The Birthday Problem.

Cascadia Ruby 2014
Scott Windsor
@swindsor

AKA: TDD, DHH, SRP, & TLA

Cascadia Conf 2014
Scott Windsor
@swindsor

Splitting up functions to support the testing process, [destroys] your system architecture and code comprehension along with it. Test at a coarser level of granularity.

— James Coplien, "Why Most Unit Testing is Waste"

"Test-Induced design Damage"

railsconf

railsconf


Test-wat?
quickmeme.com

Birthday example by DHH

```ruby
class Person
  attr_reader :birthday

  def initialize(attributes={})
    @birthday = attributes.fetch(:birthday)
  end

  def age
    Date.today.year - birthday.year
  end
end
```

Birthday stub solution

```
describe Person do
  it "age is determined by their birthday" do
    Date.stub :today, Date.new(2009) do
      seventy_niner = Person.new birthday: Date.new(1979)
      seventy_niner.age.must_equal 30
    end
  end
end
```

DHH's dependancy injection test

```
describe Person do
  it "age is determined by their birthday" do
    seventy_niner = Person.new birthday: Date.new(1979)
    seventy_niner.age(Date.new(2009)).must_equal 30
  end
end
```

DHH's dependency injection code

```
class Person
  attr_reader :birthday

  def initialize(attributes={})
    @birthday = attributes.fetch(:birthday)
  end

  def age(now = Date.today)
    now.year - birthday.year
  end
end
```

As more functionality is added…

```
describe Person do
  it "age is determined by their birthday" do
    seventy_niner = Person.new birthday: Date.new(1979)
    seventy_niner.age(Date.new(2009)).must_equal 30
  end

  it "can drink if older than 21" do
    seventy_niner = Person.new birthday: Date.new(1979)
    seventy_niner.can_drink?(Date.new(2009)).must_equal true
  end
end
```

Our design becomes “damaged”

```ruby
class Person
  attr_reader :birthday

  def initialize(attributes={})
    @birthday = attributes.fetch(:birthday)
  end

  def can_drink?(now = Date.today)
    age(now) > 21
  end

  def age(now = Date.today)
    now.year - birthday.year
  end
end
```

We can do better.

Better dependency injection

```ruby
class Person
  attr_reader :birthday, :today

  def initialize(attributes={})
    @birthday = attributes.fetch(:birthday)
    @today = attributes.fetch(:today){ Date.today }
  end

  def can_drink?
    age > 21
  end

  def age
    today.year - birthday.year
  end
end
```

Tests are much easier to follow

```ruby
describe Person do
  let(:seventy_niner) do
    Person.new birthday: Date.new(1979)
  end

  let(:seventy_niner_in_2009) do
    Person.new birthday: Date.new(1979), today: Date.new(2009)
  end

  it "age is determined by their birthday" do
    seventy_niner_in_2009.age.must_equal 30
  end

  it "can drink if older than 21" do
    seventy_niner_in_2009.can_drink?.must_equal true
  end

  it "sets today to Date.today by default" do
    seventy_niner.today.must_equal Date.today
  end
end
```

Better dependency injection?

```ruby
class Person
  attr_reader :birthday, :today

  def initialize(attributes={})
    @birthday = attributes.fetch(:birthday)
    @today = attributes.fetch(:today){ Date.today }
  end

  def can_drink?
    age > 21
  end

  def age
    today.year - birthday.year
  end
end
```

Birthday refactored tests

```
describe Person do
  it "should have a birthday" do
    seventy_niner = Person.new birthday: Date.new(1979)
    seventy_niner.birth_year.must_equal 1979
  end
end
```

Birthday refactored

```
class Person
  attr_reader :birthday

  def initialize(attributes={})
    @birthday = attributes.fetch(:birthday)
  end

  def birth_year
    birthday.year
  end
end
```

Let's add a Bouncer

```
describe Bouncer do
  let(:seventy_niner) do
    Person.new birthday: Date.new(1979)
  end

  it "should check if person can drink" do
    bouncer_in_2009 = Bouncer.new date: Date.new(2009)
    bouncer_in_2009.can_drink?(seventy_niner).must_equal true
  end
end
```

And implementation…

```
class Bouncer
  attr_reader :date

  def initialize(attributes={})
    @date = attributes.fetch(:date)
  end

  def can_drink?(person)
    date.year - person.birth_year > 21
  end
end
```

You design your code, as well as your tests.

Remember TDD is awesome.

You are awesome.