Domain Driven Design Hexagonal Architecture Rails

Declan Whelan
@dwhelan

PRINT CHOMP

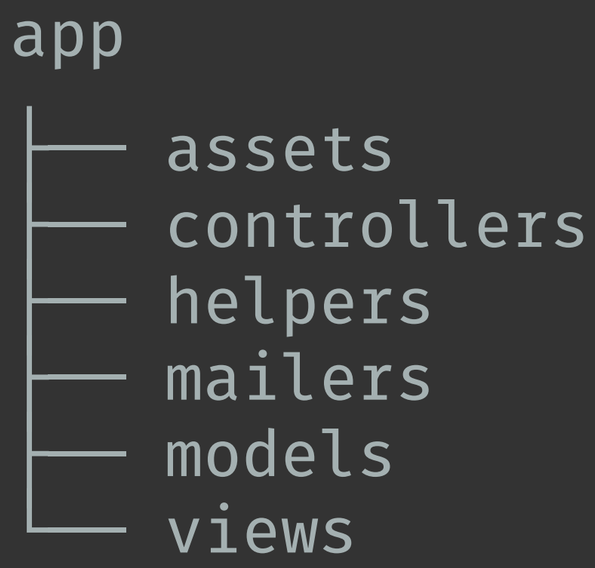
Eric Roberts
@eroberts

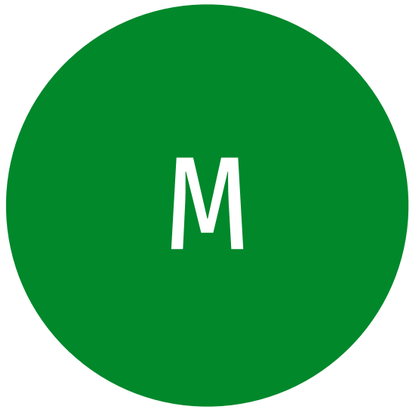
boltmade

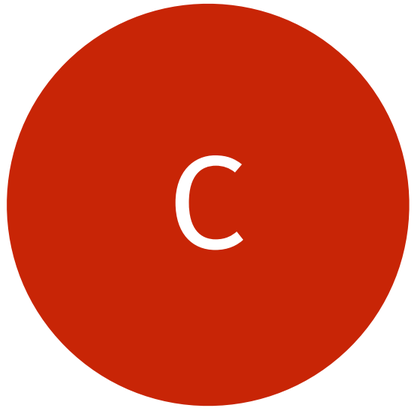
1. Embrace complexity

2. Know where you’re going

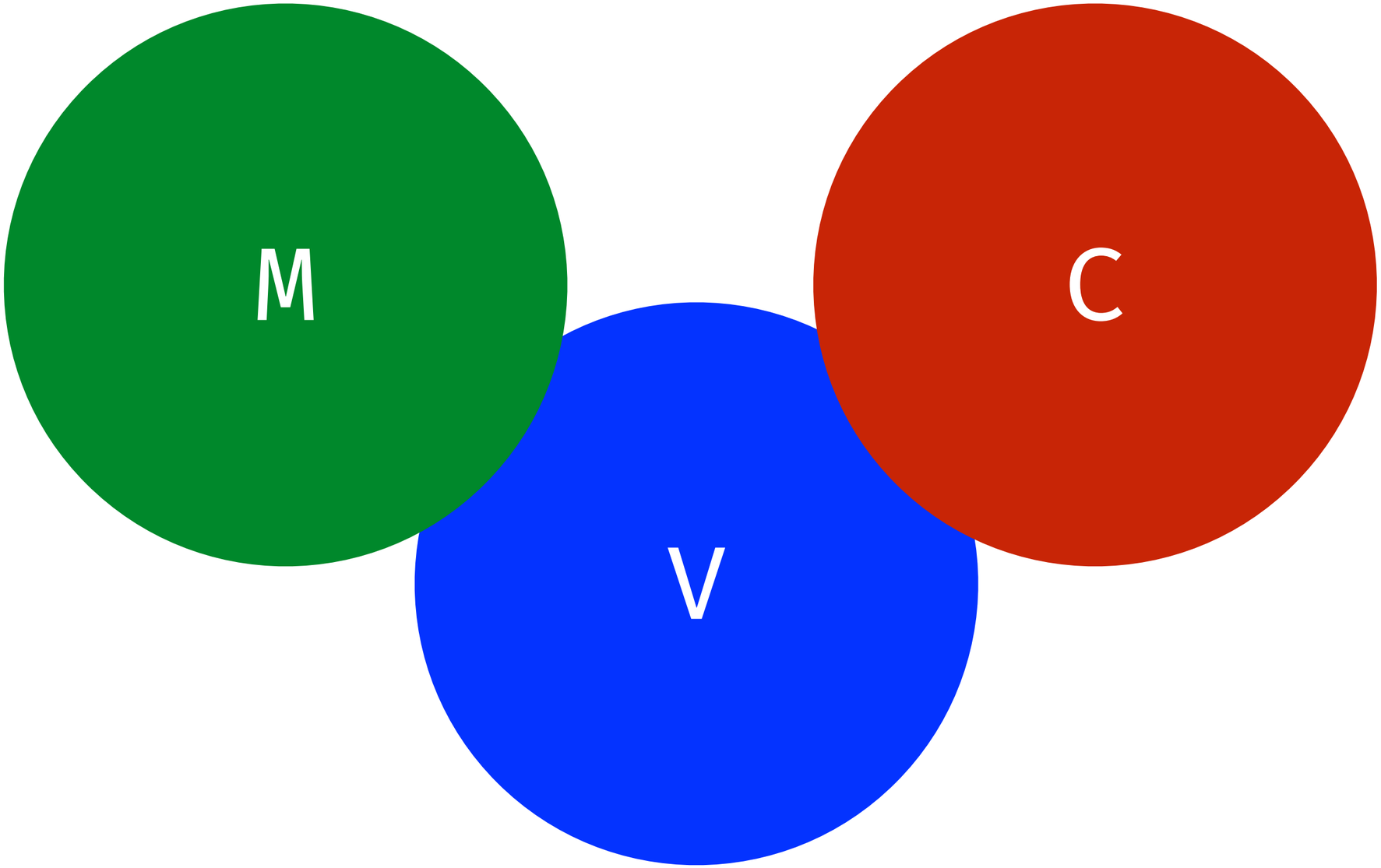
3. Be more than just a “Rails Developer”


