

Functional Object-Oriented Design in Ruby

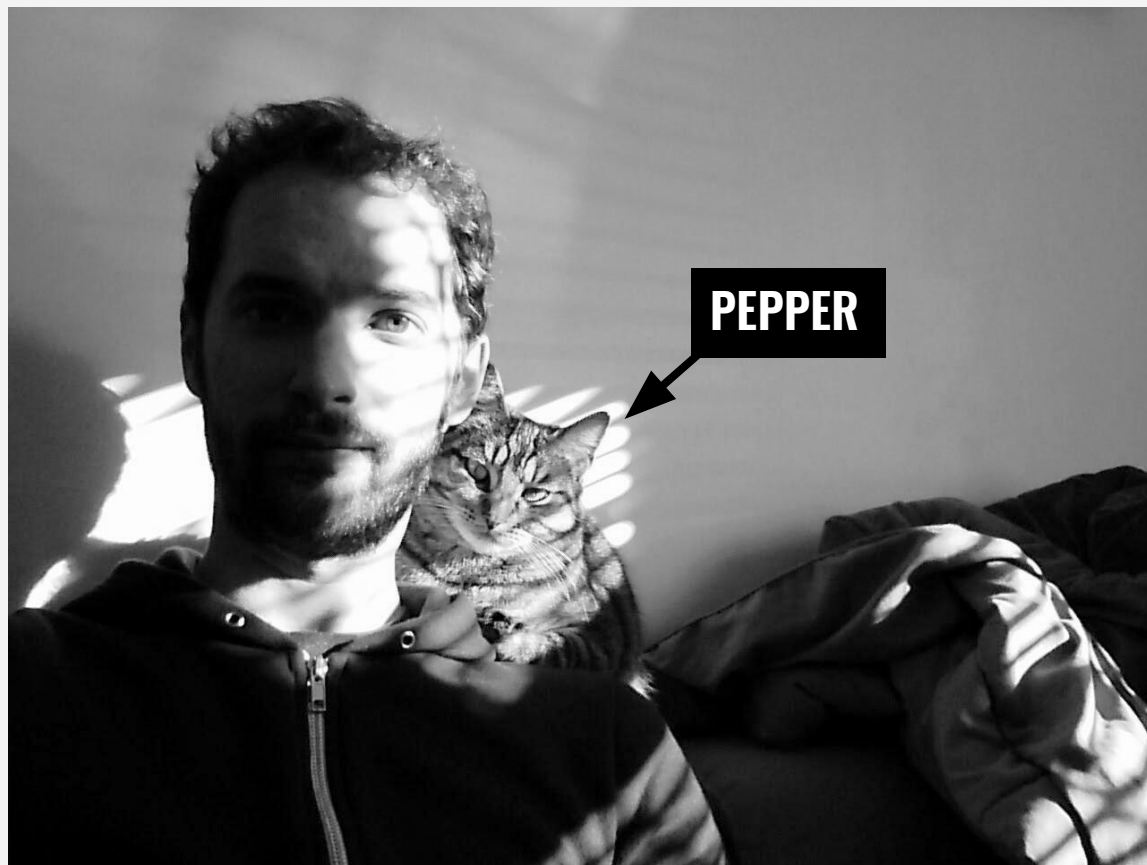
Conf & Coffee

Conf & Tea ☕

I'm Alex


I ♥ Ruby since 2015

Software Developer @ Unbounce






built on rails.com

Built On  **RAILS**


Ruby on Rails powers hundreds of thousands of web applications. It's powerful, scalable and easy to learn. The companies below rely on Rails every day.

[Submit a site](#)

 **Netflix** uses Rails


Netflix is a streaming service that allows our customers to watch a wide variety of award-winning TV shows, movies, documentaries and more on thousands of Internet-connected devices.

Netflix's Studio Acquisition Engineering team builds applications used by content buyers to discover, evaluate and procure Netflix Original content.

 **Kickstarter** uses Rails


Kickstarter helps artists, musicians, filmmakers, designers, and other creators find the resources and support they need to make their ideas a reality. To date, tens of thousands of creative projects — big and small — have come to life with the support of the Kickstarter community.

Kickstarter's core application and payment service are built with Rails.

 **Airbnb** uses Rails


Book unique homes and experiences all over the world.

Airbnb's payment stack is written in Rails.

 **Basecamp** uses Rails 5.2


Basecamp organizes your communication, projects, and client work together so you have a central source of truth.

All of Basecamp runs on a single Rails monolith.

 **Hulu** uses Rails


Watch premium original series, full seasons of hit shows, current episodes, movies and more.

Hulu's core app is built on Rails.

 **Shopify** uses Rails 5.2


The ecommerce platform made for you. Whether you sell online, on social media, in store, or out of the trunk of your car, Shopify has you covered.

Shopify's core application is built on Rails.

