# Spojenie py.test, coverage.py a ast.py, ktoré posúva TDD vývoj na novú úroveň


Tibor Arpas
tibor@infinit.sk

# Tibor Arpáš - o mne

- freelancer
- ale nie výhradne
- www.infinit.sk
- Python a nič iné od roku 2007

# O vás

# Testovanie softvéru

- ručné
  - intuitívne, bežné
  - pri každom opakovaní rovnako prácne
  - nepresné

# Testovanie softvéru

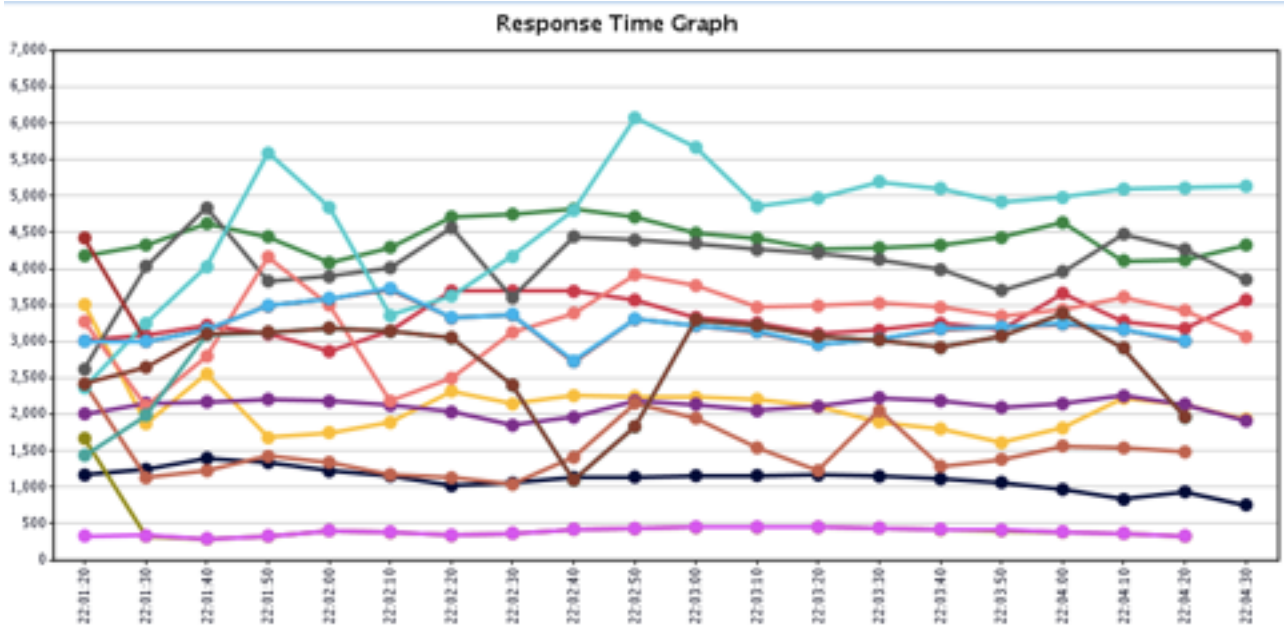
- automatizované
  - nutná investícia
  - rýchle
  - presné
  - test runner nutný pre väčšie projekty

# Užívatelia nenávidia čakanie

- 100 milisekúnd
- 1 sekunda
- 10 sekúnd

# Takže computing sa prispôsobil

- veľké zlepšenie v posledných rokoch



# Spúšťanie sady testov


EXECUTING TEST SUITE..
waiting for reply...

# Návrat mainframe-ov?

- vytvoriť iteráciu vstupov
- čakať hodiny
- dostať výsledok
- znova

# Ale ...

- čo tak spúšťať iba relevantné testy?