app
├── assets
├── controllers
├── helpers
├── mailers
├── models
└── views

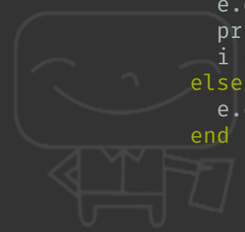

M


C

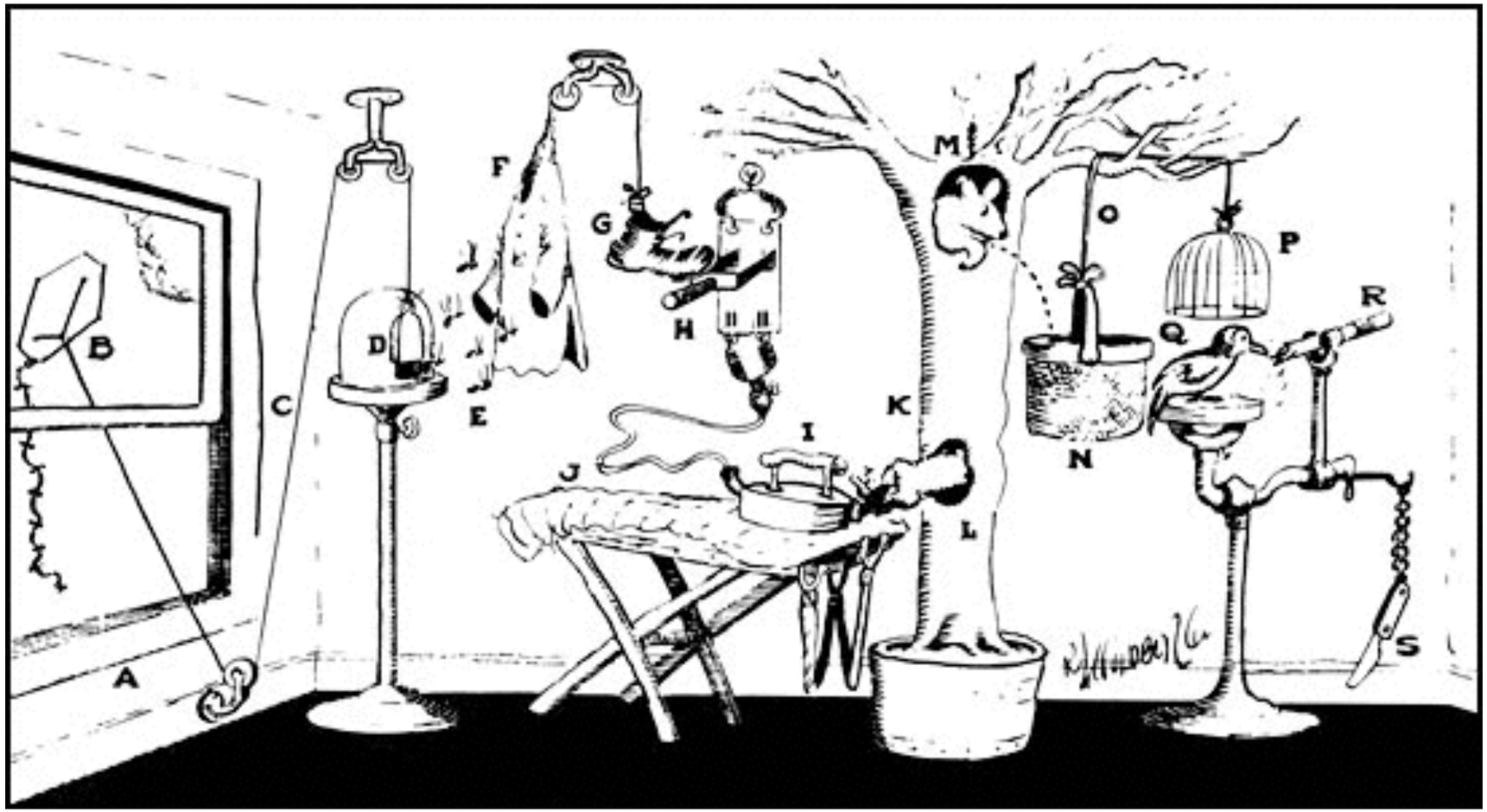

V


M
C
V

```ruby
def pricing_range_fold_and_save(new_range)
  pos = 0
  pricings = self.product_pricings
  other_range = ProductPricing.new
  if pricings.length == 0
    pricings.push(new_range)
  else
    pricings.each_with_index do |e, i|
      if i == 0 && new_range.start_date < e.start_date
        pricings.unshift(new_range)
        if new_range.end_date <= e.end_date
          e.start_date = new_range.end_date + 1.day
          break
        else
          while pricings[1].present? && new_range.end_date >= pricings[1].end_date
            pricings[1].destroy
            pricings.delete pricings[1]
          end
          if pricings[1].present?
            pricings[1].start_date = new_range.end_date + 1.day
          end
          break
        end
      elsif new_range.start_date <= e.end_date && new_range.start_date >= e.start_date

        if new_range.end_date < e.end_date
          other_range = e.dup
          other_range.start_date = new_range.end_date + 1.day
          if new_range.start_date == e.start_date
            e.destroy
            pricings.delete e
            i -= 1
          else
            e.end_date = new_range.start_date - 1.day
          end
          pricings.insert(i+1, new_range)
          pos = i+1
          pricings.insert(i+2, other_range)
          break
        else
          if new_range.start_date == e.start_date
            e.destroy
            pricings.delete e
            i -= 1
          else
            e.end_date = new_range.start_date - 1.day
          end
          pricings.insert(i+1, new_range)
          pos = i+1
          while pricings[i+2].present? && new_range.end_date >= pricings[i+2].end_date
            pricings[i+2].destroy
            pricings.delete pricings[i+2]
          end
          if pricings[i+2].present?
            pricings[i+2].start_date = new_range.end_date + 1.day
          end
          break
        end

      elsif i == pricings.size-1
        pricings[i].end_date = new_range.start_date-1.day
        pricings.push(new_range)
        break
      end
    end
  end

  pricings.each_with_index do |pricing, i|
    if i != pricings.size-1 && pricing.price ==pricings[i+1].price
      pricing.end_date = pricings[i+1].end_date
    end
    if i != 0 && pricing.end_date == pricings[i-1].end_date
      pricing.destroy
      pricings.delete pricing
    end
    if pricing.end_date < pricing.start_date
      pricing.destroy
      pricings.delete pricing
    end
  end

  pricings.each do |pricing|
    if pricing != pricings[pos]
      pricing.currency = pricings[pos].currency
      pricing.save
    end
  end
  pricings[pos].save
  return pricings

end
```



