Unit testing

Hróbjartur Thorsteinsson // hroi@vedur.is

improves code quality
and confidence

Software testing

- safeguards against code regression
- sets design targets
  - use cases fulfilled
  - enforces the programmer to follow design
- reduces worry and panic
  - feel good about code changes
- sets a measurable standard
  - for version release

Testing is full of jargon

unit test

system test

integration test

white box

black box

regression test

acceptance test

smoke test

Key concepts: Test scope!

- unit test
    i. quarantine and test small code pieces (units) of the software
    ii. smallest scope
- integration test
    i. test larger code units allowing interaction with other units
    ii. intermediate scope
- system test
    i. follows a typical ‘user case’ for the whole software
    ii. largest scope

Words describing how or why?

- white box test
  - test insides of the code - e.g. function calls within code
- black box test
  - test from outside the code - input / output only
- non-”regression” test
  - test intended to stop code from regressing - e.g. stop a bug from reappearing
- acceptance test
  - test that checks that a particular user requirement and/or use case has been met.
- smoke test
  - a light-weight system test to quickly check if the software runs. used before running lengthier tests.

Fun jargon combinations:

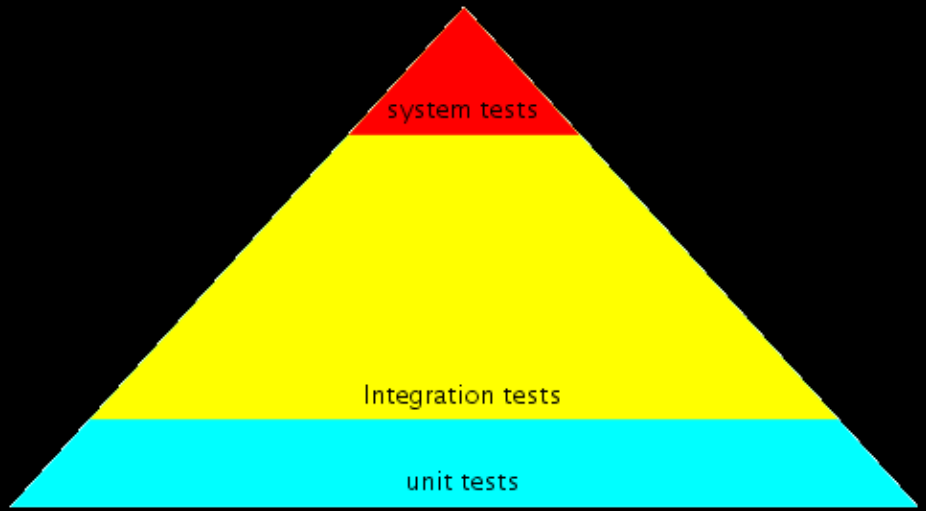
“A system test is usually a black box test”

“Is that a white box unit test?”

