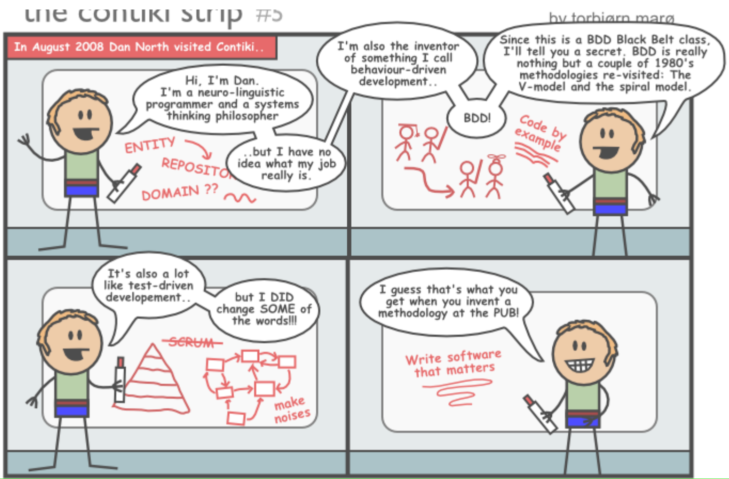
Code testing philosophies

Examples in python



the contiki strip #5
by torbjørn marø
In August 2008 Dan North visited Contiki..
Hi, I'm Dan. I'm a neuro-linguistic programmer and a systems thinking philosopher
ENTITY
REPOSITO
DOMAIN ??
..but I have no idea what my job really is.
I'm also the inventor of something I call behaviour-driven development..
BDD!
Since this is a BDD Black Belt class, I'll tell you a secret. BDD is really nothing but a couple of 1980's methodologies re-visited: The V-model and the spiral model.
Code by example
It's also a lot like test-driven developement..
but I DID change SOME of the words!!!
~~SCRUM~~
make noises
I guess that's what you get when you invent a methodology at the PUB!
Write software that matters

Bas Rustenburg, Chodera Lab @MSKCC

Hot Topics Tuesday , January 20 2015

Table of Contents

Table of Contents

Why do we write tests for code?

Writing tests for code can prevent mistakes, and guide development of software.

Common reasons for writing tests

- Catch bugs introduced during development
- Maintain backwards compatibility
- Verify output of newly implemented function
- Introduction to new developers
- Provide a scaffold for code design

Table of Contents

Development-Driven Development
Developing code, write tests later

Strategy

“My objective is to produce code, not write tests. As long as users follow my example, the code should work fine.”

- Write the code first
- Make sure it produces the output you expect
- Finally, write tests to maintain the code in further development, if time allows it.

Development-Driven Development
Developing code, write tests later

The idea: “My objective is to produce code, not write tests.”

Advantages

- Spend your development time of the part of the software that ultimately matters most
- Programmer is not burdened with maintaining strict software management methodologies

Disadvantages

- Lack of code design and structure
- Inexperienced developers introducing bugs
- Code becomes hard to inherit

Table of Contents

How does it work?

The emphasis is on the structure of the code.

- Instead of units of code, one first writes tests of the unit
- Then, you write a minimal amount of code to make the test pass
- Finally, verify that all tests succeed.
- Connect all the units and you will have the complete program.

Test-Driven Development
Developing code starting with tests

What are the advantages?

- Keeps code concise
- Clear purpose for implemented code
- Tests don't cost extra time to write
- **Y**ou **A**ren't **G**onna **N**eed **I**t.
  - No code written besides the essentials
  - Code stays simple and modular

Table of Contents

Behavior-Driven Development
Developing from the end-users perspective.

What are the key ideas?

- The intended behavior of a program determines the development strategy
- From the start, you decide *what* should be tested.
- Using domain specific language to describe behavior
    - Often close to natural language, with some object-oriented elements
- Allow non-experts to design software and tests
- Have programmers implement them

Table of Contents

Good testing practices

Some things to keep in mind when writing tests.