Pencil Sharpener



7 Patterns to Refactor Fat ActiveRecord Models

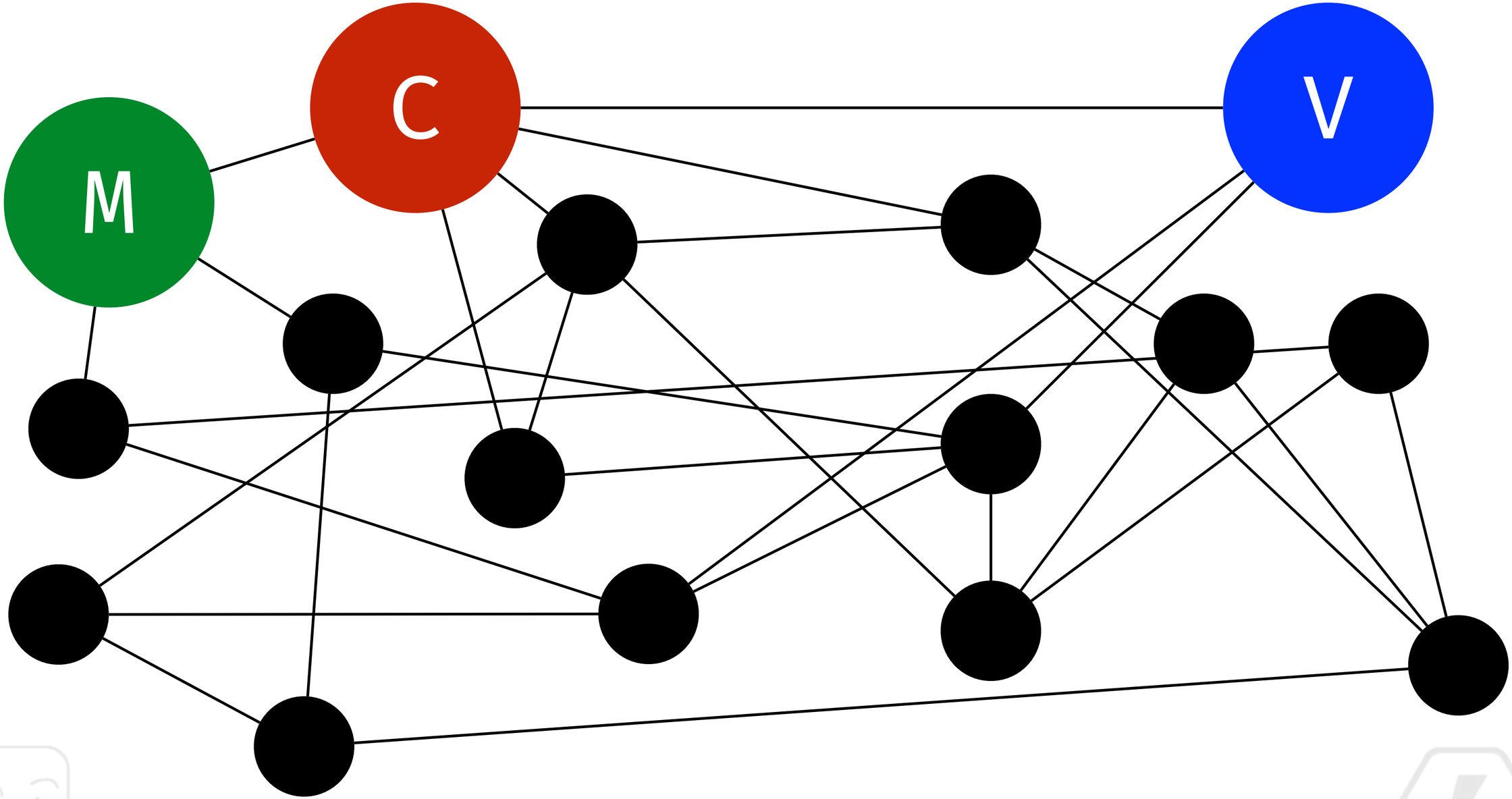
Posted by @brynary on Oct 17th, 2012

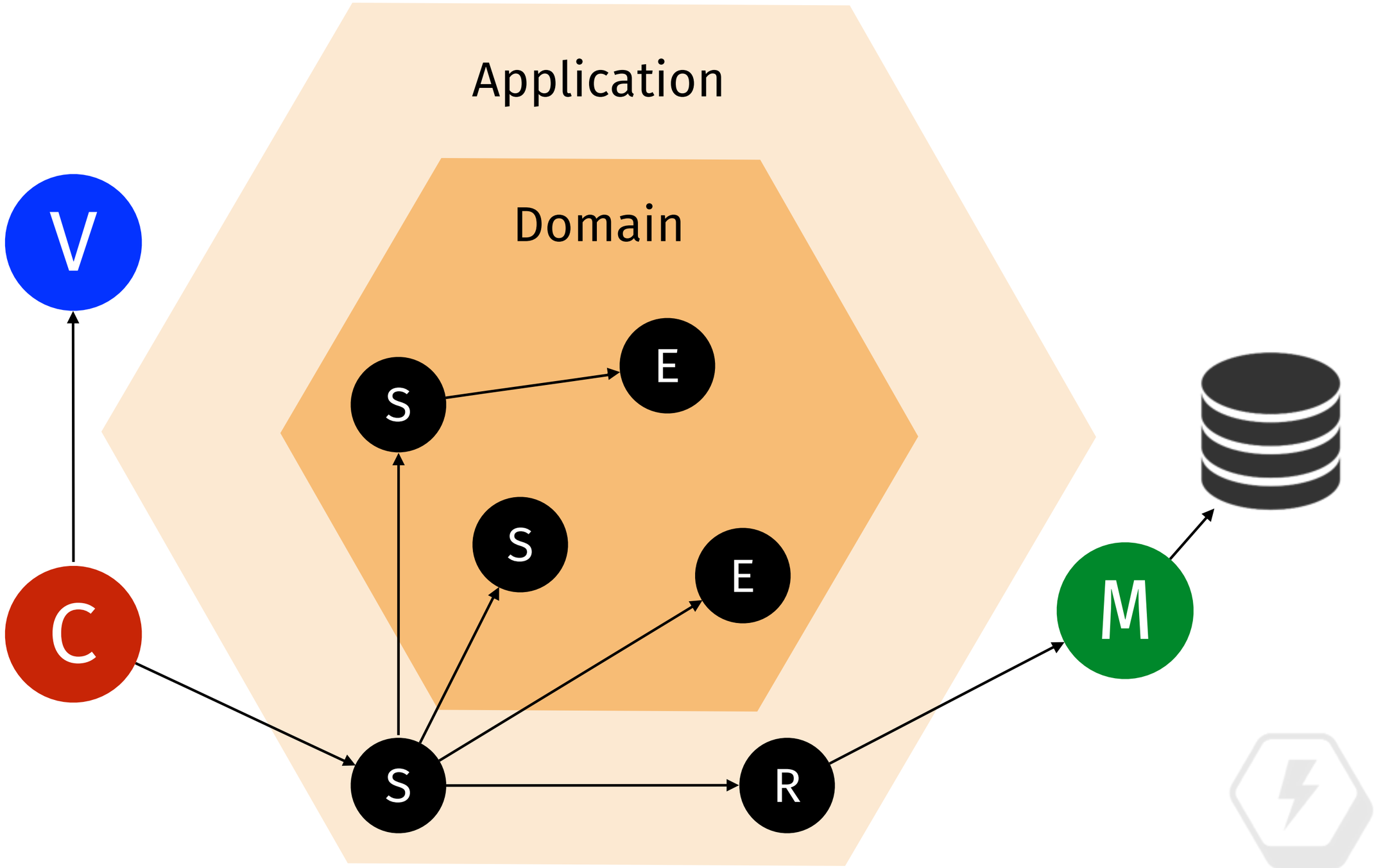
When teams use Code Climate to improve the quality of their Rails applications, they learn to break the habit of allowing models to get fat. *"Fat models"* cause maintenance issues in large apps. Only incrementally better than cluttering controllers with domain logic, they usually represent a failure to apply the Single Responsibility Principle (SRP). "Anything related to what a user does" is not a *single* responsibility.

