HCS COMPANY

PYTHON UNIT TESTING

VINCENT VAN DAM, 2021-04-06

VERSIE: 1.1

testing

“Program testing can be used to show the presence of **bugs**, but never to show their **absence**!”

EDSGER DIJKSTRA

Performace testing
Unit testing
Component testing
Contract testing
Smoke testing
API testing
Manual testing
End 2 end testing
Soak testing
Chaos testing

But why?

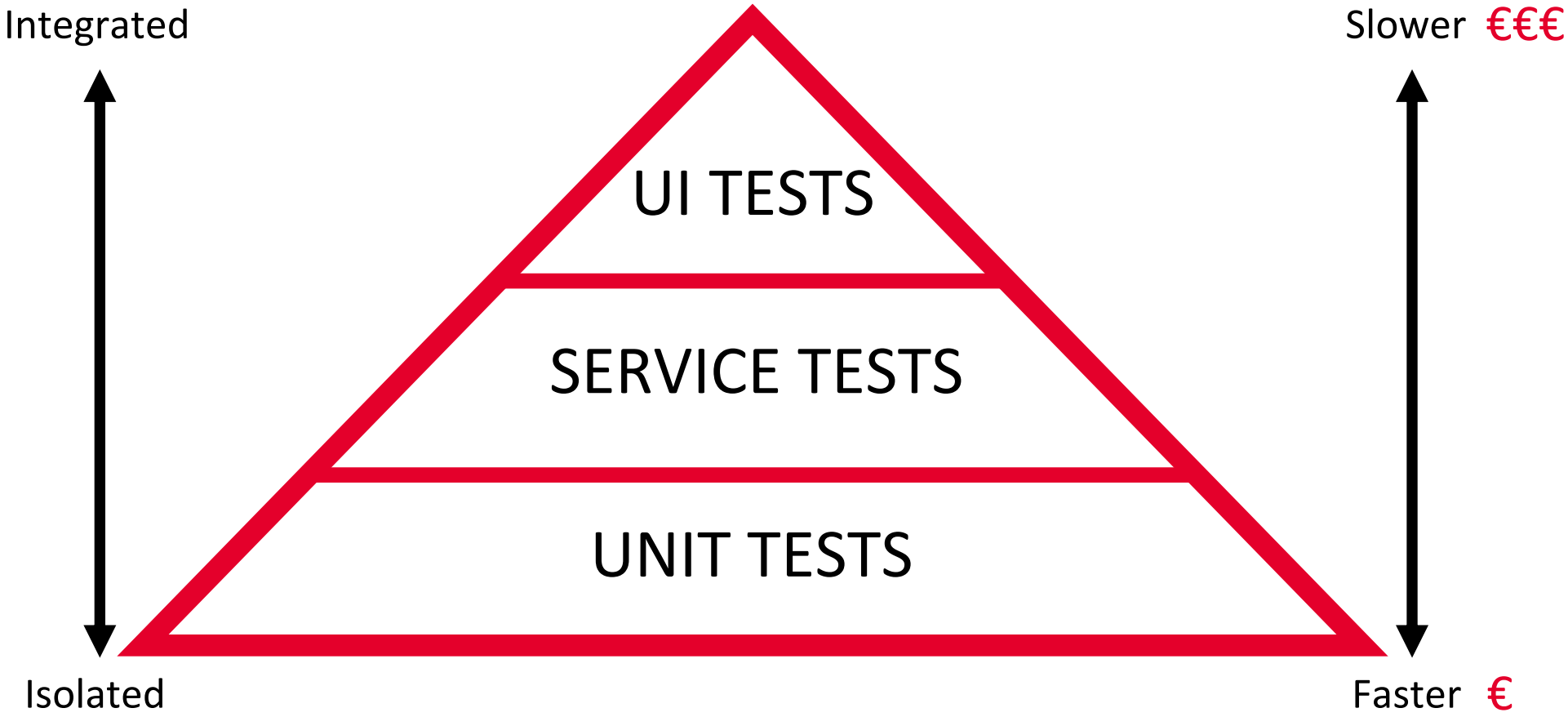
(specifically, why automated?)

