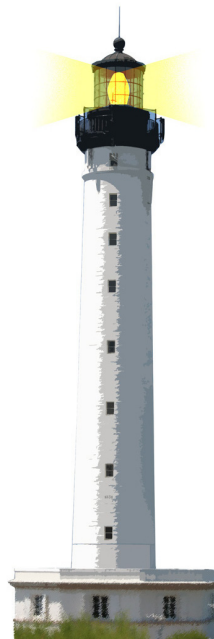


# **Xtreme Test Driven Development**

Getting a productivity boost

S. Ducasse



# 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 the test is green



# Studying an example

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

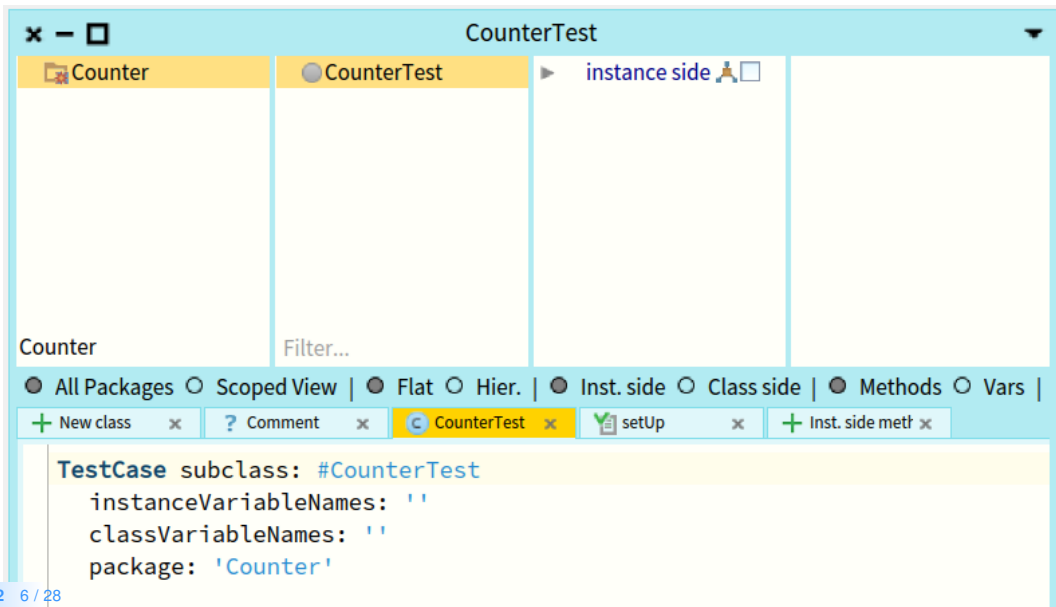


# An empty package

The screenshot shows an IDE window titled "Counter". The left sidebar displays a project tree with a folder named "Counter" highlighted. The main editor area is divided into three vertical panes, all of which are empty. Below the editor panes, there is a toolbar with several options: "All Packages" (selected), "Scoped View", "Inst. side" (selected), and "Class side". Below the toolbar, there are two buttons: "Comment" and "New class" (highlighted in yellow). At the bottom of the editor, there is a code editor showing the following code:

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

# An empty test case class



The screenshot shows an IDE window titled "CounterTest". The interface is divided into several panes. On the left, a "Counter" package is visible. The main area shows the "CounterTest" class, which is currently empty. Below the class name, there are tabs for "Counter" and "Filter...". At the bottom, a toolbar contains various icons and text: "All Packages", "Scoped View", "Flat", "Hier.", "Inst. side", "Class side", "Methods", and "Vars". Below the toolbar, there are several tabs: "New class", "Comment", "CounterTest", "setUp", and "Inst. side metr". The main editor area displays the following code:

