

Powerful Tools for Live Development with Pharo

<http://stephane.ducasse.free.fr>

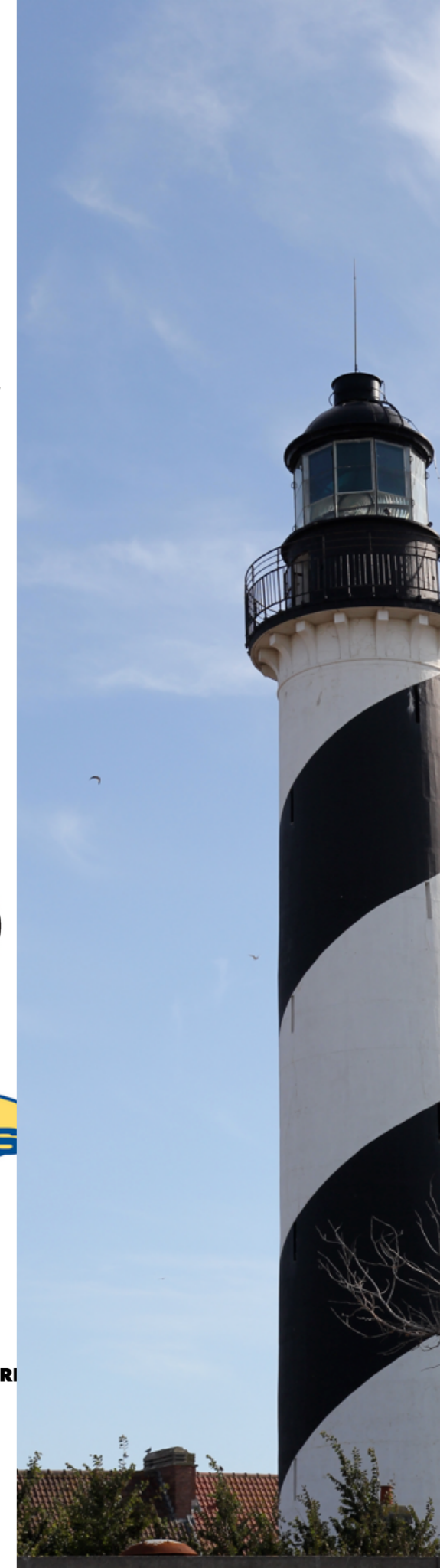
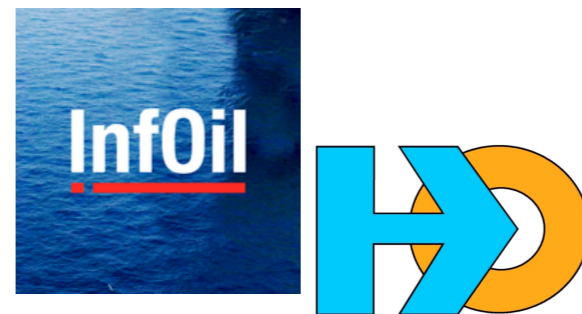
<http://www.pharo.org>

Inria



Université
de Lille





A journey in a live environment and its companion tools

- Pharo in 7 min
- Some advanced features
- Some tools
- Pharo is research friendly

Pharo!

- System: **Pure** object language + full IDE
- **Powerful, elegant** and **fun** to program
- **Living** system under your fingers
- Works on Mac OSX, Linux(es), iOS, Windows, Pi, and “*android*”
- 100% MIT

Pharo in Numbers

13 releases since 2008
Language Core + IDE +
Tools + Frameworks
710 packages (tests
included)

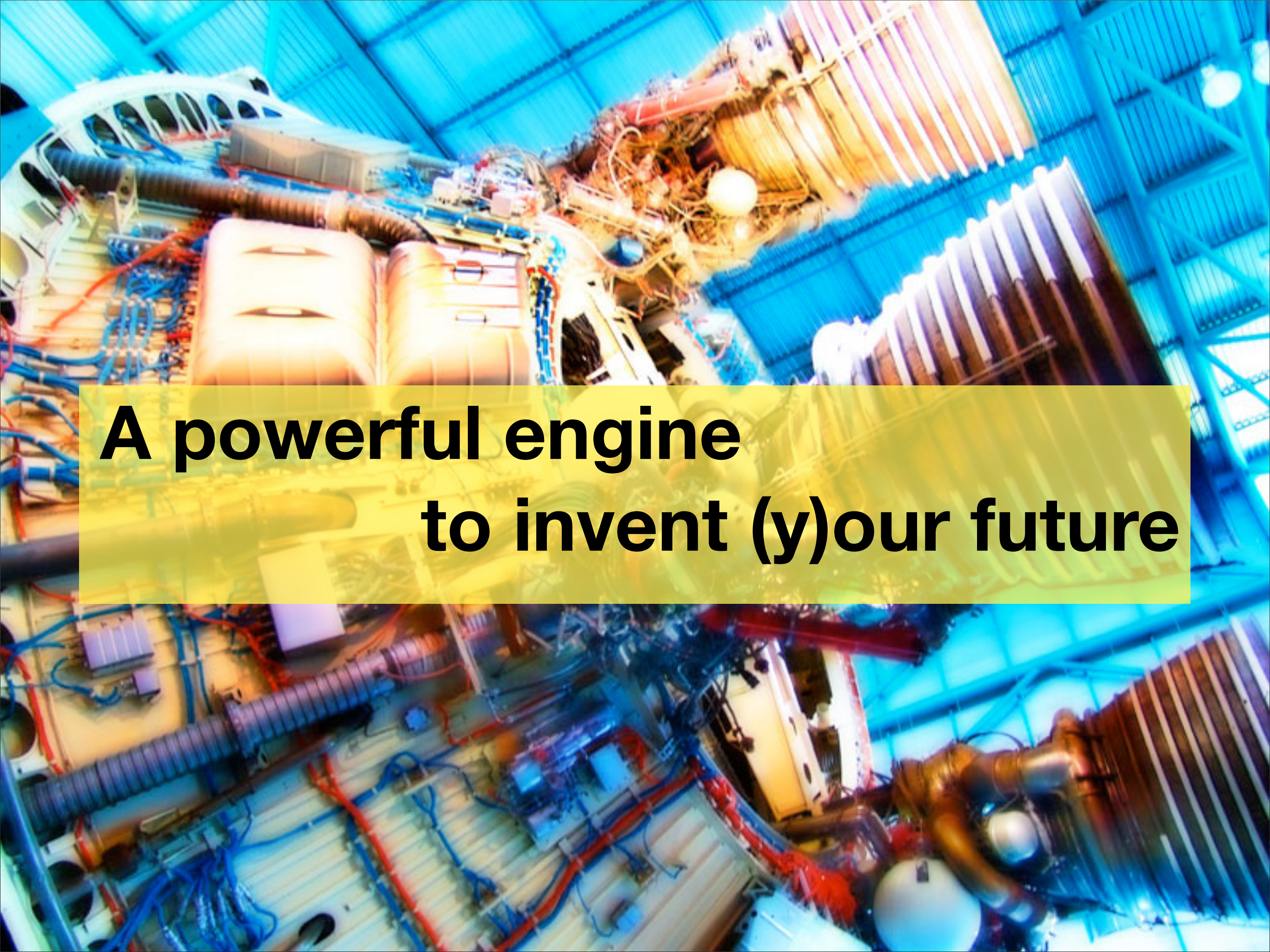
~ **27000 tests**
5 platforms
9 400 classes
130 000 methods
61 Mb (64 bits)

<http://github.com/pharo-project/Pharo> (~220
forks, 15/123
international
contributors)


Growing ecosystem
polymath
pharo-graphics
pharo-gis
pharo-container
pharo-ai



**Pharo is our vehicle
We improve it everyday**



**A powerful engine
to invent (y)our future**



**An ecosystem where
innovation/business bloom**

trentosur

Soluciones móviles para retail y trade marketing

Nos enfocamos en lo que importa del negocio sin perder de vista los detalles de su implementación.

- Primer móvil
- Plataforma Android
- En la nube

PharoCloud

Overview Pricing Blog Login Sign Up

Pharo platform as a Service: put your Smalltalk web-application online at PharoCloud in just 3 clicks

Try it for FREE

Romax TECHNOLOGY

Wind Energy

Pioneering new ways of maximising sustainable wind energy yields. Our products and services optimise asset availability, wind turbine performance and drivetrain reliability. We work with owners, operators, manufacturers, insurers and service providers worldwide.

Get in touch

WEBDRUCK.CH

Web-To-Print Solution

- Design and create individual printed matter
- eShop with credit card payment
- High quality PDF output with Printing Process integration
- Thousands of orders for seven Swiss printing companies

Quuve

Some Success Stories @ pharo.org/

success

Yesplan is veelzijdige software voor het efficiënt plannen van evenementen.

Yesplan is uiterst gebruiksvriendelijk, flexibel en makkelijk te koppelen met andere software.

Dedicated and cost-effective tools for software evolution

- Dedicated Analyses
- Dedicated tools
- Decision making

NORRIZIK-COM

The world's first online platform fully supporting risk-based test management.

BETTER FASTER CHEAPER

CSOB

airflowing

Organize your creative work

Plans and Pricing

Manage your simple way

OBJECT PROFILE

CMSBOX

Das Content Management mit System

100% Inline-Editor

2denker

Continuous API Testing

keep your services under control 24/7

t3

Elegant!