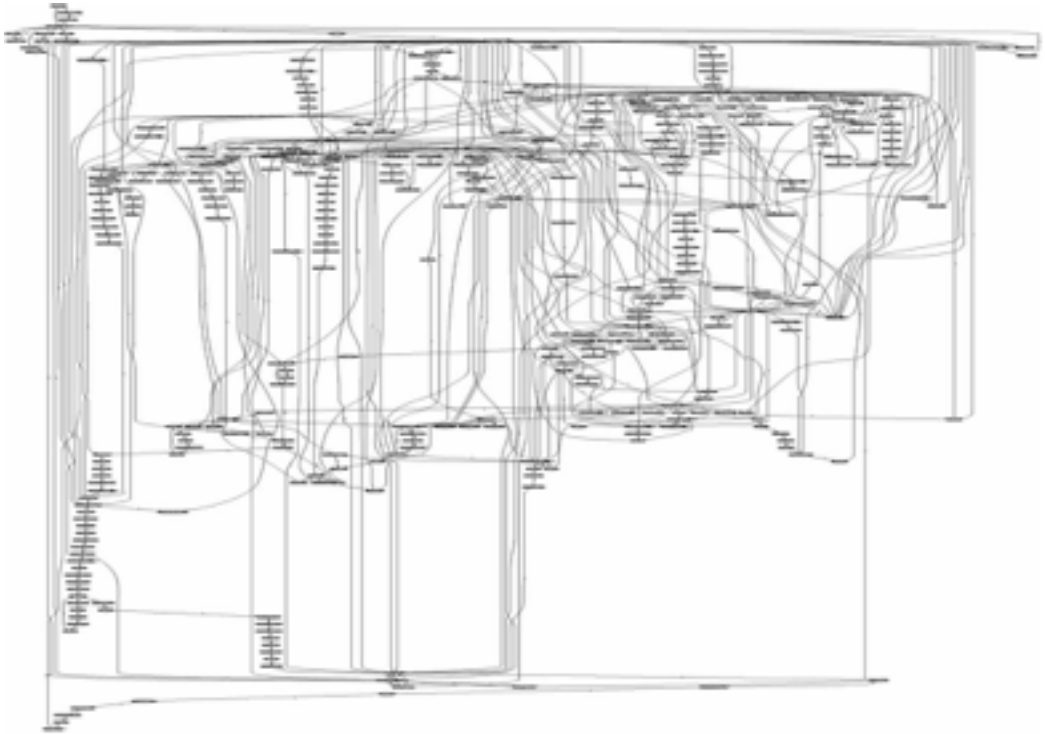
# Väčšina zmien kódu je lokálnych

- Spúšťať všetko je plytvaním

# Ručné vyberanie testov

- ako často vyberieš správne?

# Čo ak závislosti v projekte vyzerajú takto?



# Príklad “projektu” so sadou testov v jednom súbore.

test_s.py

```
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def test_add():
    assert add(1, 2) == 3

def test_subtract():
    assert subtract(4, 3) == 1

def test_both():
    assert add(1, 3) == 4
    assert subtract (2, 1) == 1
```

# Závislosti

| **method**<br>**test** | add | subtract |
|---|---|---|
| ::test_add | x | |
| ::test_subtract | | x |
| ::test_both | x | x |

