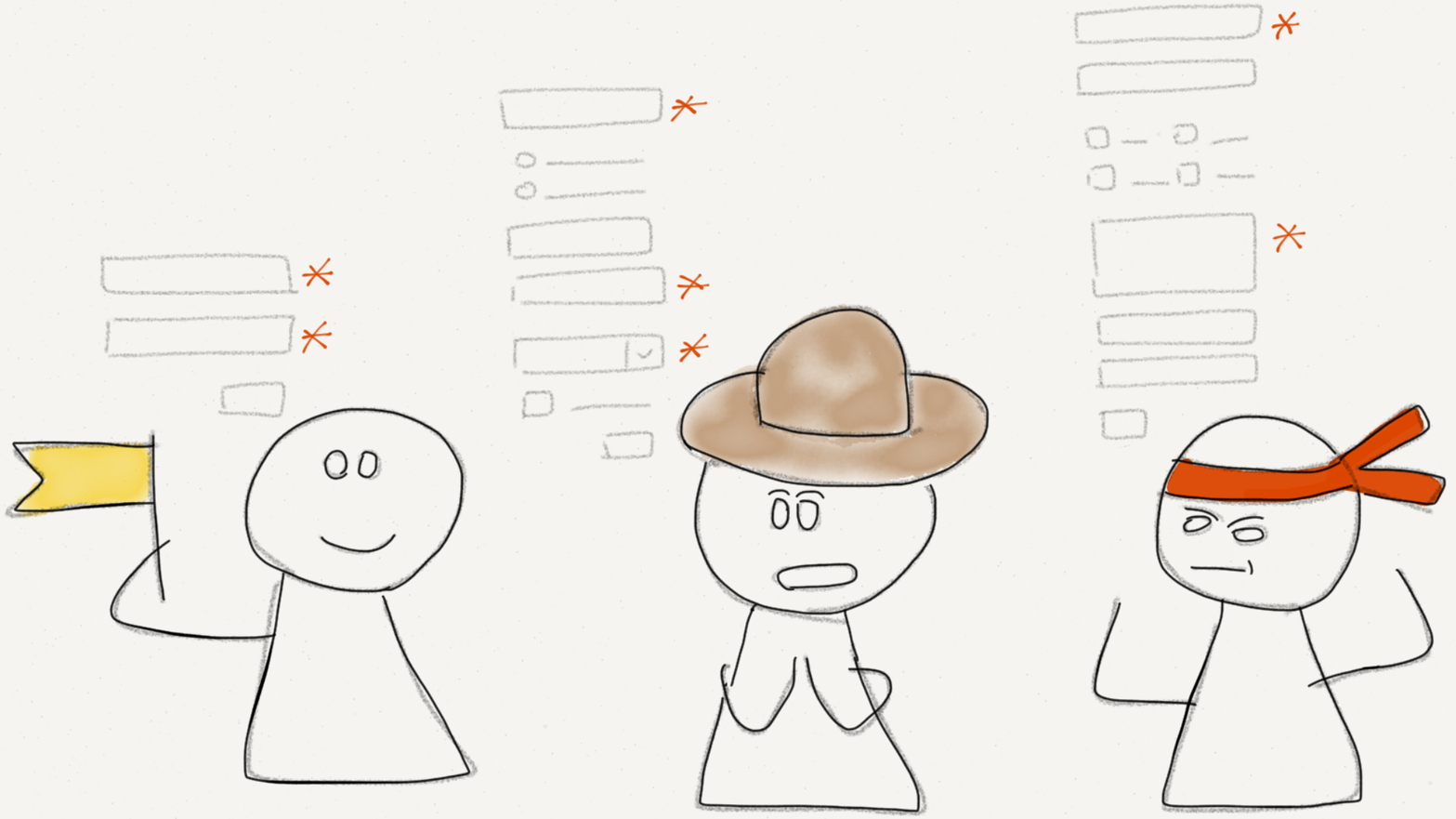
Let's implement some features!

Feature #1: Users Registration



Let's implement some features!

Feature #1: Users Registration



Single table inheritance !

Conditional validations !

Both STI and conditional validations are just workarounds

The problem is deeper!

Singe responsibility principle:

A class should have only one reason to change

Our model knows about..

- How `admin` **form** is validated
- How `org_user` **form** is validated
- How `guest_user` **form** is validated
- How `user` **data** is saved

Form object !