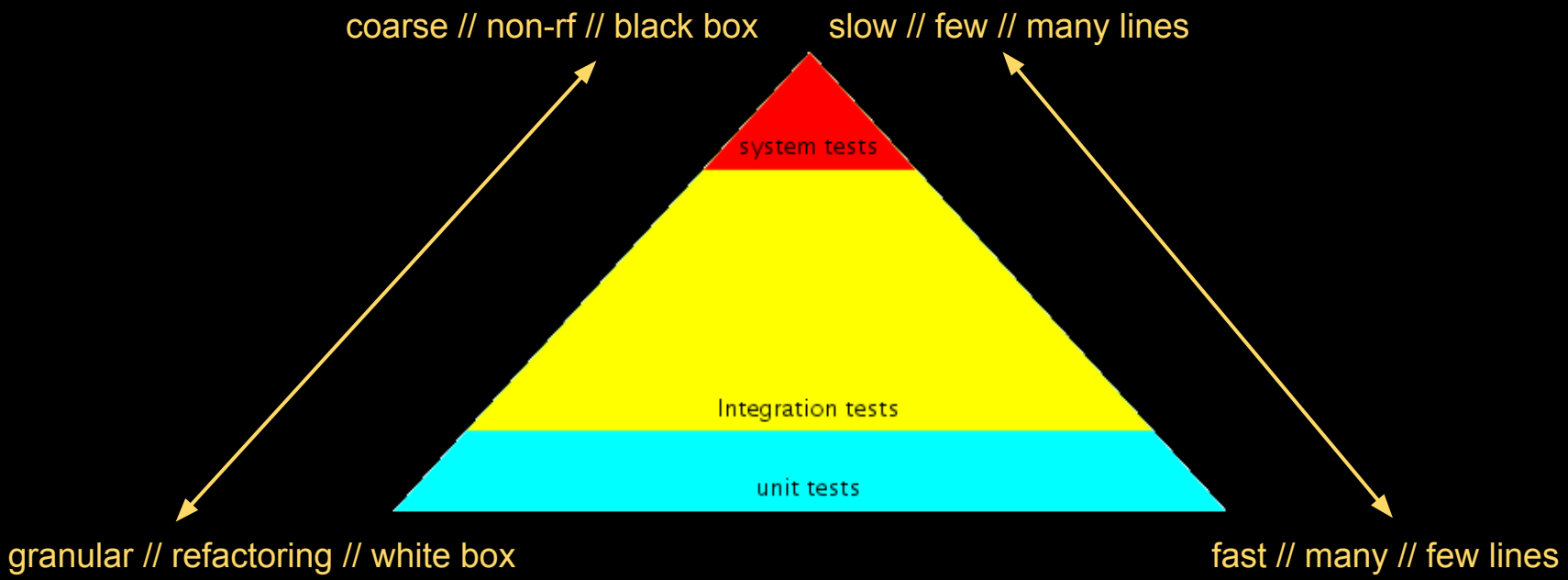
“I added a regression test for that bug”

“That’s not a unit test, that’s a low level integration test !”

