How to make the most of software testing

Greg Detre

greg@gregdetre.co.uk
@gregdetre

memrise

theguardian

The Tampa Bay Python Meetup Group
13th October, 2015

github.com/gregdetre/unit-testing-pres

SETUP

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

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

Google Hangout

http://bit.ly/1GELH73

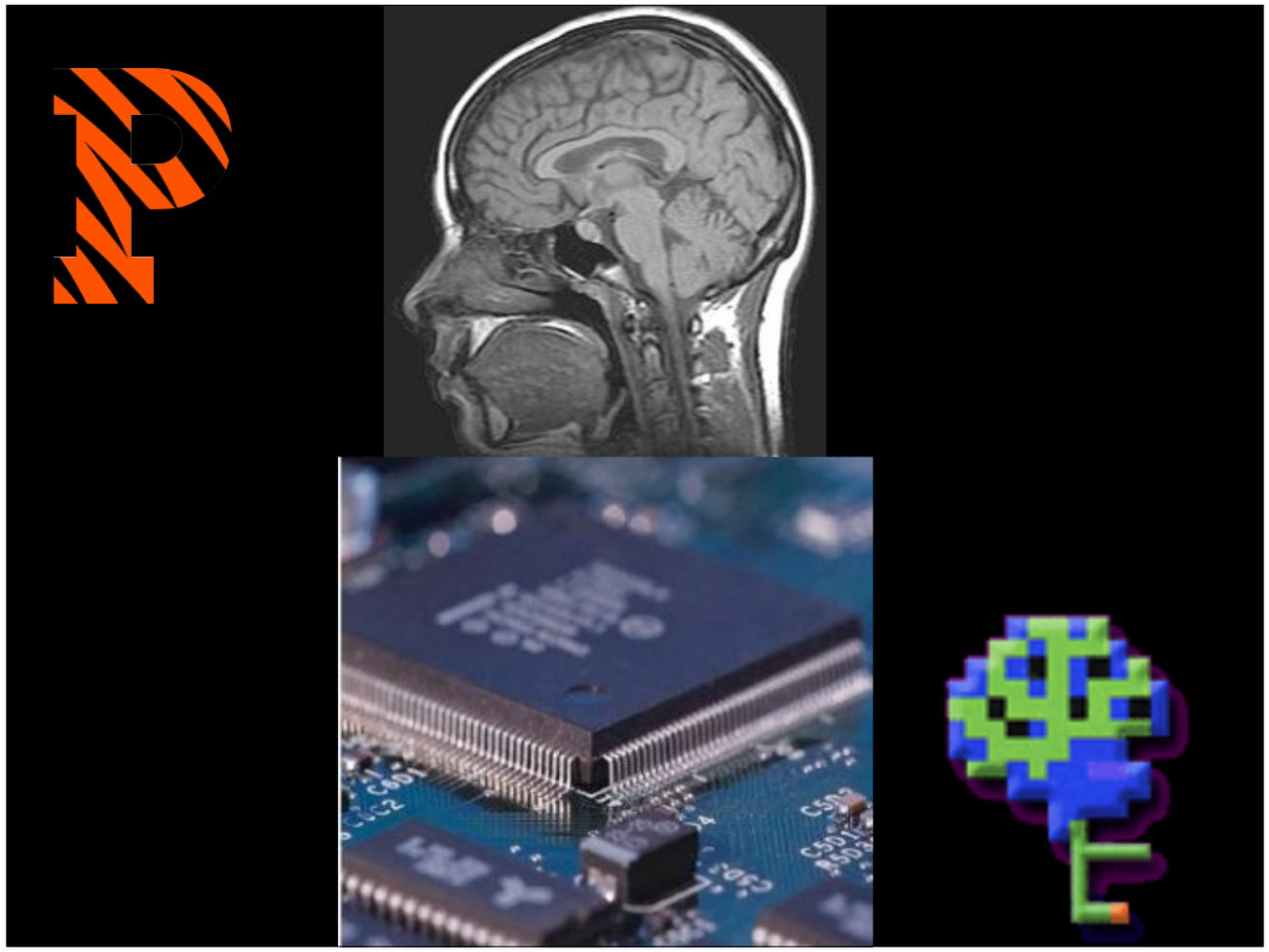
- join in, share your screen, turn off speakers, mute

```
# in test_template0.py
def test_blah():
    assert True


$ nose2 test_template0
```