Cooking Form object (step 1)

```
class Person
  attr_accessor :first_name, :last_name
  def initialize(first_name, last_name)
    @first_name, @last_name = first_name, last_name
  end
end
```

Cooking Form object (step 2)

```
class Person
  attr_accessor :first_name, :last_name
  def initialize(first_name, last_name)
    @first_name, @last_name = first_name, last_name
  end

  include ActiveModel::Validations
  validates_presence_of :first_name, :last_name
end
```

Cooking Form object (step 3)

```
class Person
  include Virtus.model
  attribute :first_name, String
  attribute :last_name, String

  include ActiveModel::Validations
  validates_presence_of :first_name, :last_name
end
```

Form object

```
class OrgUserInput
  include Virtus.model
  include ActiveModel::Validations

  attribute :login, String
  attribute :password, String
  attribute :password_confirmation, String
  attribute :organization_id, Integer

  validates_presence_of :login, :password, :password_confirmation
  validates_numericality_of :organization_id
end
```

Using Form object

```
def create
  input = OrgUserInput.new(params)
  if input.valid?
    @user = User.create(input.to_hash)
  else
    #...
  end
end
```

Form objects

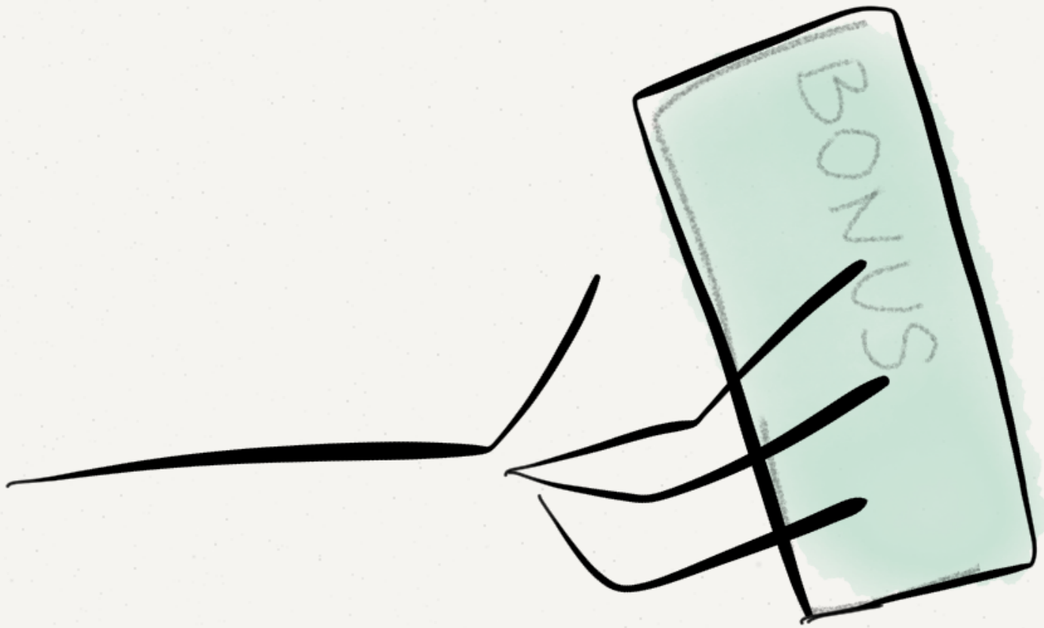
What we've got?