- Full syntax on a postcard
- Simple and powerful objet model

Pharo

```
exampleWithNumber: x

<syntaxOn: #postcard>
"A ""complete"" Pharo syntax"
| y |
true & false not & (nil isNil)
  ifFalse: [ self perform: #add: with: x ].
y := thisContext stack size + super size.
byteArray := #[2 2r100 8r20 16rFF].
{ -42 . #($a #a #'I''m' 'a' 1.0 1.23e2 3.14s2 1) }
do: [ :each |
  | var |
  var := Transcript
  show: each class name;
  show: each printString ].
^ x < y
```

method name, parameter, pragma, comment, local variable, binary message, unary message, boolean literals, nil literal, block, keyword message, pseudo variables, assignment, instance variable, integer literals, byte array, array generated at runtime, literal array, symbols, character, string, scaled decimal, local block variable, block parameter, global variable, cascade, keyword message, return instruction, other method definition examples: unary, + binaryMessageArgument, keyword: arg, keyword: arg1 withTwo: arg2

<https://www.pharo.org>

PLACE
STAMP
HERE

.....

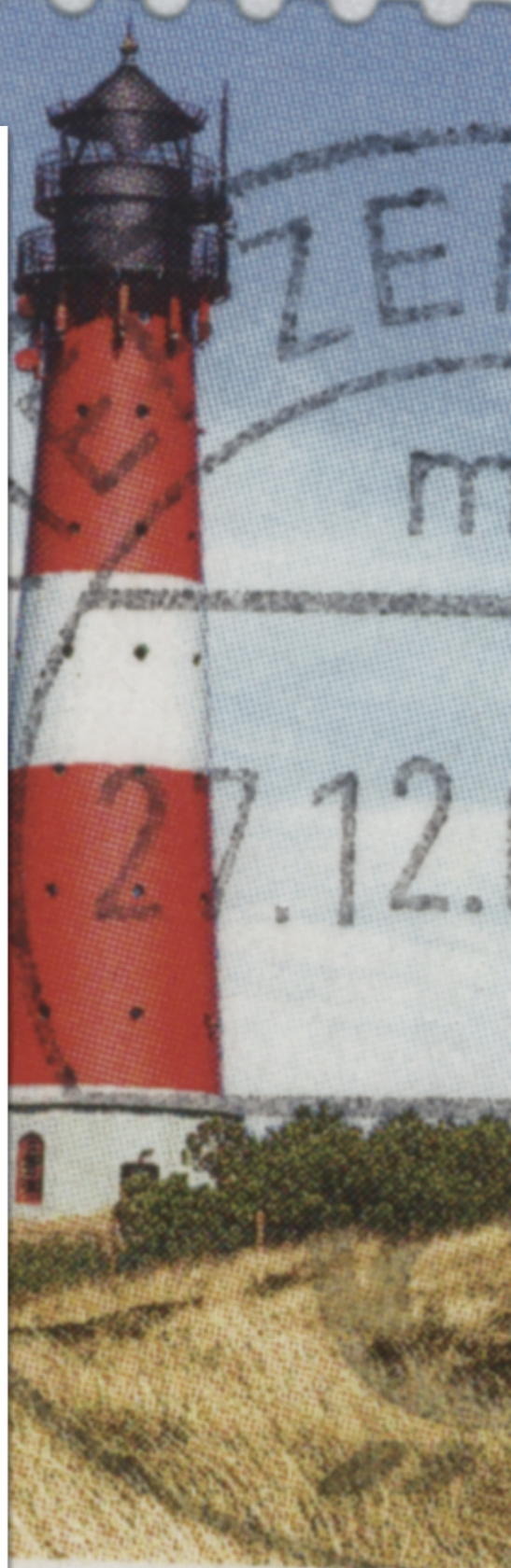
.....

.....

.....

.....

* H Ö R N U M



A Pure World of Objects

Only

objects + messages +

closures

mouse, booleans, arrays, numbers, strings, windows, scrollbars, canvas, files, trees, compilers, sound, url, socket, fonts, text, collections, stack, shortcut, streams, ...

A Fully Uniform Model

- Dynamically typed
- **Everything** is an object instance of a class
- All methods are public **virtual**
- All attributes are **protected**
- **Single** inheritance with **traits**

Less is more!

No constructors, no static methods, no operators

No type declaration, no primitive types,

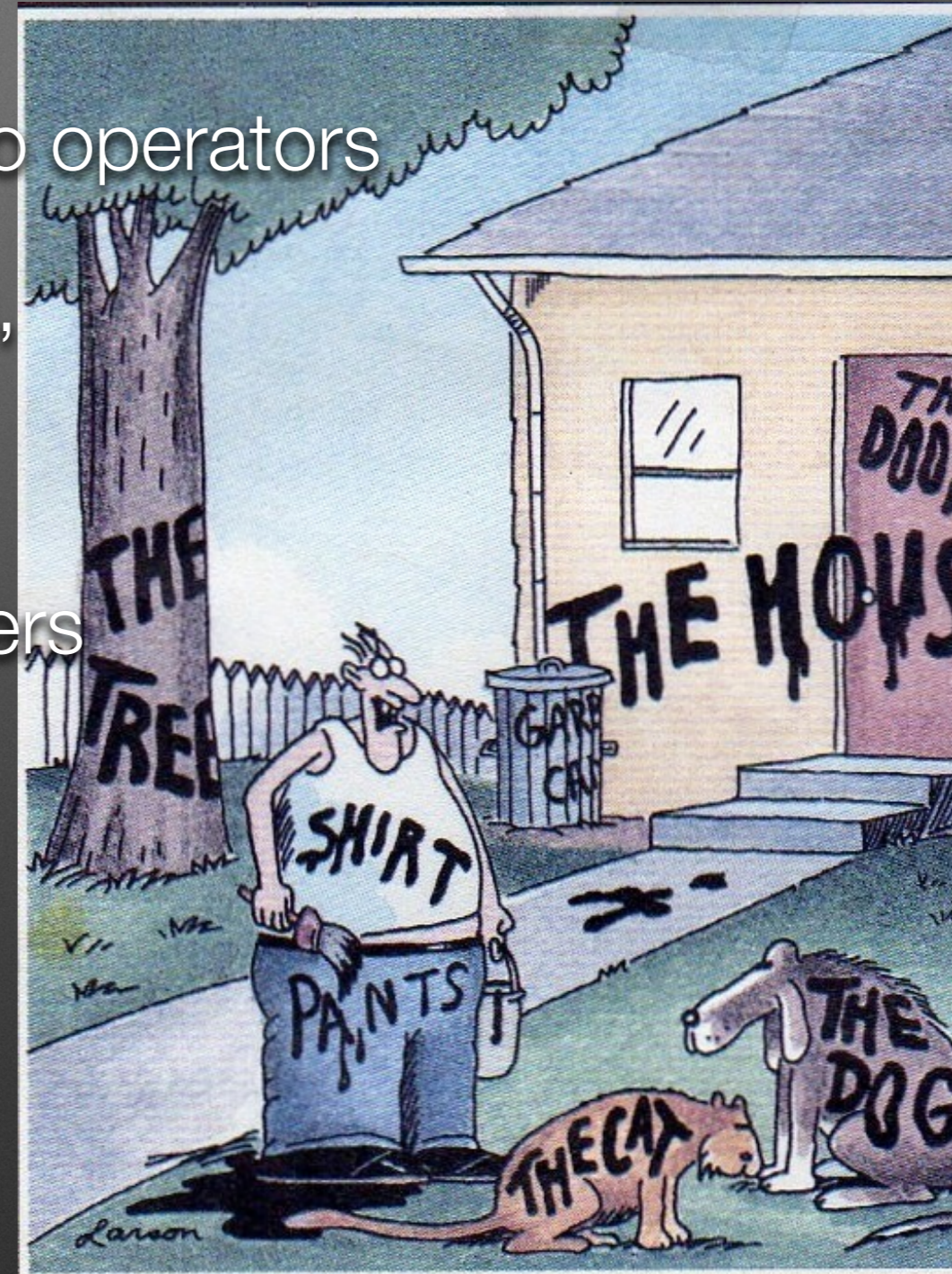
No interfaces, no need for factory

No packages/private/protected modifiers

No parametrized types

No boxing/unboxing

Still powerful



"Now! ... That should clear up a few things around here!"

Pharo is highly immersive

A large, clear aquarium tank filled with water. In the center, a large spotted shark swims horizontally. Above it, another spotted shark swims diagonally. The water is filled with many smaller, silvery fish. In the foreground, the silhouettes of several people are visible, looking into the tank. The background is a deep blue color.

Immersing...

Pharo is not a blackbox

Everything is **fully inspectable** and
reflective

A diver in a black wetsuit and scuba gear is swimming in a large aquarium. The water is filled with numerous stingrays of various sizes, some swimming near the sandy bottom and others higher up. The background features large, textured rock formations. The lighting is a mix of blue and green, creating a vibrant underwater atmosphere. The text "You are immersed and interacting with objects" is overlaid in white, bold font across the upper portion of the image.

**You are immersed and
interacting with objects**

Workspace

```
| elements lay |
```

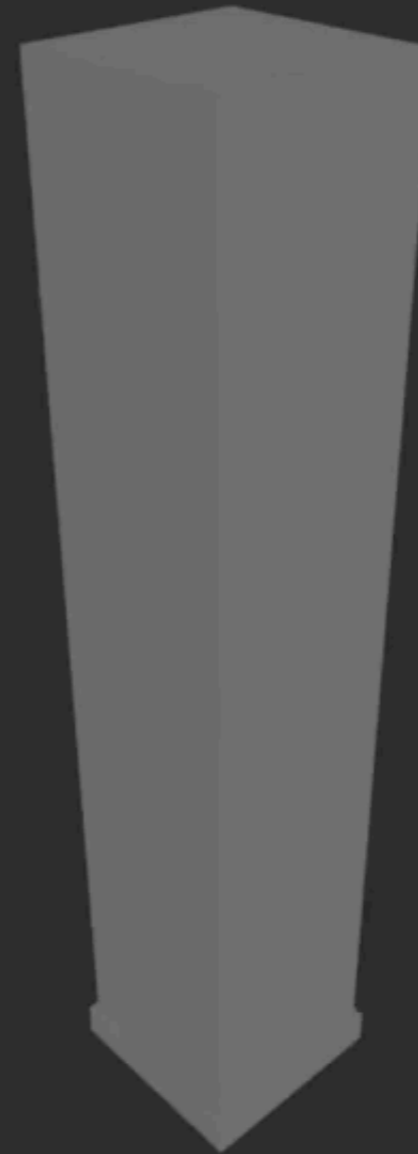
```
elements := (1 to: 5) collect: [ :ob |  
  (R3CubeShape new) elementOn: ob ].
```

```
I
```

```
lay := R3WallLayout new.  
lay on: elements.
```

```
UberPresenter present: elements
```

Uber Presenter



**We can do the same with
web app, sockets, networks,
sensors, living programming....**



**Hackers
scripting live
the coffee
machine**



Selected Infrastructure

Fully Written in Itself



Selected features

- First class instance variables (daemons, relationships...)
- Fast resumable exceptions
- Runtime classes and objects migration
- Customizable compiler
- Serializable and shareable execution stack
- Optional system virtualization
- Fully bootstrapped kernel(s) (down to 200kb)

Advanced reflective layer

- Versatile AST annotations and transformations @ runtime
- Full stack reification (continuations, exceptions...)
- Instance enumeration
- Causally connected “Software as Objects”
- Atomic bulk object swapping
- ... more but no time for that



Tools are our friends

How to find information?