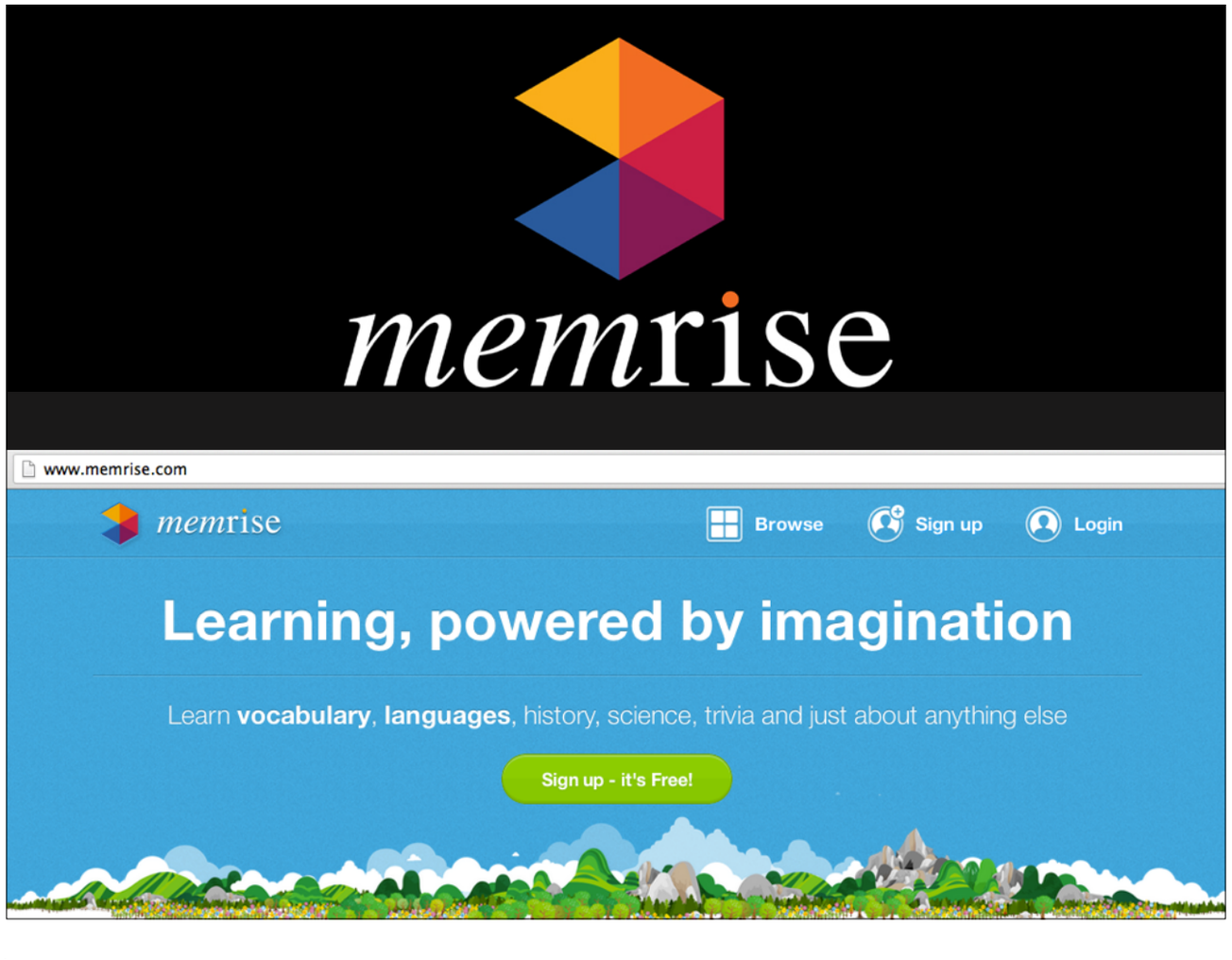
github.com/gregdetre/unit-testing-pres

ME



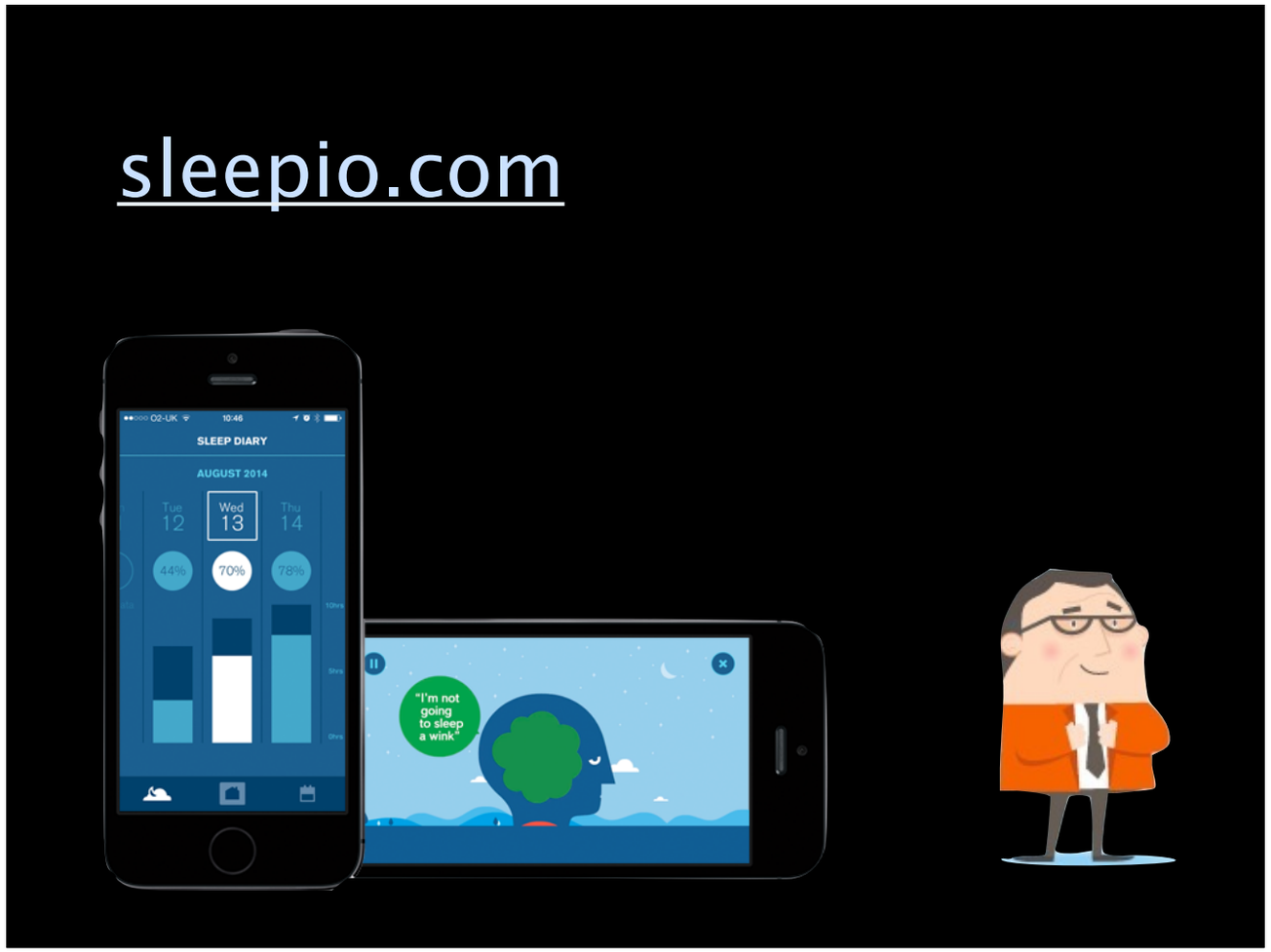
did my PhD work at Princeton in with Ken Norman in the Computational Memory Lab
I'm Greg Detre

has a PhD in the neuroscience of human memory and forgetting at Princeton
scan people's brains, including my own - it turned out to be smaller than I'd hoped



after grad school, I started a company called Memrise

Led the brand-new data science team at The Guardian



One more caveat - I haven't been working in the trenches as an engineer for the last year, and I'm very rusty on some things. I hope you'll help me along if I get stuck on something obvious!

INTRO

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.

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

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

You already test manually as you're writing code.

But that's inefficient.

EXAMPLE

Is this a telephone number?

```
# in tests.py
def test_blah(self):
    assert True

$ nose2
```

BENEFITS

Benefits of testing

Find more bugs earlier and more cheaply
Write better code
Develop faster
Easier to change
Understand what the code does
Guard against new bugs in old code
Integrate with others (API, in a team)
Feel confident

Helps you structure your code – if it's easy to test, it'll be easy to understand and refactor

Easier to read than code, how-to guide and expected behavior

Run your unit tests every time you deploy. Otherwise you might break something that used to work, and not realize it

Confidence – there may be bugs. But there are no easy bugs. No new bugs. If you have good tests, you might even be somewhat confident there are no bugs.

