relevance

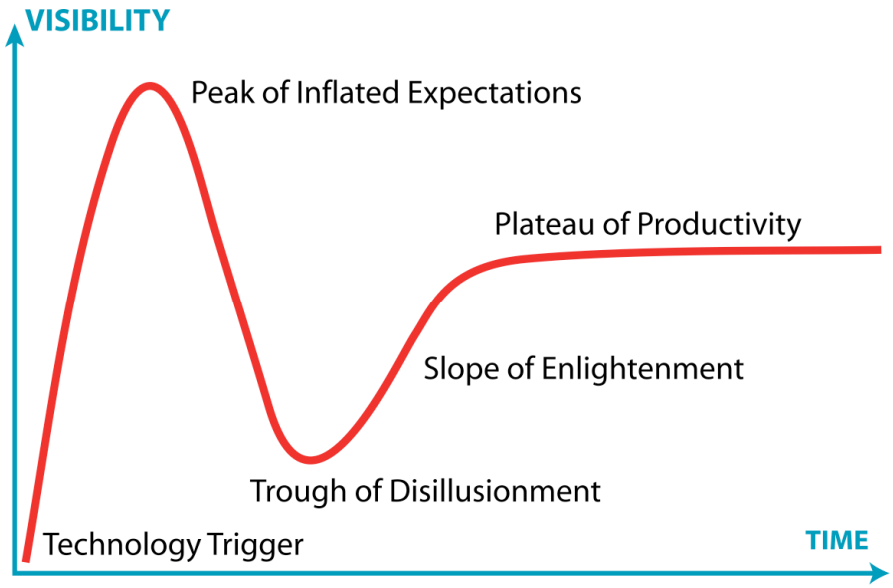
The State of the DSL Art in Ruby

Glenn Vanderburg
Relevance, Inc.
glenn@thinkrelevance.com

relevance

run›code›run

DSLs Are Overhyped



relevance

David Says. . .

The intent is to bridge programming and conversation by creating a domain language that can be shared by both. Having such a language means cutting down on the mental translation that otherwise confuses the discussion of a *product description* with the client when it's really implemented as *merchandise body*. These communications gaps are bound to lead to errors.

From *AgileWeb Development with Rails, 2e*, by Dave Thomas and David Heinemeier Hansson

relevance

Evolution

Origins

In Lisp, you don't just write your program down toward the language, you also build the language up toward your program.

—Paul Graham

Lisp

```
(task "warn if website is not alive"
      every 3 seconds
      starting now
      when (not (website-alive? "http://example.org"))
      then (notify "admin@example.org" "server down!"))
```

relevance

Functional Languages

```
keepleft (p :>: ps)
    | keepleft p = case partitionFL keepleft ps of
                     a :> b -> p :>: a :> b
    | otherwise  = case commuteWhatWeCanFL (p :> ps) of
                     a :> p' :> b -> case partitionFL keepleft a of
                                       a' :> b' -> a' :> b' +>+ p' :>: b
```

relevance

Ruby, Out of the Box

```
attr_reader   :id, :age
attr_writer   :name
attr_accessor :color
```

relevance

```
class Module
  def attr_reader (*syms)
    syms.each do |sym|
      class_eval %{def #{sym}
                     @#{sym}
                   end}
    end
  end
end
```

relevance

```ruby
class Module
  def attr_writer (*syms)
    syms.each do |sym|
      class_eval %{def #{sym}= (val)
                     @#{sym} = val
                   end}
    end
  end
end
```

relevance

Bouchard's X11 Library

ReparentWindow
window, *parent*: WINDOW
x, *y*: INT16
Errors: **Match**, **Window**

DestroySubwindows
window: WINDOW
Errors: **Window**

ChangeSaveSet
window: WINDOW
mode: {**Insert**, **Delete**}
Errors: **Match**, **Value**, **Window**

```ruby
def_remote :close_subwindows, 5, [Self]
def_remote :change_save_set, 6, [Self,
      [:change_type, ChangeMode, :in_header]]
def_remote :reparent, 7, [Self,
      [:parent, Window],
      [:point, Point]]
```

Styles Have Changed

ObjectReference Command Set (9)
ReferenceType Command (1)
Returns the runtime type of the object. The runtime type will be a class or an array.