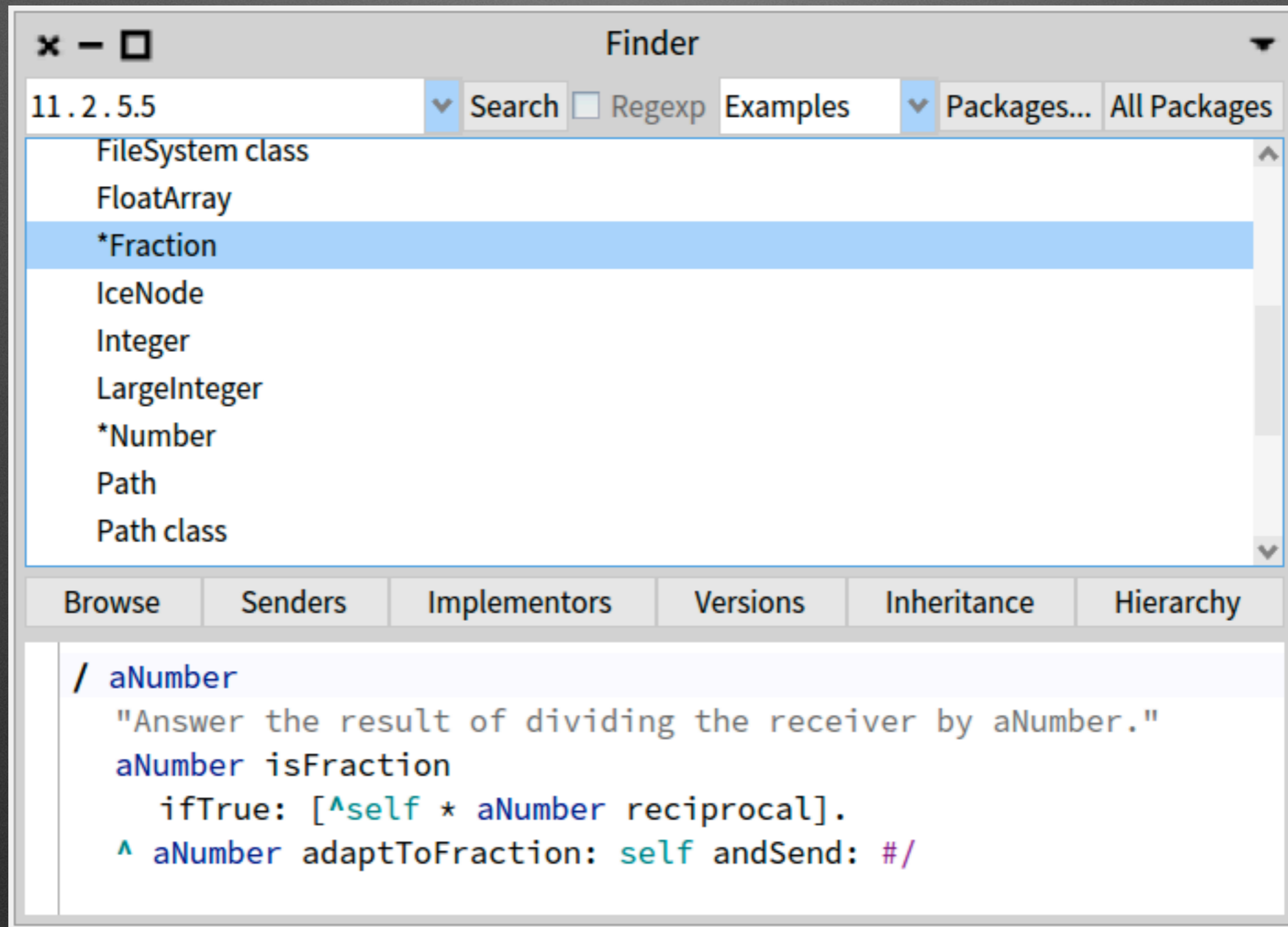
- Libraries are large
- You know what you want
- You do not know how to express it

11 ??? 2 should give 5

The screenshot shows the Finder application window with the search results for the query '11 // 2 --> 5'. The results are listed under the '11 // 2 --> 5' category, including 'Collection', 'Duration', '*Integer', 'LargeInteger', '*Number', 'Point', and '*SmallInteger'. Below the search results, there are tabs for 'Browse', 'Senders', 'Implementors', 'Versions', 'Inheritance', and 'Hierarchy'. The 'Browse' tab is selected, and the content area displays the following text:

```
Use an example to find a method in the system.  
  
    'a'. 'b'. 'ab'      will find the message #, for strings  
concatenation  
    2. -2              will find the message #negated  
    3. 6               will find the message #factorial
```

11 ??? 2 should give 5.5



The screenshot shows a window titled "Finder" with a search bar containing "11.2.5.5". The search results list several classes, with "*Fraction" selected. Below the list, there are tabs for "Browse", "Senders", "Implementors", "Versions", "Inheritance", and "Hierarchy". The "Browse" tab is active, showing the following code snippet:

```
/ aNumber
"Answer the result of dividing the receiver by aNumber."
aNumber isFraction
  ifTrue: [^self * aNumber reciprocal].
^ aNumber adaptToFraction: self andSend: #/
```

What are the messages send to \$0 that return true

The screenshot shows the Ruby Finder tool interface. The search bar contains '\$0.isDigit'. The search results are displayed in a list view, showing various messages that return true for the receiver \$0. The 'Character' class is highlighted in blue. Below the list, there are tabs for 'Browse', 'Senders', 'Implementors', 'Versions', 'Inheritance', and 'Hierarchy'. The 'isDigit' method is selected, and its implementation is shown in the bottom pane.

```
$0.isAlphaNumeric --> true
$0.isCharacter --> true
$0.isCompletionCharacter --> true
$0.isDecimalDigit --> true
$0.isDigit --> true
Character
$0.isLiteral --> true
$0.isOctetCharacter --> true
$0.isSafeForHTTP --> true
$0.shouldBePrintedAsLiteral --> true
$0.tokenish --> true
```

isDigit

```
"Return whether the receiver is a digit."
"$1.isDigit >>> true"
"$0.isDigit >>> true"

^ self.characterSet.isDigit: self
```

Tools

- Shape our mind....
- Pharo has moldable tools: **you CAN adapt them to you and your process** and not the inverse
- Build fast **your own** tools

Pharo has **amazing**
moldable tools

**Customizable object
interaction/presentations**

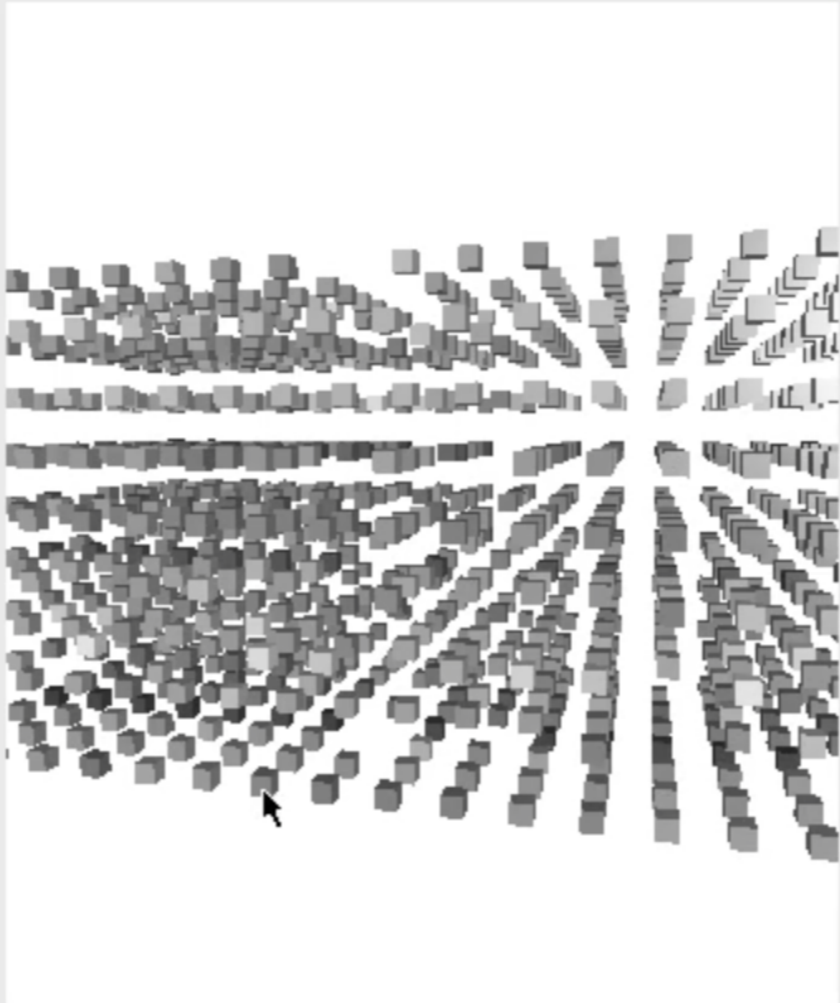
Inspecting live a 3D object

Playground

```
data := TestData new data.  
  
cube := MatrixCube new initWithData: data.  
cube view
```