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 time-intensive

Most development time is spent debugging. Unit tests help you debug much faster

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

BASIC TECHNIQUES

An ideal unit test

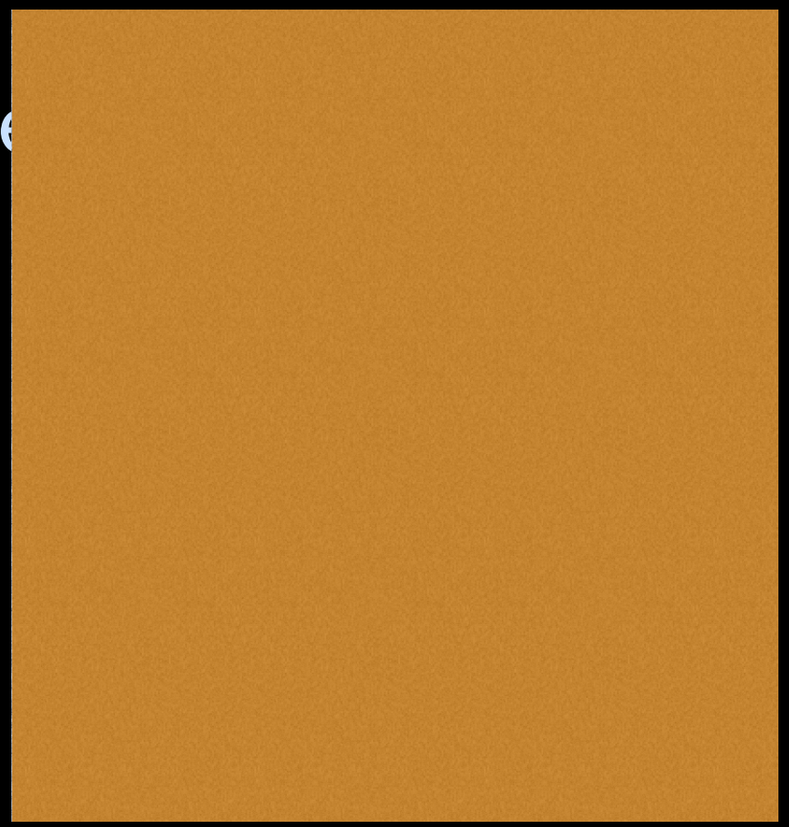
Fully automated
Tests a single logical concept in the system
Readable, concrete, trustworthy
Independent from other tests/full system
Runs fast
Consistently returns the same result?

see http://artofunittesting.com/definition-of-a-unit-test/

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

A good te



ADVANCED TECHNIQUES

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?

AS WELL AS TESTING

Combine software testing with other techniques

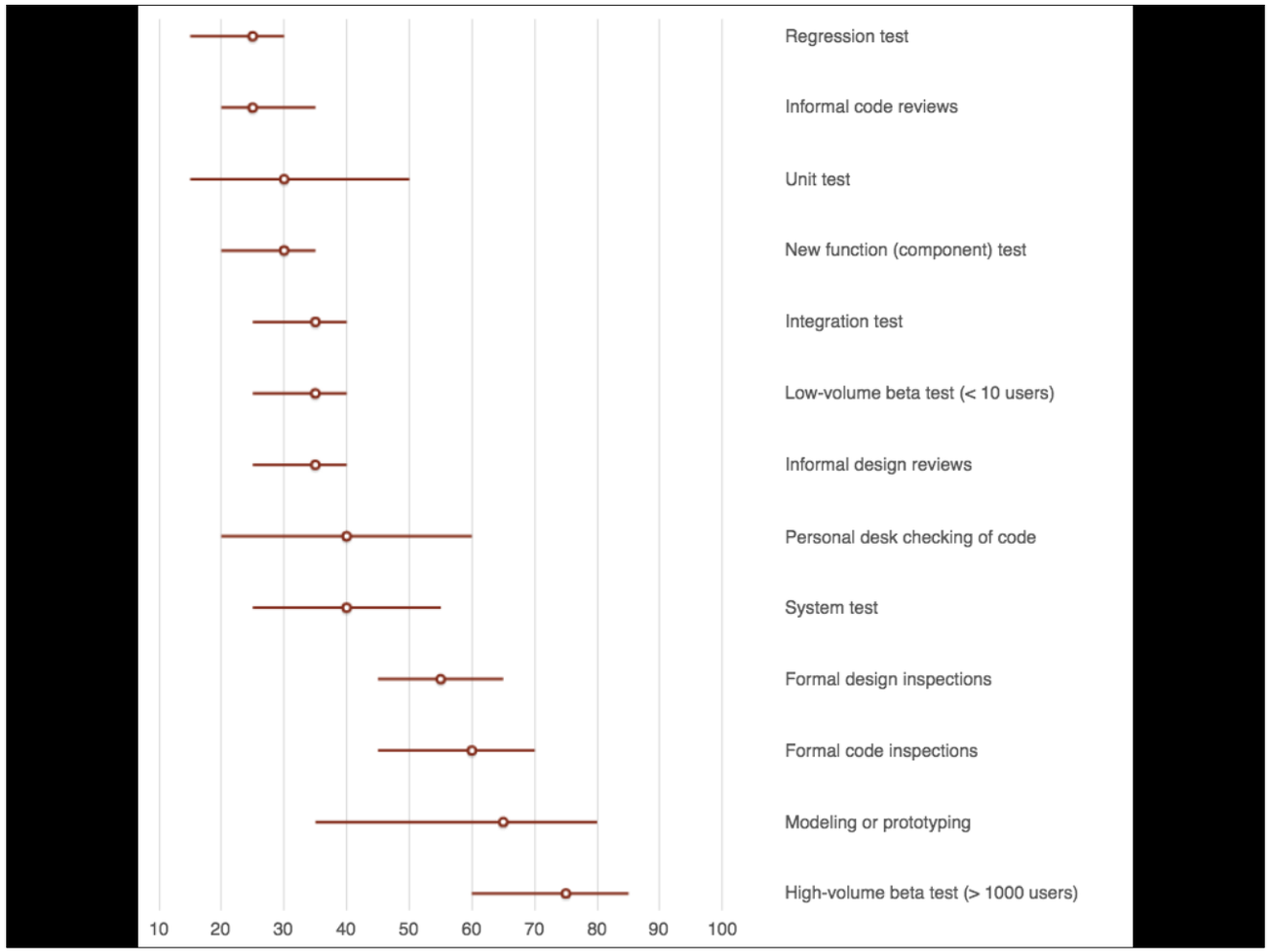
Automated testing finds a certain proportion (<40%) and type of bugs

