Behavior Driven Development

Freshen

Behavior Driven Development

Freshen

```
import unittest

import os
import sys

from freshen.noseplugin import FreshenNosePlugin
from optparse import OptionParser

class TestFreshenTestCaseName(unittest.TestCase):

    def __init__(self, method_name='runTest'):
        unittest.TestCase.__init__(self, method_name)
        self.cur_dir = os.path.dirname(os.path.abspath(__file__))

    def _make_plugin(self):
        plugin = FreshenNosePlugin()
        parser = OptionParser()

        plugin.options(parser, {})

        sys.argv = ['nosetests', '--with-freshen']
        (options, args) = parser.parse_args()

        plugin.configure(options, None)
        return plugin

    def test_should_use_feature_name_as_class_name_when_subclassing_FreshenTestCase(self):
        plugin = self._make_plugin()
        test_generator = plugin.loadTestsFromFile(self.cur_dir + '/resources/valid_no_tags_no_use_only.feature')
        test_instance = test_generator.next()

        self.assertEquals(test_instance.__class__.__name__, 'Independence of the counter.')

    def test_should_use_scenario_name_as_method_name_when_subclassing_FreshenTestCase(self):
        plugin = self._make_plugin()
        test_generator = plugin.loadTestsFromFile(self.cur_dir + '/resources/valid_no_tags_no_use_only.feature')
        test_instance = test_generator.next()

        self.assertNotEqual(getattr(test_instance, 'Print counter', None), None)
```

Cucumber

behaviour driven development with elegance and joy

1: Describe behaviour in plain text

```
Feature: Addition
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers

  Scenario: Add two numbers
    Given I have entered 50 into the calculator
    And I have entered 70 into the calculator
    When I press add
    Then the result should be 120 on the screen
```

2: Write a step definition in Ruby

```
Given /I have entered (.*) into the calculator/ do |n|
  calculator = Calculator.new
  calculator.push(n.to_i)
end
```

3: Run and watch it fail

```
$ cucumber features/addition.feature
Feature: Addition  # features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
  Scenario: Add two numbers                          # features/additi
    Given I have entered 50 into the calculator      # features/step_d
      uninitialized constant Calculator (NameError)
      ./features/step_definitons/calculator_steps.rb:2:in `Given /
      features/addition.feature:7:in `Given I have entered 50 into
    And I have entered 70 into the calculator        # features/step_d
    When I press add                                 # features/additi
    Then the result should be 120 on the screen      # features/additi
```

4. Write code to make the step pass

```
class Calculator
  def push(n)
    @args ||= []
    @args << n
  end
end
```

5. Run again and see the step pass

```
$ cucumber features/addition.feature
Feature: Addition  # features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
  Scenario: Add two numbers                          # features/additi
    Given I have entered 50 into the calculator      # features/step_d
    And I have entered 70 into the calculator        # features/step_d
    When I press add                                 # features/additi
    Then the result should be 120 on the screen      # features/additi
```