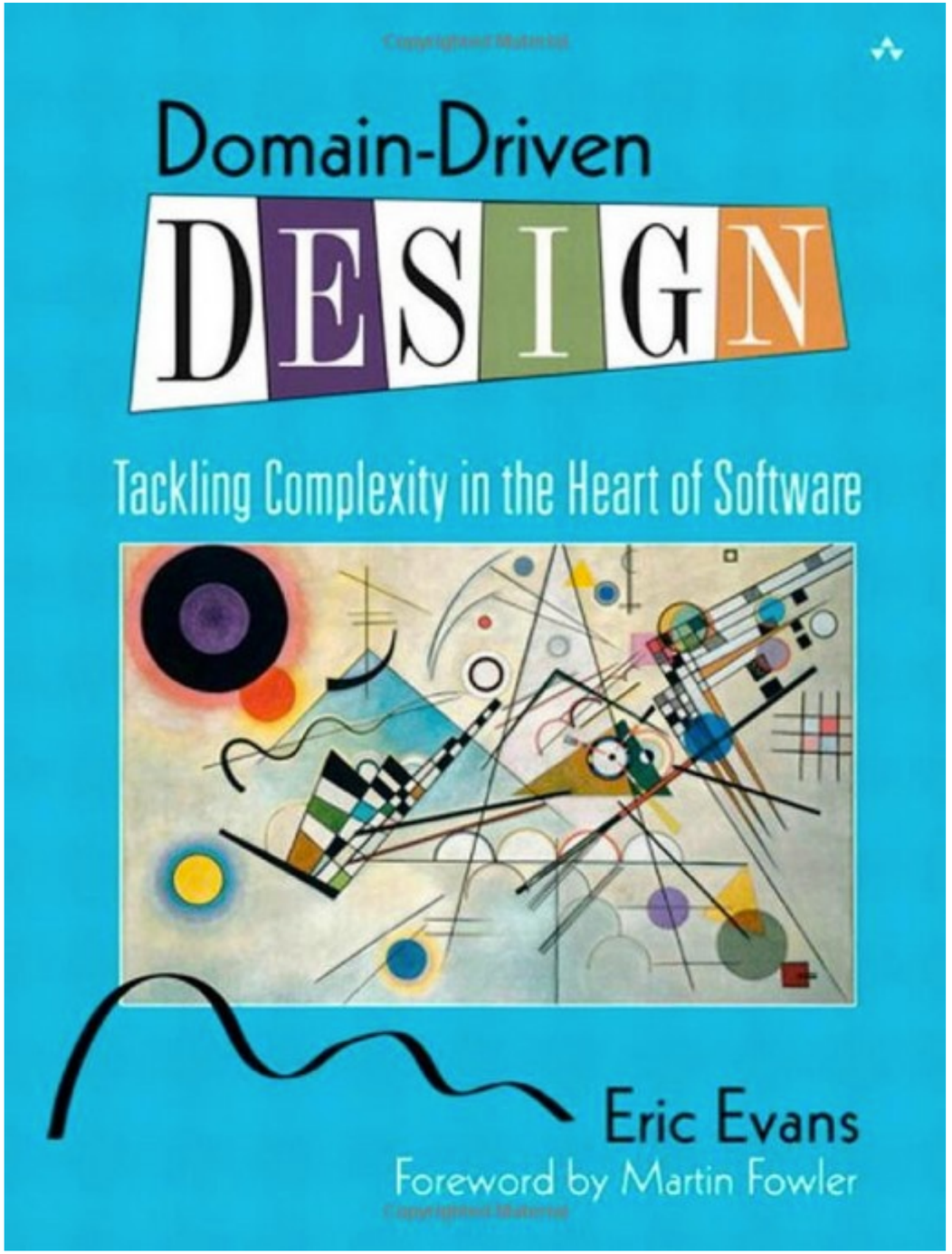
Early on, SRP is easier to apply. ActiveRecord classes handle persistence, associations and not much else. But bit-by-bit, they grow. Objects that are inherently responsible for persistence become the *de facto* owner of all business logic as well. And a year or two later you have a `User` class with over 500 lines of code, and hundreds of methods in it's *public* interface. Callback hell ensues.

As you add more intrinsic complexity (read: features!) to your application, the goal is to spread it across a coordinated set of small, encapsulated objects (and, at a higher level, modules) just as you might spread cake batter across the bottom of a pan. Fat models are like the big clumps you get when you first pour the batter in. Refactor to break them down and spread out the logic evenly. Repeat this

Popular Posts


M
C
V


Application
Domain
V
C
S
E
S
E
S
R
M


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

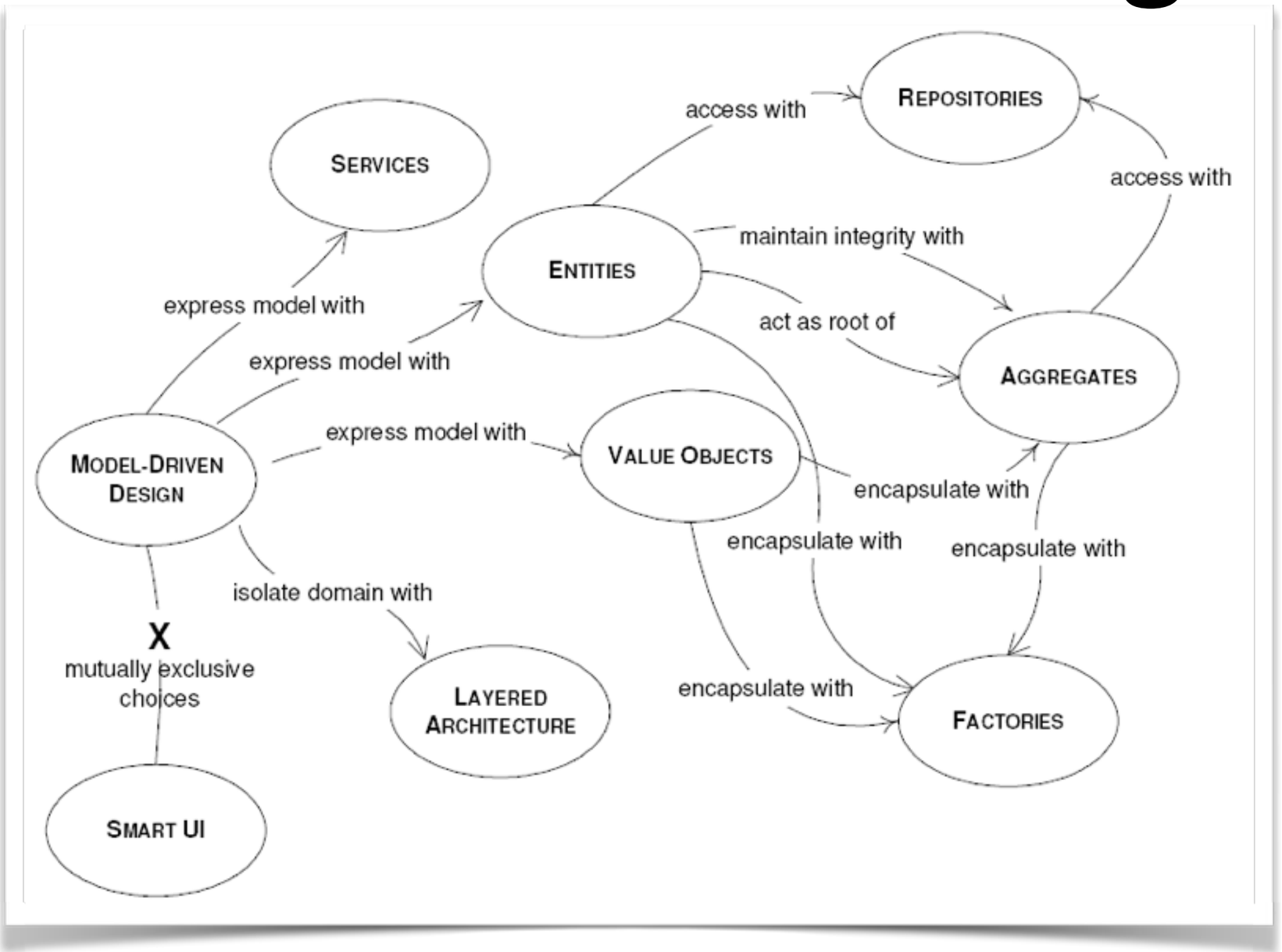
Complexity

The critical complexity of most software projects is in understanding the domain itself.