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

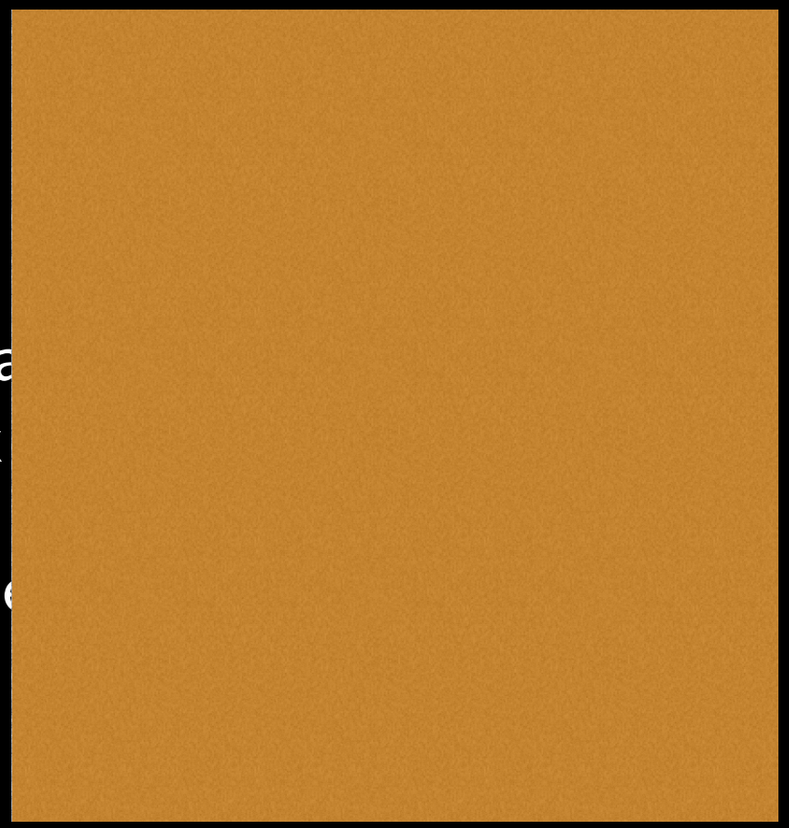
Different kinds of automated testing

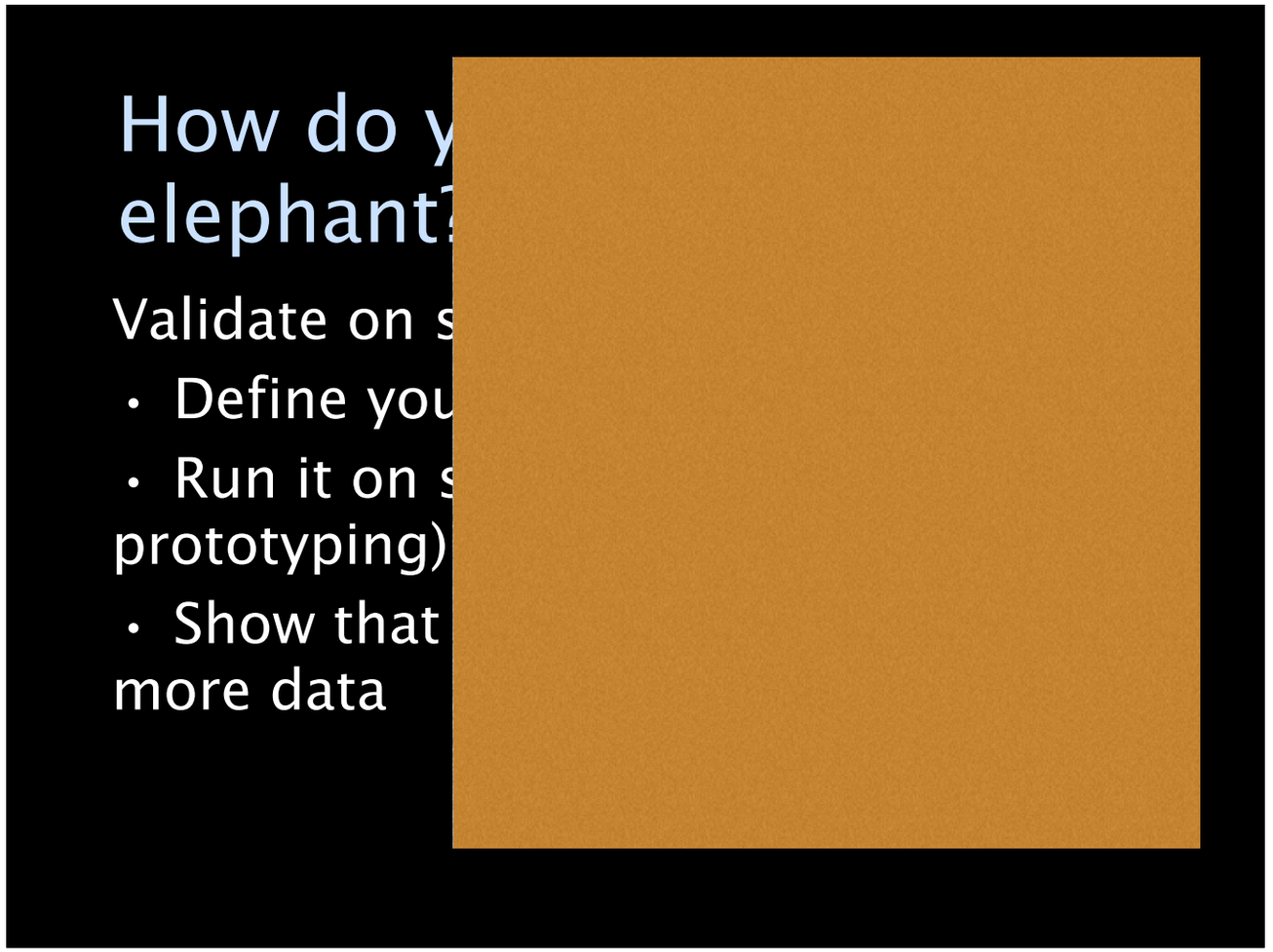
unit vs integration tests

DATA & ALGORITHMS

3 teams:

- Write the a