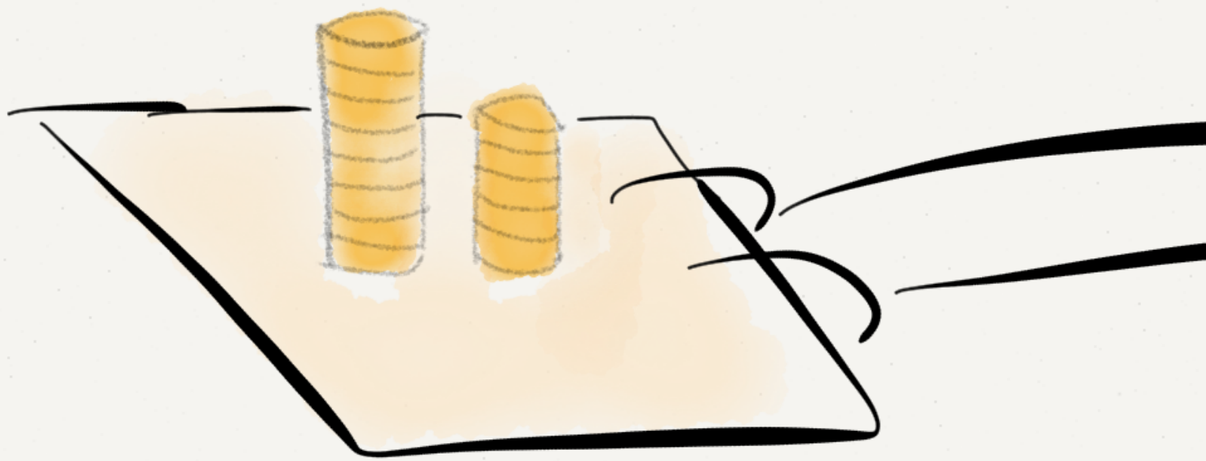
4 simple objects instead of one complex

:)

Let's implement more features!

Feature #2: Bonuscode redeem





```ruby
def redeem
  unless bonuscode = Bonuscode.find_by_hash(params[:code])
    render json: {error: 'Bonuscode not found'}, status: 404 and return
  end
  if bonuscode.used?
    render json: {error: 'Bonuscode is already used'}, status: 404 and return
  end
  unless recipient = User.find_by_id(params[:receptor_id])
    render json: {error: 'Recipient not found'}, status: 404 and return
  end

  ActiveRecord::Base.transaction do
    amount = bonuscode.mark_as_used!(params[:receptor_id])
    recipient.increase_balance!(amount)

    if recipient.save && bonuscode.save
      render json: {balance: recipient.balance}, status: 200
    else
      render json: {error: 'Error during transaction'}, status: 500
    end
  end
end
```

```ruby
def redeem
  unless bonuscode = Bonuscode.find_by_hash(params[:code])
    render json: {error: 'Bonuscode not found'}, status: 404 and return
  end
  if bonuscode.used?
    render json: {error: 'Bonuscode is already used'}, status: 404 and return
  end
  unless recipient = User.find_by_id(params[:receptor_id])
    render json: {error: 'Recipient not found'}, status: 404 and return
  end
```

```ruby
end
```

```ruby
def redeem
  [redacted]

  ActiveRecord::Base.transaction do
    amount = bonuscode.mark_as_used!(params[:receptor_id])
    recipient.increase_balance!(amount)

    if recipient.save && bonuscode.save
      render json: {balance: recipient.balance}, status: 200
    else
      render json: {error: 'Error during transaction'}, status: 500
    end
  end
end
```

```ruby
def redeem
  unless bonuscode = Bonuscode.find_by_hash(params[:code])
    render json: {error: 'Bonuscode not found'}, status: 404 and return
  end
  if bonuscode.used?
    render json: {error: 'Bonuscode is already used'}, status: 404 and return
  end
  unless recipient = User.find_by_id(params[:receptor_id])
    render json: {error: 'Recipient not found'}, status: 404 and return
  end

  ActiveRecord::Base.transaction do
    amount = bonuscode.mark_as_used!(params[:receptor_id])
    recipient.increase_balance!(amount)

    if recipient.save && bonuscode.save
      render json: {balance: recipient.balance}, status: 200
    else
      render json: {error: 'Error during transaction'}, status: 500
    end
  end
end
```

Single responsibility principle

```ruby
def redeem
  unless bonuscode = Bonuscode.find_by_hash(params[:code])
    render json: {error: 'Bonuscode not found'}, status: 404 and return
  end
  if bonuscode.used?
    render json: {error: 'Bonuscode is already used'}, status: 404 and return
  end
  unless recipient = User.find_by_id(params[:receptor_id])
    render json: {error: 'Recipient not found'}, status: 404 and return
  end

  ActiveRecord::Base.transaction do
    amount = bonuscode.mark_as_used!(params[:receptor_id])
    recipient.increase_balance!(amount)

    if recipient.save && bonuscode.save
      render json: {balance: recipient.balance}, status: 200
    else
      render json: {error: 'Error during transaction'}, status: 500
    end
  end
end
```

```
bonuscode.redeem_by(user)
```

or

```
user.redeem_bonus(code)
```

?

Service Object!

