Unit testing
- better code faster

Greg Detre

greg@gregdetre.co.uk
@gregdetre

memrise

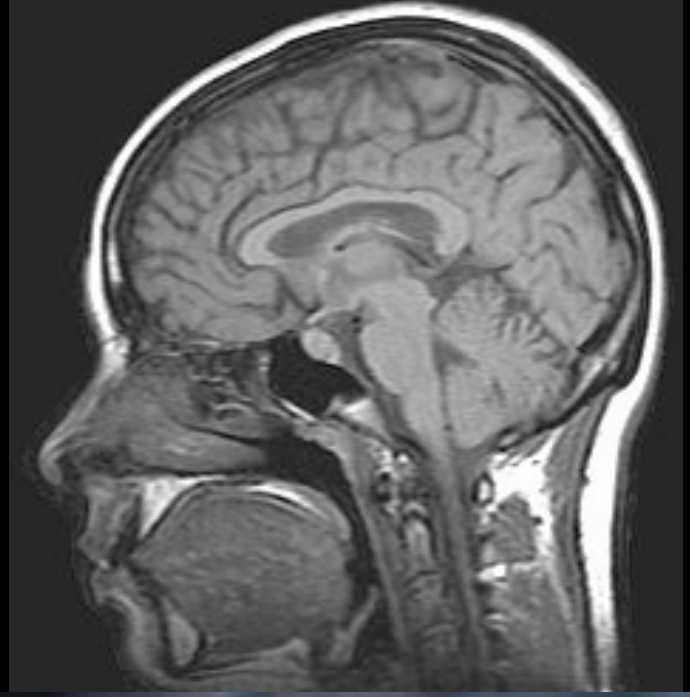
theguardian

BarCamp Tampa 2015
17th October, 2015

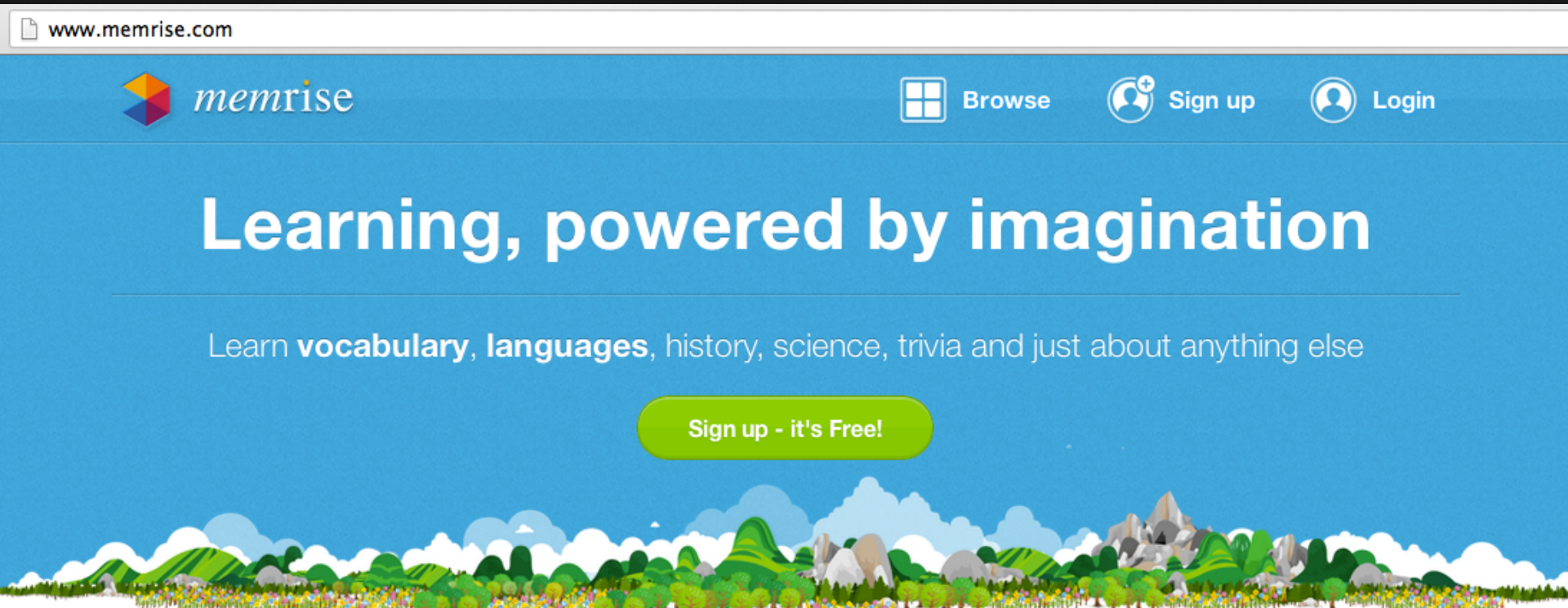
github.com/gregdetre/unit-testing-pres

github.com/gregdetre/unit-testing-pres

ME



memrise


www.memrise.com
memrise
Browse
Sign up
Login
Learning, powered by imagination
Learn vocabulary, languages, history, science, trivia and just about anything else
Sign up - it's Free!


g

Sleepio


O2-UK
10:46
SLEEP DIARY
AUGUST 2014
Tue 12
Wed 13
Thu 14
44%
70%
78%
10hrs
5hrs
0hrs
"I'm not going to sleep a wink"

WHAT IS UNIT TESTING?

What is unit testing?

Take the smallest piece of testable software in the application, isolate it from the remainder of the code, and determine whether it behaves exactly as you expect.

- msdn.microsoft.com/en-us/library/aa292197(v=vs.71).aspx

e.g. if I call this function with input X, I expect to get output Y back

```
def adds2(x):
    return x + 2

def test_adds2_basic():
    assert adds2(10) == 12
```

e.g. if I call this function with input X, I expect to get output Y back

`assert` means ‘I expect’

You already test manually as you're writing code.

But that's inefficient.

SETUP

Docs (including this presentation)

github.com/gregdetre/unit-testing-pres/

- tell Greg your GitHub account name and I'll give you commit access or submit pull requests
- follow the README.md *Setup* instructions

Python unit testing frameworks

unittest - standard library

nose - just like unittest, but nicer

```
$ pip install nose2
$ nose2
```

plugins, e.g.

- colored output
- autodiscovery
- coverage
- debug on error

INTERACTIVE

Goal

to write a function that identifies all and only legitimate email addresses

for now, don't look online

```
def isitanemail(e):
    return '@' in e
```

Hands up if you've written a unit test before

Next steps?

- Define exceptions, then test & build them
- Completely rewrite isitanemail() while making tests pass - open book
- Modularize - check TLD, check illegal characters
- Black vs white box

BENEFITS

Find more bugs earlier (and more cheaply)

*Shull et al (2002) estimate that non-severe defects take approximately **14 hours of debugging effort after release**, but **only 7.4 hours before release .***

Develop faster

Manually testing as you go is slow and incomplete

Most development time is spent debugging. Unit tests take more time up front, but help you debug much faster

- Help you isolate what is and is not working.
- You make a change that fixes the issue you were thinking about, but your tests highlight that you've inadvertently created a problem elsewhere.
- Your tests pause the debugger exactly where the problem is.
- Find new bugs when you introduce them (when it's easy to fix), not months later
- Help realize if the error is elsewhere from where it manifests