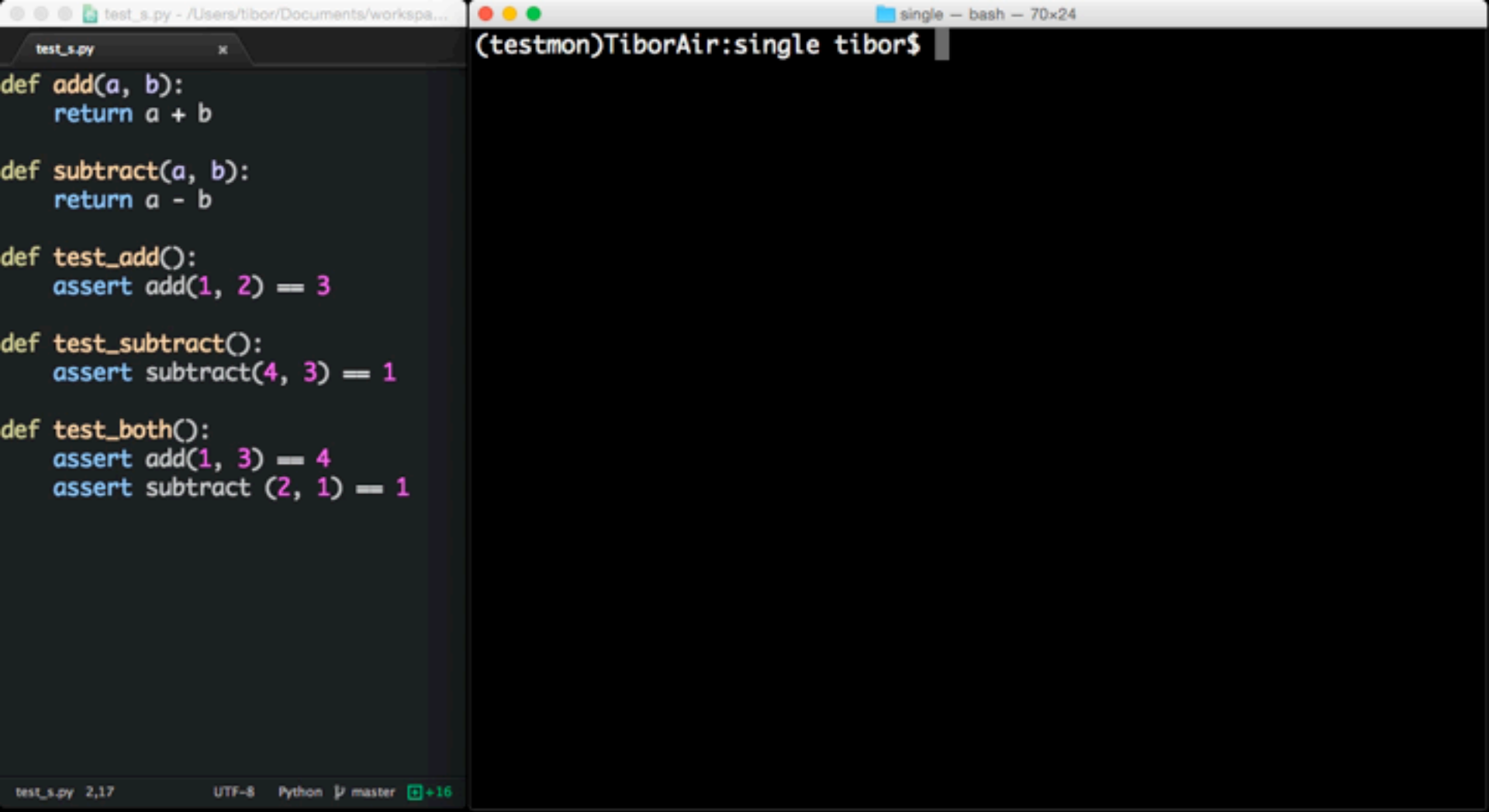
# subtract vs. test_add

test_s.py

```
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def test_add():
    assert add(1, 2) == 3

def test_subtract():
    assert subtract(4, 3) == 1

def test_both():
    assert add(1, 3) == 4
    assert subtract (2, 1) == 1
```

# Závislosti

| **method**<br>**test** | add | subtract |
|---|---|---|
| ::test_add | x | |
| ::test_subtract | | x |
| ::test_both | x | x |


test_s.py - /Users/tibor/Documents/workspa...
test_s.py
def add(a, b):
return a + b
def subtract(a, b):
return a - b
def test_add():
assert add(1, 2) == 3
def test_subtract():
assert subtract(4, 3) == 1
def test_both():
assert add(1, 3) == 4
assert subtract (2, 1) == 1
test_s.py 2,17
UTF-8
Python
master
+16
single — bash — 70×24
(testmon)TiborAir:single tibor$