Failsafe – regression
Increase development speed
Documentation

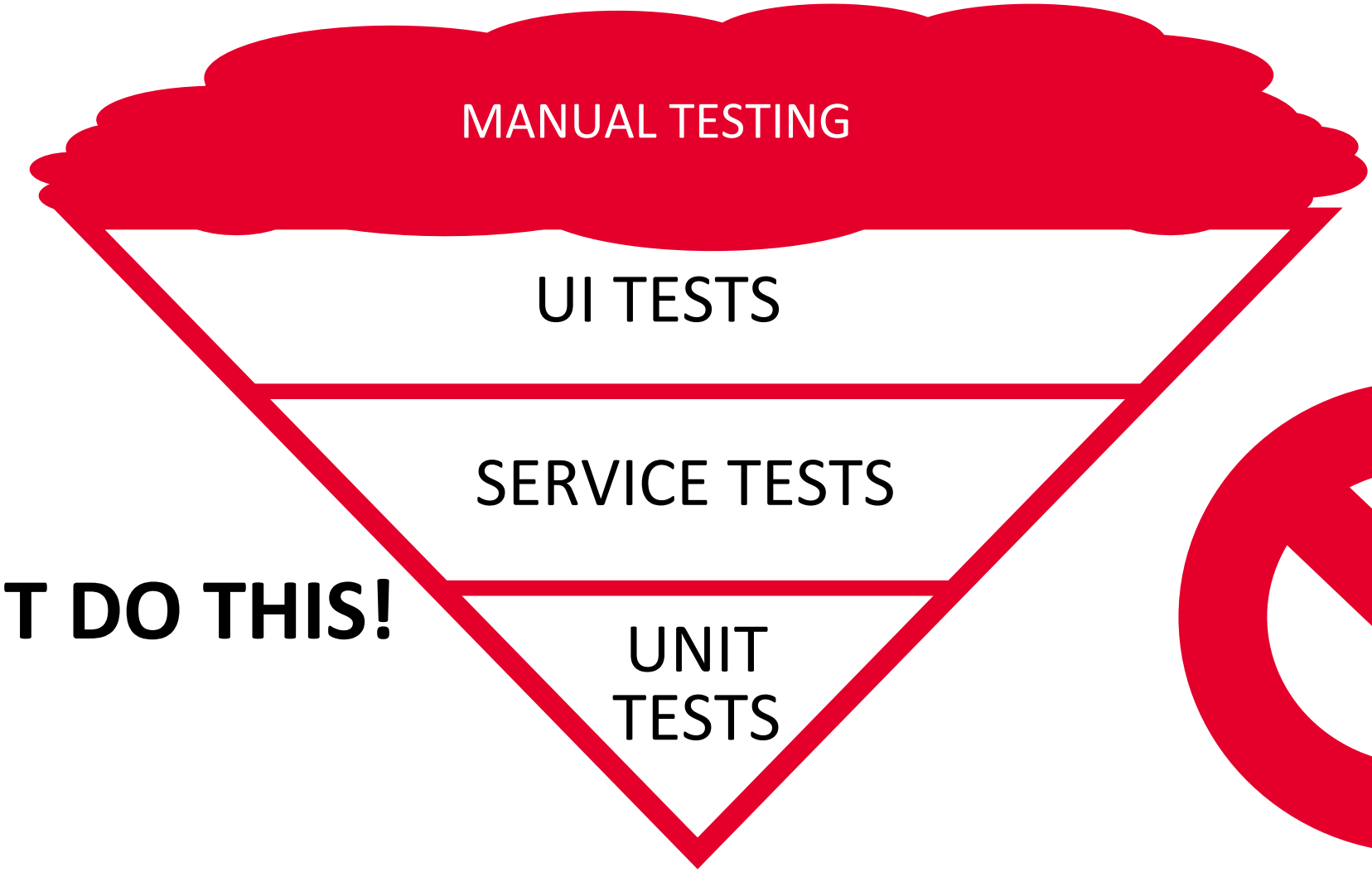
Quality!