Easy to change/refactor code confidently

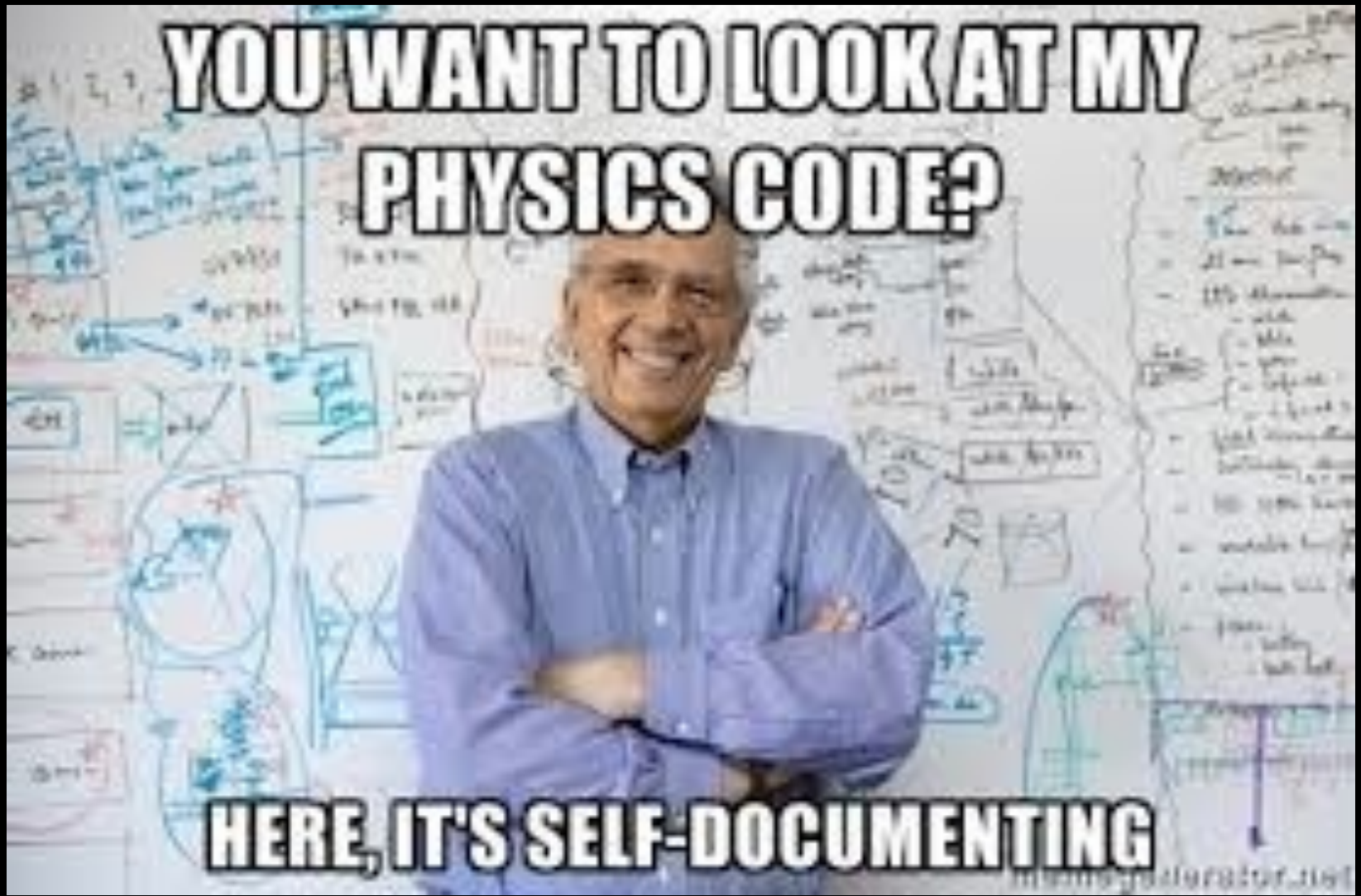
You know when you're done

Write better code

Writing tests encourages you to:

- Reduce dependencies
- Think about the interface
- Minimise entrances and complexity

Self-documenting



Benefits of testing

Find more bugs earlier and more cheaply

Develop faster

Write better code

Self-documenting

Guard against new bugs in old code

Integrate with others (API, in a team)

More predictable progress

Feel confident you've done a good job

WHAT MAKES A GOOD UNIT TEST?