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

# A first test

CounterTest

Counter CounterTest instance side

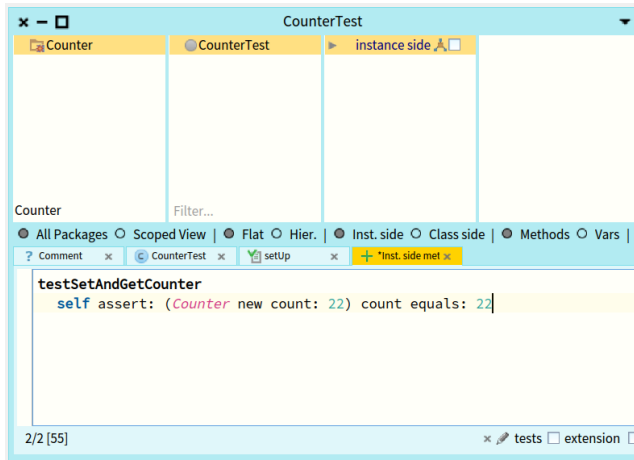
Counter Filter...

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

? Comment x CounterTest x setUp x + \*Inst. side met x

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

# A first test

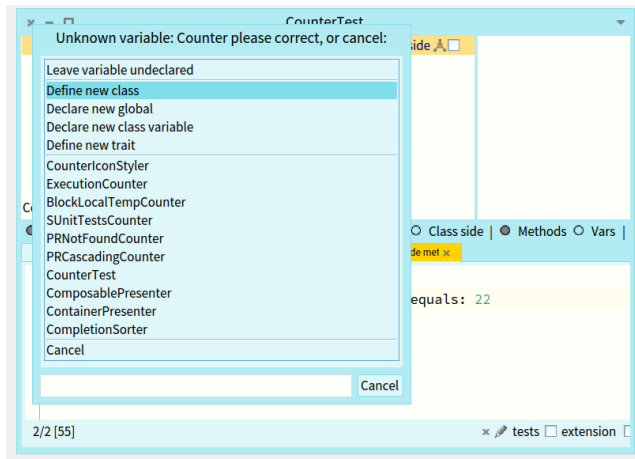


- Method is about to be compiled
- The system knows the class does not exist!



# Define a class

- At compile time...



# Define a class (II)

The screenshot shows an IDE window titled "CounterTest" with a breadcrumb trail: Counter > CounterTest > instance side. A dialog box titled "Information Required" is open, prompting to "Edit class definition:". The dialog contains a text area with the following code:

```
Object subclass: #Counter
  instanceVariableNames: ""
  classVariableNames: ""
  category: 'Counter'
```

At the bottom of the dialog are "OK" and "Cancel" buttons. In the background, a code editor shows the start of a `testSetAndGet` method with `self` and `asser` visible.

# Test defined but not executed

The screenshot shows an IDE window titled "CounterTest>>testSetAndGetCounter". The interface is divided into several panes:

- Left Pane:** Shows a package structure with "Counter" and "CounterTest".
- Middle Pane:** Shows a tree view with "Counter" (selected), "CounterTest", "instance side", and "tests".
- Right Pane:** Shows a list of methods, with "testSetAndGetCounter" selected.

