# Getting simple

Designing and Maintaining Software (DAMS)

Louis Rose

# Habitable Software

**Lean**er

Less **Complex**

Loosely **Coupled**

More **Cohesive**

Avoids **Duplication**

**Clear**er

More **Extensible**

???

# Tactics

*Favour shorter methods (and classes)*

*Favour “functional style” to loops*

*Favour method calls to conditionals*

# Loops

**Conjecture 1**: Looping is a low-level evil

```
class Pizza
  def title
    if @toppings.empty?
      "Margherita"
    else
      title = ""
      @toppings.each do |topping|
        title += " and " unless title.empty?
        title += topping
      end
      title
    end
  end
end
```

# Loop like it's 1999

Remember this?

```
class Pizza
  def title
    if @toppings.empty?
      "Margherita"
    else
      title = ""
      index = 0
      while index < @toppings.length
        topping = @toppings[index]
        title += " and " unless title.empty?
        title += topping
        index += 1
      end
      title
    end
  end
end
```

# Loop like it’s 1999

Or was it this?

```
class Pizza
  def title
    if @toppings.empty?
      "Margherita"
    else
      title = ""
      index = 0
      while index <= @toppings.length
        topping = @toppings[index]
        title += " and " unless title.empty?
        title += topping
        index += 1
      end
      title
    end
  end
end
```

# Loop like it’s 1999

Or this??!

```
class Pizza
  def title
    if @toppings.empty?
      "Margherita"
    else
      title = ""
      index = 0
      while index <= @toppings.length - 1
        topping = @toppings[index]
        title += " and " unless title.empty?
        title += topping
        index += 1
      end
      title
    end
  end
end
```

# Loop like it’s 2099

Let’s never do that again

```
class Pizza
  def title
    if @toppings.empty?
      "Margherita"
    else
      title = ""
      @toppings.each do |topping|
        title += " and " unless title.empty?
        title += topping
      end
      title
    end
  end
end
```

# Loop like it’s 2099

And most modern languages have now learnt from FP

```
class Pizza
  def title
    if @toppings.empty?
      "Margherita"
    else
      title = ""
      @toppings.each do |topping|
        title += " and " unless title.empty?
        title += topping
      end
      title
    end
  end
end
```

# Loop like it’s 2099

Ruby: see the Enumerable API

```
class Pizza
  def title
    if @toppings.empty?
      "Margherita"
    else
      @toppings.join(" and ")
    end
  end
end
```

# Loop like it’s 2099

Ruby: see the Enumerable API

```
class Pizza
  def title
    if @toppings.empty?
      "Margherita"
    else
      @toppings.join(" and ")
    end
  end
end
```

# Conditionals

**Conjecture 2**: Conditionals are a low-level evil

```
class Pizza
  def title
    if @toppings.empty?
      "Margherita"
    else
      @toppings.join(" and ")
    end
  end
end
```

# Conditionals

Toppings is an array. Is it the right abstraction?

```
class Pizza
  def title
    if @toppings.empty?
      "Margherita"
    else
      @toppings.join(" and ")
    end
  end
end
```

# Conditionals

Toppings is an array. Is it the right abstraction?

```
class Pizza
  def title
    if @toppings.empty?
      "Margherita"
    else
      @toppings.join(" and ")
    end
  end
end
```

# Conditionals

Toppings is an array. Is it the right abstraction?

```
class Pizza
  def title
    if @toppings.empty?
      "Margherita"
    else
      @toppings.join(" and ")
    end
  end
end
```

# Conditionals

If only Ruby arrays knew about pizzas…

```
class Pizza
  def title
    @toppings.pizzaify
  end
end
```

# Conditionals

Array is not the right abstraction.

```
class Pizza
  def initialize(toppings = [])
    @toppings = toppings
  end

  def title
    @toppings.pizzaify
  end
end
```

# Conditionals

Array is not the right abstraction.

```
class Pizza
  def initialize(toppings = [])
    if toppings.empty?
      @toppings = Plain.new
    else
      @toppings = Topped.new(toppings)
    end
  end

  def title
    @toppings.pizzaify
  end
end
```

# Conditionals

Abstractions should come from the problem domain.

```
class Pizza
  def initialize(toppings = [])
    if toppings.empty?
      @toppings = Plain.new
    else
      @toppings = Topped.new(toppings)
    end
  end

  def title
    @toppings.pizzaify
  end
end
```

```
class Plain
  def pizzaify
    "Margherita"
  end
end

class Topped
  def initialize(toppings = [])
    @toppings = toppings
  end

  def pizzaify
    @toppings.join(“ and ")
  end
end
```

# Conditionals

Pizzaify wasn't the greatest name…

```
class Pizza
  def initialize(toppings = [])
    if toppings.empty?
      @toppings = Plain.new
    else
      @toppings = Topped.new(toppings)
    end
  end

  def title
    @toppings.title
  end
end
```

```
class Plain
  def title
    "Margherita"
  end
end

class Topped
  def initialize(toppings = [])
    @toppings = toppings
  end

  def title
    @toppings.join(" and ")
  end
end
```

# Hang on…

| Classes | Methods | LOC | Cyclomatic Complexity (>1) |
|---|---|---|---|
| 1 | 2 | 12 | Pizza#title |
| 3 | 5 | 25 | Pizza#initialize |

# Conditionals

But these kinds of conditionals tend to breed…

```
class Pizza
  def title
    if @toppings.empty?
      "Margherita"
    else
      @toppings.join(" and ")
    end
  end

  def cost
    cost = 4
    if @toppings.empty?
      cost += 1
    else
      cost += @toppings.size * 2
    end
  end
end
```

# Conditionals

But these kinds of conditionals tend to breed…

```
class Pizza
  def title
    if @toppings.empty?
      "Margherita"
    else
      @toppings.join(" and ")
    end
  end

  def cost
    cost = 4
    if @toppings.empty?
      cost += 1
    else
      cost += @toppings.size * 2
    end
  end
end
```

# Conditionals

Prefer method calls to conditionals.

```
class Pizza
  def title
    @toppings.title
  end

  def cost
    @toppings.cost + 4
  end
end
```

```
class Plain
  def title
    "Margherita"
  end

    def cost
    1
  end
end

class Topped
  def title
    @toppings.join(" and ")
  end

    def cost
    @toppings.size * 2
  end
end
```

# Aha!

| Classes | Methods | LOC | Cyclomatic Complexity (>1) |
|---|---|---|---|
| 1 | 3 | 20 | Pizza#title<br>Pizza#cost |
| 3 | 8 | 34 | Pizza#initialize |

# Null Object Pattern

Encapsulate the absence of an object

Active Nothing

# Summary

Complex methods hinder habitability.

Avoid loops by using a
declarative, functional style.

Avoid conditionals by using objects
and method calls.

# Also important

Strategy and Visitor patterns for
avoiding conditionals in other situations.

Further resources on designing
to avoid conditionals:

“Nothing is Something”
Sandi Metz (RailsConf 2015)

“Unconditional Programming”
Michael Feathers (2013 blog post)