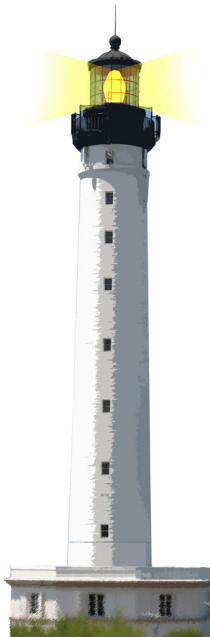
Learning Object-Oriented
Programming and Design with TDD

Why testing is important

S. Ducasse
http://stephane.ducasse.free.fr



Pharo
http://www.pharo.org

WXSYY

Software Testing

- It is a vast and important topic
- Multiple conferences on the topic
- Heavily used in industry
- Many companies forgot to invest in tests and usually they pay the price

In this lecture, we just gives a little overview.

Goal

Minimal survey

- Why tests are important?
- What are their advantages?
- What are the techniques to write good tests?

Multiple kinds of tests

Multiple kinds

- Integration testing: verify that different modules or services used by your application work well together.
- Functional testing: focus on the business requirements of an application.
- Unit tests (low level)

Others

- Acceptance testing: formal tests executed to verify if a system satisfies its business requirements.
- Performance testing: behaviors of the system when it is under significant load.
- Smoke testing: check basic functionality of the application. to give you the assurance that the major features of your system are working as expected

We will focus on Unit test.

Don’t and Do

- Do not prevent bugs to appear
- Do identify that a bug appear

Tests are you insurance that when something breaks you know it.

A Test

- You write it once
- Run it million times...

Tests

- Can improve customers trust
- Give you the parachute to change your software
- Guarantee that old bugs won’t reappear

Good test suite

- check extreme cases (e.g. null 0 and empty)
- check complex cases(e.g. exceptions)
- 1 test for each bug (at least)
- good coverage
- check abstractions
- check units independently

Tests support understanding

```
testConvert
  self assert: Color white convert = '#FFFFFF'.
  self assert: Color red convert = '#FF0000'.
  self assert: Color black convert = '#000000'
```

Tests support understanding

```
testBitShift
    self assert: (2r11 bitShift: 2) equals: 2r1100.
    self assert: (2r1011 bitShift: − 2) equals: 2r10.
```

```
testShiftOneLeftThenRight
    "Shift 1 bit left then right and test for 1"
    1 to: 100 do: [:i |
        self assert: ((1 bitShift: i) bitShift: i negated) = 1]
```

Advantages

- Give simple and reproducible examples
- Explain an API
- Give a first up to date documentation
- Check conformity of new code
- Offer a first client to new code
- Force modular design

For Understanding support

Good Unit tests are:

- deterministic
- automatic
- self explained
- simple
- unit

Increasing Trust

- Accelerate bug detection
- Help validation of changes
- Ease refactorings
- Prevent regressions

For Increasing Trust

Good Unit tests are:

- change less frequently than the rest
- good code coverage
- deterministic

Colateral advantages

- Improve feeling of customers who care
- Allow for automatic bug fixing
- Improve type inference
- Provide examples to variable values

Testing Tips: Fix the world

You cannot test a changing system

- Fix part
- Test a fixed subset

For example, test minimal local fonts and not the ones on the machine.

Testing Tips: Mocks

Mocks are simple faked objects that represent a part that is not under test but participate to the object context.

- Example, how to test that an object reads well an output from the network?
  - provide a Mock object playing the network

Testing Tips: Mocks

- Some mocks framework let you teach your mock objects how to respond to messages.
- https://github.com/dionisiydk/Mocketry
- JMock
- "Growing Object-Oriented Software Guided by Tests"

Three excellent testing practices

- During dev, write tests first
  - Specify what you want
  - You are done when the tests run
- When you redesign/improve your software
  - refactor in small steps and
  - run the tests to stop any regression
  - fix what is broken (get the bar green)
- During debugging
  - write a test that demonstrates the bug
  - then fix it.

Conclusion

- Tests are important
- Tests are your life insurance
- Tests identify bugs
- The world changes continuously.
- Software models the world so it will BREAK.

A course by Stéphane Ducasse

http://stephane.ducasse.free.fr

Reusing some parts of the Pharo Mooc by

Damien Cassou, Stéphane Ducasse, Luc Fabresse

http://mooc.pharo.org