

Avoiding Sneaky Testing Antipatterns




Hi!
I'm Sarah

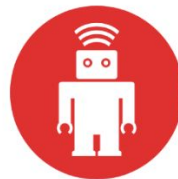


thoughtbot

sarah.lima@thoughtbot.com 

sarahraqueld 

thoughtbot/ **factory_bot**



A library for setting up Ruby objects as test data.

 257
Contributors

 158k
Used by

 8k
Stars

 3k
Forks



Anti-patterns are the opposite of best practices.

They seem to work, but the larger context or the long-term consequences are often not considered.

```
class User
```

```
  def save
```

```
    ...
```

```
  end
```

```
end
```

... you find a bug and you create this test

```
it 'does not return false when the email is valid' do
  user = User.new(email: 'valid@example.com')
  expect { user.save }.not_to be false
end
```

```
it 'does not return false when the email is valid' do

  expect { user.save }.not_to be false

end
```

```
expect { user.save }.not_to raise_error
```



```
it 'returns true with a valid email' do
  user = User.new(email: 'valid@example.com')
  expect(user.save).to be true
end
```

False Positives

False Positives

How to avoid:

Make sure you see the test fail

Mystery Guest

Let's not!

```
RSpec.describe User do
  let(:user) { create(:user) }

  it 'validates the presence of the name' do
    expect(user).to validate_presence_of(:name)
  end
end
```

The test reader is not able to see the cause and effect between fixture and verification logic because part of it is done outside the Test Method.

Arrange - Act - Assert

Test Hooks

```
class ApplicationController < ActionController::Base
  unless Rails.env.test?
    before_filter :require_login
  end
end
```


Test Hooks

Clearance

CI Tests passing maintainability ? docs Reviewed by Hound

Rails authentication with email & password.

Clearance is intended to be small, simple, and well-tested. It has opinionated defaults but is intended to be easy to override.

```
visit root_path(as: user)
```

Leaking Domain Knowledge

```
def calculate_total_price(offer)
  total_fees = offer.charges.sum { |charge| charge.amount }
  offer.face_value + total_fees
end
```

Leaking Domain Knowledge

```
RSpec.describe "Offer Price" do
  it "returns the sum of all charges amounts with the face value" do
    offer = Offer.new(
      face_value: 1000,
      charges: [
        Charge.new(type: "fee", amount: 50),
        Charge.new(type: "tax", amount: 100)
      ]
    )

    final_price = calculate_total_price(offer)
    expect(final_price).to eq(...)
  end
end
```

Leaking Domain Knowledge

```
RSpec.describe "Offer Price" do
  it "returns the sum of all charges amounts with the face value" do
  end

  it "other scenario" do
  end

  it "other scenario" do
  end

  def expected_price
  end
end
```

Leaking Domain Knowledge

```
RSpec.describe "Offer Price" do
  it "returns the sum of all charges amounts with the face value" do
    offer = {...}

    final_price = calculate_offer_price(offer)
    expect(final_price).to eq(1150)
  end
end
```

Leaking Domain Knowledge

Instead of duplicating the domain logic, pre-calculate the expected results with the help of a domain expert and hard-code the results into your tests.

Expressive

Expressive
Maintainable

Expressive

Maintainable

Isolated

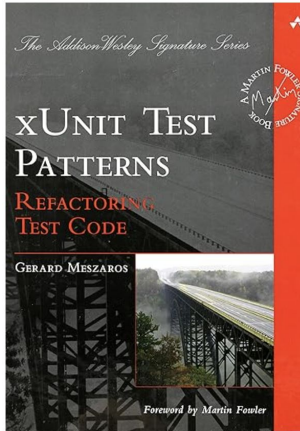
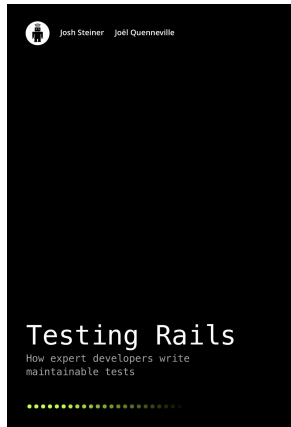
Expressive

Maintainable

Isolated

Reliable

References



books.thoughtbot.com/assets/testing-rails.pdf

blog.thoughtbot.com/tags/testing

thoughtbot.com/upcase/videos/testing-antipatterns