a RWView

Raw Live Meta

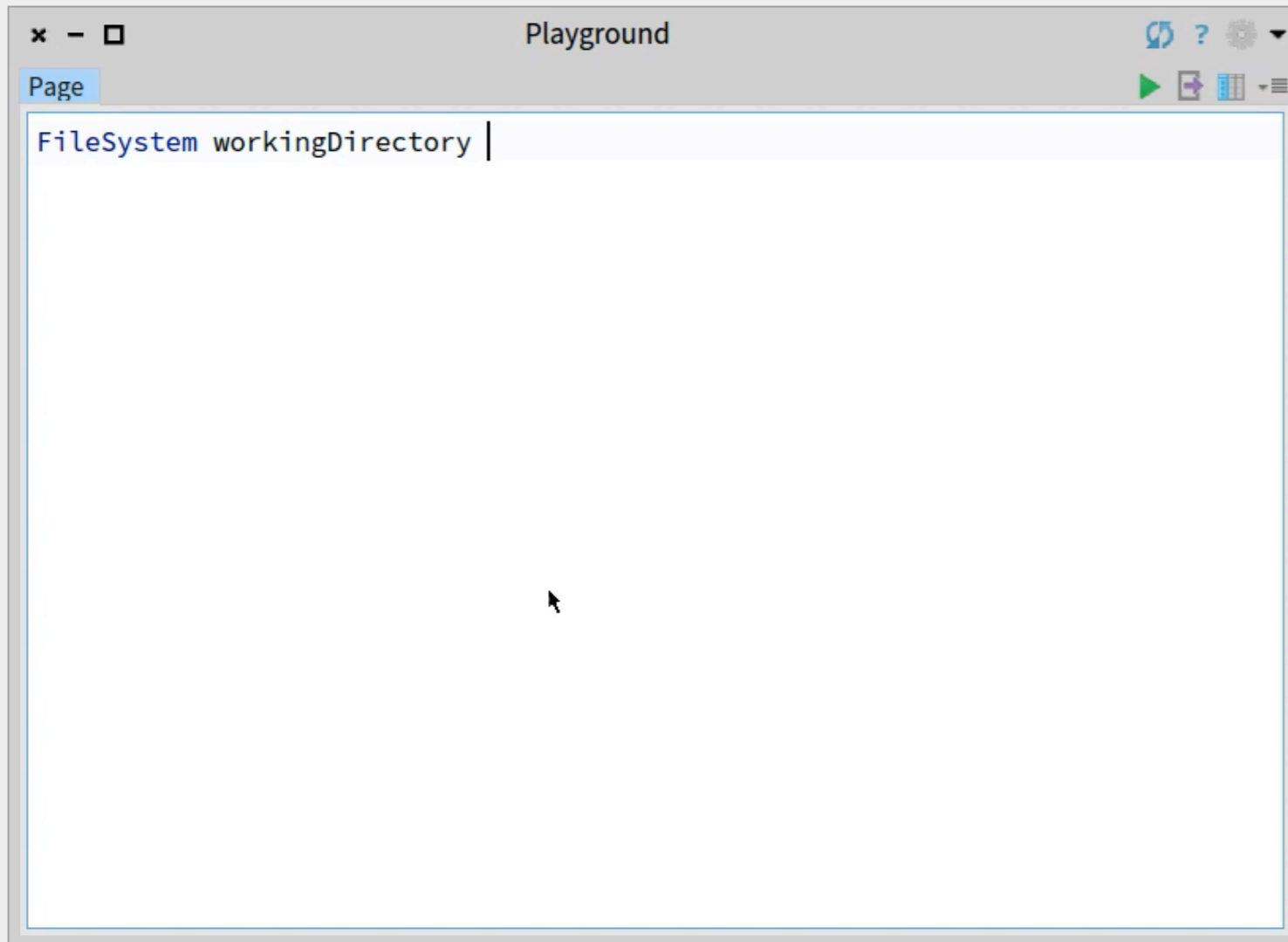


an Array [4 items] ('2000-01-01' 'Malawi' 'France'... x

Index	Item
1	'2000-01-01'
2	'Malawi'
3	'France'
4	'400.0'

enter search query (example: "each = 5")

The views of a file reference



It is cool but it is not magic
You can define your own

Implementing a pane!

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

- Left Pane:** A tree view showing a package structure under "Public". The "Extensions" folder is expanded, listing various file system-related packages like "FileSystem-Disk", "FileSystem-Memory", etc.
- Middle Pane:** A list of classes and methods. "AbstractFileReference" is selected, showing its subclasses like "FileLocator", "FileReference", "FileSystem", etc.
- Right Pane:** A list of methods. "instance side" is selected, showing methods like "extensions", "flags", "ToDeprecate", "accessing", etc.
- Bottom Pane:** A code editor showing the implementation of the "gtInspectorPngIn" method. The code is as follows:

```
gtInspectorPngIn: composite
  <gtInspectorPresentationOrder: 0>
  composite morph
    title: 'Picture';
    display: [ self binaryReadStreamDo: [ :stream | PNGReadWriter formFromStream: stream ] ];
    when: [ self isFile and:
      [ self mimeTypes notNil and:
        [ self mimeTypes first matches: ZnMimeType imagePng ] ] ]
```

At the bottom of the IDE, there is a status bar showing "8/8 [9]" and "GT-InspectorExtensions-Core extension F +".

Files are boring...
What about *inside* the system?



A class is an object we can inspect!

The screenshot shows a Python Playground window with the following components:

- Page:** A text editor on the left containing the word `Point`.
- Playground:** The main interface with a toolbar containing a play button, a plus icon, a list icon, and a menu icon.
- Tab:** A tab titled "a Point class (Point)" is active.
- Inspector:** A table-like view showing the attributes of the object. The table has two columns: "Variable" and "Value".

Variable	Value
self	Point
superclass	Object
methodDict	a MethodDictionary [103 items] (size 103)
format	65538
layout	a FixedLayout
organization	a ClassOrganization
subclasses	nil
name	#Point
classPool	a Dictionary [0 items] /

Below the table, the object's string representation is shown as `"Point"`, and the `self` attribute is highlighted in blue.

“A class has a method dictionary”
they said... let us verify

The screenshot shows a Ruby Playground window with two panes. The left pane displays the internal structure of a `Point` class, and the right pane displays the contents of its `MethodDictionary`.

Left Pane: a Point class (Point)

Variable	Value
self	Point
superclass	Object
methodDict	a MethodDictionary [103 items] (size 103)
format	65538
layout	a FixedLayout
organization	a ClassOrganization
subclasses	nil
name	#Point

Below the table, the source code for the `Point` class is visible:

```
"Point"  
self
```

Right Pane: a MethodDictionary [103 items] (size 103)

Key	Value
#reflectedAbout:	Point>>#reflectedAbout:
#rotateBy:centerAt:	Point>>#rotateBy:centerAt:
#adaptToNumber:andSend:	Point>>#adaptToNumber:andSend:
#squaredDistanceTo:	Point>>#squaredDistanceTo:
#adaptToCollection:andSend:	Point>>#adaptToCollection:andSend:
#theta	Point>>#theta
#transposed	Point>>#transposed
#-	Point>>#-
#fourDirections	Point>>#fourDirections
#crossProduct:	Point>>#crossProduct:
#scaleFrom:to:	Point>>#scaleFrom:to:
#veryDeepCopyWith:	Point>>#veryDeepCopyWith:

Dissecting one method object

The screenshot shows the Ruby Inspector tool window titled "Inspector on a CompiledMethod (Point>>#degrees)". The main content area displays the internal structure of the method object, organized into a table with "Variable" and "Value" columns. The variables listed are self, literal1 through literal8, and bc 89. Below the table, the source code for the method is shown, starting with the string "Point>>#degrees" and the keyword self.

Variable	Value
{ } self	Point>>#degrees
▶ Σ literal1	90.0
▶ Σ literal2	270.0
▶ ¶ literal3	#asFloat
▶ ¶ literal4	#arcTan
▶ ¶ literal5	#radiansToDegrees
▶ Σ literal6	360.0
▶ Σ literal7	180.0
▶ ¶ literal8	#ifTrue:ifFalse:
▶ Σ bc 89	0

```
"Point>>#degrees"  
self
```

I do not want to be a compiler!

