

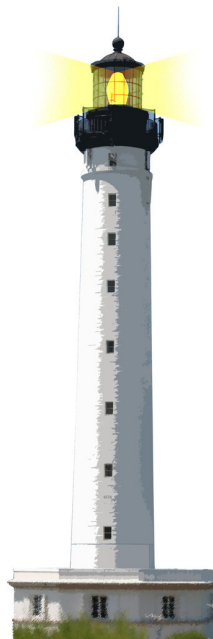
Xtreme Test Driven Development

Getting a productivity boost

S. Ducasse



<http://www.pharo.org>



Outline

- TDD on **steroids**"
- Live programming at **its best**
- Smart tools
- Absolutely **gorgeous** development flow



Principle

Do **not break** the flow

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



Studying an example

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



An empty package

The screenshot shows a software development environment window titled "Counter". The interface is divided into several panes. On the left, a "Counter" package folder is highlighted in yellow. Below this, a "Counter" label is visible. The main area is divided into three vertical panes. The left pane contains a "Filter..." label. The right pane is empty. At the bottom, a toolbar contains buttons for "All Packages", "Scoped View", "Inst. side", and "Class side". Below the toolbar, a "Comment" button and a "New class" button are visible. The "New class" button is highlighted in yellow. To the right of the toolbar, there are green and blue arrows. The bottom pane displays the following code:

```
Object subclass: #NameOfSubclass
  instanceVariableNames: ''
  classVariableNames: ''
  package: 'Counter'
```

An empty test case class

The screenshot shows the IntelliJ IDE interface with a project named 'CounterTest'. The top toolbar includes icons for 'x', '-', and a square. Below the toolbar, the 'CounterTest' window is active, showing a tree view with 'Counter' and 'CounterTest' folders. The 'CounterTest' folder is selected, and the 'instance side' view is active. The main editor area is empty, showing a filter input field. The bottom toolbar includes radio buttons for 'All Packages', 'Scoped View', 'Flat', 'Hier.', 'Inst. side' (selected), 'Class side', 'Methods', and 'Vars'. Below the toolbar, there are tabs for 'New class', 'Comment', 'CounterTest' (selected), 'setUp', and 'Inst. side meth'. The code editor shows the following code:

```
TestCase subclass: #CounterTest
  instanceVariableNames: ''
  classVariableNames: ''
  package: 'Counter'
```

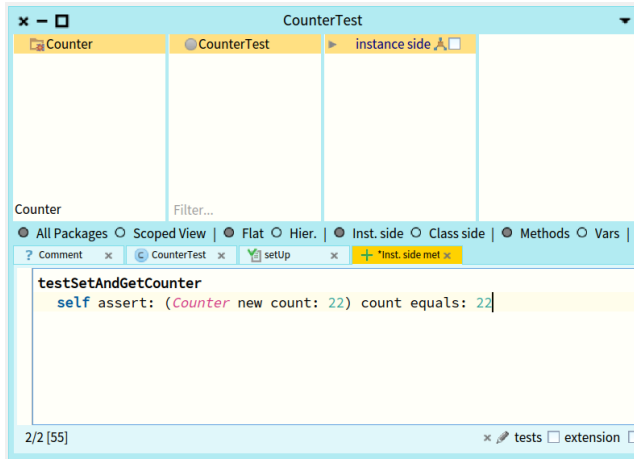
A first test

The screenshot shows the IntelliJ IDEA IDE interface during a test run. The top toolbar includes a close button (x), a maximize button (square), and the window title "CounterTest". Below the toolbar, the "CounterTest" tab is active, showing the "instance side" view. The left sidebar displays the "Counter" class. The main editor area shows the "testSetAndGetCounter" method with the following code:

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

The bottom status bar indicates the current view is "Inst. side met" (Instance side method).