(Use case, interactor)

Cooking Service object (step 1)

```ruby
class RedeemBonuscode
  def redeem
    unless bonuscode = Bonuscode.find_by_hash(params[:code])
      render json: {error: 'Bonuscode not found'}, status: 404 and return
    end
    if bonuscode.used?
      render json: {error: 'Bonuscode is already used'}, status: 404 and return
    end
    unless recipient = User.find_by_id(params[:receptor_id])
      render json: {error: 'Recipient not found'}, status: 404 and return
    end

    ActiveRecord::Base.transaction do
      amount = bonuscode.mark_as_used!(params[:receptor_id])
      recipient.increase_balance!(amount)

      if recipient.save && bonuscode.save
        render json: {balance: recipient.balance}, status: 200
      else
        render json: {error: 'Error during transaction'}, status: 500
      end
    end
  end
end
```

Cooking Service object (step 2)

```
class RedeemBonuscode
  def run!(params)
    unless bonuscode = Bonuscode.find_by_hash(params[:code])
      raise BonuscodeNotFound.new
    end
    if bonuscode.used?
      raise BonuscodeIsAlreadyUsed.new
    end
    unless recipient = User.find_by_id(params[:receptor_id])
      raise RecipientNotFound.new
    end

    ActiveRecord::Base.transaction do
      amount = bonuscode.mark_as_used!(params[:receptor_id])
      recipient.increase_balance!(amount)
      recipient.save! && bonuscode.save!
    end
    recipient.balance
  end
end
```

Cooking Service object (step 3)

```
def redeem
  use_case = RedeemBonuscode.new

  begin
    recipient_balance = use_case.run!(params)
  rescue BonuscodeNotFound, BonuscodeIsAlreadyUsed, RecipientNotFound => ex
    render json: {error: ex.message}, status: 404 and return
  rescue TransactionError => ex
    render json: {error: ex.message}, status: 500 and return
  end

  render json: {balance: recipient_balance}
end
```

Service objects
What we've got?

Business logic/controller separation

:)

Are we happy now?

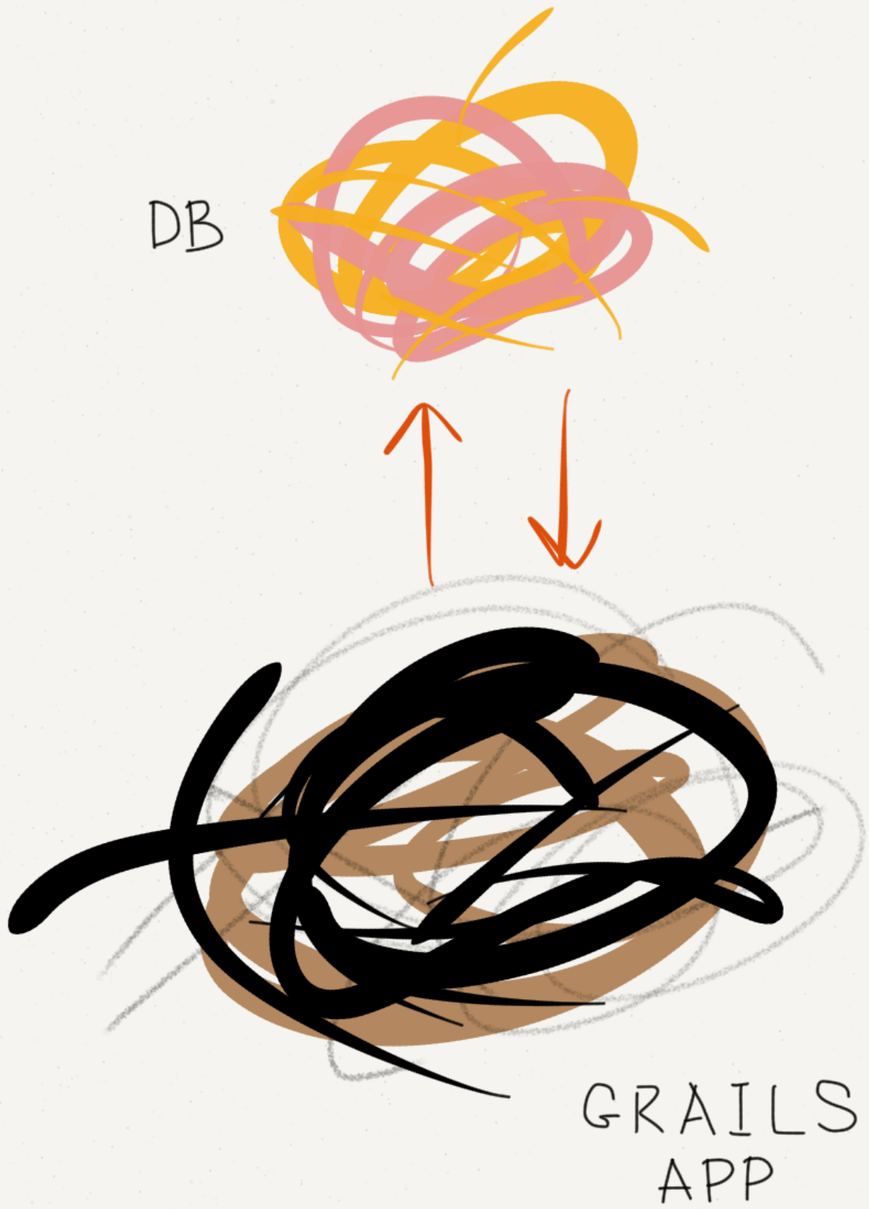
Not really happy

```
user.gender_id
user.status_id

if image = user.profile.image
  images = [image] + image.image_variants

  original = images.select do |img|
    img.settings.label == 'Profile'
  end.first

  resized = images.select do |img|
    img.settings.label == 'thumbnail'
  end.first
end
```