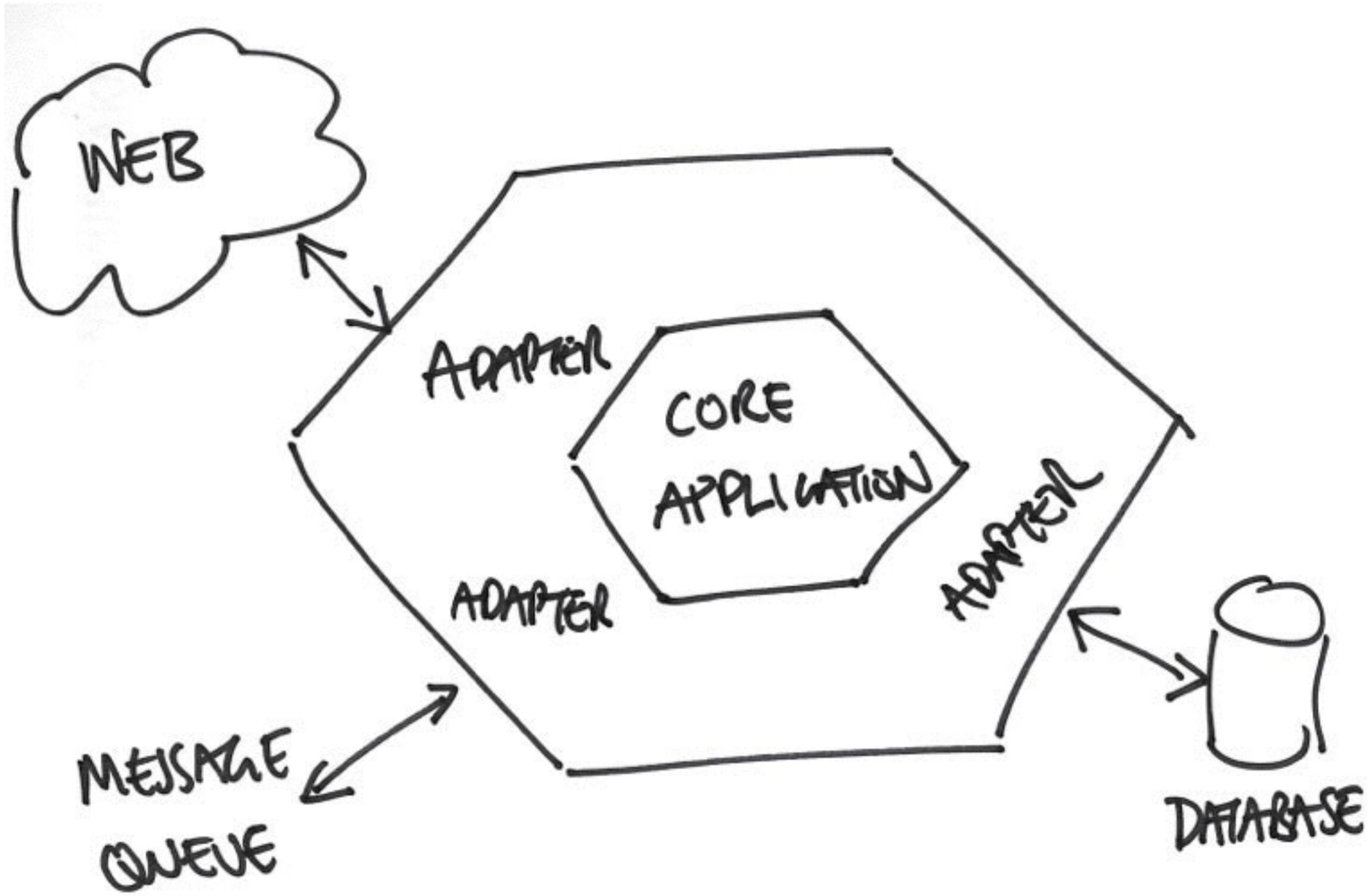
Eric Evans


Ubiquitous Language

Domain Driven Design



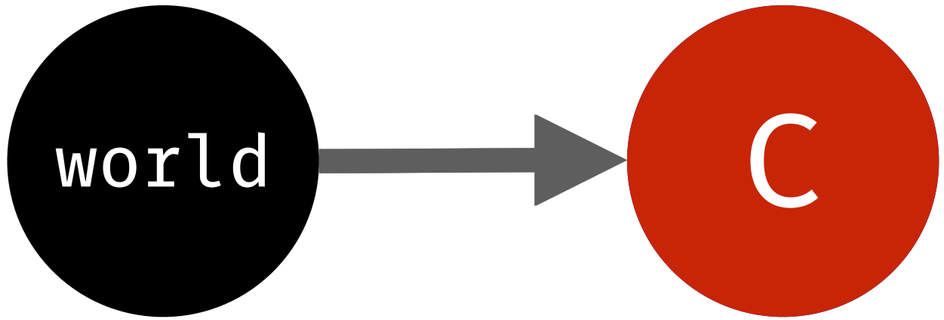
Hexagonal Architecture



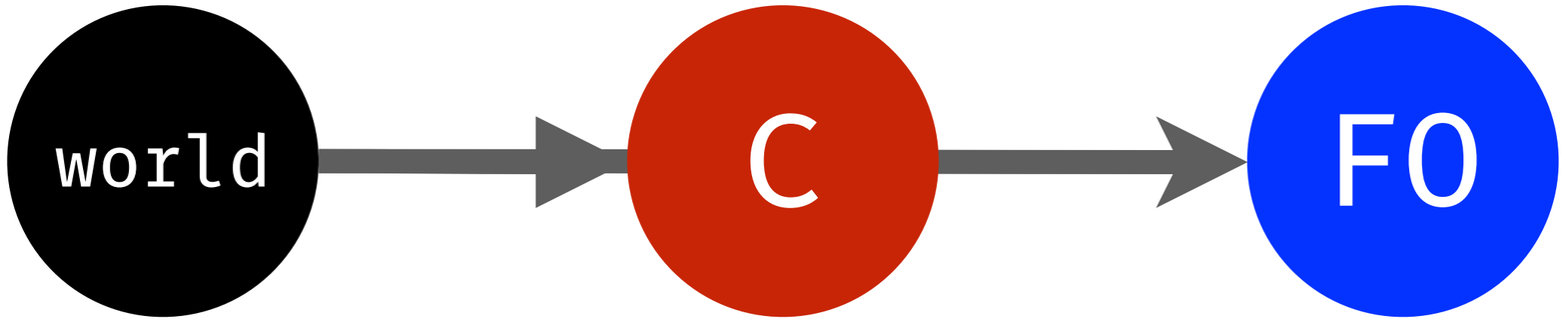
Matt Wynne

http://blog.mattwynne.net/2012/05/31/hexagonal-rails-objects-values-and-hexagons/

Form Object



Form Object



STARTUP TRAIN

Select your ticket

All tickets include first class travel from Toronto to Montreal and full access to the International Startup Festival

- **For Investors and All Others**
  ($750 + $97.50 HST)

  This ticket grants you access to all Startup Festival activities; from the cocktail party on July 9th, through to closing of the event on July 11th (Open House Day activities on July 12th also included). OFF EVENTS NOT INCLUDED.