What makes a bad unit test?

```
def test_isitanemail_ints:
     for num in range(1, 1000000000):
          # e.g. ‘123@example.com’
          e = ‘%i@example.com’ % num
          assert isitanemail(e)
```

```
# test_raw_input.py
def test_isitanemail_ask():
    e = ‘x@example.com’
    print ‘Returned for %s:’ % e, isitanemail(e)
    assert raw_input(‘Ok?’) == ‘y’
```

```
from random import sample

def randstr():
     # e.g. ‘a1+’, ‘c3b’, ‘!.!’
     return ‘’.join(sample(‘abc123#+!.’, 3))

def test_isitanemail_rand():
     e = ‘%s@%s.%s’ % (randstr(), \
                       randstr(), \
                       randstr())
     assert isitanemail(e)
```

```
from blah.dns import validate_dns
from your.database import query_user_by_email

def isitanemail(e):
     if ‘@‘ not in e:
          return False
     if not validate_dns(e):
          return False
     if not query_user_by_email(e):
          return False
```

An ideal unit test

Fully automated

Focus on a single, minimal ‘unit’

Readable, concrete

Independent

- from other tests/full system (see ‘mocks’)

Fast

Consistent

Comprehensive

Tips

- Concrete and easy to understand
  - DAMP not DRY -"Descriptive and Meaningful Phrases”
  - Implement the test a different way from the original function
- Write your tests early in the development cycle
- Keep them up to date, and always passing
- Run tests with every deploy
- Make sure your test fails before it passes
- Test a representative sample

TEST-DRIVEN DEVELOPMENT

Test-driven development

Think about your interface.

Write stub functions.

Write tests against those stub functions. They'll all fail.

Slowly fill out the stubs until your tests pass.

You're done!

CHOOSE YOUR OWN ADVENTURE

Other bug-finding techniques (code review, QA etc)

Data, algorithm and analysis testing

Testing performance

When is it hard to unit test?

More interactive

Nose plugins…

AS WELL AS TESTING

Combine software testing with other techniques

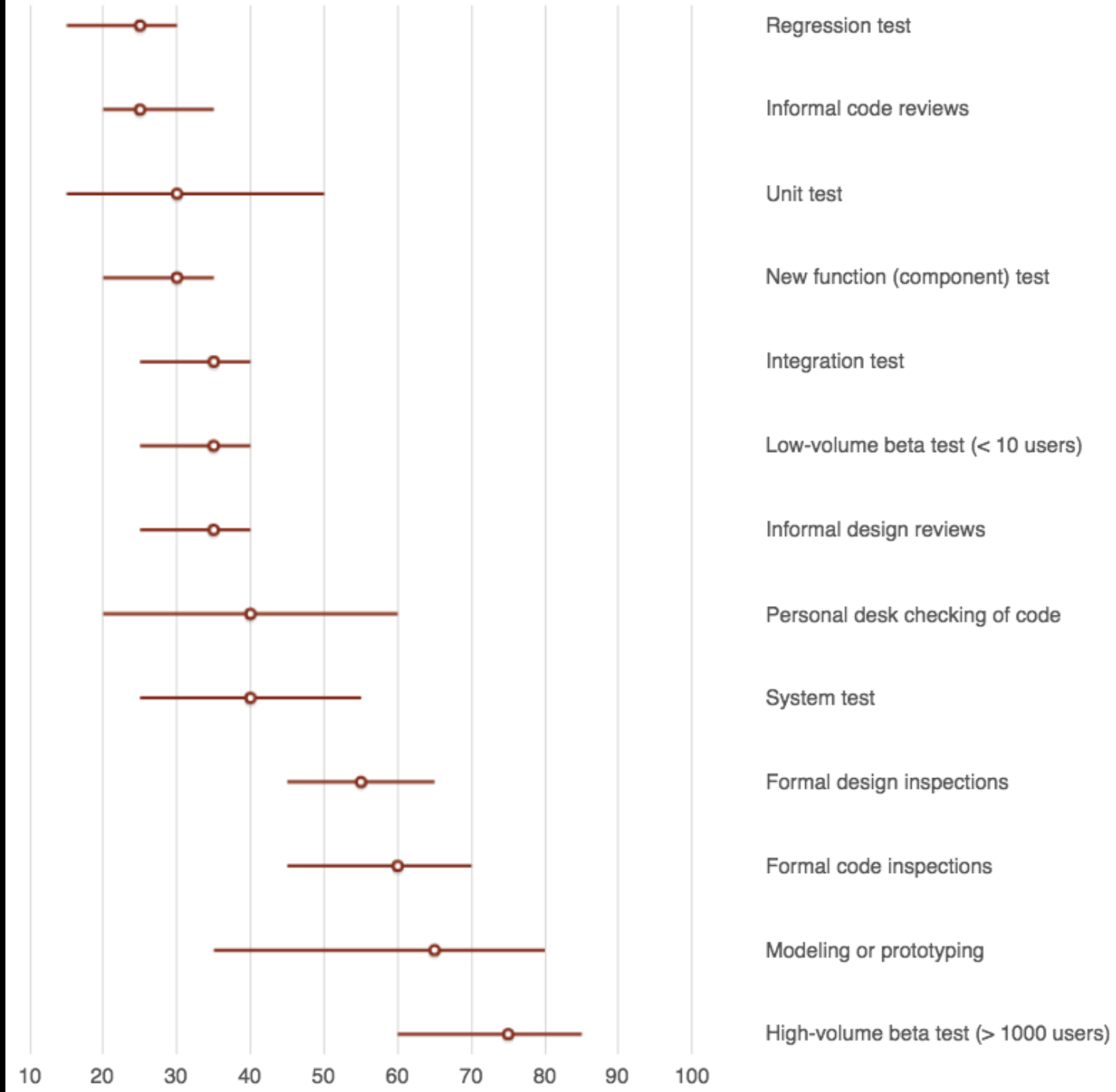
Automated testing finds a certain proportion (<40%) and type of bugs