Not really happy

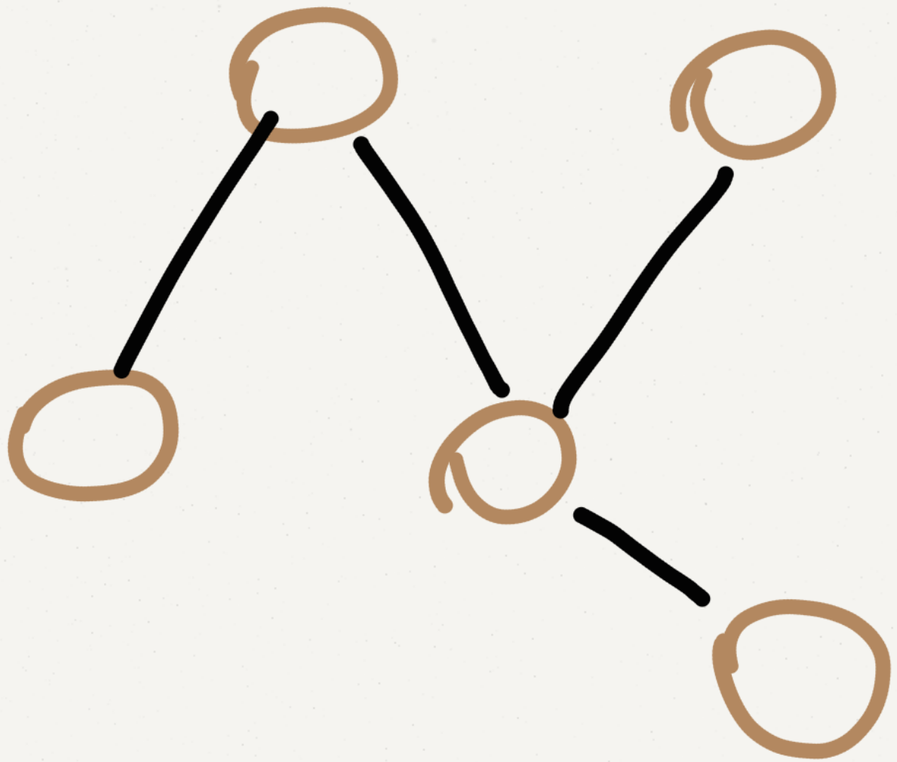
```
class Image < ActiveRecord::Base
  self.table_name = 'image'
  has_and_belongs_to_many :image_variants,
      join_table: "image_image", class_name: "Image",
      association_foreign_key: :image_images_id
  belongs_to :settings,
      foreign_key: :settings_id, class_name: 'ImageSettings'
  belongs_to :asset
end
```

Hey kid, want some DDD?