Attempt to at least:

- Focus on small units of functionality and ensure correctness
- Tests must be able to run independently (see Mock for help)
- Keep them fast! Slowness discourages running tests.
- Write a test for debugging.
- Keep your code clear, keep your tests clearer!

Inspired by: The Hitchhiker's Guide to Python

`https://github.com/kennethreitz/python-guide`

Mock

Making sure that a method was called:

```
# mock_example.py
from mock import Mock

class Myclass(object):
  def closer(self, something):
      something.close()

mycase = Myclass()
mock = Mock()
mycase.closer(mock)
mock.close.assert_called_with()
```

http://www.voidspace.org.uk/python/mock/index.html

Table of Contents

How to test code?
Example python libraries

unittest

A solid testing library that provides many functions to help you test units of code.

```python
# unittest_example.py
import unittest

def my_function(x, y):
    """Subtract y from x"""
    return x - y

class MyTest(unittest.TestCase):
    def test_subtracting(self):
        self.assertEqual(my_function(7, 4), 3)
    def test_subtracting_typerror(self):
        self.assertRaises(TypeError, my_function, [7, '4'])
if __name__ == '__main__':
    unittest.main()
```

https://docs.python.org/2/library/unittest.html

How to test code?
Example python libraries

doctest

Write test in your documentation strings.

```
# doctest_example.py
def my_function(x, y):
    """Subtract y from x
    Examples
    --------
    >>> my_function(7, 4)
    3
    >>> my_function(7, '4')
    Traceback (most recent call last):
    ...
    TypeError: unsupported operand type(s) for -: 'int' and 'str'
    """
    return x - y

if __name__ == "__main__":
    #Execute script as: python doctest_example.py -v
    import doctest
    doctest.testmod()
```

https://docs.python.org/2/library/doctest.html

How to test code?
Example python libraries

nosetests

Adds test discovery on top of available tools.

```
# test_nose_example.py
def my_function(x, y):
    """Subtract y from x"""
    return x - y

def test_my_function():
    assert my_function(7,4) == 3
```

Try commands such as:

- nosetests
- nosetests unittest_example.py
- nosetests --with-doctest

https://nose.readthedocs.org/en/latest

Cucumber and Behave
Examples of Behavior-Driven Development tools

What are they all about?

- Cucumber (Ruby), Behave(Python), Specflow (.NET) et al.
- The behavior of the program is written down in a business readable, domain specific language called *Gherkin*
- This is then parsed and matched to functions called *step functions*
- The step functions are written in actual programming languages
- They test other underlying code and take input from the Gherkin parser

`https://github.com/cucumber/cucumber/wiki/Gherkin`

Gherkin syntax

A simple example

```
# example.feature
Feature: An example of behavior driven design

Scenario: I want to subtract two integers
Given our subtractor is installed
When I subtract 4 from 7
Then the result should be 3

Scenario: I want subtract floats
Given our subtractor is installed
When I subtract 3.5 from 7.0
Then it should return 3.5 within 0.0001 error
```

Step methods
The underlying python code

A simple example

```
# Example steps
from behave import given, when, then

# Given our subtractor is installed
@given('our subtractor is installed')
def step_impl(context):
    pass
# When I subtract 4 from 7
@when('I subtract {y} from {x}')
def step_impl(context,y,x):
    if '.' in x or '.' in y:
      x = float(x)
      y = float(y)
    else:
      x = int(x)
      y = int(y)

    context.result = x - y
# Then the result should be 3
@then('the result should be {result}')
def step_impl(context, result):
    if '.' in result:
      result = float(result)
    else:
      result = int(result)
    assert (context.result - result) <= 0.0001
```

Useful sources of information
..the wiki is not enough..

Urls

- Hitchhiker's Guide to Python : `https://github.com/kennethreitz/python-guide`
- doctest, unittest: `https://docs.python.org/2/library/development.html`
- nose: `https://nose.readthedocs.org`
- mock: `http://www.voidspace.org.uk/python/mock/mock.html`
- behave: `http://pythonhosted.org/behave/`
- cucumber: `http://cukes.info`