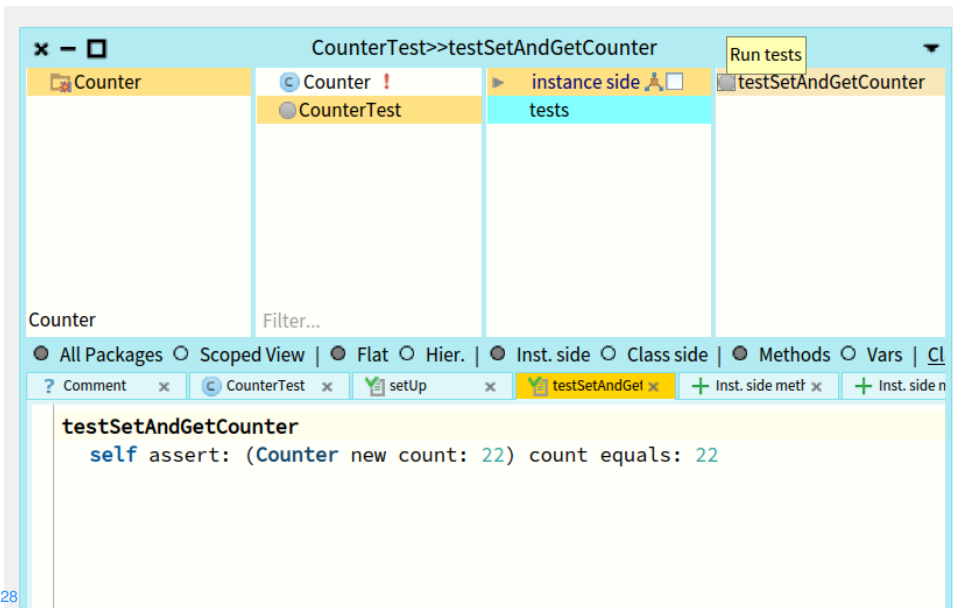
Below the panes, there are view options: "All Packages", "Scoped View", "Flat", "Hier.", "Inst. side" (selected), "Class side", "Methods", "Vars", and "CL".

The bottom pane displays the code for the selected method:

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

The IDE interface includes a toolbar with icons for "Comment", "CounterTest", "setUp", "testSetAndGet", "Inst. side met", and "Inst. side n".

# Running the test



The screenshot shows an IDE window titled "CounterTest>>testSetAndGetCounter". The interface is divided into several panes:

- Left Pane:** A tree view showing the package structure. "Counter" is selected, and "CounterTest" is highlighted.
- Middle Pane:** A tree view showing the test structure. "instance side" is selected, and "tests" is highlighted.
- Right Pane:** A list of test methods. "testSetAndGetCounter" is selected.
- Bottom Pane:** A code editor showing the implementation of the test method:

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

At the bottom of the IDE, there is a toolbar with various options: "All Packages", "Scoped View", "Flat", "Hier.", "Inst. side", "Class side", "Methods", "Vars", and "Cl". Below the toolbar, there are tabs for "Comment", "CounterTest", "setUp", "testSetAndGet", "Inst. side meth", and "Inst. side n".

# 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 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
```

# 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 Evaluator

Type	Variable	Value
implicit	self	a Counter

# Edit the method in the debugger (III)

The screenshot shows an IDE debugger window titled "Instance of Counter did not understand #count:". The window is divided into several sections:

- Stack:** A table showing the call stack. The top frame is highlighted in yellow.
- Source:** A code editor showing the source code for the `count:` method. The line `count := anInteger` is highlighted in yellow.
- Variables:** A table showing the current state of variables.

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

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 an IDE window titled "Instance of Counter did not understand #count:". A dialog box is open with the message "Unknown variable: count please correct, or cancel:". The dialog has three options: "Declare new temporary variable", "Declare new instance variable" (which is highlighted), and "Cancel".

Below the dialog, the "Source" view shows the following code:

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

The "Variables" view at the bottom shows the following table:

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 un...  
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 Evaluator

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
```

# 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 an IDE window titled "CounterTest>>testSetAndGetCounter". The interface is divided into several panes:

- Left Pane:** A tree view showing the project structure. "Counter" is expanded, and "CounterTest" is selected.
- Middle Pane:** A tree view showing the "instance side" of the selected class. "tests" is selected.
- Right Pane:** A tree view showing the "testSetAndGetCounter" test method, which is highlighted with a green circle, indicating it passed.

Below the panes, there are navigation and filter options:

- Buttons for "All Packages", "Scoped View", "Flat", "Hier.", "Inst. side", "Class side", "Methods", and "Vars".
- A search bar labeled "Filter...".

At the bottom, there is a code editor showing the test method:

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

The IDE also shows a tab bar with several open tabs: "Comment", "CounterTest", "setUp", "testSetAndGet", "Inst. side met...", and "Inst. side n...".



# 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

A course by

S. Ducasse, 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/>