 **500px** uses Rails

A community for passionate photographers everywhere to connect, get inspired and grow their skills.

500px's core application is built on Rails.

 **Bonatch** uses Rails

Functional Object-Oriented Design

ABSTRACTIONS

Nouns & Verbs

- Two important abstractions in human language
- *Separate concepts*

Noun



[photo](#)

Verb



[photo](#)

Nouns

Verbs

Nouns & Verbs

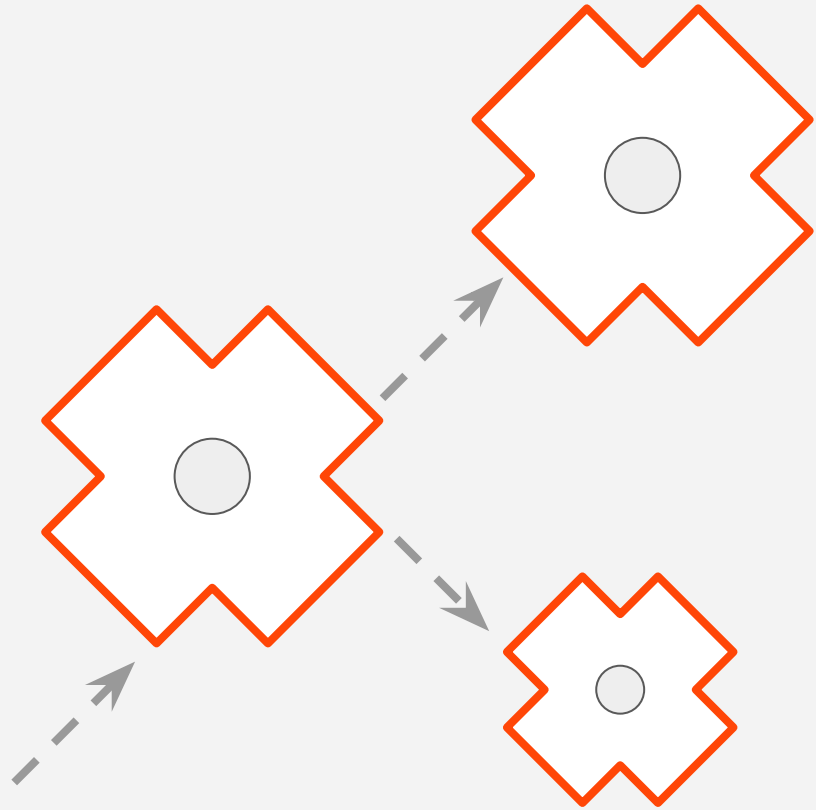
- Separate concepts in speech
- Good to keep them separate in our applications

FUNCTIONS & OBJECTS

OBJECTS offer *encapsulation*

Object-Oriented Programming

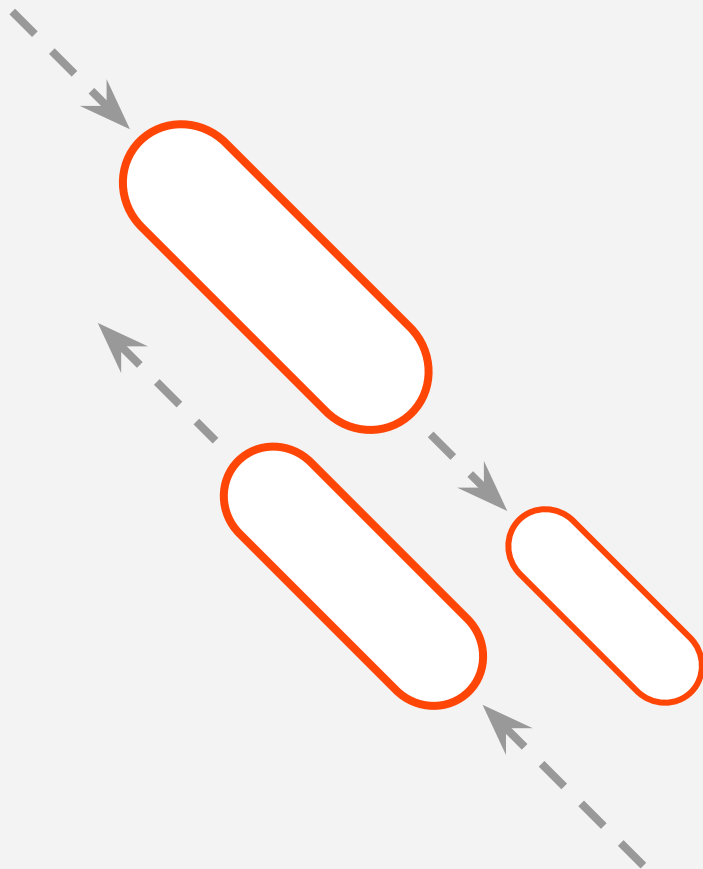
- Objects contain data *and* behaviour
- Objects pass messages between each other to get work done



FUNCTIONS offer *purity*

Functional Programming

- Functions operate on immutable data
- Functions are composed together to get work done



Functional Object-Oriented Design

FOOD



photo

What does good look like?

