Ruby

Unit Testing / Minitest

Por qué escribimos tests?

- ¿Cómo sabemos que el código **funciona**?
  - No tenemos idea hasta que lo ejecutamos
- ¿Cómo **modificamos el código** con la garantía de que no **rompimos nada**?
- Sirve de **documentación** para los desarrolladores.

Test::Unit / Minitest::Test

- Ruby toma muy en serio el test unitario.
- En Ruby 1.8 - Test::Unit fue sobrecargado con librerías innecesarias por lo que en Ruby 1.9 fue reducido al mínimo.
- El nuevo framework tiene como nombre oficial Minitest, pero es un reemplazo sin impacto a nivel de código.

Test::Unit / Minitest::Test

- Miembro de la familia de los XUnit (Junit, CppUnit).
- La idea básica es extender de Test::Unit::TestCase
- Los métodos llevan como prefijo **test_**
- Si uno de los métodos falla - los demás continuarán (Esto es bueno!)
- Se pueden utilizar los métodos **setup()** y **teardown()** para configurar un comportamiento que se ejecutará antes de **cada test**.

Ejemplo: calculator.rb

```ruby
class Calculator

  attr_reader :name

  def initialize(name)
    @name = name
  end

  def add(one, two)
    one - two
  end

  def substract(one, two)
    one + two
  end

  def divide(one, two)
    one / two
  end
end
```

```ruby
require 'minitest/autorun'
require 'minitest/test'
require_relative 'calculator'
class CalculatorTest < Minitest::Test
  def setup
    @calc = Calculator.new('test')
  end

  def setup
    @calc = Calculator.new('test')
  end

  def test_divide_by_zero
    assert_raises ZeroDivisionError do
      @calc.divide(1, 0)
    end
  end
  def test_addition
    assert_equal 4, @calc.add(2,2)
  end
  def test_substraction
    assert_equal 2, @calc.substract(4,2)
  end
  def test_division
    assert_equal 2, @calc.divide(4,2)
  end
end
```

Ejecución del test

```
Run options: --seed 38964

# Running:

F..F

Finished in 0.001000s, 4000.0000 runs/s, 4000.0000 assertions/s.

  1) Failure:
CalculatorTest#test_addition [C:/vanessa/repositorios/codium/cursos/ruby/ruby-on-rails-intro/modulo-2/leccion-15/calculator_test.rb:19]:
Expected: 4
  Actual: 0


  2) Failure:
CalculatorTest#test_substraction [C:/vanessa/repositorios/codium/cursos/ruby/ruby-on-rails-intro/modulo-2/leccion-15/calculator_test.rb:22]:
Expected: 2
  Actual: 6

4 runs, 4 assertions, 2 failures, 0 errors, 0 skips
[Finished in 0.3s with exit code 1]
```

Si corregimos calculator.rb

```
Run options: --seed 9023

# Running:

....

Finished in 0.001000s, 4000.0000 runs/s, 4000.0000 assertions/s.

4 runs, 4 assertions, 0 failures, 0 errors, 0 skips
[Finished in 0.3s]
```

Entonces

- Las aserciones nos permiten ejercitar el código
- Las pruebas unitarias nos dan la confianza para modificar el código...

Mas información: Intro to Minitest