pytest - Framework for Unit Testing

Applying test-driven development within Ansys

Alex Kaszynski
Jorge Martinez Garrido

November 9, 2022

Ansys

Table of Contents

1. Introduction
2. Overview of pytest
    - Basic assertions with pytest
    - Array assertions
    - Marking tests
    - Parameterizing tests
    - Plugins
    - Continuous integration and development
3. Where to find help

Introduction

Ansys

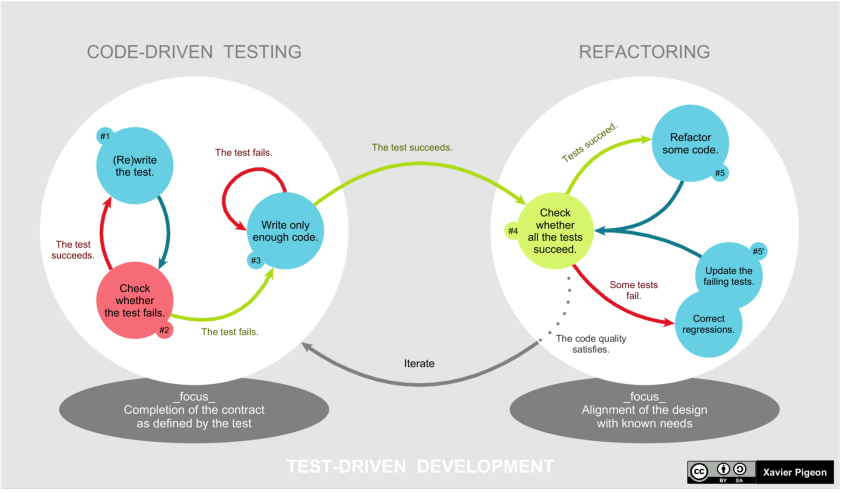
Introduction

- **What is test-driven development (TDD)?**
  Software requirements are converted to test cases before software is fully developed.
  As soon as the code base gets modified, a collection of tests get executed to guarantee the integrity of the new code.

- **What are its main advantages?**
  - Improved design of the system.
  - Better code quality and flexibility.
  - Good documentation as a byproduct.
  - Lower development costs and increased developer productivity.

- **pytest is a and powerful and mature testing framework widely used in Python.**

Test Driven Development - Overview



Overview of pytest

Ansys

Content

1. Introduction
2. Overview of pytest
    - Basic assertions with pytest
    - Array assertions
    - Marking tests
    - Parameterizing tests
    - Plugins
    - Continuous integration and development
3. Where to find help

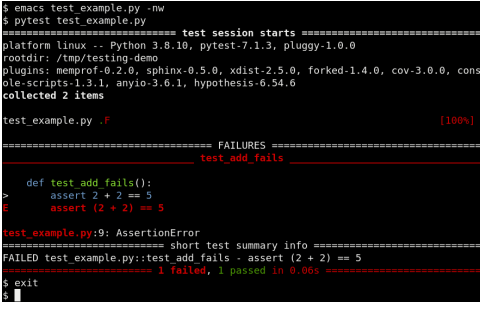
Basic assertions with pytest

- Tests are defined using functions starting with `test`
- Use the assert statement to compare current and expected results.
- Run your tests by invoking pytest.

Basic Assertion

```python
# content of test_sample.py
def inc(x):
    return x + 1


def test_answer():
    assert inc(3) == 5
```



Using NumPy for Floating Point Assertions

- For floating point values, absolute and relative tolerances are required.
- NumPy provides the assert_allclose function for this purpose.
- You can also use assert_equal to check if two arrays are exactly the same.

Array Assertions

```
>>> x = [1e-5, 1e-3, 1e-1]
>>> y = np.arccos(np.cos(x))
>>> np.testing.assert_allclose(
...      x, y, rtol=1e-5, atol=0,
... )

>>> np.testing.assert_array_equal(
...      [1.0, 2.33333, np.nan],
...      [np.exp(0), 2.33333, np.nan],
... )
```

```
collected 2 items

test_array.py .F                                                    [100%]

================================== FAILURES ===================================
_____________________________ test_arccos_failed ______________________________

    def test_arccos_failed():
>       assert_allclose(x, y, rtol=1e-9, atol=0)
E       AssertionError:
E       Not equal to tolerance rtol=1e-09, atol=0
E
E       Mismatched elements: 1 / 3 (33.3%)
E       Max absolute difference: 4.13743513e-13
E       Max relative difference: 4.13743495e-08
E        x: array([1.e-05, 1.e-03, 1.e-01])
E        y: array([1.e-05, 1.e-03, 1.e-01])

test_array.py:14: AssertionError
----------------------------- memory consumption estimates -----------------------------
test_array.py::test_arccos_failed  - 252.0 KB
--------------------------- short test summary info ----------------------------
FAILED test_array.py::test_arccos_failed - AssertionError:
========================== 1 failed, 1 passed in 0.10s ==========================
```