Out Data

objectID	*object*	The object ID

Reply Data

byte	*refTypeTag*	Kind of following reference type.
referenceTypeID	*typeID*	The runtime reference type.

```
JDWP.add_command_set :ObjectReference, 9 do |set|
  set.add_command :ReferenceType do |cmd|
    cmd.description = "Returns the runtime type of the object.  The ..."
    cmd.out_data :objectID, :object, "The object ID"
    cmd.reply_data :byte, :refTypeTag, "Kind of following reference type."
    cmd.reply_data :referenceTypeID, :typeID, "The runtime reference ..."
  end
end
```

Dave's Summer Project

- Dave Thomas, RubyConf 2002
  "How I Spent My Summer Vacation"

```
class RegionTable < Table
  table "region" do
    field autoinc,      :reg_id,        pk
    field varchar(100), :reg_name
    field int,          :reg_affiliate, references(AffiliateTable,
                                                   :aff_id)
  end
end
```

Rails

```ruby
class CreateRegions < ActiveRecord::Migration
  def self.up
    create_table :regions do |t|
      t.string     :name
      t.belongs_to :affiliate
    end
  end

  def self.down
    drop_table :regions
  end
end
```

```ruby
class Region < ActiveRecord::Base
  belongs_to :affiliate
end
```

RSpec

```ruby
describe Codebase do

  it "should load from a hash with optional attributes omitted" do
    cb = Codebase.load({:name => 'a', :base_path => '/b/c'})
    cb.name.should      == 'a'
    cb.base_path.should == '/b/c'
    cb.name.should      == 'a'
  end

end
```

What Makes Internal DSLs Special?

relevance

General-Purpose Constructs

- Types
- Literals
- Declarations
- Expressions
- Operators
- Statements
- Control Structures

relevance

Specialized Constructs

- Context-dependence
- Commands and sentences
- Units
- Large vocabularies
- Hierarchy

relevance

Contexts

```
Interval = new_struct(:start, :end) do
  def length
    self.end - self.start
  end
end


create_table :regions do |t|
  t.string     :name
  t.belongs_to :affiliate
end
```

relevance

Implementing Contexts

```ruby
def new_struct (*args, &block)
  struct_class = Class.new
  struct_class.class_eval { attr_accessor *args }
  # define initialize method
  struct_class.class_eval(&block) if block_given?
  struct_class
end

def create_table(table_name, options = {})
  table_definition = TableDefinition.new(options)

  yield table_definition

  if options[:force] && table_exists?(table_name)
    drop_table(table_name, options)
  end

  execute table_definition.to_sql
end
```

Commands and Sentences

```ruby
field autoinc,:reg_id,        pk
field int,    :reg_affiliate, references(AffiliateTable,
                                         :aff_id)
```

Commands and Sentences

```
field autoinc,  :reg_id,  pk
```


Modern Sentences

```
has_many :favorites, :order => :position,
         :conditions => {:state => 'public'}

has_many :roles, :through => :projects, :uniq => true

validates_length_of :login, :within => 3..40,
                    :on => :create

validates_presence_of :authority, :if => :in_leadership_role
              :message => "must be authorized for leadership."
```

Implementing Sentences

```
has_many :roles,  :through => :projects,
                  :uniq => true
```

```
def declaration(thing, options={})
  # validate and process options
  # create and store metadata
  # define custom methods
end
```

relevance

Units

- General-purpose languages deal with scalars
- Most domain-specific languages deal with quantities expressed using units.
- From Rails:

```
# A time interval
3.years + 13.days + 2.hours
# Four months from now, on a Monday
4.months.from_now.next_week.monday
```

relevance

Implementing Units

```ruby
# Augment the built-in classes
class Numeric
  def minutes; self * 60; end
  def hours; self * 60.minutes; end
  # etc.
end
```


Large Vocabularies

- Roman numerals:

```ruby
Roman.CCXX
Roman.XLII
```

- XmlMarkup class:

```ruby
xm.em("emphasized")
xm.a("A Link", :href => "http://example.com/")
xm.target(:name => "compile", "option" => "fast")
```

Large Vocabularies

```
class Roman
  def self.method_missing (method_id)
    str = method_id.id2name
    roman_to_int(str)
  end
end
```

relevance

Hierarchy

```
xml.html {
  xml.head {
    xml.title("History")
  }
  xml.body {
    xml.h1("Header")
    xml.p("paragraph")
  }
}
```

relevance

Implementing Hierarchy

- Called from `method_missing`:

```
def element (elem_name, opts={})
  write "<#{elem_name}#{encode_opts(opts)}"
  if block_given?
    puts ">#{yield}</#{elem_name}>"
  else
    puts "/>"
  end
end
```