- **For Startups**
  ($600 + $78.00 HST)

  Must be a Startup to purchase this ticket. Your Startup must be 3 years or younger. 1 person per ticket. This ticket grants you access to all Startup Festival activities; from the cocktail party on July 9th, through to closing of the event on July 11th (Open House Day activities on July 12th also included). OFF EVENTS NOT INCLUDED.

Get on board July 9th

Full name

Email address

Add a passenger for $847.50

Card Number

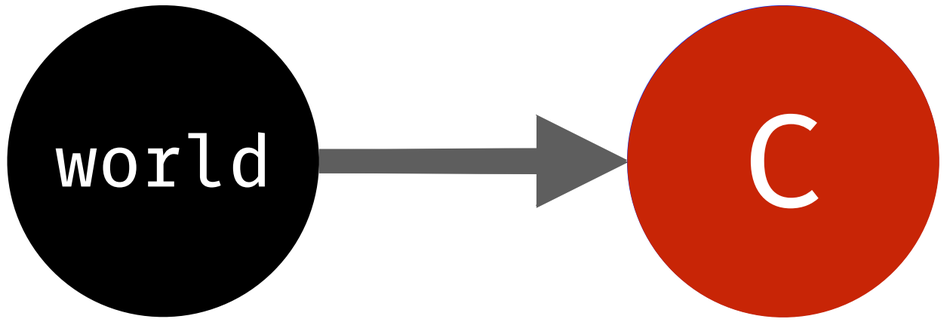
CVC | MM | YYYY

Pay $847.50

```ruby
class TicketForm
  include ActiveModel::Model

  validates :trip, :price, :passengers, presence: true

  def passengers
    @passengers ||= [Passenger.new]
  end

  def tickets
    @tickets ||= self.passengers.map do |passenger|
      Ticket.new(...)
    end
  end
end
```

```ruby
class TicketsController < ApplicationController

  def create
    @ticket_form = TicketForm.new(params)
    tickets = @ticket_form.tickets

    if @ticket_form.valid? && TicketCharger.new(tickets).charge!
      redirect_to success_url
    else
      render 'new'
    end
  end

end
```

Request Object

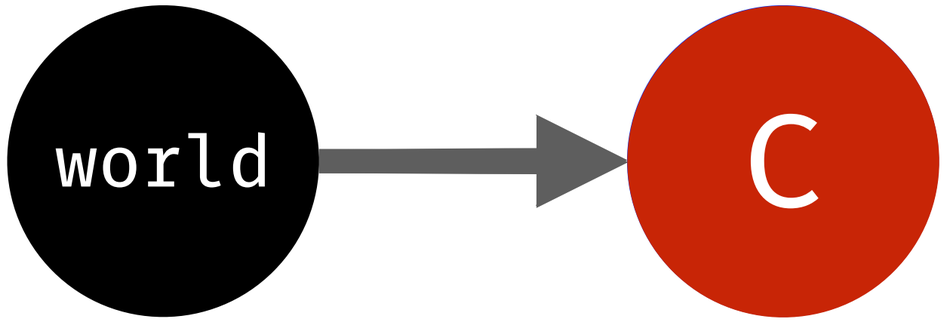


Request Object

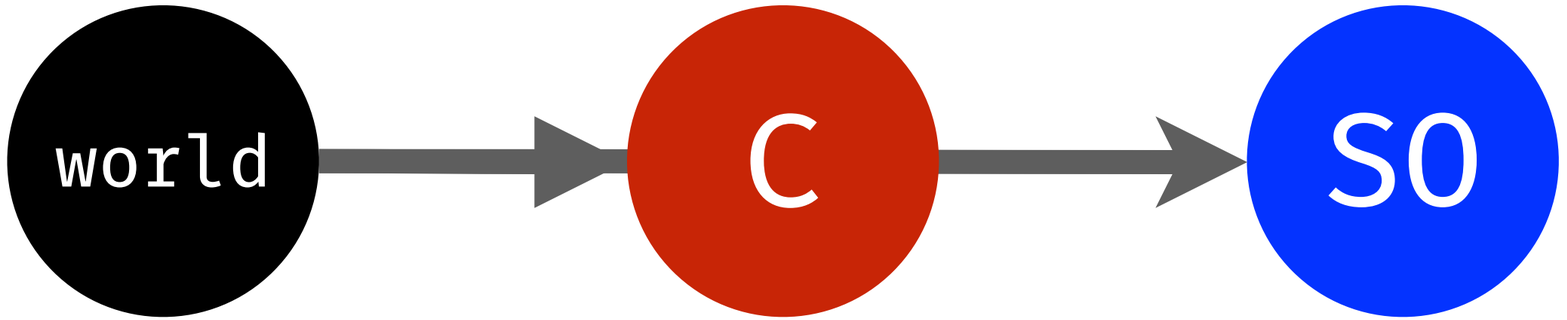
```ruby
class CreateOrderRequest

  include Virtus.value_object
  include ActiveModel::Validations

  attribute :customer, Customer
  validates :customer, nested: true, presence: true

  attribute :billing, Billing
  validates :billing, nested: true, presence: true

  attribute :shipping, Shipping
  validates :shipping, nested: true, presence: true

end
```

```ruby
class ApplicationController < ActionController::Base
  before_filter :validate_request

  def validate_request
    handle_error(request_object) unless request_object.valid?
  end

  def request_object
    @request_object ||= request_class.new(request_parameters)
  end

  def request_class
    "#{action_name}#{resource_name}Request".constantize
  end

  def handle_error(request_object)
    [...]
  end
end
```