Domain Driven Design

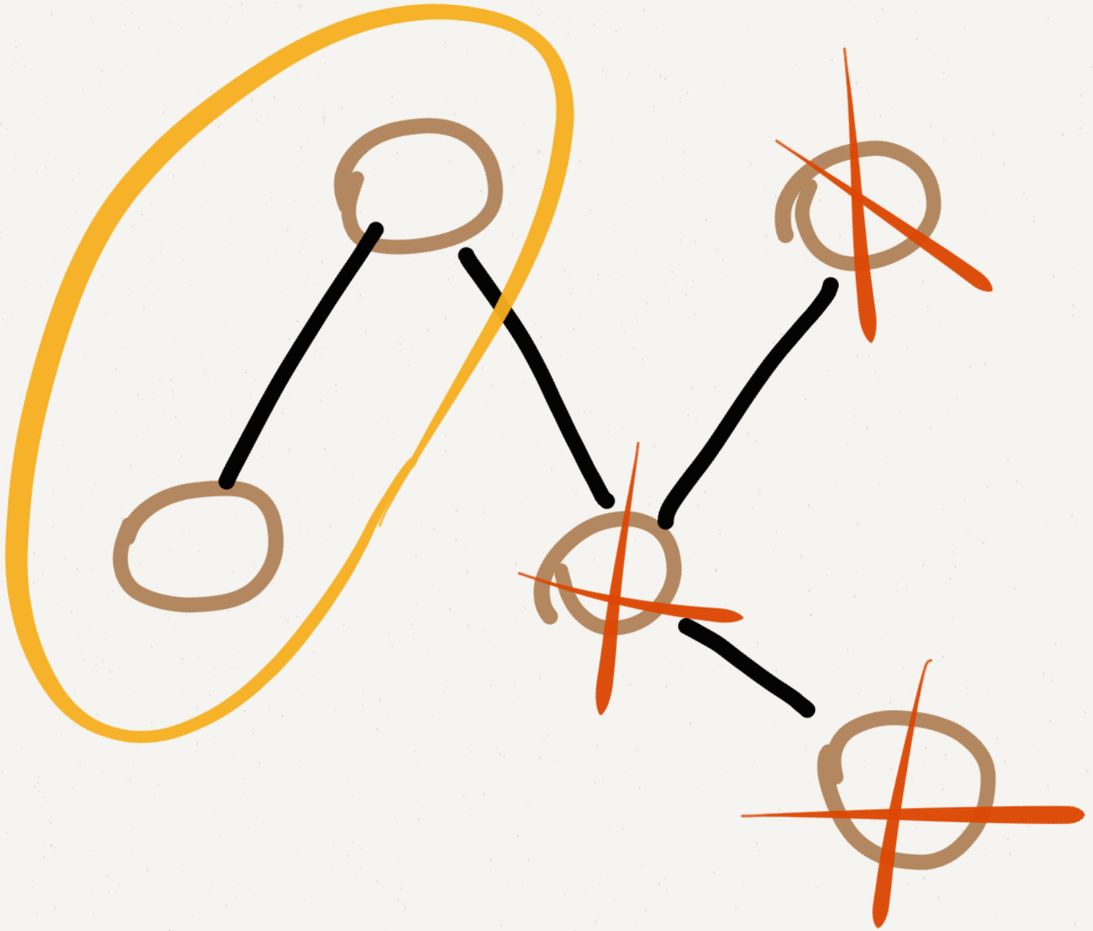
Domain-specific objects

- User
- Profile
- Image
- ImageVariant
- ImageSettings



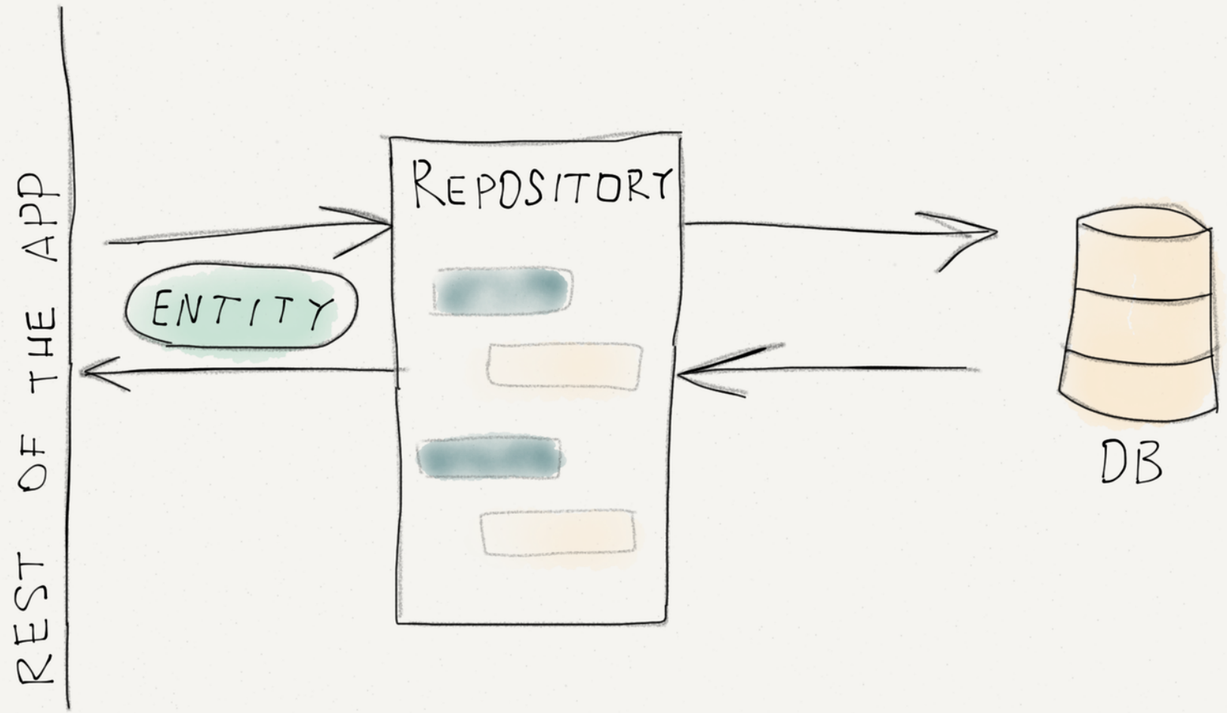
Domain-specific objects

- **User**
- **Profile**
- ~~Image~~
- ~~ImageVariant~~
- ~~ImageSettings~~



Repository !

Repositories + Entities



Entity Example

```
class User
  include Virtus.model

  attribute :id, Integer
  attribute :username, String
  attribute :enabled, Boolean
  attribute :date_created, DateTime
  attribute :last_updated, DateTime
  attribute :roles, Array

  def new_record?
    !id
  end
end
```

Repository

```
class UserRepository
  def find(id)
    columns = :id, :username, :enabled, :date_created, :last_updated
    if dataset = table.select(columns).where(id: id)

      user = User.new(dataset.first)
      user.roles = get_roles(id)
      user
    end
  end
end
```

Repository

```
class ProfileRepository
  def find(ids)
    dataset = table.join(:profile_status, id: :profile_status_id)
                   .join(:gender, id: :profile__gender_id)
                   .select_all(:profile).select_append(*extra_fields)
                   .where(profile__id: ids)

    dataset.all.map do |record|
      Profile.new(record)
    end
  end
end
```

```ruby
def persist(profile)
  status_id = get_status_id(profile.status)
  gender_id = get_gender_id(profile.gender)

  [redacted]

end

def get_status_id(status)
  DB[:online_profile_status].select(:id).where(name: status).get(:id)
end

def get_gender_id(gender)
  DB[:gender].select(:id).where(name: gender).get(:id)
end
```

```
def persist(profile)
  [redacted]

  hash = { username: profile.username, picture_id: profile.image_id,
           profile_status_id: status_id, gender_id: gender_id }

  if profile.new_record?
    profile.id = table.insert(hash.merge!(date_created: Time.now.utc))
  else
    table.where(id: profile.id).update(hash.merge!(last_updated: Time.now.utc))
  end
end
```

Repositories

What we've got?

DB structure doesn't affect app :)

Repositories

What we've got?

No ActiveRecord magic anymore!

:/

Repositories

What we've got?