Basis for agile development

Mike Cohn's Test Pyramid



Ice Cream Cone Testing



DON’T DO THIS!

Unit testing

(in python)

UNIT TEST

```python
def greet(name):
    return f"Hello {name}!"
```

```python
import unittest

from example import hello

class Test_Hello(unittest.TestCase):

    def test_hello(self):
        self.assertEqual(hello.greet("Kitty"), "Hello Kitty!")
```

UNIT TEST

```python
def greet(name):
    return f"Hello {name}!"
```

```python
import unittest

from example import hello

class Test_Hello(unittest.TestCase):

    def test_hello(self):
        self.assertEqual(hello.greet("Kitty"), "Hello Kitty!")
```

```
$ python -m unittest discover tests/ -p "*_test.py"
.
----------------------------------------------------------------------
Ran 1 test in 0.000s


OK
```

UNIT TEST

```python
def greet(name):
    return f"Hello {name}!"
```

```
$ python -m unittest discover tests/ -p "*_test.py"
F
======================================================================
FAIL: test_hello (hello_test.Test_Hello)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "example/tests/hello_test.py", line 8, in test_hello
    self.assertEqual(hello.greet("Kitty"), "Hello Spongebob!")
AssertionError: 'Hello Kitty!' != 'Hello Spongebob!'
- Hello Kitty!
+ Hello Spongebob!

----------------------------------------------------------------------
Ran 1 test in 0.001s

FAILED (failures=1)
```

Asserts

```
self.assertEqual(hello.greet("Kitty"), "Hello Kitty!")
```

METHOD	CHECKS THAT
assertEqual(a, b)	a == b
assertNotEqual(a, b)	a != b
assertTrue(x)	bool(x) is True
assertFalse(x)	bool(x) is False
assertIs(a, b)	a is b
assertIsNot(a, b)	a is not b
assertIsNone(x)	x is None
assertIsNotNone(x)	x is not None
assertIn(a, b)	a in b
assertNotIn(a, b)	a not in b
assertIsInstance(a, b)	isinstance(a, b)
assertNotIsInstance(a, b)	not isinstance(a, b)

USE LISTS TO EFFICIENTLY IMPLEMENT 'LOTS' OF TEST CASES

```
def test_firstname(self):
    tests = [
        [ "Spongebob",                "Spongebob" ],
        [ "Spongebob Squarepants",    "Spongebob" ],
        [ "Mr. Spongebob Squarepants", "Spongebob" ],
        [ "Mr Spongebob Squarepants", "Spongebob" ],
        [ "John Doe",                 "John"      ],
    ]

    for test in tests:
        res = hello.firstname(test[0])
        self.assertEqual(list(res), test[1])
```

LAB 1

PRACTICAL EXAMPLE: TICTACTOE

- https://katacoda.com/hcs-company/scenarios/python-unit-testing

More Unit testing

(still in python)