Service Object



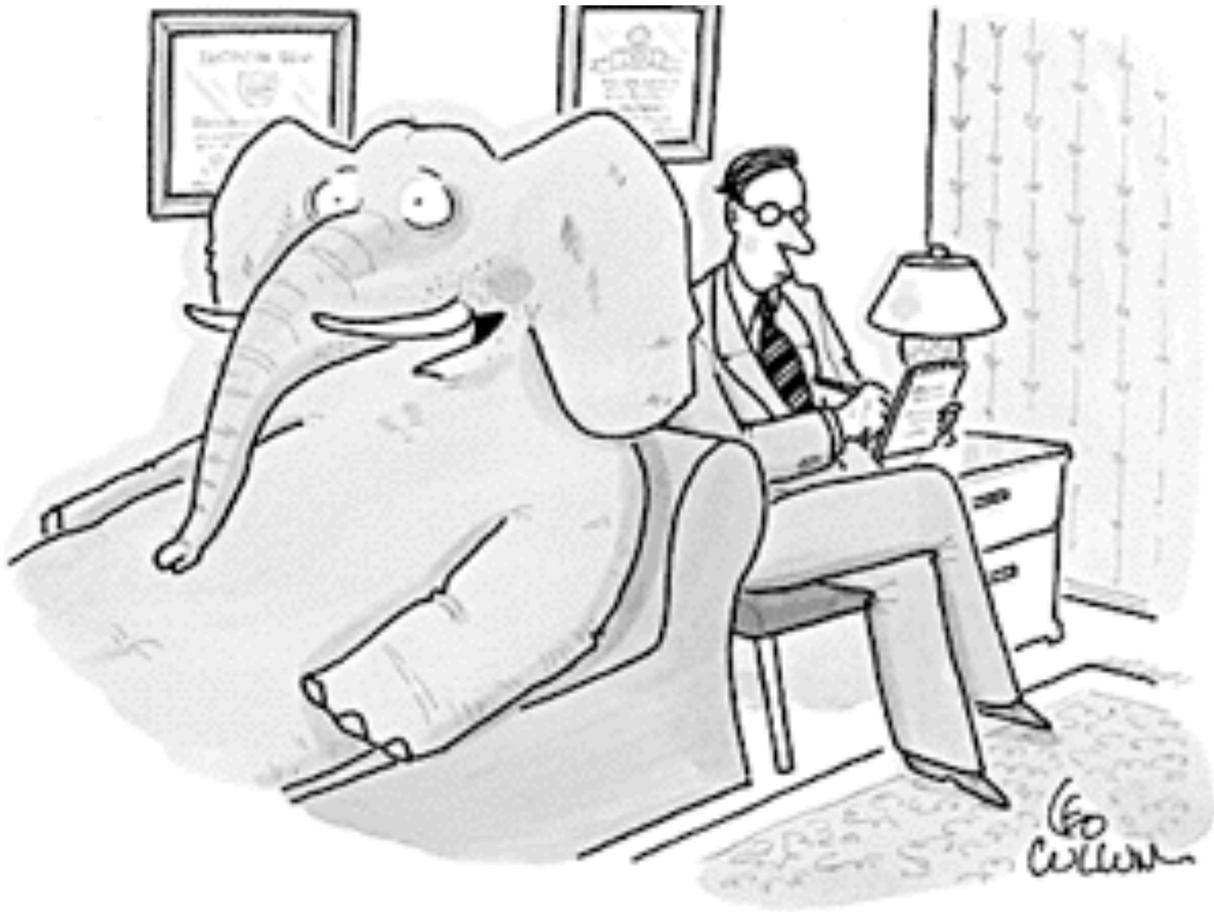
Service Object



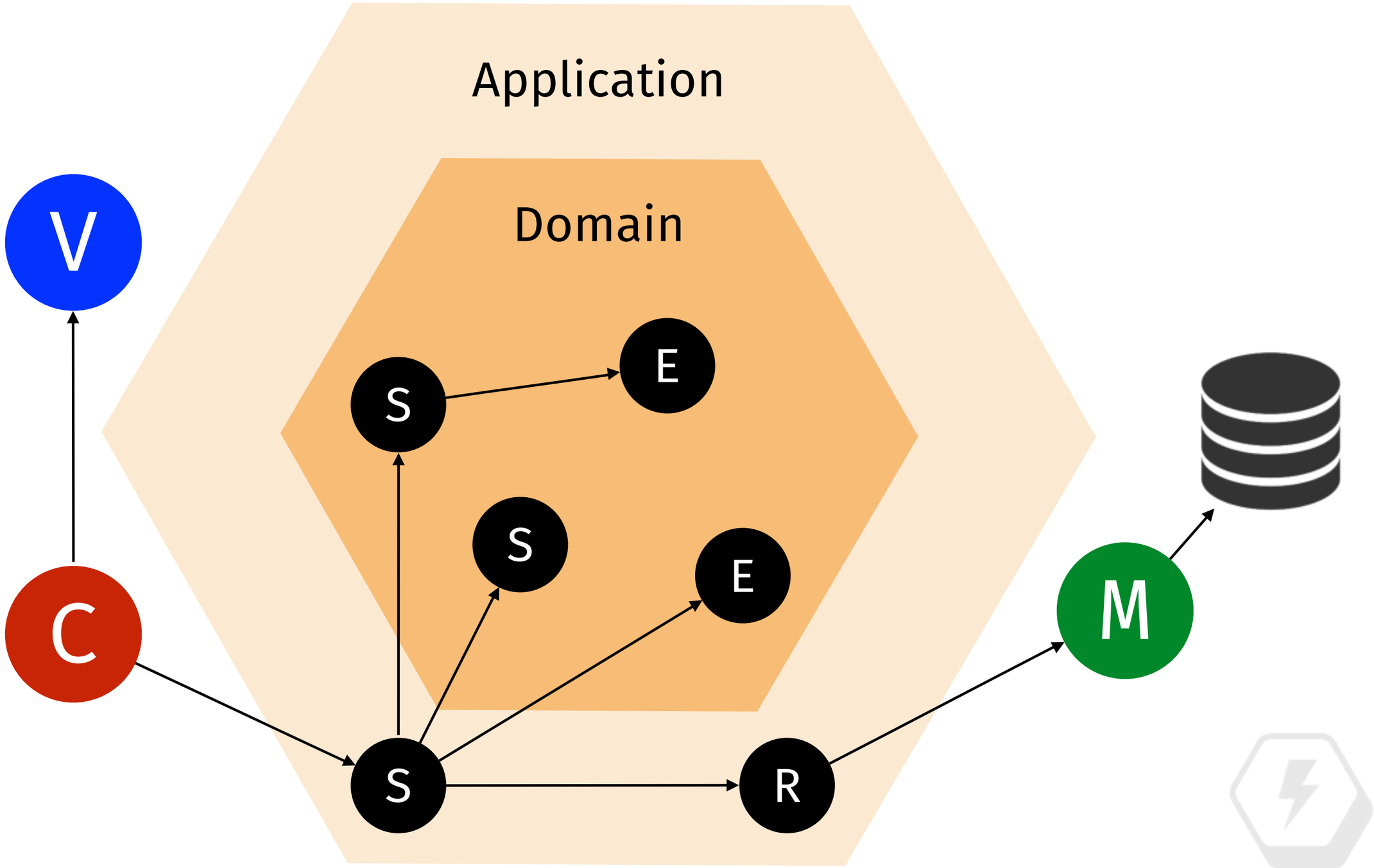
```ruby
class OrderService

  def create(order)
    authorize!(order)

    repository.save!(order)

    purchase(order) do |transaction|
      repository.save!(order)
    end
  end

  def purchase(order, &block)
    PaymentService.new.purchase(order, &block)
  end

end
```

```ruby
class OrdersController < ApiController

  def create
    order       = request_object.to_order
    transaction = OrderService.new.create(order)

    if transaction.success?
      order_created(order)
    else
      payment_failed(transaction)
    end
  end

end
```