Marking Tests

- Marking tests allows to filter and select desired tests from the test suite.
- Marking is performed using the `pytest.mark.<name>` decorator.
- Select marked tests by executing `pytest -m <name> -vv tests`

pytest.mark.<name>

```python
from time import sleep
import pytest

@pytest.mark.slow
def test_a_very_slow_logic():
    sleep(10)

@pytest.mark.fast
def test_a_very_fast_test():
    ...
```

```
  rootdir: /tmp/testing-demo

$ pytest -m "not fast" --durations 0 -vv
============================= test session starts ==============================
platform linux -- Python 3.8.10, pytest-7.1.3, pluggy-1.0.0 -- /home/akaszyns/py
thon/py38/bin/python
cachedir: .pytest_cache
hypothesis profile 'default' -> database=DirectoryBasedExampleDatabase('/tmp/tes
ting-demo/.hypothesis/examples')
rootdir: /tmp/testing-demo, configfile: pytest.ini
plugins: memprof-0.2.0, sphinx-0.5.0, xdist-2.5.0, forked-1.4.0, cov-3.0.0, cons
ole-scripts-1.3.1, anyio-3.6.1, hypothesis-6.54.6
collected 2 items / 1 deselected / 1 selected

test_demo.py::test_talk_to_network PASSED                                [100%]
======================== memory consumption estimates ========================
test_demo.py::test_talk_to_network  - 1.0 MB

============================= slowest durations ==============================
0.03s call     test_demo.py::test_talk_to_network
0.00s setup    test_demo.py::test_talk_to_network
0.00s teardown test_demo.py::test_talk_to_network
====================== 1 passed, 1 deselected in 0.15s =======================
$ exit
```

Parametrizing tests

- Parameterizing a allows you to run the same test with different inputs.
- This allows you to reuse test code and test a variety of inputs.
- Using the pytest.mark.parametrize decorator.

pytest.mark.parametrize

```python
from numpy.testing import assert_allclose
import pytest

@pytest.mark.parametrize(
    "a, b, c",
    [(0.1, 0.2, 0.3), (8, 12, 20)]
)
def test_add_two_floats(a, b, c):
    assert a + b == c
```

```
.1, anyio-3.6.1, hypothesis-6.54.6
collected 2 items

test_para.py::test_raises[1-0-ZeroDivisionError] PASSED                  [ 50%]
test_para.py::test_raises[mystr-1-TypeError] PASSED                      [100%]

============================== 2 passed in 0.09s ===============================
$ emacs test_para.py -nw
$ pytest test_para.py -k two_ints
============================= test session starts ==============================
platform linux -- Python 3.8.10, pytest-7.1.3, pluggy-1.0.0
rootdir: /tmp/testing-demo
plugins: sphinx-0.5.0, xdist-2.5.0, forked-1.4.0, cov-3.0.0, console-scripts-1.3
.1, anyio-3.6.1, hypothesis-6.54.6
collected 402 items / 2 deselected / 400 selected

test_para.py ..................................................................... [ 14%]
..................................................................... [ 32%]
..................................................................... [ 50%]
..................................................................... [ 68%]
..................................................................... [ 86%]
.....................................................                [100%]

====================== 400 passed, 2 deselected in 0.36s =======================
```

A plethora of plugins

Plugins add or modify the behavior of `pytest`:

pytest-xdist

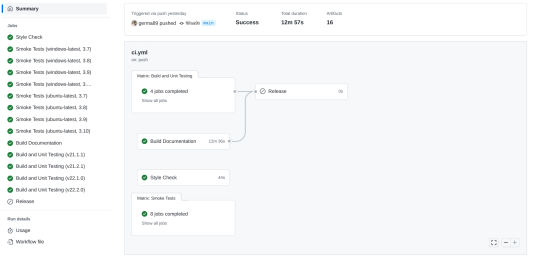
```
$ pip install pytest-xdist
$ pytest -n auto
```

- `pytest` integrates with the unittest module from Python Standard Library.
- Validate documentation examples with doctests.
- Check code coverage with pytest-cov
- Include Matplotlib testing with pytest-mpl
- Find edge cases with hypothesis

Check out the official list at pytest - Plugin List.

Continuous integration and development

- At Ansys, Continuous integration and development (CI/CD) takes place in ADO or GitHub.
- It is performed using workflow and configured via YAML files and you can test across a variety of environments using a matrix workflow.
- Automate testing using `pytest` across multiple OSes and platforms.
- Integrate with other third party apps like codecov to provide metrics and insights.



workflow.yaml

```yaml
testimport:
  name: Smoke Tests
  runs-on: ${{ matrix.os }}
  strategy:
    matrix:
      os: [windows-latest, ubuntu-latest]
      python-version: ['3.7', '3.8', '3.9', '3.10']
```

Where to find help

Ansys

Content

1. Introduction
2. Overview of pytest
    - Basic assertions with pytest
    - Array assertions
    - Marking tests
    - Parameterizing tests
    - Plugins
    - Continuous integration and development
3. Where to find help

Additional Information

Visit the PyAnsys Developer's Guide for more information on how to implement testing.

PyAnsys Dev Guide - testing
https://dev.docs.pyansys.com/how-to/testing.html

PyAnsys Dev Guide - Continuous Integration
https://dev.docs.pyansys.com/how-to/continuous-integration.html

pytest - Documentation
https://docs.pytest.org/en/latest/


Ansys