If you want to ship high quality code, you should invest in more than one of formal code review, design inspection, testing, and quality assurance.

… according to Basili and Selby (1987), code reading detected 80 percent more faults per hour than testing…

https://kev.inburke.com/kevin/the-best-ways-to-find-bugs-in-your-code/

Consider code reviews, QA, beta testing, pair programming, and dog-fooding your own product.


Regression test
Informal code reviews
Unit test
New function (component) test
Integration test
Low-volume beta test (< 10 users)
Informal design reviews
Personal desk checking of code
System test
Formal design inspections
Formal code inspections
Modeling or prototyping
High-volume beta test (> 1000 users)
10
20
30
40
50
60
70
80
90
100

Different kinds of automated testing

unit vs integration tests

DATA & ALGORITHMS

How do you eat an elephant?

Validate on small data, build up

- Define your metric
- Run it on small data (quick, while prototyping)
- Show that you get better as you add more data

Fake data

Generate data that looks exactly the way you expect

Can be hard to do, but often helps you think things through

Confirm that the output looks as it should

Useful for orienting audience in presentations

Nonsense/scrambled data

Set a trap. Feed your algorithm nonsense data. It had better tell you the results aren't significant!

Easy: shuffle labels or feed in random numbers as data

This. Will Save. Your. Bacon.

e.g. guard against peeking

compare against ground truth if you have one

e.g. previous implementation, simpler version of the algorithm, doing it once by hand on real data

Defensive coding

- Pepper your code with asserts and sanity checks
  - e.g. confirm the dimensions, range of values, type of values
- Fail immediately if things are wrong
  - that way you'll notice early on in time and near to the cause of the problem
  - rather than 2 weeks later and in a downstream part of the analysis

WHAT ABOUT...?

What about…

- If the interface is changing very fast?
- If most of the work is being done by an external library?
- If the hard part is in the integration, not the pieces?
- If it requires a lot of infrastructure to be in place?
- If I’m in a really big hurry?

Which of these is the odd one out?

PERFORMANCE

Will this function/ algorithm/query scale?

measure

    time1 = running on some small N

    time2 = running on 100N

    coefficient = float(time2) / time1

assert(coefficient < 200)

THE END

APPENDIX

Resources

https://docs.python.org/2/library/unittest.html
http://nose2.readthedocs.org/en/latest/index.html

Mark Pilgrim's (free) Dive Into Python chapters 13 and 14 on unit testing

- http://www.diveintopython.net/unit_testing/index.html#roman.intro

http://nedbatchelder.com/text/test0.html

http://kev.inburke.com/kevin/the-best-ways-to-find-bugs-in-your-code/

back to Benefits