6. Repeat 2-5 until green like a cuke

```
$ cucumber features/addition.feature
Feature: Addition  # features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
  Scenario: Add two numbers                          # features/additi
    Given I have entered 50 into the calculator      # features/step_d
    And I have entered 70 into the calculator        # features/step_d
    When I press add                                 # features/step_d
    Then the result should be 120 on the screen      # features/step_d
```

```
Feature: Basic operations on the list
  In order to handle variable number of objects
  As a list user
  I want to add, remove and check length of the list

  Scenario: one element on empty list
    Given I have an empty list
    When a "bar" string is appended to the list
    Then the length of the list is now 1

  Scenario: two elements on empty list
    Given I have an empty list
    When a "foo" string is appended to the list
    And a "bar" string is appended to the list
    Then the length of the list is now 2

  Scenario: add and remove element
    Given I have an empty list
    When a "foo" string is appended to the list
    And the last element of the list is removed
    Then the length of the list is now 0

  Scenario: dates on the list
    Given I have an empty list
    When a 11:00 time is appended to the list
    And a 12:00 time is appended to the list
    And the last element of the list is removed
    Then the all elements of the list are times earlier than 11:30
```

```
$ nosetests --with-yanc --with-freshen
UUUU
----------------------------------------------------------------------
Ran 4 tests in 0.223s

OK (UNDEFINED=4)
$
```

```
$ nosetests --with-yanc --with-freshen -v
Basic operations on the list: one element on empty list ... UNDEFINED: "I have an empty list" # list.feature:7
Basic operations on the list: two elements on empty list ... UNDEFINED: "I have an empty list" # list.feature:12
Basic operations on the list: add and remove element ... UNDEFINED: "I have an empty list" # list.feature:18
Basic operations on the list: dates on the list ... UNDEFINED: "I have an empty list" # list.feature:24

----------------------------------------------------------------------
Ran 4 tests in 0.250s

OK (UNDEFINED=4)
$
```

```
======================================================================
UNDEFINED: Basic operations on the list: add and remove element
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/usr/lib/python2.6/dist-packages/twisted/internet/defer.py", line 893, in _inlineCallbacks
    result = g.send(result)
  File "/usr/lib/pymodules/python2.6/freshen/test/async.py", line 56, in _run_deferred
    result = callback()
  File "/usr/lib/pymodules/python2.6/freshen/test/async.py", line 34, in <lambda>
    steps.append(lambda s=step: self.runStep(s, 3))
  File "/usr/lib/pymodules/python2.6/freshen/test/base.py", line 66, in runStep
    return self.step_runner.run_step(step)
  File "/usr/lib/pymodules/python2.6/freshen/core.py", line 31, in run_step
    step_impl, args = self.step_registry.find_step_impl(step)
  File "/usr/lib/pymodules/python2.6/freshen/stepregistry.py", line 233, in find_step_impl
    raise UndefinedStepImpl(step)
UndefinedStepImpl: "I have an empty list" # list.feature:18

======================================================================
UNDEFINED: Basic operations on the list: dates on the list
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/usr/lib/python2.6/dist-packages/twisted/internet/defer.py", line 893, in _inlineCallbacks
    result = g.send(result)
  File "/usr/lib/pymodules/python2.6/freshen/test/async.py", line 56, in _run_deferred
    result = callback()
  File "/usr/lib/pymodules/python2.6/freshen/test/async.py", line 34, in <lambda>
    steps.append(lambda s=step: self.runStep(s, 3))
  File "/usr/lib/pymodules/python2.6/freshen/test/base.py", line 66, in runStep
    return self.step_runner.run_step(step)
  File "/usr/lib/pymodules/python2.6/freshen/core.py", line 31, in run_step
    step_impl, args = self.step_registry.find_step_impl(step)
  File "/usr/lib/pymodules/python2.6/freshen/stepregistry.py", line 233, in find_step_impl
    raise UndefinedStepImpl(step)
UndefinedStepImpl: "I have an empty list" # list.feature:24

----------------------------------------------------------------------
Ran 4 tests in 0.214s

FAILED (UNDEFINED=4)
$ nosetests --with-yanc --with-freshen --undefined-steps-cause-test-failure
```

```python
from freshen import Given, When, Then, NamedTransform, scc
import datetime

@Given('I have an empty list')
def i_have_an_empty_list():
    scc.i = list()

@When('a "(.*)" string is appended to the list')
def a_string_is_appended_to_the_list(string):
    scc.i.append(string)

@When('the last element of the list is removed')
def the_last_element_of_the_list_is_removed():
    scc.i.pop()

@Then('the length of the list is now ([0-9]+)')
def the_length_of_the_list_is(length):
    assert len(scc.i) == int(length)
```

```
$ nosetests --with-yanc --with-freshen
....
----------------------------------------------------------------------
Ran 4 tests in 0.230s

OK
$ nosetests --with-yanc --with-freshen -v
Basic operations on the list: one element on empty list ... ok
Basic operations on the list: two elements on empty list ... ok
Basic operations on the list: add and remove element ... ok
Basic operations on the list: dates on the list ... ok

----------------------------------------------------------------------
Ran 4 tests in 0.174s

OK
$
```

```python
@NamedTransform('{time}', r'([0-9:]{5})', r'^([0-9:]{5})$')
def transform_time_to_datetime(time_):
    d = datetime.datetime.strptime(time_, '%H:%M')
    return d

@When('a {time} time is appended to the list')
def a_time_is_appended_to_the_list(timee):
    assert isinstance(timee, datetime.datetime)
    scc.i.append(timee)

@Then('all elements of the list are a time earlier than {time}')
def all_elements_of_the_list_are_a_time_earlier(timee):
    for element in scc.i:
        assert element < timee
```

```
Feature: Basic operations on the list
  In order to handle variable number of objects
  As a list user
  I want to add, remove and check length of the list

  Scenario: one element on empty list
    Given I have an empty list
    When a "bar" string is appended to the list
    Then the length of the list is now 1

  Scenario: two elements on empty list
    Given I have an empty list
    When a "foo" string is appended to the list
    And a "bar" string is appended to the list
    Then the length of the list is now 2

  Scenario: add and remove element
    Given I have an empty list
    When a "foo" string is appended to the list
    And the last element of the list is removed
    Then the length of the list is now 0

  Scenario: dates on the list
    Given I have an empty list
    When a 11:00 time is appended to the list
    And a 12:00 time is appended to the list
    And the last element of the list is removed
    Then the all elements of the list are times earlier than 11:30
```

```
Scenario Outline: Add two numbers
  Given I have entered <input_1> into the calculator
  And I have entered <input_2> into the calculator
  When I press <button>
  Then the result should be <output> on the screen

Examples:
  | input_1 | input_2 | button | output |
  | 20      | 30      | add    | 50     |
  | 2       | 5       | add    | 7      |
  | 0       | 40      | add    | 40     |
```

Dlaczego tak?

Dlaczego tak?

Przenośność

Dlaczego tak?

W DSL mogą pisać różni ludzie

Dlaczego tak?

QA

Dlaczego tak?

Bussines  IT