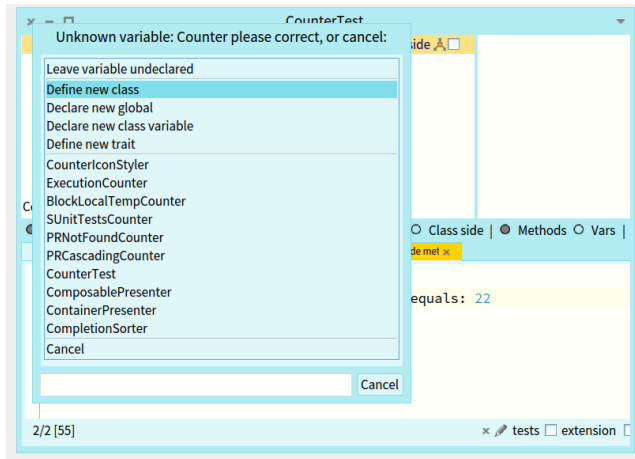
A first test



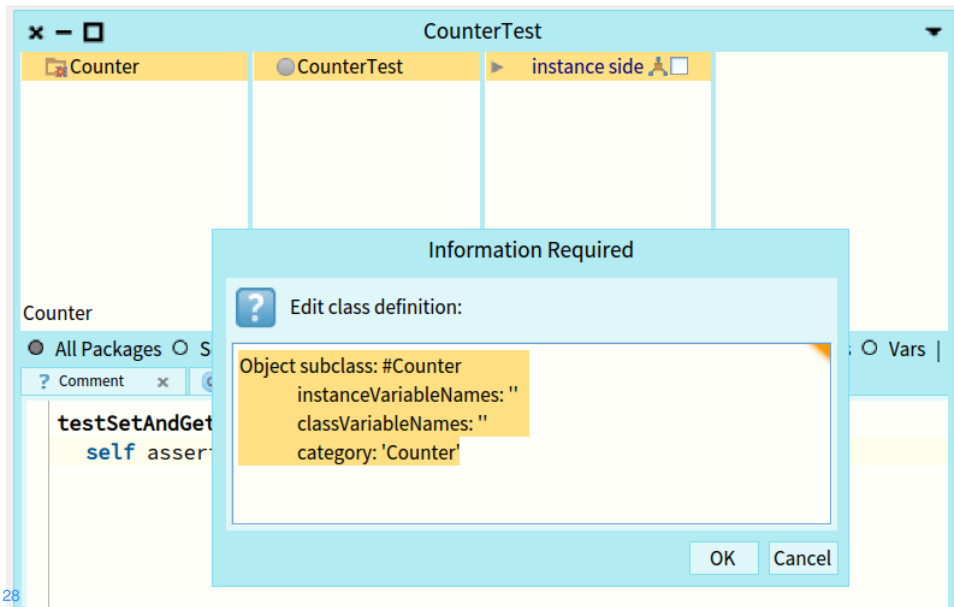
- Method is about to be compiled
- The system knows the class does not exist!

Define a class

- At compile time...



Define a class (II)