Inspector on a CompiledMethod (Point>>#degrees)

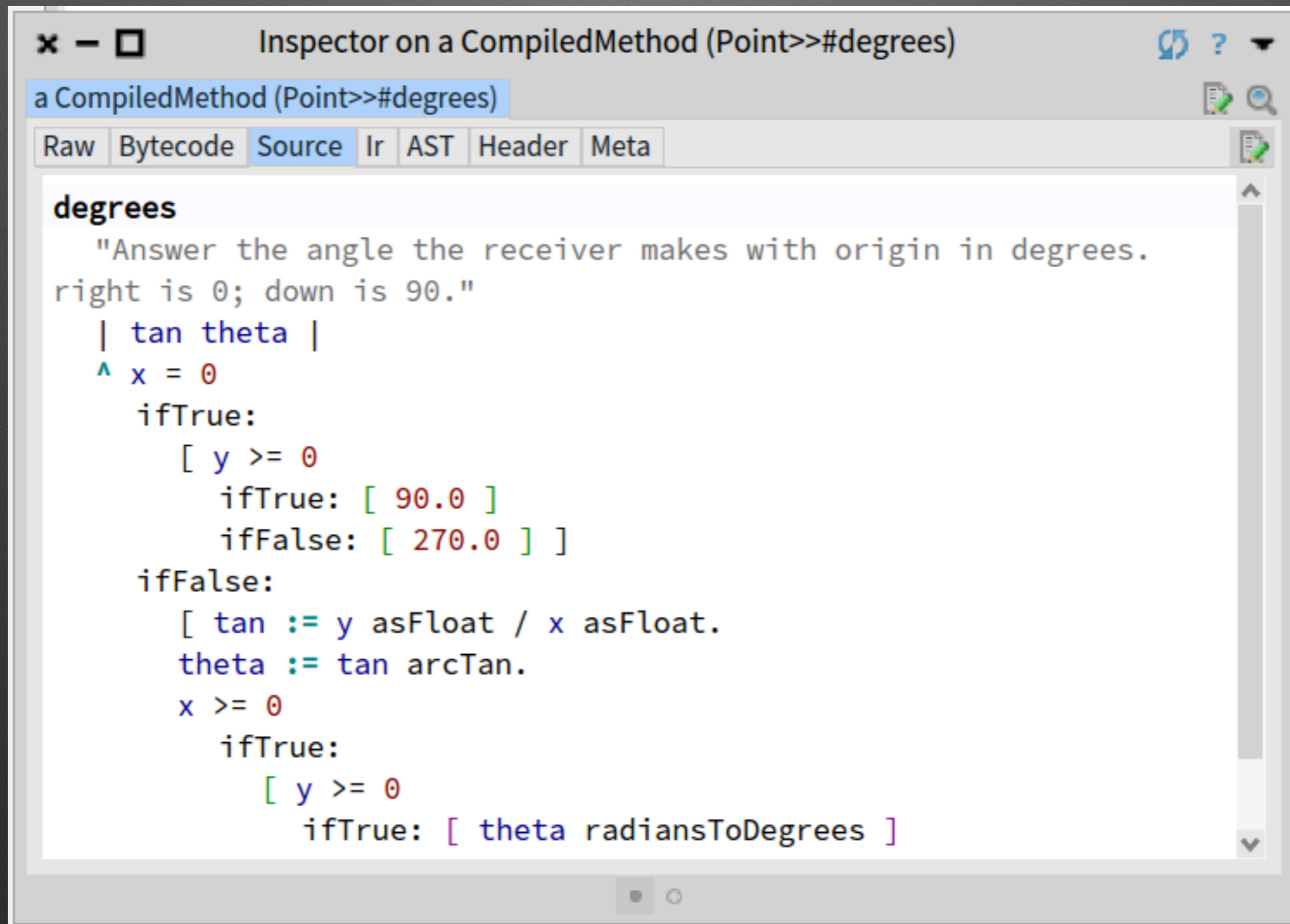
a CompiledMethod (Point>>#degrees)

Raw Bytecode Source Ir AST Header Meta

Variable	Value
▶ Σ bc 89	0
▶ Σ bc 90	117
▶ Σ bc 91	182
▶ Σ bc 92	172
▶ Σ bc 93	9
▶ Σ bc 94	1
▶ Σ bc 95	117
▶ Σ bc 96	181
▶ Σ bc 97	153
▶ Σ bc 98	32

"Point>>#degrees"
self

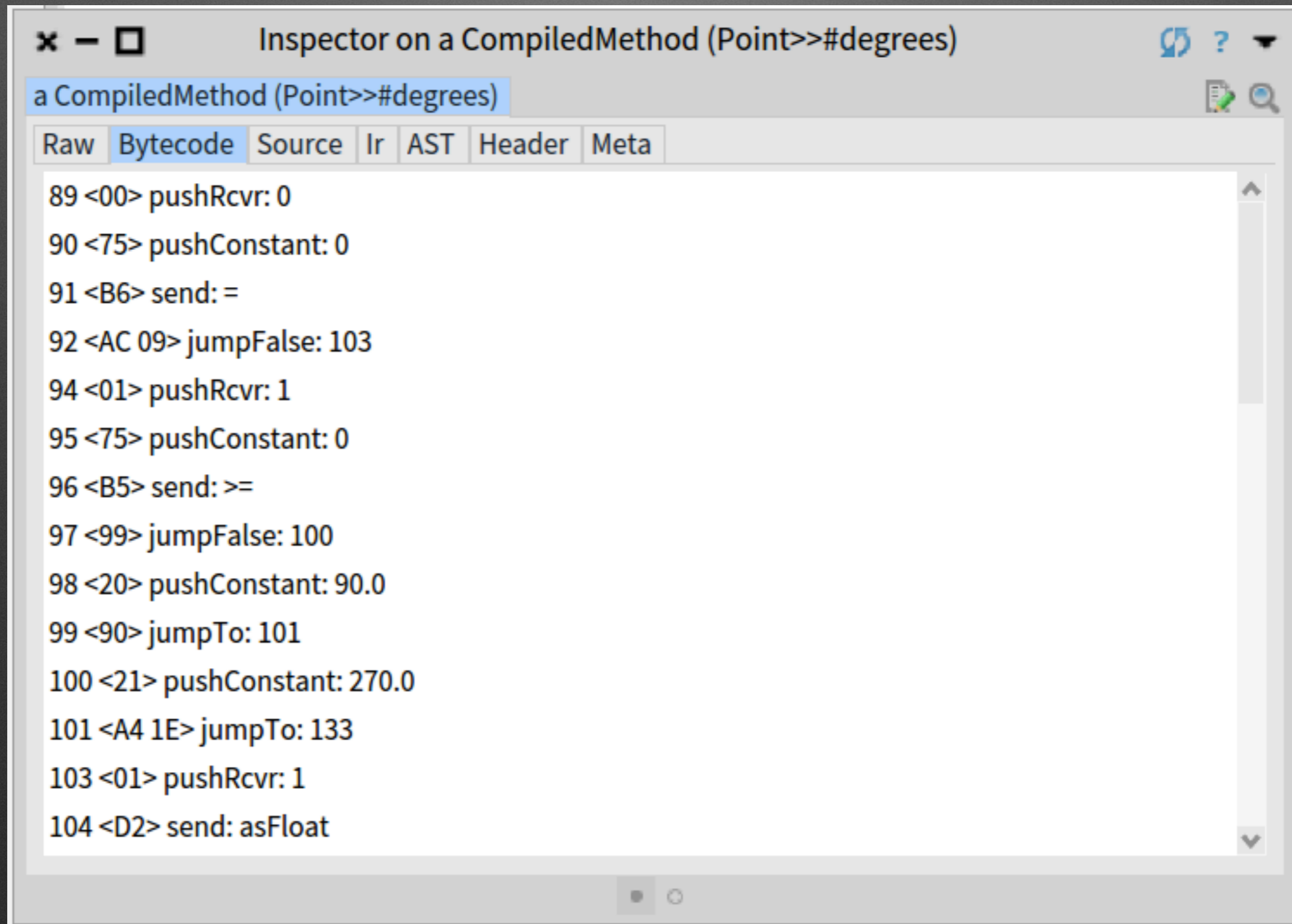
It looks like a method



The screenshot shows an IDE window titled "Inspector on a CompiledMethod (Point>>#degrees)". The window contains a tabbed interface with "Source" selected. The source code is as follows:

```
degrees
  "Answer the angle the receiver makes with origin in degrees.
  right is 0; down is 90."
  | tan theta |
  ^ x = 0
  ifTrue:
    [ y >= 0
      ifTrue: [ 90.0 ]
      ifFalse: [ 270.0 ] ]
  ifFalse:
    [ tan := y asFloat / x asFloat.
      theta := tan arcTan.
      x >= 0
        ifTrue:
          [ y >= 0
            ifTrue: [ theta radiansToDegrees ]
```

Numbers are not that obscure



The screenshot shows a debugger window titled "Inspector on a CompiledMethod (Point>>#degrees)". The window displays the bytecode for a compiled method. The bytecode is organized into a table with columns for "Raw", "Bytecode", "Source", "Ir", "AST", "Header", and "Meta". The "Bytecode" column is selected, and the following instructions are visible:

```
89 <00> pushRcvr: 0
90 <75> pushConstant: 0
91 <B6> send: =
92 <AC 09> jumpFalse: 103
94 <01> pushRcvr: 1
95 <75> pushConstant: 0
96 <B5> send: >=
97 <99> jumpFalse: 100
98 <20> pushConstant: 90.0
99 <90> jumpTo: 101
100 <21> pushConstant: 270.0
101 <A4 1E> jumpTo: 133
103 <01> pushRcvr: 1
104 <D2> send: asFloat
```

And mapping them to the good abstraction helps

The image shows a screenshot of the Ruby Inspector application. The main window is titled "Inspector on a CompiledMethod (Point>>#degrees)". It is split into two panes. The left pane shows the AST of a compiled method, and the right pane shows the source code of the corresponding RBMessageNode.