- White-box
working
- The hostil

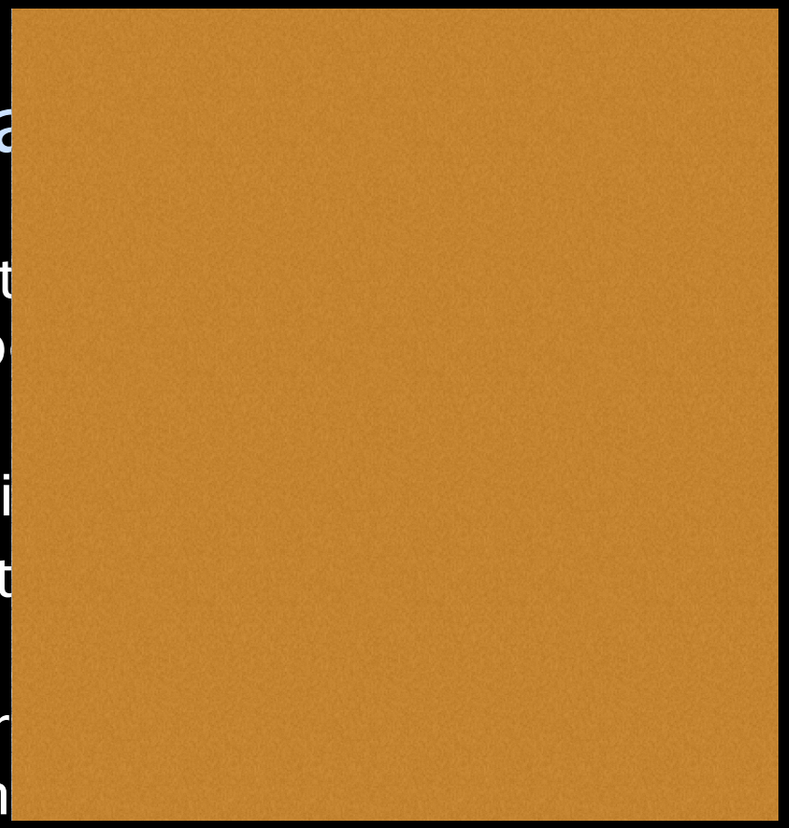




how do you eat an elephant? one bite at a time. start small, with a tiny subset of your data. that way, the algorithm runs quickly while you're prototyping

