Xtreme Test Driven Development

Getting a productivity boost

S. Ducasse, L. Fabresse, G. Polito, and P. Tesone

http://www.pharo.org
Outline

- TDD on steroids
- Live programming at its best
- Smart tools
- Absolutely gorgeous development flow
Do **not break** the flow

- Write a test
- When it breaks, define the method **on the fly in the debugger**
- **Resume and continue** until the test is green
Studying an example

- A dead simple counter. Nothing simpler.
- Focus on the essence of the process!
- You can do it.
An empty package

Object subclass: #NameOfSubclass
  instanceVariableNames: ''
  classVariableNames: ''
  package: 'Counter'
An empty test case class

```
import Counter
import CounterTest

class CounterTest(TestsCase):
    instanceVariableNames: 
    classVariableNames: 
    package: 
```
A first test

```plaintext
testSetAndGetCounter
    self assert: (Counter new count: 22) count equals: 22
```
A first test (II)

- Method is about to be compiled
- The system knows the class does not exist!
Define a class

- At compile time...
Define a class (II)
Test defined but not executed

```plaintext
CounterTest>>testSetAndGetCounter

self assert: (Counter new count: 22) count equals: 22
```
Running the test

```ruby
testSetAndGetCounter

self assert: (Counter new count: 22) count equals: 22
```
First Error

Instance of Counter did not understand #count:

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Other</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>CounterTest</td>
<td>testSetAndGetCounter</td>
<td></td>
<td>Counter</td>
</tr>
<tr>
<td>CounterTest(TestCase)</td>
<td>performTest</td>
<td></td>
<td>SUnit-Core</td>
</tr>
<tr>
<td>CounterTest(TestCase)</td>
<td>runCase</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FullBlockClosure(BC)</td>
<td>ensure:</td>
<td></td>
<td>Kernel</td>
</tr>
</tbody>
</table>

**Source**

```ruby
testSetAndGetCounter
  self assert: (Counter new count: 22) count equals: 22
```

**Variables**

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>implicit</td>
<td>self</td>
<td>CounterTest&gt;&gt;#testSetAndGetCounter</td>
</tr>
<tr>
<td>attribute</td>
<td>expectedFails</td>
<td>an Array [0 items] ()</td>
</tr>
<tr>
<td>attribute</td>
<td>testSelector</td>
<td>#testSetAndGetCounter</td>
</tr>
<tr>
<td>implicit</td>
<td>thisContext</td>
<td>CounterTest&gt;&gt;#testSetAndGetCounter</td>
</tr>
</tbody>
</table>
Create a method on the fly

Create the missing class or method in the user prompted class, and restart the debugger at the location where it can be edited.

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Other</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>CounterTest</td>
<td>testSetAndGetCounter</td>
<td></td>
<td>Counter</td>
</tr>
<tr>
<td>CounterTest(TestCase)</td>
<td>performTest</td>
<td></td>
<td>SUnit-Core</td>
</tr>
<tr>
<td>CounterTest(TestCase)</td>
<td>runCase</td>
<td></td>
<td>Kernel</td>
</tr>
<tr>
<td>FullBlockClosure(BlockClosure)</td>
<td>ensure:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```ruby
testSetAndGetCounter
  self assert: (Counter new count: 22) count equals: 22
```
Create a method on the fly (II)
Edit the method in the debugger
Add an instance variable on the fly

```
count: anInteger
   count := anInteger
```

Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>implicit</td>
<td>self</td>
<td>a Counter</td>
</tr>
<tr>
<td>parameters</td>
<td>anInteger</td>
<td>22</td>
</tr>
<tr>
<td>implicit</td>
<td>thisContext</td>
<td>Counter&gt;&gt;count:</td>
</tr>
<tr>
<td>implicit</td>
<td>stack</td>
<td>22</td>
</tr>
</tbody>
</table>
Compile....

Instance of Counter did not understand #count:

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Other</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter</td>
<td>count:</td>
<td></td>
<td>Counter</td>
</tr>
<tr>
<td>CounterTest</td>
<td>testSetAndGetCounter</td>
<td></td>
<td>Counter</td>
</tr>
<tr>
<td>CounterTest(TestCase)</td>
<td>performTest</td>
<td></td>
<td>SUnit-Core</td>
</tr>
<tr>
<td>CounterTest(TestCase)</td>
<td>runCase</td>
<td>[self setUp. self performTest]</td>
<td>SUnit-Core</td>
</tr>
</tbody>
</table>

```
count: anInteger
```
Continue the execution...

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Other</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter</td>
<td>count:</td>
<td></td>
<td>Counter</td>
</tr>
<tr>
<td>CounterTest</td>
<td>testSetAndGetCounter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CounterTest</td>
<td>performTest</td>
<td></td>
<td>SUnit-Core</td>
</tr>
<tr>
<td>CounterTest</td>
<td>runCase</td>
<td></td>
<td>SUnit-Core</td>
</tr>
</tbody>
</table>

```
count: anInteger
  count := anInteger
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>implicit</td>
<td>self</td>
<td>a Counter</td>
</tr>
<tr>
<td>parameters</td>
<td>anInteger</td>
<td>22</td>
</tr>
<tr>
<td>attributes</td>
<td>count</td>
<td>nil</td>
</tr>
</tbody>
</table>
Supporting the programmer flow

- The system
  - created a new method for us
  - Removed the stack element with Error
  - Replaced it with a call to the new method
  - Relaunched execution
- We edited it and recompiled the method
- The system Continued execution
New method

The system:

- Created a new method
- Removed the stack element with Error
- Replaced it with a call to the new method

```small
count: anInteger
    self shouldBeImplemented
```

- `shouldBeImplemented` is just an exception so that the debugger stops again
Same story....
Debugger also precompiles methods

Instance of Counter did not understand #count

Stack

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Other</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Counter</td>
<td>count</td>
<td></td>
<td>Counter</td>
</tr>
<tr>
<td>CounterTest</td>
<td>testSetAndGetCounter</td>
<td></td>
<td>Counter</td>
</tr>
<tr>
<td>CounterTest(TestCase)</td>
<td>performTest</td>
<td></td>
<td>SUnit-Cor</td>
</tr>
<tr>
<td>CounterTest(TestCase)</td>
<td>runCase</td>
<td>[self setUp. self performTest SUnit-Cor]</td>
<td></td>
</tr>
</tbody>
</table>

Source

```
count
  ^ count
```

Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>implicit</td>
<td>self</td>
<td>a Counter</td>
</tr>
<tr>
<td>attribute</td>
<td>count</td>
<td>22</td>
</tr>
<tr>
<td>implicit</td>
<td>thisContext</td>
<td>Counter&gt;&gt;count</td>
</tr>
<tr>
<td>implicit</td>
<td>stackTop</td>
<td>nil</td>
</tr>
</tbody>
</table>
Test is green

testSetAndGetCounter

self assert: (Counter new count: 22) count equals: 22
One Cycle

- Run all the tests
- Ready to commit
- New test
Why XTDD is powerful

- Avoid *guessing* context when coding
- Much much better context
  - inspect that *specific* instance state
  - talk to that *specific* object
- Inspectable / interactable context
- Tests are not a side effect artifact but the *driving* force
Protip from expert Pharo developers

- Grab **as fast as** possible one object
- **Cristalize** your scenario with a test
- Xtreme TDD
- Loop
Advanced Object-Oriented Design and Development with Pharo

A course by
S.Ducasse, L. Fabresse, G. Polito, and P. Tesone

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/