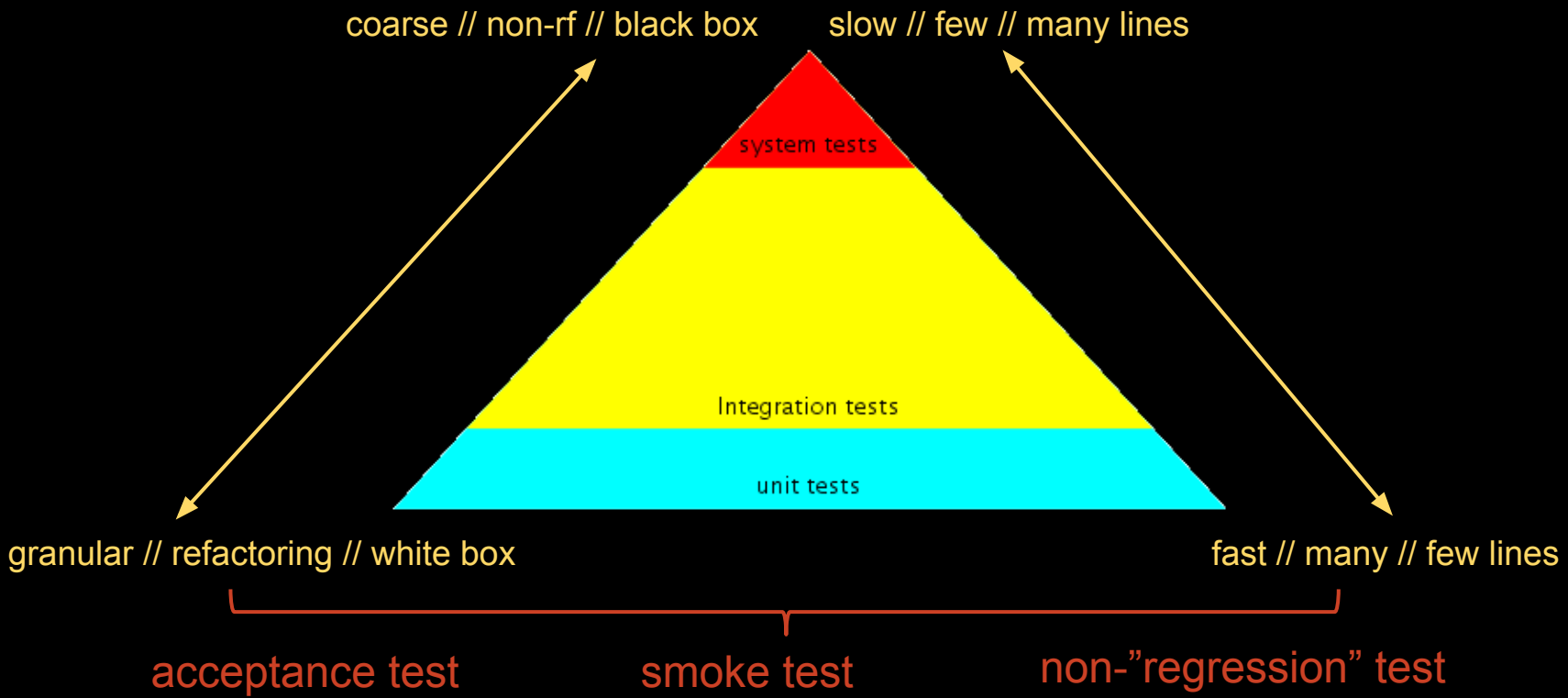
Test pyramid



Test pyramid



Test pyramid



Unit testing

1. test smallest code units (code lines / functions / methods)
2. isolated from other methods and classes 'as much as is practicable'
3. one behaviour, function or class method at a time
4. each test, very fast (<10ms / test) !
5. 'ideally' write test, then code
    i. test driven development
    ii. force code slaves to follow your design - management tool
6. aim to have very few lines in tests (~ 4 - 6)
    i. leads to great code refactoring
    ii. if many lines for setting up a test → your code can likely be broken up into even smaller units

Stubs and Mocks

- isolate your functions / methods / classes
- Stubs and Mocks are “fake” code that is used to replace external functions / methods / classes.
- Stubs/Mocks set conditions
  - give some fake test response/data
  - or record a call from the code being tested
- behaviour of the code unit can now be tested
  - what did the code unit return?
  - what call did the mock object receive from the code unit?
  - how did the code unit react to the mock values returned by the mock?

* Stubs and Mocks are often used interchangeably, but are slightly different in scope - not important for now. Mocking is considered more common and a more flexible way of testing in python.

Testing

Three simple steps,

1. setup
    i. create fake input data
    ii. instanciate w/wo fake input
    iii. mock out dependencies
2. run *just one function/method call !
3. assert *just one simple assert, …. ideally

Python example

```
class PhoneBookLayer():
        db = MyDataBase()

        def __init__(self):
                pass

        def insert(self, record):
                “”” Insert a new phone book record, semi-colon sep. string:  ‘name;address;phone_number’ “””

                name, address, phone_number = record.split(‘;’)
                self.db.add(‘phone_book’, name, address,    int(phone_number)    )
```

Python example

```
import unittest
from mock import Mock

class TestPhoneBookLayer( unittest.TestCase ):
        def setUp(self):
                self.pbl = PhoneBookLayer()
                self.pbl.db.add = Mock()

        def test_insert(self):
                self.pbl.insert(“Veðurstofan;150 Reykjavík;5226000”)
                self.pbl.db.add.assert_called_with(‘phone_book’,
                                                ’Veðurstofan’, ’150 Reykjavík’, 5226000)
```

Command line tools

- nose (to run tests)
- coverage (to see how well your code is tested)
- pylint (helps debug, follow good python practice)
- pyreverse (visualize code with UML diagrams)