Object-Oriented Programming

- Small classes, single responsibility
- Prefer composition over inheritance
- Inject dependencies

What does good look like?

Object-Oriented Programming

- Small classes, single responsibility
- Prefer composition over inheritance
- Inject dependencies

Functional Programming

- Small functions, single responsibility
- Compose functions to build complex behaviour
- Only depend on a function's arguments

What's different?

CO-LOCATION

STATE

1. Co-location of data and behaviour

To separate or not?

Co-located

- Behaviour lives close to the data that it works with
- Changing behaviour happens in one place
- Learning how to work with the data is intuitive
- Behaviour often coupled to the data
- Easy to end up with too many responsibilities

To separate or not?

Co-located

- Behaviour lives close to the data that it works with
- Changing behaviour happens in one place
- Learning how to work with the data is intuitive
- Behaviour often coupled to the data
- Easy to end up with too many responsibilities

Separated

- Behaviour is often reusable
- Data is lightweight, can be passed around
- Makes data transformations explicit
- Changing behaviour might need to happen in many places

Data

Behaviour

Nouns

Verbs

~~Nouns~~ **Data**

Verbs

~~Nouns~~ **Data**

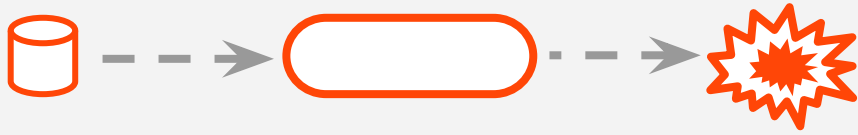
~~Verbs~~ **Behaviour**

VERBS are *side effects*

Changing the world

- Be *explicit* about when you change the world
- Interactors, Service Objects, Command Objects, etc
- Single responsibility
- Single public interface (#call)

Data + Behaviour == Action

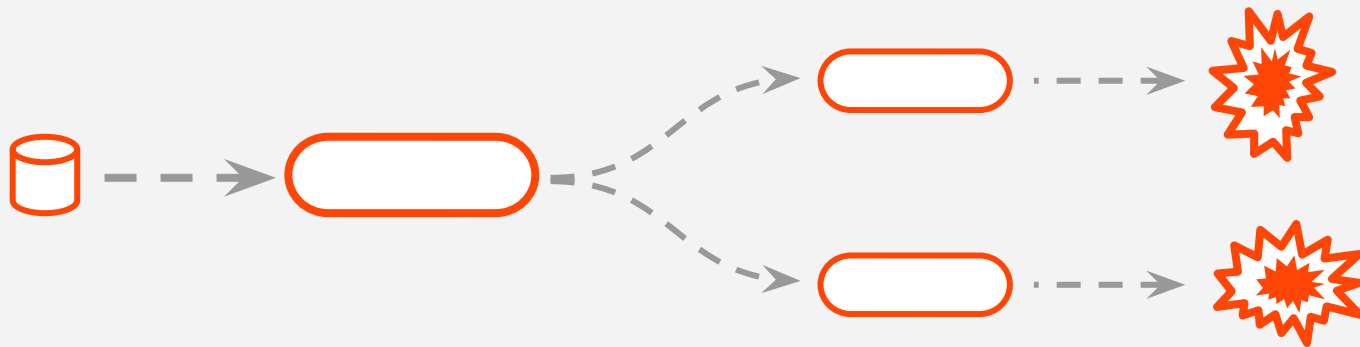


Data + Behaviour == Action

- Data and behaviour still coupled, but loosely
- Good naming can help enumerate all possible actions

```
services
├── subscription
│   ├── activate.rb
│   ├── cancel.rb
│   └── renew.rb
└── user
    ├── create.rb
    ├── delete.rb
    └── update.rb
```

