

# Xtreme Test Driven Development

Getting a productivity boost

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



# 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 the Xcode IDE interface. At the top, a window titled "Counter" is open. On the left, a project navigator shows a folder named "Counter". The main workspace is divided into three empty panes. Below the workspace, 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 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 open, displaying a tree view on the left with 'Counter' and 'CounterTest' folders. The 'CounterTest' folder is selected, showing an 'instance side' view. The main editor area is empty, with a 'Filter...' input field. The bottom toolbar contains various icons and tabs: 'All Packages', 'Scoped View', 'Flat', 'Hier.', 'Inst. side' (selected), 'Class side', 'Methods', 'Vars', 'New class', 'Comment', 'CounterTest', '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 icons for running tests and a search icon. The main window is titled "CounterTest" and is divided into four panes. The left pane shows the project structure with "Counter" and "CounterTest" folders. The second pane shows the "CounterTest" class with a "Filter..." input field. The third pane shows the "instance side" of the test run, with a "Filter..." input field. The bottom pane shows the test results, with a table containing the following information:

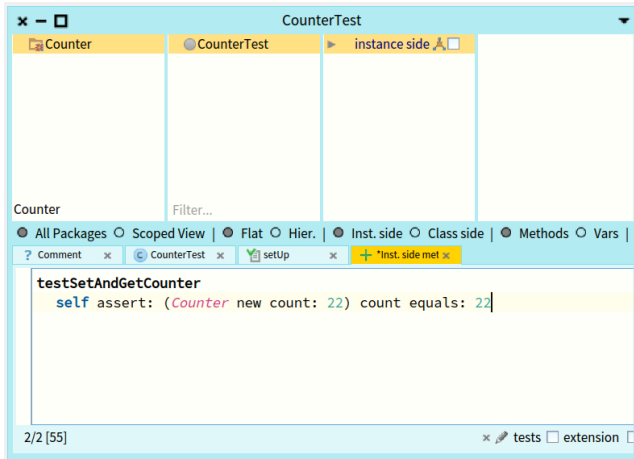
Test Name	Result	Duration	Message
testSetAndGetCounter	Passed	0.001s	

Below the table, the test code is displayed:

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

The bottom status bar shows the current file is "CounterTest" and the test method is "testSetAndGetCounter".

# A first test (II)

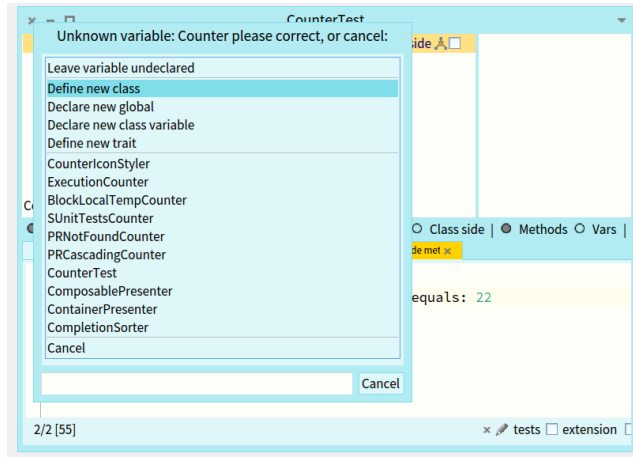


- Method is about to be compiled
- The system knows the class does not exist!