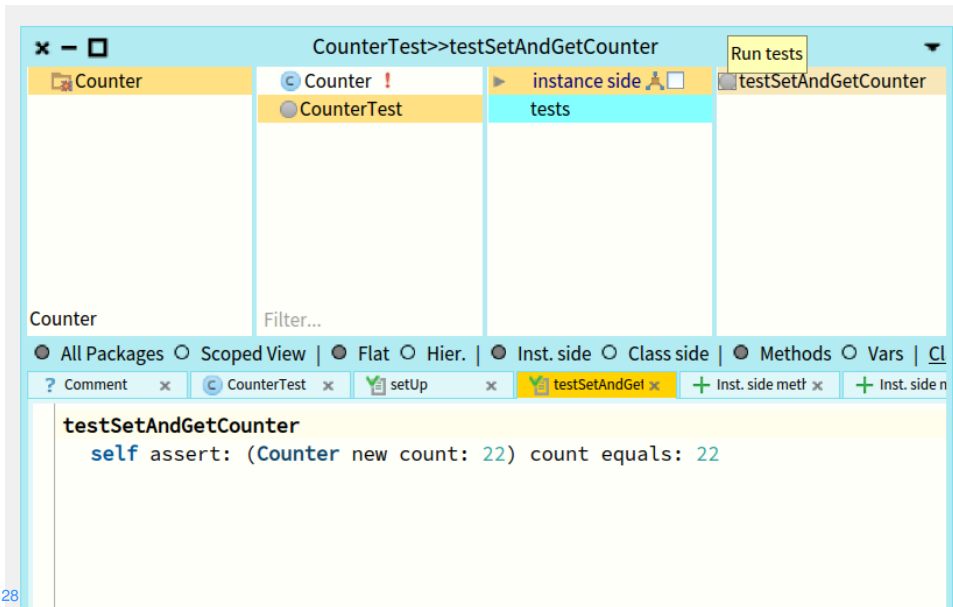


Test defined but not executed

The screenshot shows an IDE window titled "CounterTest>>testSetAndGetCounter". The interface is divided into several panes. On the left, a "Counter" package is shown. The middle pane displays a tree view with "Counter" (marked with a red exclamation mark) and "CounterTest". The right pane shows a list of tests, with "testSetAndGetCounter" selected. Below the panes, a toolbar contains options for "All Packages", "Scoped View", "Flat", "Hier.", "Inst. side", "Class side", "Methods", "Vars", and "CL". A tab bar at the bottom shows several open tabs: "Comment", "CounterTest", "setUp", "testSetAndGet", and two "Inst. side mettr" tabs. The main editor area displays the code for the "testSetAndGetCounter" test:

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

Running the test



The screenshot shows the IntelliJ IDEA IDE interface for running tests. The title bar indicates the current context is `CounterTest>>testSetAndGetCounter`. A **Run tests** tooltip is visible over the `testSetAndGetCounter` entry in the tests list.

The interface is divided into several panes:

- Left Pane:** Shows the project structure with `Counter` and `CounterTest`.
- Middle Pane:** Displays the test hierarchy. Under `CounterTest`, the `tests` folder is expanded, showing `testSetAndGetCounter`.
- Bottom Pane:** Shows the source code for the selected test. The code is:

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

The bottom status bar includes various filters and tabs for the test runner, such as `Comment`, `CounterTest`, `setUp`, `testSetAndGet`, and `Inst. side met`.

First Error

Instance of Counter did not understand #count: Bytecode GT

Stack

+ Create ▶ Proceed ↺ Restart ↩ Step into ↪ Step over ↪ Step through ≡

Class	Method	Other	Package
CounterTest	testSetAndGetCounter		Counter
CounterTest(TestCase)	performTest		SUnit-Core
CounterTest(TestCase)	runCase	[self setUp. self performTest	SUnit-Core
FullBlockClosure(BlockClosure)	ensure:		Kernel

Source

Where is? Browse

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

Variables Evaluator

Type	Variable	Value
implicit	self	CounterTest>>#testSetAndGetCounter
attribute	expectedFails	an Array [0 items] ()
attribute	testSelector	#testSetAndGetCounter
implicit	thisContext	CounterTest>>testSetAndGetCounter

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.

Instance of Counter d

Stack

+ Create ▶ Proceed ↺ Restart ↻ Step into ↗ Step over ↘ Step through ⋮

Class	Method	Other	Package
CounterTest	testSetAndGetCounter		Counter
CounterTest(TestCase)	performTest		SUnit-Core
CounterTest(TestCase)	runCase	[self setUp. self performTest	SUnit-Core
FullBlockClosure(BlockClosure)	ensure:		Kernel

Source

Where is? Browse

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

Create a method on the fly (II)

Instance of Counter did not understand #count: Bytecode GT

Stack

► Proceed ◀ Restart ⚙ Step into ⚙ Step over ⚙ Step through -≡

Class	Method	Other	Package
Counter	count:		Counter
CounterTest	testSetAndGetCounter		Counter
CounterTest(TestCase)	performTest		SUnit-Core
CounterTest(TestCase)	runCase	[self setUp. self performTest	SUnit-Core

Source

Where is? Browse

```
count: anInteger
self shouldBeImplemented.
```

Variables

Type	Variable	Value
implicit	self	a Counter

Edit the method in the debugger (III)

Instance of Counter did not understand #count: Bytecode GT

Stack

► Proceed ◀ Restart ▶ Step into ▶ Step over ▶ Step through ≡

Class	Method	Other	Package
Counter	count:		Counter
CounterTest	testSetAndGetCounter		Counter
CounterTest(TestCase)	performTest		SUnit-Core
CounterTest(TestCase)	runCase	[self setUp. self performTest	SUnit-Core

Source

Where is? Browse

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

Variables Evaluator

Type	Variable	Value
implicit	self	a Counter
parameter	anInteger	22
implicit	thisContext	Counter>>count:
implicit	stack top	22