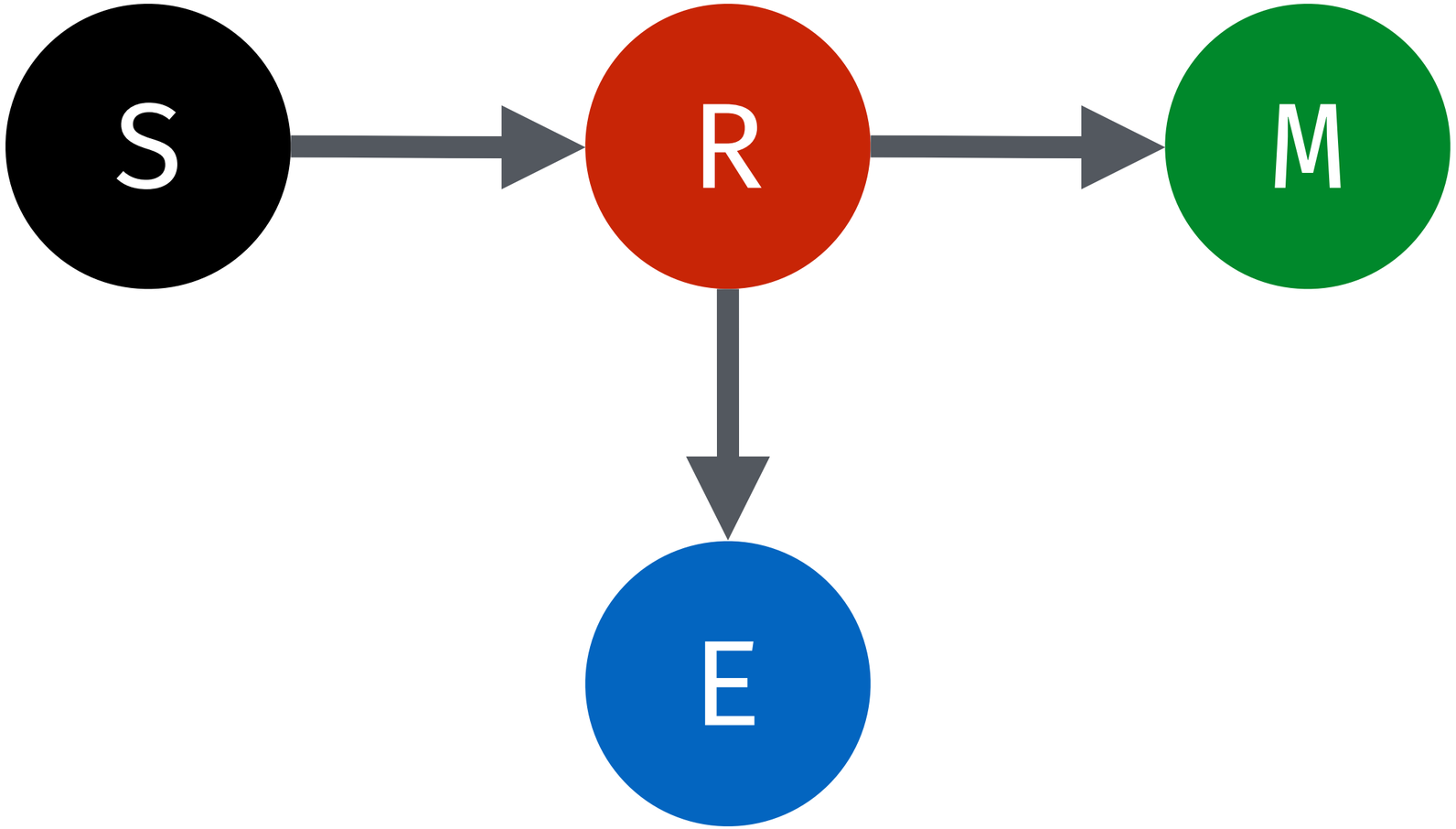
OK, so now what?



“I’m right there in the room, and no one even acknowledges me.”


Application
Domain
V
C
S
E
S
E
S
R
M

Repository



```ruby
class Repository

  class << self

    attr_accessor :mapper

    def save!(domain)
      record    = mapper.export(domain, record)
      response  = record.save!
      domain.id = record.id
      response
    end

  end

  def self.method_missing(method_name, *args, &block)
    Scope.new(mapper).send(method_name, *args, &block)
  end

end
```

```ruby
class Scope

  attr_accessor :mapper, :scope

  def initialize(mapper)
    @mapper = mapper
    @scope  = mapper.export_class
  end

  def method_missing(method_name, *args, &block)
    @scope = scope.send(method_name, *args, &_map_block(block))

    scope.is_a?(ActiveRecord::Relation) ? self : _map(scope)
  end

  def _map_block(block)
    Proc.new { |*args| block.call(_map(*args)) } if block
  end

  def _map(object)
    if object.is_a?(mapper.export_class)
      mapper.map(object)
    elsif object.is_a?(Enumerable)
      object.map { |e| e.is_a?(mapper.export_class) ? mapper.map(e) : e }
    else
      object
    end
  end

  def respond_to?(method_name, include_private = false)
    scope.respond_to?(method_name, include_private) || super
  end
end
```

```ruby
class Mapper
  class << self

    attr_reader :base_class, :export_class

    def maps(mapping)
      @base_class, @export_class = mapping.first
    end

    def map(object)
      if object.is_a? base_class
        export(object)
      else
        import(object)
      end
    end

    def export(base, record=nil)
      return unless base

      if record
        record.assign_attributes(base.attributes)
      else
        record = export_class.new(base.attributes)
      end

      record
    end

    def import(record)
      base_class.new(record.attributes) if record
    end
  end
end
```

```ruby
class Repository

  class << self

    attr_accessor :mapper

    def save!(domain)
      record    = IdentityMap.get(mapper.export_class, domain.id)
      record    = mapper.export(domain, record)
      response  = record.save!
      IdentityMap.add(record)
      domain.id = record.id
      response
    end

  end

  def self.method_missing(method_name, *args, &block)
    Scope.new(mapper).send(method_name, *args, &block)
  end

end
```

```ruby
class IdentityMap

  class << self
    def add(record)
      raise ArgumentError.new('Record cannot be added with a nil id') unless record.id
      repository[key(record.class)][record.id] = record
    end

    def remove(record)
      repository[key(record.class)].delete(record.id)
    end

    def get(klass, id)
      repository[key(klass)][id]
    end

    def clear
      repository.clear
    end

    def repository
      Thread.current[:identity_map] ||= Hash.new { |h,k| h[k] = {} }
    end

    def key(klass)
      klass
    end

  end
end
```

So, what’s the point again?

1. Embrace complexity

2. Know where you’re going

3. Be more than just a “Rails Developer”

Continue the discussion

https://github.com/dwhelan/hex-ddd

Declan Whelan
@dwhelan

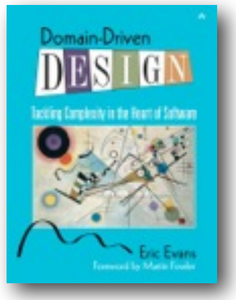
PRINT CHOMP

Eric Roberts
@eroberts

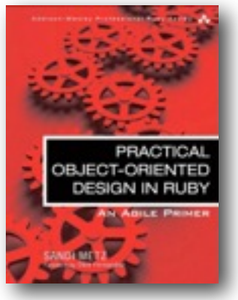
boltmade

Reading

Patterns of Enterprise Application Architecture
Martin Fowler



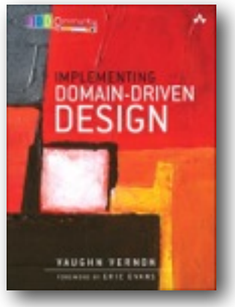
Domain Driven Design
Tackling Complexity in the Heart of Software
Eric Evans



Practical Object Oriented Design in Ruby
An Agile Primer
Sandi Metz

Reading

Clean Code
A Handbook of Agile Software Craftsmanship
Robert C. Martin



Implementing Domain Driven Design
Vaughn Vern

Photo/Video Credits

https://www.youtube.com/watch?v=-JLbAePwoHQ

http://www.docstoc.com/docs/47342986/Rube-Goldberg-Project-Rube

https://sitespecific2013awe.blogs.lincoln.ac.uk/tag/tower-of-babel/

http://collabcubed.com/2012/01/17/roskilde-festival-plywood-dome/

http://mitchjackson.com/white-elephants/