Fake data

Generate dat
way you exp
Can be hard
you think thi
Confirm that
should
Useful for or
presentation

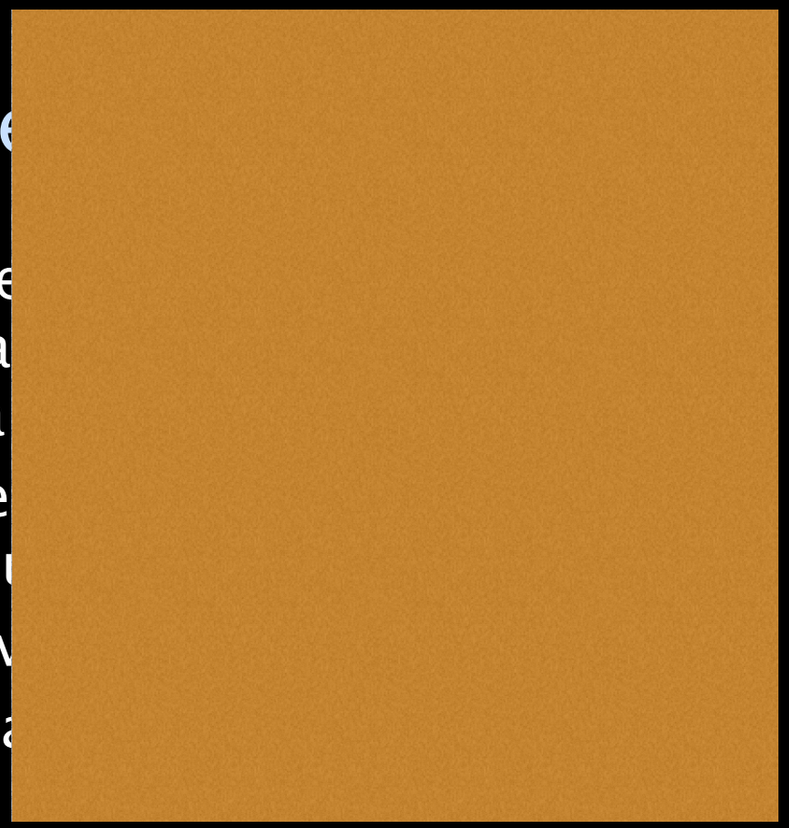


Nonsense

Set a trap. Fe
nonsense da
the results a

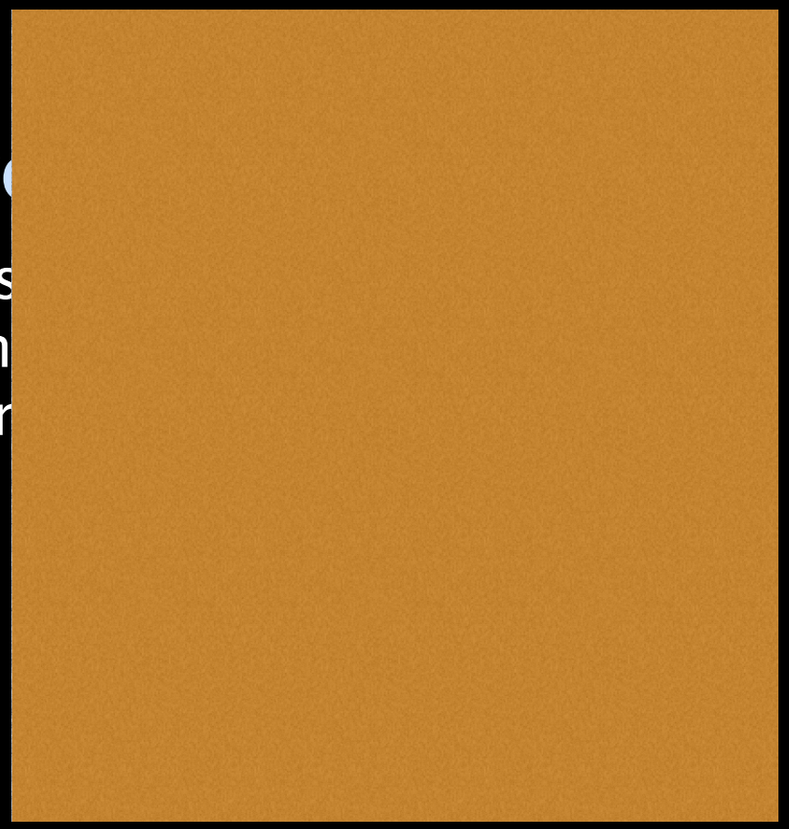
Easy: shuffle
in random nu

This. Will Sav
e.g. guard a



compare
truth if y

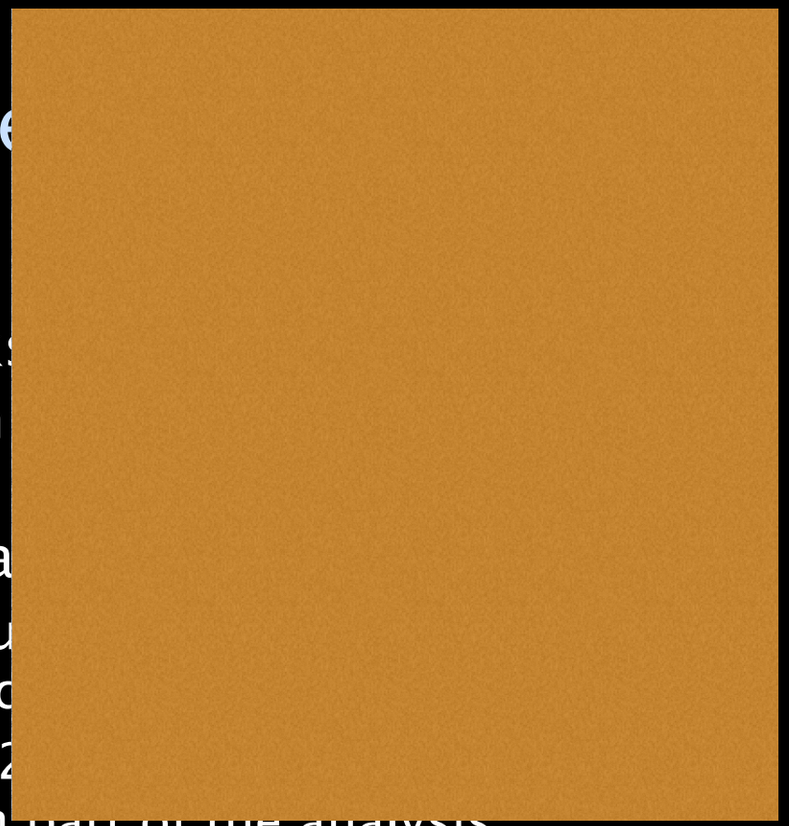
e.g. previous
version of th
by hand on r



Defensive

Pepper your
sanity check
- e.g. confirm
values, type

