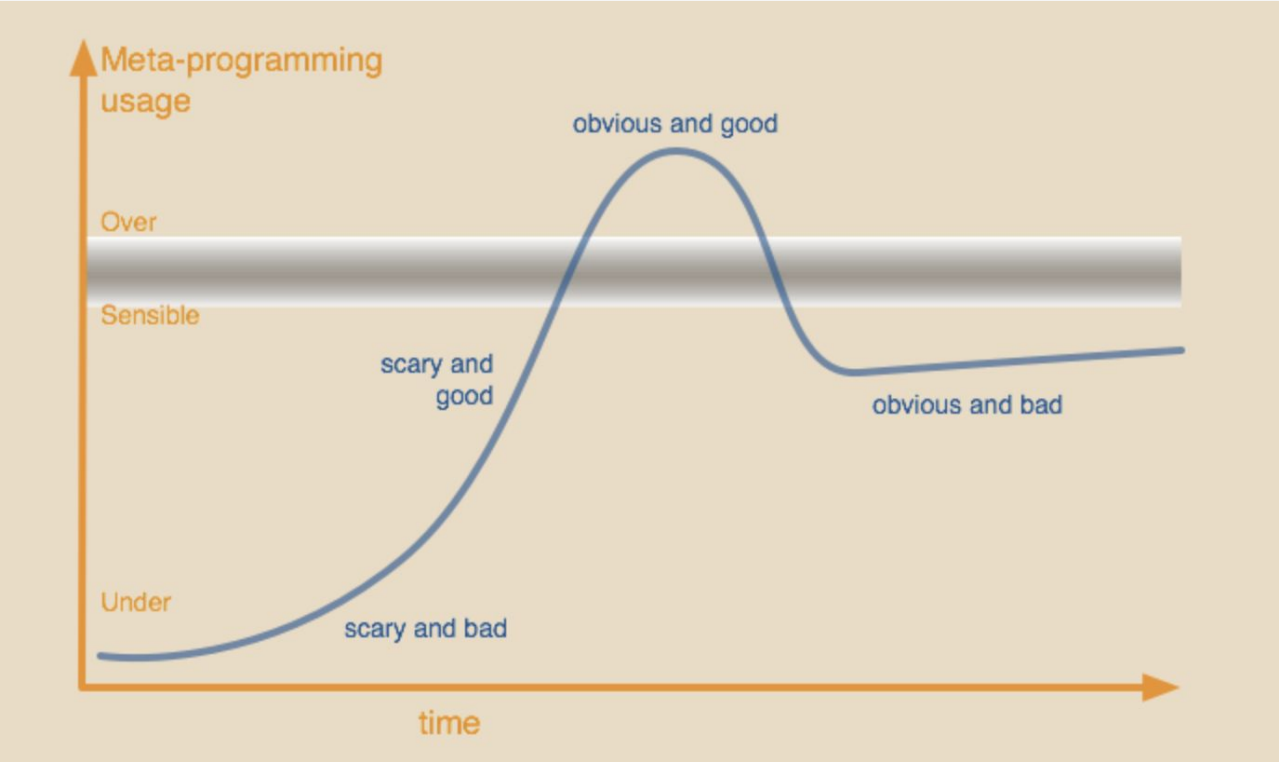
An Introduction to Ruby Metaprogramming

Karl Parkinson

What is Metaprogramming

- Programming about programming.
- Writing code about writing code.
- In Ruby, it is the ability to modify already existing objects, classes, and modules dynamically at runtime.
- Can be confusing to write and read, but is very powerful.

Use Wisely


Meta-programming usage
obvious and good
Over
Sensible
scary and good
obvious and bad
Under
scary and bad
time

Monkey Patching An Existing Class

```ruby
s = "hello"
puts s.upcase # HELLO

class String
  def upcase
    # Uh oh, upcase now becomes downcase. Monkey patching can get you in trouble. Be carefu
    downcase
  end
end

puts s.upcase # hello, thanks to monkey patch
```

respond_to?

```ruby
class Foo

  def method_a
    puts "method a"
  end

end

puts Foo.new.respond_to?("method_a") # true
puts Foo.new.respond_to?("method_b") # false
```

Runtime Method Creation

```ruby
class Foo
  def method_missing(method, *args, &block)
    puts "method #{method} does not exist."
  end
end

class Bar
  # Can create method on the fly at runtime using method_missing and define_method
  def method_missing(method, *args, &block)
    self.class.send(:define_method, method) do
      puts "Created a method called #{method}!"
    end
    self.send(method, *args, &block)
  end
end

Foo.new.do_something
Bar.new.do_something
Bar.new.do_something_else
```

Class Eval and Instance Eval

- https://gist.github.com/KarlParkinson/44092bba9ff1b51f6cd0a8cf12d3047f

Hook Methods

- Runtime “hooks” exist in ruby
- included, extended, prepended, inherited
- https://gist.github.com/KarlParkinson/a1c987d8e0d5941b2b2dabfc33942b1b

Practical Example

- Problem: Want a module that can be included in an arbitrary class that will log running time in seconds of all methods of that class.
- Metaprogramming solution: https://gist.github.com/KarlParkinson/d359264d8aac4be090eb2bdb008d7f29

A Brief Explainer of How This Works

- Ruby is an Object Oriented language
- **Everything** in Ruby is an object, including classes. They are instances of type Class.
- A class can be manipulated and interacted with at runtime just like any other object, because they are just another object.
- Every class has an Eigenclass, which is an invisible class unique to each object.
- The Eigenclass is a class like any other, it can have variables and methods. Class methods are simply instance methods of the Eigenclass.

Further Reading

- https://yehudakatz.com/2009/11/15/metaprogramming-in-ruby-its-all-about-the-self/
- https://stackoverflow.com/questions/2505067/class-self-idiom-in-ruby
- https://suchdevblog.com/lessons/ExplainingRubySingletonClass.html#what-s-the-eigenclass
- https://medium.com/rubycademy/understanding-the-eigenclass-in-less-than-5-minutes-dcb8ca223eb4
- https://www.geeksforgeeks.org/ruby-hook-methods/
- https://apidock.com/ruby/Module/class_eval
- https://apidock.com/ruby/Object/instance_eval
- https://www.toptal.com/ruby/ruby-metaprogramming-cooler-than-it-sounds
- https://github.com/KarlParkinson/ruby-metaprogramming-talk