Code coverage

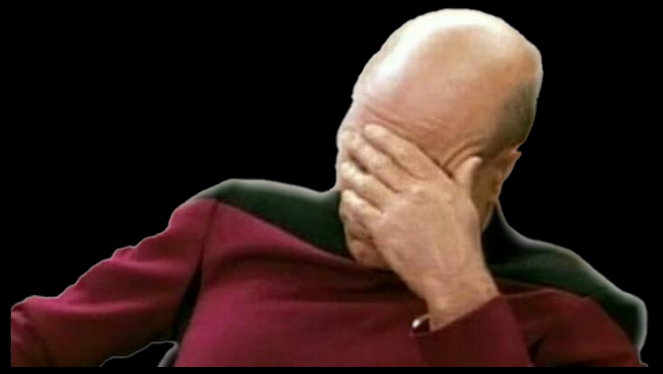
```
$ coverage run -m pytest --cov tictactoe --cov-report term-missing
============================= test session starts ==============================
platform darwin -- Python 3.9.1, pytest-6.2.2, py-1.10.0, pluggy-0.13.1
rootdir: tictactoe
plugins: cov-2.11.1
collected 5 items


tests/board_test.py ...                                                  [ 60%]
tests/game_test.py ..                                                    [100%]


---------- coverage: platform darwin, python 3.9.1-final-0 -----------
Name                      Stmts   Miss  Cover   Missing
-------------------------------------------------------
tictactoe/__init__.py         0      0   100%
tictactoe/board.py           41     12    71%   24-28, 40, 44, 46, 50-53
tictactoe/game.py            41     32    22%   13-20, 23-56
-------------------------------------------------------
TOTAL                        82     44    46%



============================== 5 passed in 0.12s ===============================
```

Decision coverage

```
def magic_function():
    if condition:
        print("many statements")
        print("will satisfy a")
        print("high code coverage")
        print("but if the else is")
        print("not tested, it will")
        print("produce a low")
        print("decision coverage")
    else:
        print("just some single statement")

    return
```

CONTROL LOOPS

```
player = board.X
while not b.has_winner() and b.can_move():
    if player == board.X:
        move = input("your move (x,y)? ")
        if not is_valid_move(move):
            print ("invalid move entered...\n")
            continue
        c = get_move_coord(move)
    else:
        c = get_computer_coord(b)

    b.put(*c, player)

    if player == board.X:
        player = board.O
    else:
        player = board.X
```



INVERSION OF CONTROL

```
def some_method():
    ## lots of business logic
    ## to be tested
    cli = Client()
    cli.magic_api_call()

def main():
    some_method()
```

INVERSION OF CONTROL

```
def some_method(cli):
    ## lots of business logic
    ## to be tested
    cli.magic_api_call()

def main():
    cli = Client()
    some_method(cli)
```

Dependency Injection

```
public class SomeClass {
    @Inject
    KubernetesClient client;
}
```

```
public class OtherClass {
    @Produces
    public KubernetesClient kubernetesClient() {
        // here you would create a custom client
        return new DefaultKubernetesClient();
    }
}
```

java example, there is no standard dependency injection in python…

Mocks

```
def some_method(cli):
    ## lots of business logic
    ## to be tested
    return cli.magic_api_call()
```

```
from unittest import mock

def test_MockPlay(self):
    with mock.patch.object(client.Client, 'magic_api_call',
                           return_value="Hello!"):

        cli = client.Client()
        res = some_method(cli)
        self.assertEqual(res, "Hello!")
```

CONTROL LOOPS

```
player = board.X
while not b.has_winner() and b.can_move():
    if player == board.X:
        move = input("your move (x,y)? ")
        if not is_valid_move(move):
            print ("invalid move entered...\n")
            continue
        c = get_move_coord(move)
    else:
        c = get_computer_coord(b)

    b.put(*c, player)

    if player == board.X:
        player = board.O
    else:
        player = board.X
```

How to make this testable?

- Step 1: move control loop to a method
- Step 2: use inversion of control to make it testable
- Step 3: create a test for the control loop providing mocked instances

```
def play(b, players):
    turn = 0
    while not b.has_winner() and b.can_move():
        c = players[turn].get_move_coord(b)
        b.put(*c, players[turn].piece)
        print()
        print(b.to_string())
        turn = (turn+1)%2
    return b.get_winner()
```

LAB 2

PRACTICAL EXAMPLE: TICTACTOE

- https://katacoda.com/hcs-company/scenarios/python-unit-testing

Wrap up...

Wrap up...

- Test pyramid
- Intro into unit tests with python
- Code coverage / decision coverage
- Inverse of dependency and mocking

FIN!

Questions?