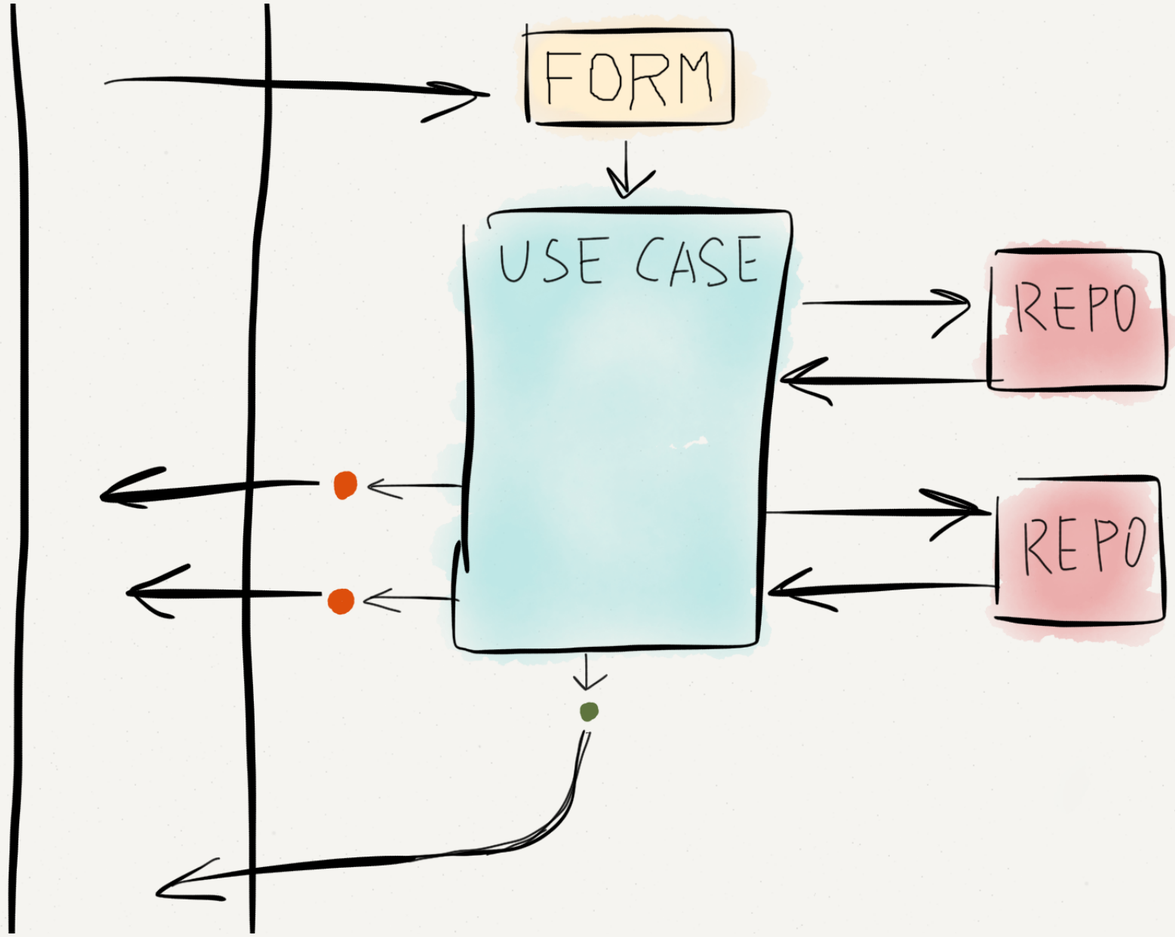
Had to write a lot of low-level code :(

Repositories

What we've got?

You have control on what's happening :)

Whole picture



Not depending on Rails anymore!

Rails is just a tool, not a core

Handles incoming requests
Sends responses

Form Object

1 model ⇎ 1 form

Service Object

business logic][requests/responses

Repository

1 table in DB ⇔ 1 model

How to stop being Rails-developer

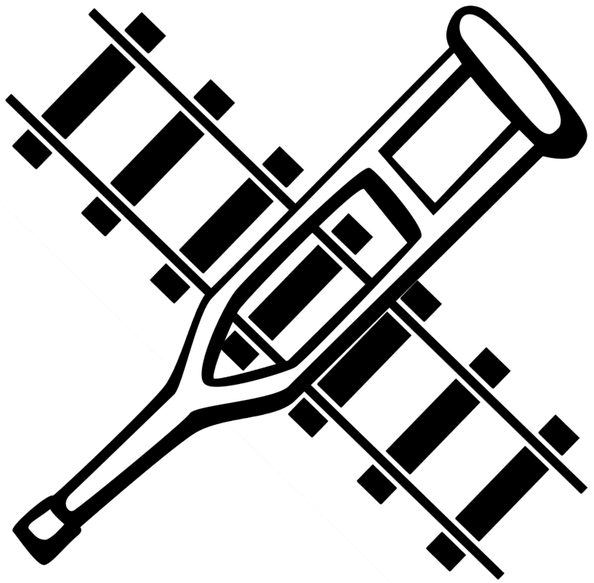
How to stop being Rails-developer

- Don't miss the point where Rails is not enough.
- Learn & explore complexity management methods
- Proactively look for an opportunity to aaply them
- Use somebody's solution if you're sure you understand the concept.

Existing tools

- **Interactor** - github.com/collectiveidea/interactor
- **Ruby object mapper** - rom-rb.org
- **Lotus** - lotusrb.org
- **reform** - github.com/apotonick/reform
- **trailblazer** - github.com/apotonick/reform

hawkins.io
blog.arkency.com



RailsHurts.com

inem.at/railsisrael

THANKS