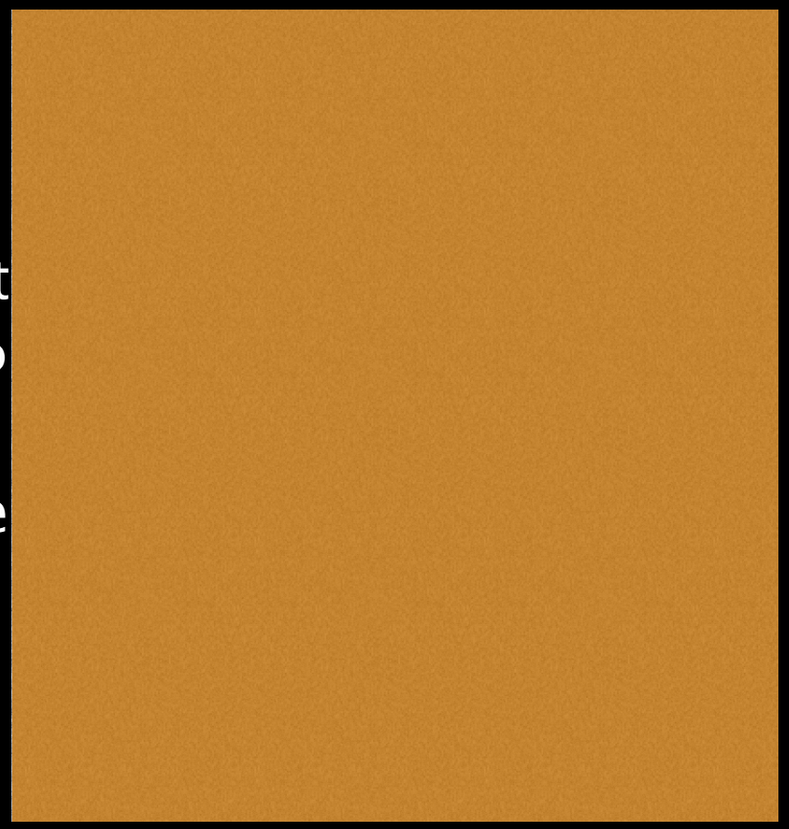
Fail immedia
- that way you
near to the c
- rather than 2
downstream part of the analysis



Scripts

Version cont
including co

Commit ofte



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?

Alternative methods, e.g. code review?

If I'm in a really big hurry?

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)

maybe cast to floats as part of the assert

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