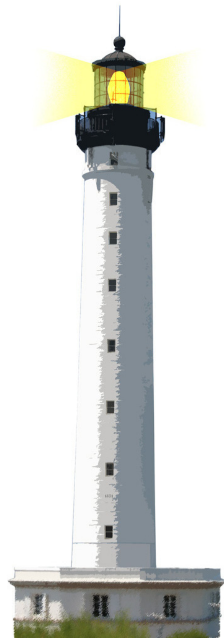
SUnit: Unit Tests in Pharo

Damien Cassou, Stéphane Ducasse and Luc Fabresse

W5S06



Agile Programming

Pharo programming culture adheres to agile programming

- Incremental, early feedback
- Be prepared for changes
- Driven by human
- Supported by automated tests

SUnit

- Extremely simple (4 classes)
- Originally developed by K. Beck (agile programming father)
- Got copied all over the places

A Test

In a test, we

- Create a context: Create an empty set
- Send a stimulus: Add twice the same element
- Check the results: Check that the set contains only one element

Set TestCase

```
TestCase subclass: # SetTestCase
   ...
```

```
SetTestCase >> testAdd
  | empty |
  empty := Set new.   "Context"
  empty add: 5.   "Stimulus"
  empty add: 5.
  self assert: empty size = 1.   "Check"

SetTestCase run: #testAdd
```

In a Subclass of TestCase

Each method starting with test*:

- Represents a test
- Is automatically executed

The results of the test are collected in a TestResult object

Another Example

```
testAdjacentRunsWithEqualsAttributesAreMerged
  "this demonstrates that adjancent runs with equal attributes
      are merged. "
  | runArray |
  runArray := RunArray new.
  runArray
    addLast: TextEmphasis normal times: 5;
    addLast: TextEmphasis bold times: 5;
    addLast: TextEmphasis bold times: 5.
  self assert: (runArray runs size = 2).
```

Failures and Errors

- A **failure** is a failed assertion, i.e., an anticipated problem that you test
- An **error** is a condition you didn't check for

To Test That an Error Must Be Raised

```
SetTestCase >> removeElementNotInSet
 self
  should: [ Set new remove: 1 ]
  raise: NotFound
```

To Test That an Error Must Not Be Raised

```
SetTestCase >> removeElementNotInSet
 self
  shouldnt: [ Set new add: 1 ]
  raise: NotFound
```

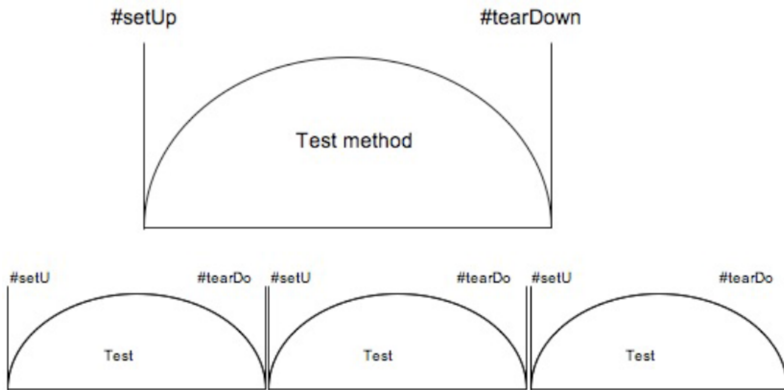
Duplicating the Context

```
SetTestCase >> testOccurrences
 | empty |
 empty := Set new.
 self
  assert: (empty occurrencesOf: 0)
  equals: 0.
 empty add: 5; add: 5.
 self
  assert: (empty occurrencesOf: 5)
  equals: 1
```

- `empty := Set new.` is repeated between tests
- We can factor it out

setUp and tearDown Messages

- Executed systematically before and after each test run
- setUp allows us to specify and reuse the context
- tearDown to clean after



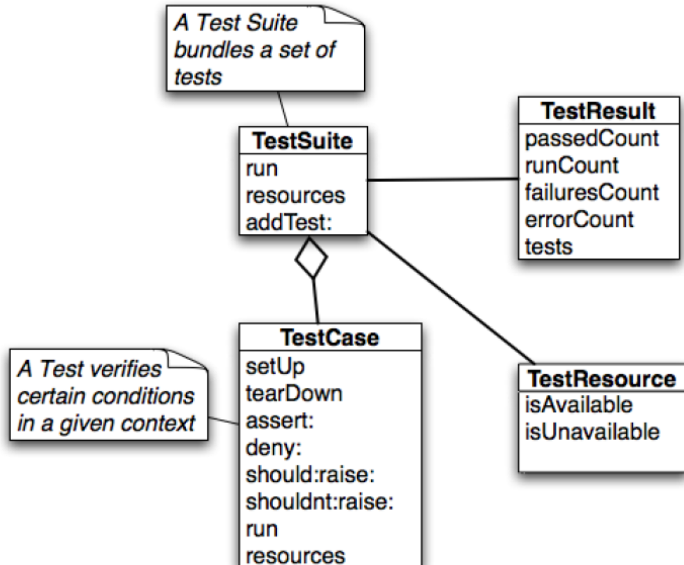
Defining a setUp Method

```
SetTestCase >> setUp
  empty := Set new
```

setUp is executed for you before any test execution

```
SetTestCase >> testOccurrences
  self
    assert: (empty occurrencesOf: 0)
    equals: 0.
  empty add: 5; add: 5.
  self
    assert: (empty occurrencesOf: 5)
    equals: 1
```

SUnit Core



TestSuite, TestCase and TestResult

- A TestCase represents one test
  - e.g. the method: SetTestCase >> testOccurenceOf
- A TestSuite is a group of tests
  - SUnit automatically builds a suite from the methods starting with 'test*'
- A TestResult represents a test execution results

Test Resources

- A TestResource is an object which is needed by a number of Test Cases, and whose instantiation is so costly in terms of time or resources that it becomes advantageous to only initialize it once for a Test Suite run.
- A TestResource is invoked once before any test is run. (read Pharo by Example SUnit Chapter)

What You Should Know

- How to write simple tests
- Reuse a bit the context by defining `setUp` methods

Summary

- Unit tests are easy to create and run
- Create one test and run it million times!
- Use them as your life insurance
- There exists other libraries for Mock (BabyMock) or different styles of testing

A course by

and

in collaboration with

Inria 2016

Except where otherwise noted, this work is licensed under CC BY-NC-ND 3.0 France
https://creativecommons.org/licenses/by-nc-nd/3.0/fr/