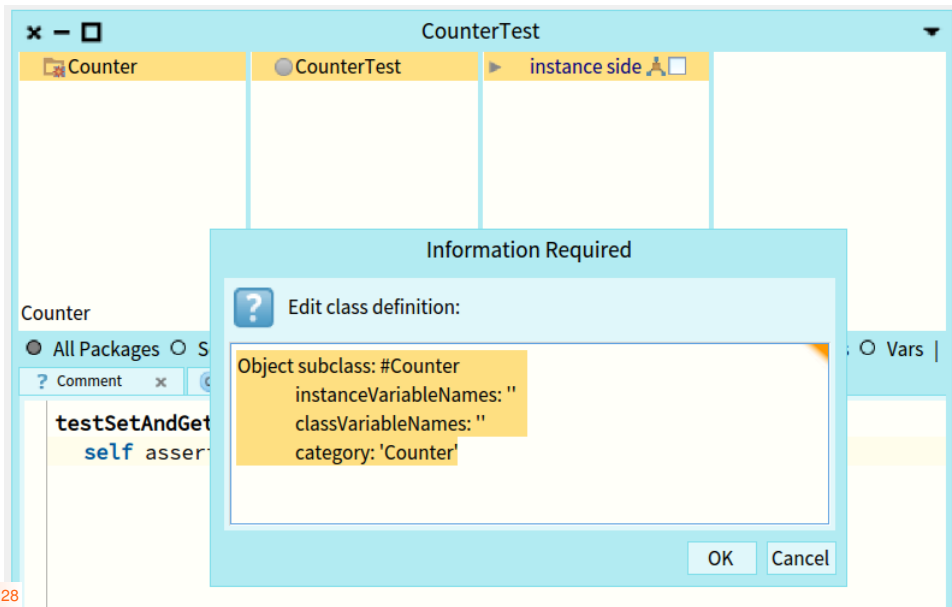


# Define a class

- At compile time...



## Define a class (II)



# Test defined but not executed

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" (selected) 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", "Inst. side mettr", and "Inst. side n". 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 IDE's test runner interface. The title bar reads "CounterTest>>testSetAndGetCounter". The interface is divided into several panes:

- Left Pane:** A tree view showing the project structure. "Counter" is selected.
- Middle-Left Pane:** A list of test classes. "Counter" (with a red exclamation mark) and "CounterTest" are listed.
- Middle-Right Pane:** A list of test methods. "instance side" and "tests" are listed.
- Right Pane:** A list of test results. "testSetAndGetCounter" is listed.

Below the panes, there is a toolbar with various options:

- View Options:** "All Packages", "Scoped View", "Flat", "Hier.", "Inst. side", "Class side", "Methods", "Vars", and "C".
- Test Runners:** "Comment", "CounterTest", "setUp", "testSetAndGet", "Inst. side meth", and "Inst. side n".

The main area displays the test code for **testSetAndGetCounter**:

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

# 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

Type	Variable	Value
implicit	self	a Counter

# Edit the method in the debugger

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 runtime error dialog box open. The error message is "Unknown variable: count please correct, or cancel:". The dialog box has three options: "Declare new temporary variable", "Declare new instance variable" (which is highlighted), and "Cancel". Below these options is a text input field and a "Cancel" button.

The background shows the IDE's interface. The top bar has "Bytecode" and "GT" tabs. The "Source" tab is active, showing the following code:

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



The "Variables" tab is also active, showing a table of variables:


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


The "Package" pane on the right shows the following packages:

- Other
- Package
- Counter
- Counter
- SUnit-Core
- [self setUp. self performTest SUnit-Core]

# 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
- 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

```
count: anInteger  
  self shouldBelImplemented
```

- `shouldBelImplemented` 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. On the left, a "Counter" class is shown. The middle pane displays a tree view with "Counter" (marked with a red exclamation mark) and "CounterTest" (marked with a green circle). The right pane shows the "testSetAndGetCounter" method, which is also marked with a green circle. Below the tree view, there are tabs for "Counter", "Filter...", "All Packages", "Scoped View", "Flat", "Hier.", "Inst. side", "Class side", "Methods", "Vars", and "CL". The "Inst. side" tab is selected. The bottom pane shows the code for the "testSetAndGetCounter" method:

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

The code is displayed in a light yellow background, indicating a successful test run.



# 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



Produced as part of the course on <http://www.fun-mooc.fr>

# 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/>