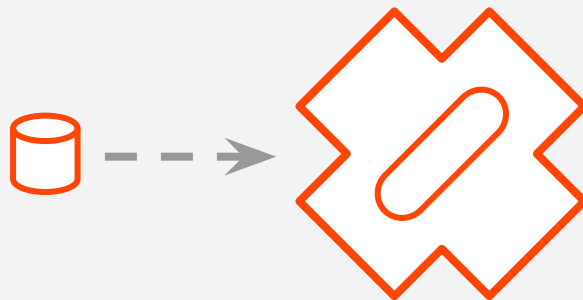
Action Objects are composable



Action Object == First-class Function

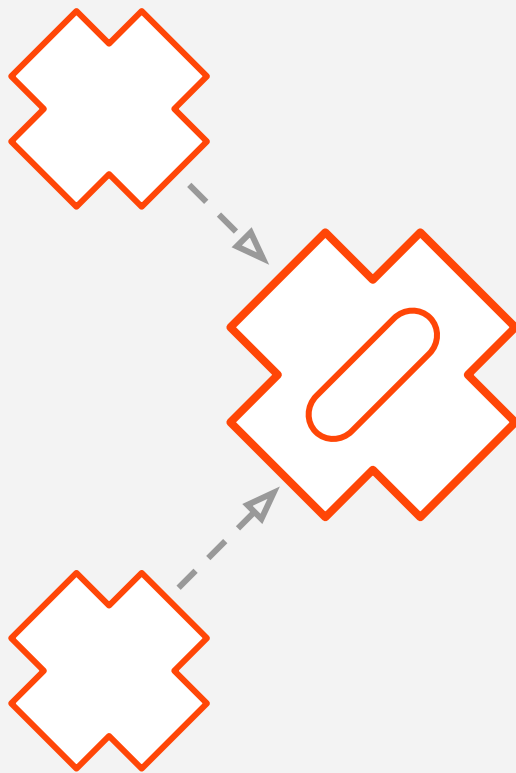
Function Objects

- Single responsibility
- Single public interface (#call)
- Private methods for code clarity / organization
- Has state, but *immutable*
- Dependencies are injected



Dependency Injection

- Makes an object's collaborators explicit
- Easy to test
- Can give you determinism



2. State

STATE IS BAD*

```
class Subscription

  def cancel
    @canceled = true
  end

  # ... 200 lines later...

  def amount
    @canceled ? 0 : amount
  end
end
```

```
# accounts/index.html.erb
```

```
...
```

```
<% account.balance += 150 %>
```

```
Your balance is <%= account.balance %>
```

```
...
```

State is bad... when it's *mutable*

- Hard to reason about
- Hard to reproduce bugs
- Hard to confidently change behaviour

State can be OK!

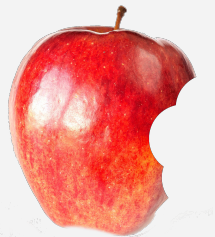
- Immutable state is good state
- State for presentation and consumption of data
- Safe to pass around

Mutable State

You have an apple

You eat the apple

You've changed the apple's state



Mutable State

You have an apple

You eat the apple

You've changed the apple's state

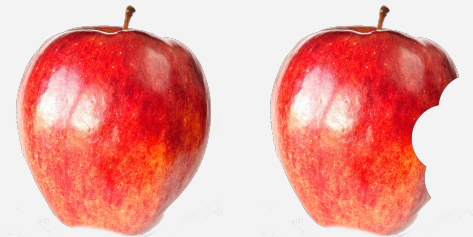


Immutable State

You have an apple

You eat the apple

Now you have two apples!

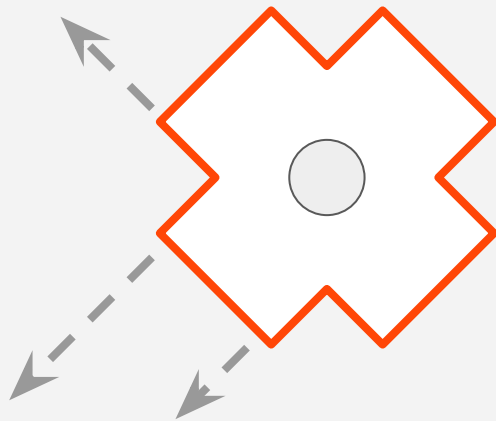


VERBS are *side effects*

NOUNS are *things*

Look, but don't touch

- Pure functions, immutable data
- Value Objects, Query Objects, etc
- Rich interfaces for interacting with your data



```
class Price

  attribute :price_in_cents, Integer

  def humanized_price
    “$#{price_in_cents / 100.0}”
  end

  # ...

end
```

```
class SubscriptionPresenter

  attribute :subscription, Subscription
  attribute :plan, Plan

  def plan_status
    "Your plan #{plan.name} will renew on #{subscription.renewal_date}"
  end

  # ...

end
```

Immutable state is good state

- Wrap immutable data in a nice interface
- No side effects - use with confidence!

Recap

- Encapsulate behaviour in function objects
- Extend behaviour with composition
- Model data with immutable state
- Inject dependencies

Functional Object-Oriented Design

SIMPLICITY

Ruby ♥ FP + OO

- Everything is an Object*
- Lambdas == Functions as data
- Blocks == anonymous functions
- Callable objects with #call

Community and Tooling

- **dry-rb**
 - **dry-view** for functional views
 - **dry-struct** for elegant function objects
- **Web frameworks**
 - Hanami
 - Roda



Mmm, FOOD...

Thanks!