Add an instance variable on the fly

The screenshot shows the IntelliJ IDEA IDE with a light blue theme. At the top, a title bar reads "Instance of Counter did not understand #count:". Below it, a message box says "Unknown variable: count please correct, or cancel:". A context menu is open over the message box, with options: "Declare new temporary variable", "Declare new instance variable" (highlighted in blue), and "Cancel". Below the menu is a text input field and a "Cancel" button. To the right of the message box, a toolbar contains "Restart", "Step into", "Step over", and "Step through" buttons. Below the toolbar is a "Package" list showing a tree structure: "Other" (selected), "Package", "Counter" (highlighted), "Counter", "SUnit-Core", and "[self setUp. self performTest SUnit-Core]". Below the "Package" list is a "Source" tab with a search icon and "Where is?" and "Browse" buttons. The "Source" tab displays the following code:

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

At the bottom, there is a "Variables" tab and an "Evaluator" tab. The "Variables" tab is active and shows a table with the following data:

Type	Variable	Value
implicit	self	a Counter
parameter	anInteger	22
implicit	thisContext	Counter>>count:
implicit	stack top	22

Compile....

✕ - □ Instance of Counter did not understand #count: Bytecode GT ▼

Stack ▶ Proceed 🔄 Restart 🔍 Step into 🔍 Step over 🔍 Step through ☰

Class	Method	Other	Package
Counter	count:		Counter
CounterTest	testSetAndGetCounter		Counter
CounterTest(TestCase)	performTest		SUnit-Core
CounterTest(TestCase)	runCase	[self setUp. self performTest	SUnit-Core

Source 🔍 Where is? 📄 Browse

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

Continue the execution...

Instance of Counter did not understand message 'runCase'

Relinquish debugger control and proceed execution from the current point of debugger control.cmd+r

Bytecode GT ▾

Stack

► Proceed ◀ Restart ▶ Step into ▶ Step over ▶ Step through ▸

Class	Method	Other	Package
Counter	count:		Counter
CounterTest	testSetAndGetCounter		Counter
CounterTest(TestCase)	performTest		SUnit-Core
CounterTest(TestCase)	runCase	[self setUp. self performTest]	SUnit-Core

Source

Where is? Browse

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

Variables

Type	Variable	Value
implicit	self	a Counter
parameter	anInteger	22
attribute	count	nil

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
- **Continued** execution



New method

The system created a new method

- Removed the stack element with Error
- Replace it with a **call** to the new method

```
count: anInteger  
self shouldBeImplemented
```

- `shouldBeImplemented` is just an exception so that the debugger stops again



Same story....

✕ - □

Instance of Counter did not understand #count

Bytecode GT ▾

Stack

+ Create ▶ Proceed 🔄 Restart ⚡ Step into ↗ Step over ↘ Step through ⋮

Class	Method	Other	Package
CounterTest	testSetAndGetCounter		Counter
CounterTest(TestCase)	performTest		SUnit-Cor
CounterTest(TestCase)	runCase	[self setUp. self performTest	SUnit-Cor
FullBlockClosure(BlockClosure)	ensure*		Kernel

Source

🔍 Where is? 📄 Browse

testSetAndGetCounter

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

Variables

Evaluator

Debugger also precompiles methods

Instance of Counter did not understand #count Bytecode GT

Stack

Proceed Restart Step into Step over Step through

Class	Method	Other	Package
Counter	count		Counter
CounterTest	testSetAndGetCounter		Counter
CounterTest(TestCase)	performTest		SUnit-Cor
CounterTest(TestCase)	runCase	[self setUp. self performTest	SUnit-Cor

Source

Where is? Browse

```
count
^ count
```

Variables Evaluator

Type	Variable	Value
implicit	self	a Counter
attribute	count	22
implicit	thisContext	Counter>>count
implicit	stack top	nil

Test is green

The screenshot shows the RubyMine IDE interface. The top toolbar indicates the current view is 'Instance side' and 'Tests'. The main pane displays the test results for 'CounterTest', showing a green circle icon and the text 'testSetAndGetCounter'. The bottom pane shows the source code for the 'testSetAndGetCounter' method, which contains a single line of code: `self assert: (Counter new count: 22) count equals: 22`. The status bar at the bottom left shows the date '2022' and the page number '24 / 28'.

CounterTest>>testSetAndGetCounter

Counter

Counter !

CounterTest

instance side

testSetAndGetCounter

tests

Counter

Filter...

All Packages Scoped View | Flat Hier. | Inst. side Class side | Methods Vars | CL

? Comment x CounterTest x setUp x testSetAndGet x + Inst. side metr x + Inst. side n

testSetAndGetCounter

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

2022 24 / 28

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 artefact 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



A course by

S. Ducasse, L. Fabresse, G. Polito, and Pablo 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/>