Left Pane: a CompiledMethod (Point>>#degrees)

- Raw
- Bytecode
- Source
- Ir
- AST
- Header
- Meta

```
▼ RBMethodNode(degrees "Answer the angle the receiver makes with)
  ▼ RBSequenceNode(| tan theta | ^ x = 0 ifTrue: [ y >= 0 ifTrue:)
    RBTemporaryNode(tan)
    RBTemporaryNode(theta)
  ▼ RBReturnNode(^ x = 0 ifTrue: [ y >= 0 ifTrue: [ 90.0 ]
    ▼ RBMessageNode(x = 0 ifTrue: [ y >= 0 ifTrue: [ 90.0 ]
      ▼ RBMessageNode(x = 0)
        RBInstanceVariableNode(x)
        RBLiteralValueNode(0)
      ▼ RBBlockNode([ y >= 0 ifTrue: [ 90.0 ] ifFalse: [ 270.0 ]])
        ▼ RBSequenceNode(y >= 0 ifTrue: [ 90.0 ] ifFalse: [ 270.0 ]
          ▼ RBMessageNode(y >= 0 ifTrue: [ 90.0 ] ifFalse: [ 270.0 ]
            ▶ RBMessageNode(y >= 0)
            ▶ RBBlockNode([ 90.0 ])
            ▶ RBBlockNode([ 270.0 ])
```

Right Pane: a RBMessageNode (RBMessageNode(y >= 0))

- Raw
- Tree
- Scopes
- Source cc...
- AST Dump
- Meta

```
degrees
  "Answer the angle the receiver makes with
  origin in degrees. right is 0; down is 90."
  | tan theta |
  ^ x = 0
  ifTrue:
    [ y >= 0
      ifTrue: [ 90.0 ]
      ifFalse: [ 270.0 ] ]
  ifFalse:
    [ tan := y asFloat / x asFloat.
      theta := tan arcTan.
      x >= 0
        ifTrue:
          [ y >= 0
            ifTrue: [ theta radiansToDegrees ]
            ifFalse: [ 360.0 + theta
              radiansToDegrees ] ]
```

Yes pushRcvr: 1 means the second field!

The image shows a debugger window titled "Inspector on a CompiledMethod (Point>>#degrees)". It contains two panes. The left pane, titled "a CompiledMethod (Point>>#degrees)", shows a list of bytecode instructions. The right pane, titled "a SymbolicBytecode (94 <01> pushRcvr: 1)", shows the corresponding symbolic code for the selected instruction.

CompiledMethod (Point>>#degrees) instructions:

- 89 <00> pushRcvr: 0
- 90 <75> pushConstant: 0
- 91 <B6> send: =
- 92 <AC 09> jumpFalse: 103
- 94 <01> pushRcvr: 1**
- 95 <75> pushConstant: 0
- 96 <B5> send: >=
- 97 <99> jumpFalse: 100
- 98 <20> pushConstant: 90.0
- 99 <90> jumpTo: 101
- 100 <21> pushConstant: 270.0
- 101 <A4 1E> jumpTo: 133
- 103 <01> pushRcvr: 1
- 104 <D2> send: asFloat
- 105 <00> pushRcvr: 0
- 106 <D2> send: asFloat

SymbolicBytecode (94 <01> pushRcvr: 1) source:

```
origin in degrees. right is 0; down is 90."
| tan theta |
^ x = 0
  ifTrue:
    [ y >= 0
      ifTrue: [ 90.0 ]
      ifFalse: [ 270.0 ] ]
  ifFalse:
    [ tan := y asFloat / x asFloat.
      theta := tan arcTan.
      x >= 0
        ifTrue:
          [ y >= 0
            ifTrue: [ theta radiansToDegrees ]
            ifFalse: [ 360.0 + theta
radiansToDegrees ] ]
          ifFalse: [ 180.0 + theta
radiansToDegrees ] ]
```

Pharo Pro devs do XtremeTDD

- Get **productivity boost**
- Xtreme TDD
 - write test, test fails and
 - **code in debugger**

Counter

- ▶ Commander2
- ▶ Commander2-Deprecation
- ▶ Commander2-Tests
- ▶ Commander2-UI
- ▶ Commander2-UI-Tests
- ▶ Compression
- ▶ Compression-Tests
- ▶ ConfigurationCommandLi
- ▶ ConfigurationCommandLi
- ▶ Counter
- ▶ Debugger-Actions

Filter... Filter...

All Packages Scoped View | Inst. side Class side

? Comment x + New class x

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

Slots

Hot update on the fly
customizable debugger

Halt

Bytecode

Stack

Proceed Restart Into Over Through

PDFCellElement	getSubElementsWith:styleSheet:
PDFCellElement(PDFComposite)	generateCodeSegmentsCollectionWi
PDFCellElement(PDFComposite)	generateCodeSegmentWith:styleShe
PDFDataTableElement(PDFComposite)	generateCodeSegmentsCollectionWi [:aSubElement aSubElement generateCodeSe
Array(SequenceableCollection)	collect:

Source

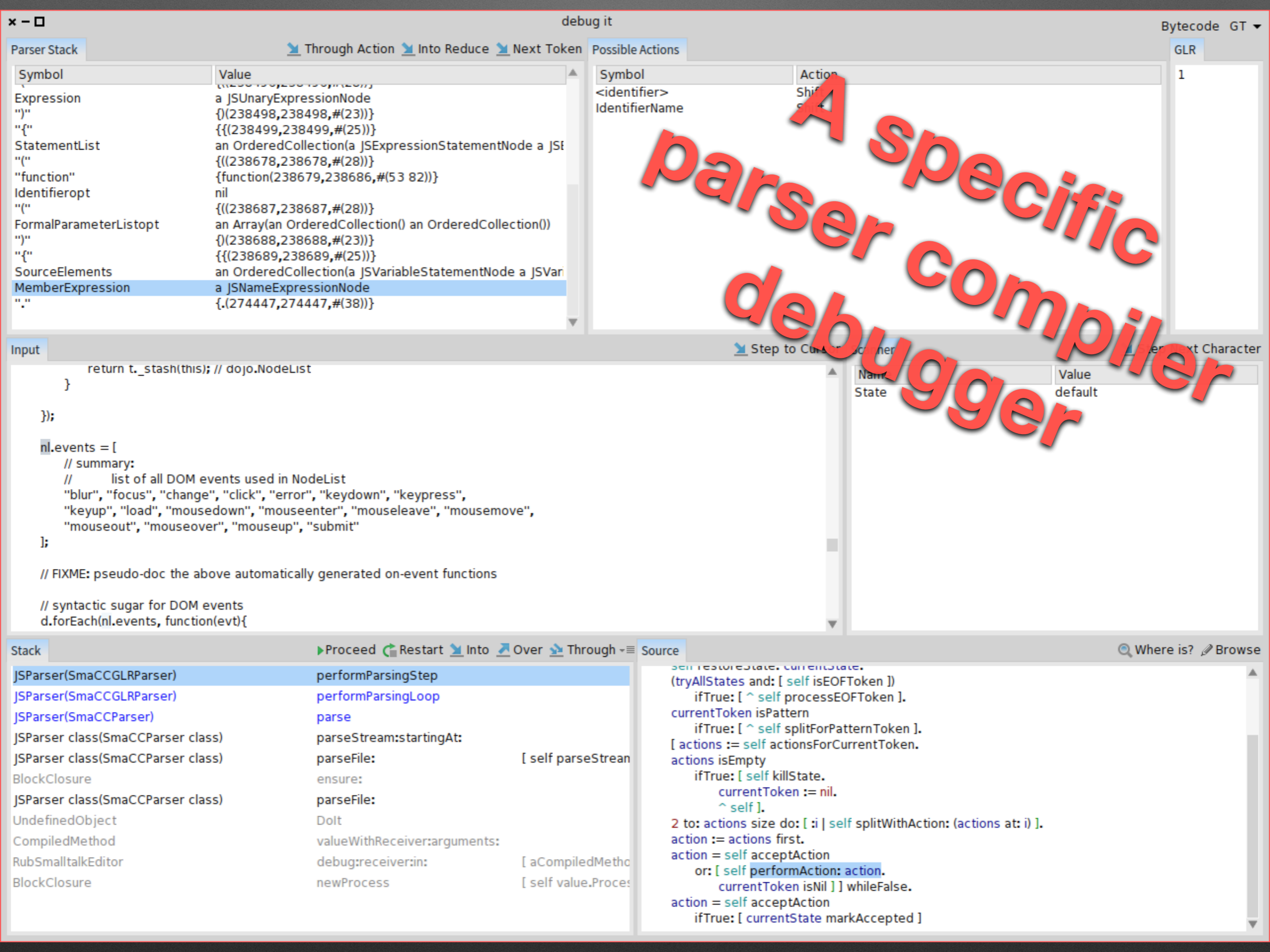
Where is? Browse

```
generateCodeSegmentsCollectionWith: aPDFGenerator styleSheet: compositeStyleSheet format: aFormat
^ (self getSubElementsWith: aPDFGenerator styleSheet: compositeStyleSheet)
  collect: [ :aSubElement |
    aSubElement
      generateCodeSegmentWith: aPDFGenerator
      styleSheet: (aSubElement buildCompositeStyleSheetFrom: compositeStyleSheet)
      format: aFormat ]
```

Variables

Type	Variable	Value
implicit	self	a PDFCellElement
parameter	aFormat	a PDFA4Format
parameter	aPDFGenerator	a PDFGenerator
parameter	compositeStyleSheet	a StyleSheet

dimension: 80 mm @ 20 mm;



A specific
parser compiler
debugger

Symbol	Value
Expression	a JSUnaryExpressionNode
"{"	{(238498,238498,#(23))}
"{"	{{(238499,238499,#(25))}
StatementList	an OrderedCollection(a JSExpressionStatementNode a JS
"{"	{{(238678,238678,#(28))}
"function"	{function(238679,238686,#(53 82))}
Identifieropt	nil
"{"	{{(238687,238687,#(28))}
FormalParameterListopt	an Array(an OrderedCollection() an OrderedCollection())
"{"	{(238688,238688,#(23))}
"{"	{{(238689,238689,#(25))}
SourceElements	an OrderedCollection(a JSVariableStatementNode a JSVari
MemberExpression	a JSNameExpressionNode
"."	.(274447,274447,#(38))

Symbol	Action
<identifier>	Shift
IdentifierName	Shift

```
return t._stash(this); // dojo.NodeList
}
});
nl.events = [
  // summary:
  // list of all DOM events used in NodeList
  "blur", "focus", "change", "click", "error", "keydown", "keypress",
  "keyup", "load", "mousedown", "mouseenter", "mouseleave", "mousemove",
  "mouseout", "mouseover", "mouseup", "submit"
];
// FIXME: pseudo-doc the above automatically generated on-event functions
// syntactic sugar for DOM events
d.forEach(nl.events, function(evt){
```