# Nápad transformovaný do nástroja: testmon.org

## pytest-testmon

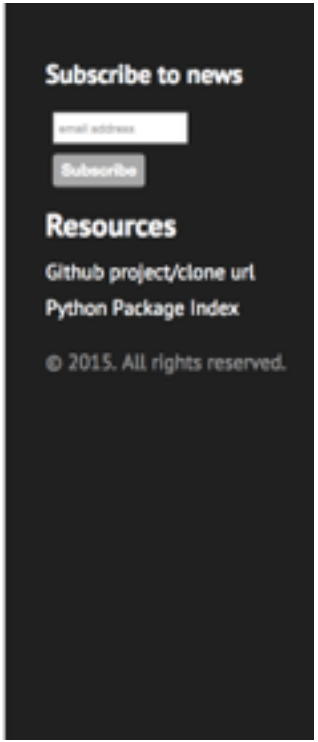
build passing

This is a py.test plug-in which automatically selects and re-executes only tests affected by recent changes. How is this possible in dynamic language like Python and how reliable is it? Read here: Determining affected tests

New versions usually have new dataformat, don't forget to rm .testmondata after each upgrade.

### Usage

```
pip install pytest-testmon

# build the dependency database and save it to .testm
py.test --testmon

# list of watched project files ordered by tests whic
py.test --by-test-count

# change some of your code (with test coverage)

# only run tests affected by recent changes
py.test --testmon
```



Subscribe to news

email address

Subscribe

Resources

Github project/clone url

Python Package Index

© 2015. All rights reserved.

# coverage.py

```
..
coverage.start()
code_to_track()
coverage.stop()

coverage.get_data()
..
```

Ned Batchelder : Blog | Code | Text | Site

**coverage.py**

» Home : Code

*Created 24 May 2009, last updated 12 December 2013*

Coverage.py is a tool for measuring code coverage of Python programs. It monitors your program, noting which parts of the code have been executed, then analyzes the source to identify code that could have been executed but was not.

Coverage measurement is typically used to gauge the effectiveness of tests. It can show which parts of your code are being exercised by tests, and which are not.

The latest version is coverage.py 3.7.1, released 13 December 2013. It is supported on Python versions 2.3 through 3.4, and PyPy 2.1.

**Quick start**

Getting started is easy:

1. Install coverage.py from the coverage page on the Python Package Index, or by using "pip install coverage". For a few more details, see Installation.
2. Use `coverage run` to run your program and gather data:

# ast.py (standard library)

```
import ast

tree = ast.parse(source_code, file_name)
```

príklady ktoré mi pomohli porozumieť:

- https://github.com/mitsuhiko/pyastutil -> codegen.py
- ast.dump

# py.test plug-in

- plugin API py.test -u je pekné
- testmon je relatívne málo riadkov kódu

# Zaujímavý aspekt - crowdfunding

Story | Updates 4 | Comments 1 | Funders 25

testmon
for Python

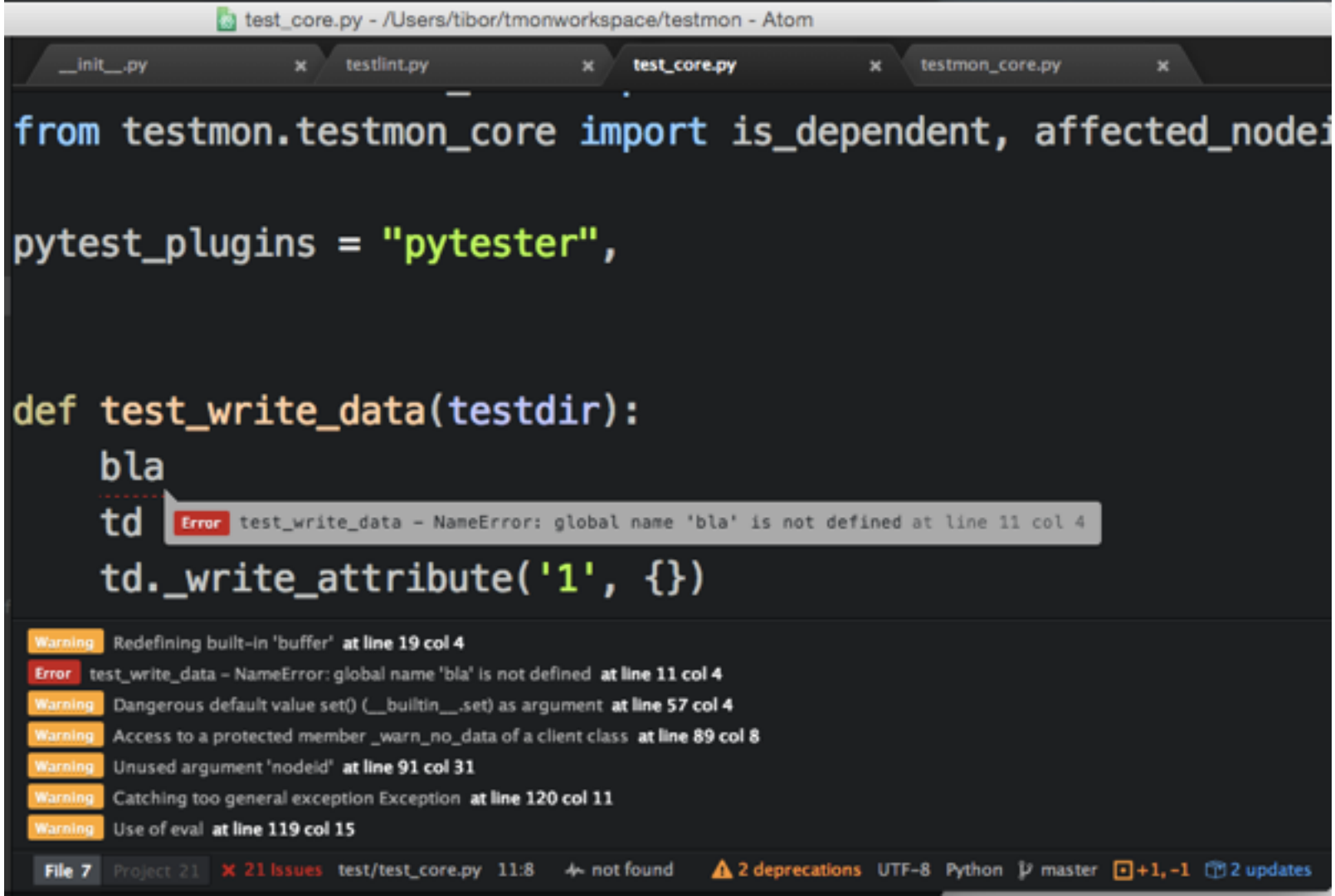
Unit
Integration
Functional
Acceptance
Test
Runner

**€1,424** EUR
raised in 2 months

102% funded — No time left

€1,400 EUR goal
Fixed Funding

**CAMPAIGN CLOSED**
This campaign ended on March 19, 2015


test_core.py - /Users/tibor/tmonworkspace/testmon - Atom
__init__.py
testlint.py
test_core.py
testmon_core.py
from testmon.testmon_core import is_dependent, affected_nodei
pytest_plugins = "pytester",
def test_write_data(testdir):
bla
td
Error test_write_data – NameError: global name 'bla' is not defined at line 11 col 4
td._write_attribute('1', {})
Warning Redefining built-in 'buffer' at line 19 col 4
Error test_write_data – NameError: global name 'bla' is not defined at line 11 col 4
Warning Dangerous default value set() (__builtin__.set) as argument at line 57 col 4
Warning Access to a protected member _warn_no_data of a client class at line 89 col 8
Warning Unused argument 'nodeid' at line 91 col 31
Warning Catching too general exception Exception at line 120 col 11
Warning Use of eval at line 119 col 15
File 7
Project 21
21 Issues
test/test_core.py
11:8
not found
2 deprecations
UTF-8
Python
master
+1, -1
2 updates

# záver

- používajte
- dajte feedback
- zdieľajte

# Kontakt znova

Tibor
tibor@infinit.sk

Questions?