Perspective

Ruby's DSL Strengths

- Dynamic and reflective
- Blocks allow writing new control structures
- Declarations are executable
- Built-in contexts
- Only slightly less malleable than Lisp (no macros)

Syntax Matters

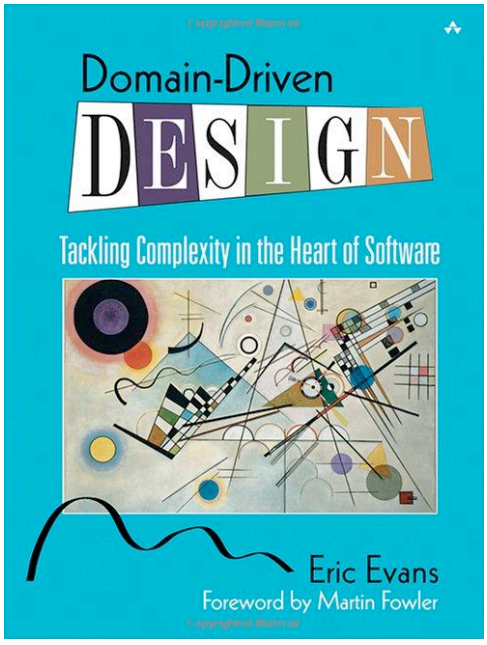
- Neutral and unobtrusive
- Enough to distinguish different kinds of constructs
- Not enough to complicate straightforward statements
- Most punctuation is optional

DSLs != Magic Pixie Dust

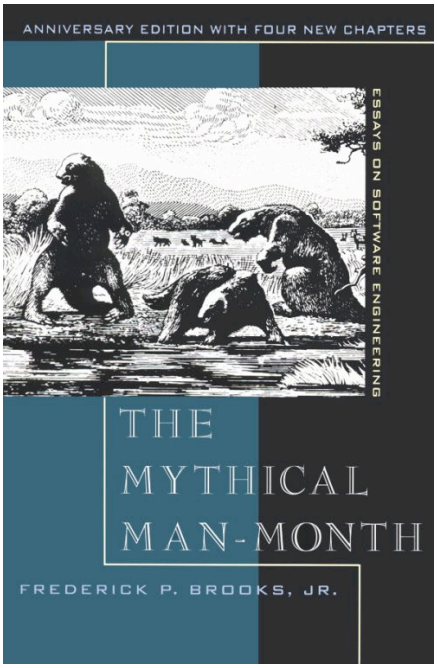
Photo credit: Tracey Parker

relevance

Domain Language



Essence and Accident



Good Software Design

- Eliminate as much of the accidental complexity as possible.
- *Separate* the rest.

Language and program evolve together. Like the border between two warring states, the boundary between language and program is drawn and redrawn, until eventually it comes to rest along the mountains and rivers, the natural frontiers of your problem.

In the end your program will look as if the language had been designed for it. And when language and program fit one another well, you end up with code which is clear, small, and efficient.

—Paul Graham


```
def create
  @post = Post.new(params[:post])

  respond_to do |format|
    if @post.save
      flash[:notice] = 'Post was successfully created.'
      format.html { redirect_to @post }
      format.xml  { render :xml      => @post,
                           :status   => :created,
                           :location => @post }
    else
      format.html { render :action => "new" }
      format.xml  { render :xml    => @post.errors,
                           :status => :unprocessable_entity }
    end
  end
end
```

```ruby
def new
  @post = Post.new

  respond_to do |format|
    format.html
    format.xml  { render :xml => @post }
  end
end
```

relevance

DSLs != Polyjuice Potion

Photo credit: Jo Naylor

relevance

Barrier to Understanding?

- The language is **for people who understand the domain.**
- The things that are implicit are **accidental complexity.**
- Learning the language **aids in understanding the domain.**

Good API Design

- Creating DSLs with everyday constructs is *powerful.*
- You can refactor to them as you find duplication, complexity.
- Internal DSLs are just a part of good API design in Ruby.

Library design is language design.

—Bell Labs Proverb

relevance

What are DSLs Really Good For?

- Solid domain modeling
- More and better options when refactoring
- Customer communication
- Clean separation of essence and accident

relevance