Name	Value
State	default

Stack	Source
JSParser(SmaCCGLRParser)	performParsingStep
JSParser(SmaCCGLRParser)	performParsingLoop
JSParser(SmaCCParser)	parse
JSParser class(SmaCCParser class)	parseStream:startingAt:
JSParser class(SmaCCParser class)	parseFile: [self parseStream
BlockClosure	ensure:
JSParser class(SmaCCParser class)	parseFile:
UndefinedObject	Dolt
CompiledMethod	valueWithReceiver:arguments:
RubSmalltalkEditor	debug:receiver:in: [aCompiledMetho
BlockClosure	newProcess [self value.Proces

```
self restoreState: currentState.
(tryAllStates and: [ self isEOFToken ])
  ifTrue: [ ^ self processEOFToken ].
currentToken isPattern
  ifTrue: [ ^ self splitForPatternToken ].
[ actions := self actionsForCurrentToken.
actions isEmpty
  ifTrue: [ self killState.
            currentToken := nil.
            ^ self ].
2 to: actions size do: [ :i | self splitWithAction: (actions at: i) ].
action := actions first.
action = self acceptAction
  or: [ self performAction: action.
        currentToken isNil ] ] whileFalse.
action = self acceptAction
  ifTrue: [ currentState markAccepted ]
```

Halt in OCDBox>>name:

Bytecode	Breakpoints	Sindarin
<input type="checkbox"/>	Breakpoint self	Reflecting Ex...
<input checked="" type="checkbox"/>	Breakpoint self	OCDBox>>initialize
<input checked="" type="checkbox"/>	Breakpoint self	OCDBox>>name:
<input checked="" type="checkbox"/>	Breakpoint self	OCDBox>>name:
<input checked="" type="checkbox"/>	Halt OCDBox	OCDBox>>name:
<input type="checkbox"/>	Breakpoint OCDBox	OCDBox>>name:
<input type="checkbox"/>	Breakpoint self	OCDBox>>removeElement:
<input type="checkbox"/>	Breakpoint self	OrderedCollection>>remove
<input type="checkbox"/>	Halt StHaltCacheTest	StHaltCacheTest>>testInitial
<input checked="" type="checkbox"/>	Halt StHaltCacheTest	StHaltCacheTest>>testInitial

```

1 name: anObject
2 self halt.
3 name := anObject
  
```

Specific view

Receiver in: a StDebuggerContext (OCDBox>>name:)

Variable	Value
[arg] anObject	'i'
self	an OCDBox
elements	an OrderedCollection [0 items] ()
name	"
Temps	a Dictionary [1 item] (#anObject->'i')
anObject	'i'
stackTop	an OCDBox

Raw	Breakpoints	Meta
<input checked="" type="checkbox"/>	Halt	OCDBox
		OCDBox>>name:
1		
1		"an OCDBox"
2		self

Bytecode Breakpoints Sindarin

```

33 <4C> self
34 <80> send: halt
35 <D8> pop
36 <40> pushTemp: 0
37 <C9> popIntoRcvr: 1
38 <58> returnSelf
  
```

Variable	Value
stackTop	an OCDBox
0 [anObject]	'i'
rcvr: 0 [elements]	an OrderedCollection [0 items]
rcvr: 1 [name]	"

object centric +

adv scripting (AST + call stack)

Variable	Value
[arg] anObject	'i'

Raw	Breakpoints	Meta
<input checked="" type="checkbox"/>	Halt	OCDBox

Variable	Value
stackTop	an OCDBox

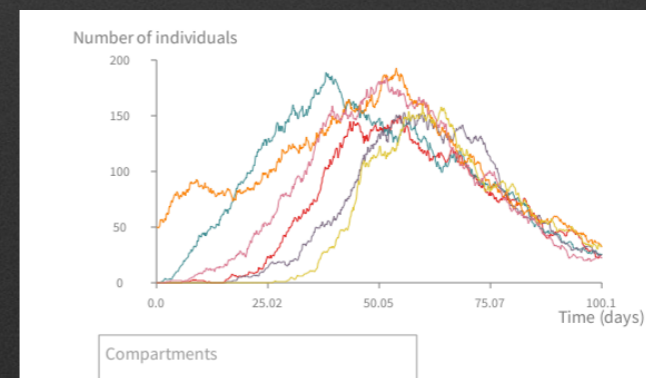
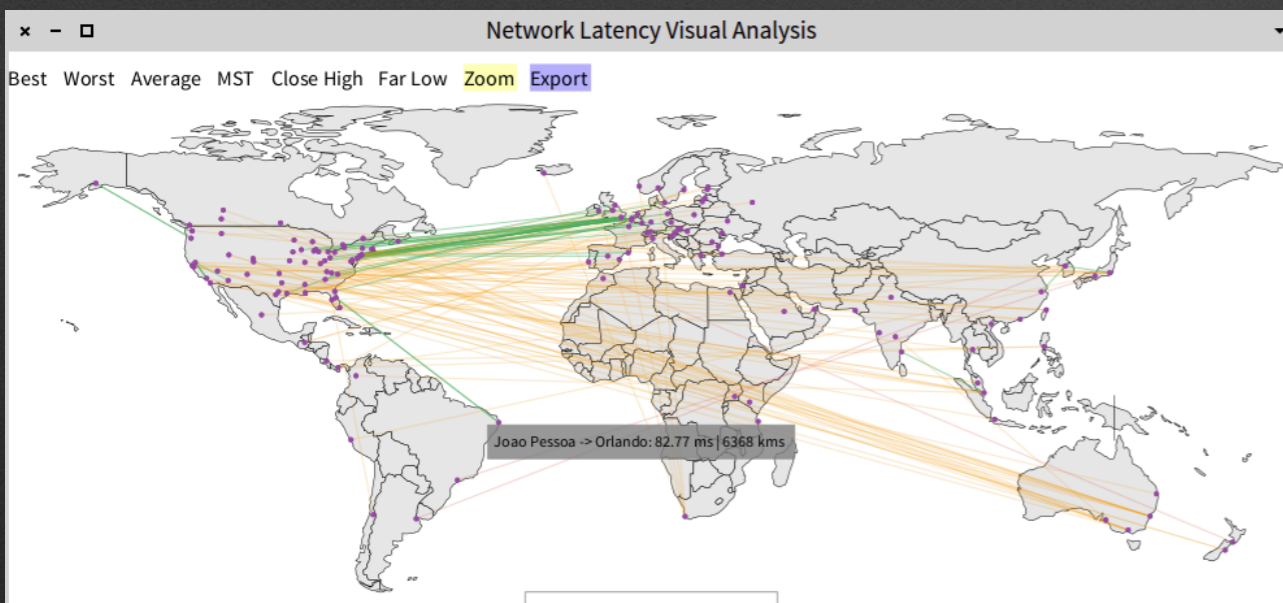
Live visualisation scripting

- The next level
- Roassal 3.0 by Prof. A. Bergel/Object Profile University of Chile at Santiago
- Simply gorgeous
- Check <http://agilevisualization.com>

Includes a DSL for Scripting visualisations

```
b := RTMondrian new.  
  b shape rectangle  
    withBorder;  
  width: [ :cls | cls numberOfVariables * 5 ];  
  height: [ :cls | cls numberOfMethods ].
```

```
b nodes: Collection withAllSubclasses.  
b edges connectToAll: [ :cls | cls subclasses ].  
b layout tree.  
b normalizer  
  normalizeColorAsGray: [ :cls |  
cls numberOfLinesOfCode ].  
b
```



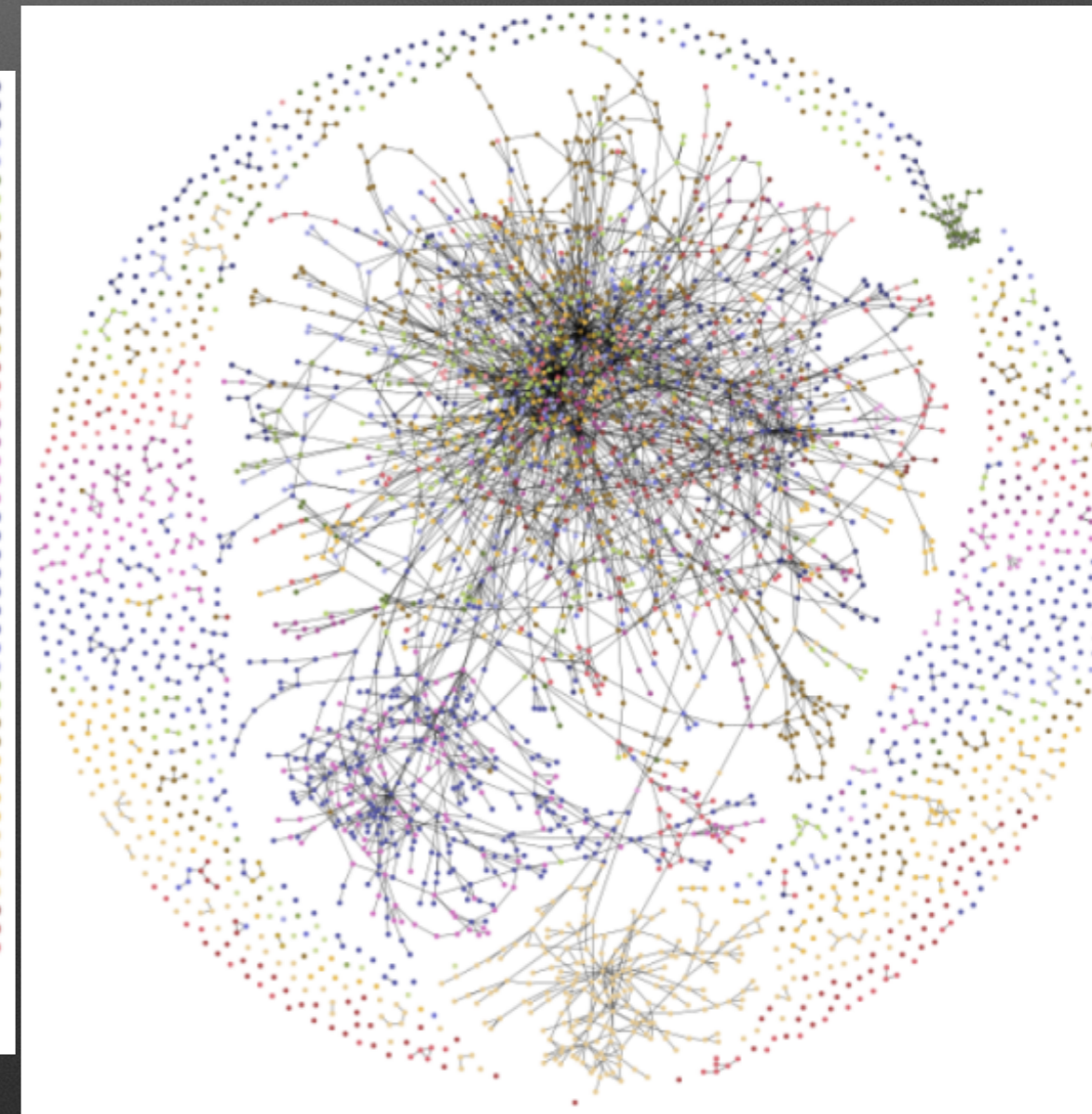
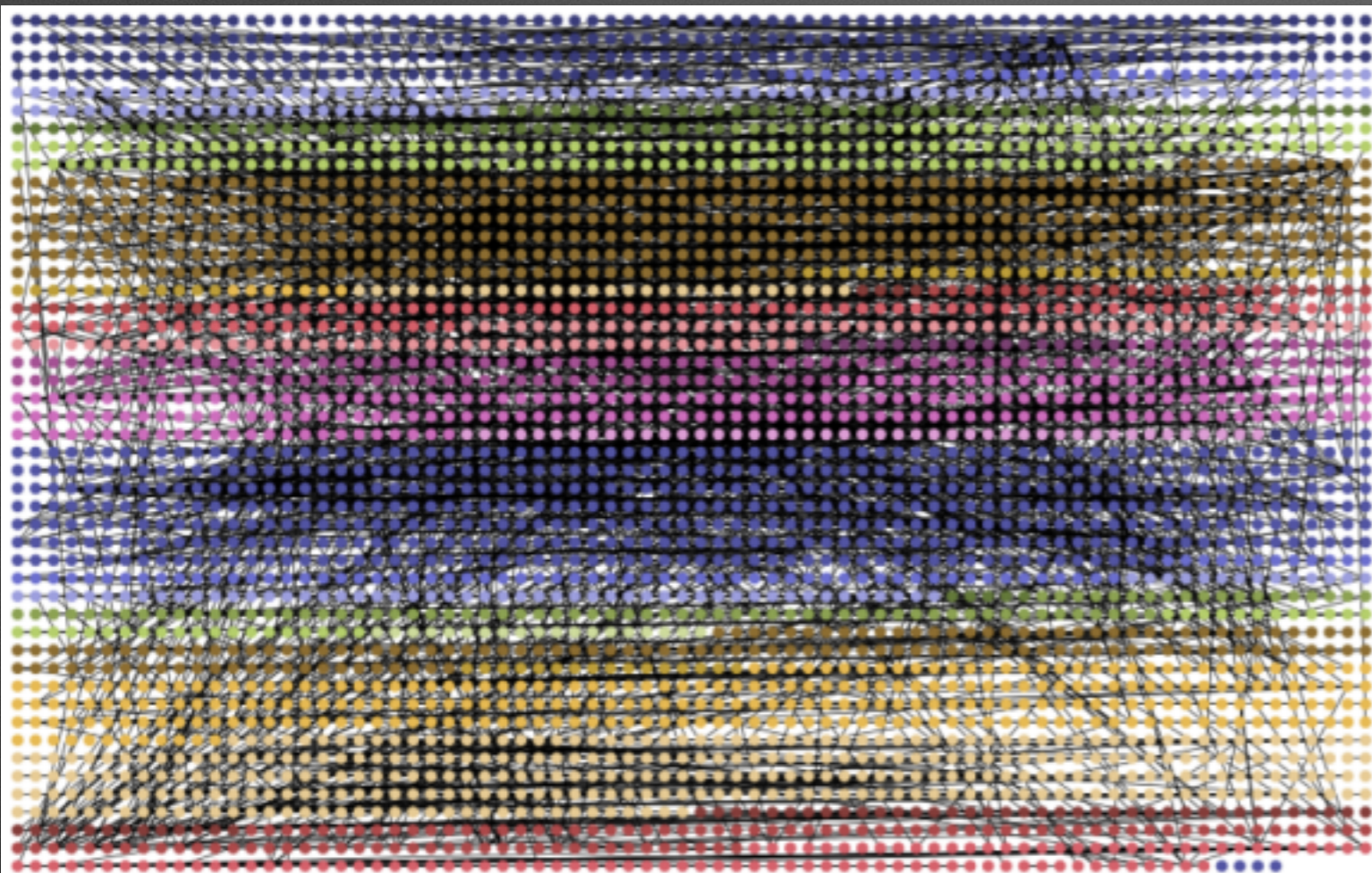
One hour about Basic

COPYRIGHT 1975 BY BILL GATES AND PAUL ALLEN

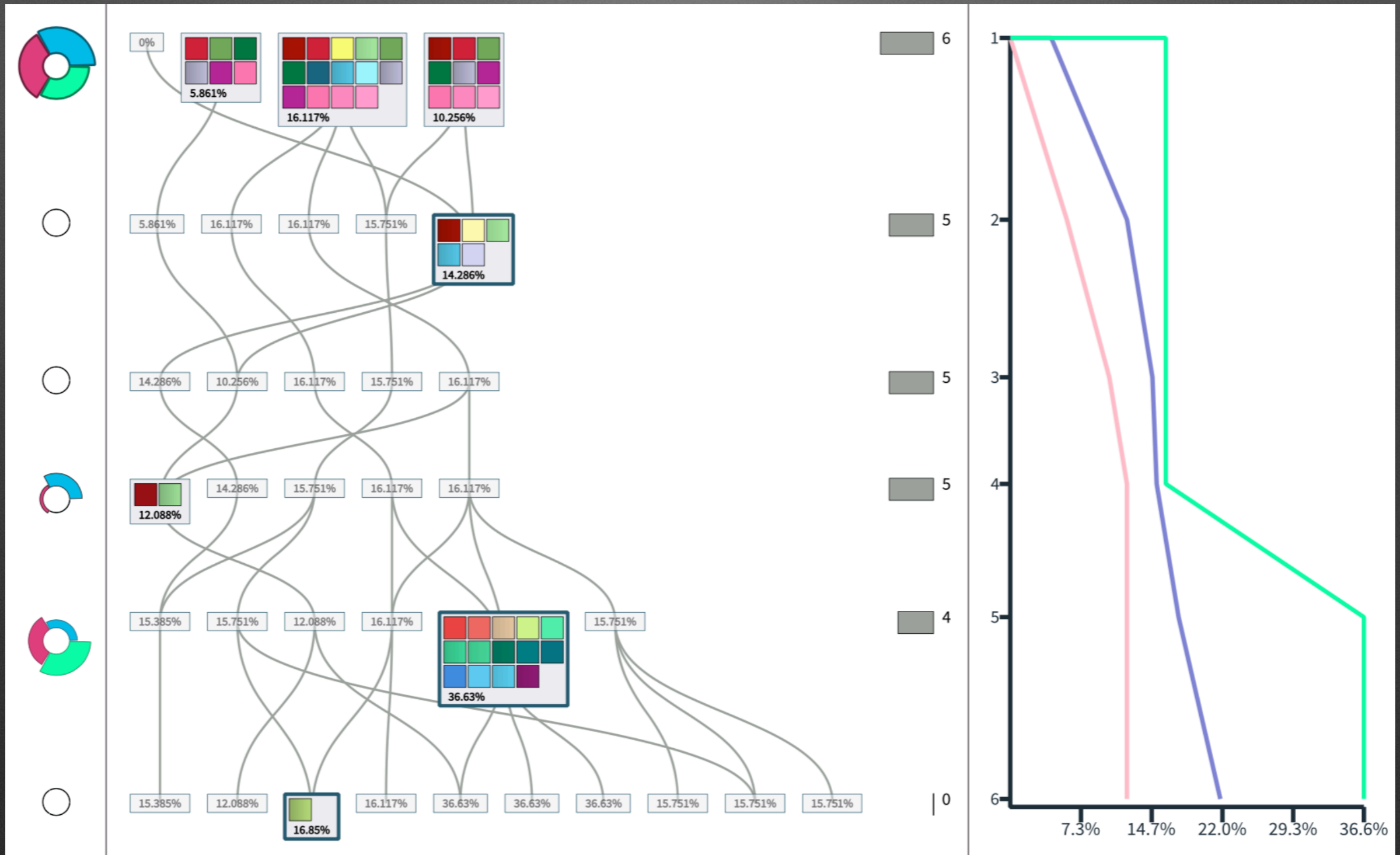
ORIGINALLY WRITTEN ON THE PDP-10 FROM
FEBRUARY 9 TO APRIL 9 1975

BILL GATES WROTE A LOT OF STUFF.
PAUL ALLEN WROTE A LOT OF OTHER STUFF AND FAST CODE.
MONTE DAVIDOFF WROTE THE MATH PACKAGE (F4I.MAC).

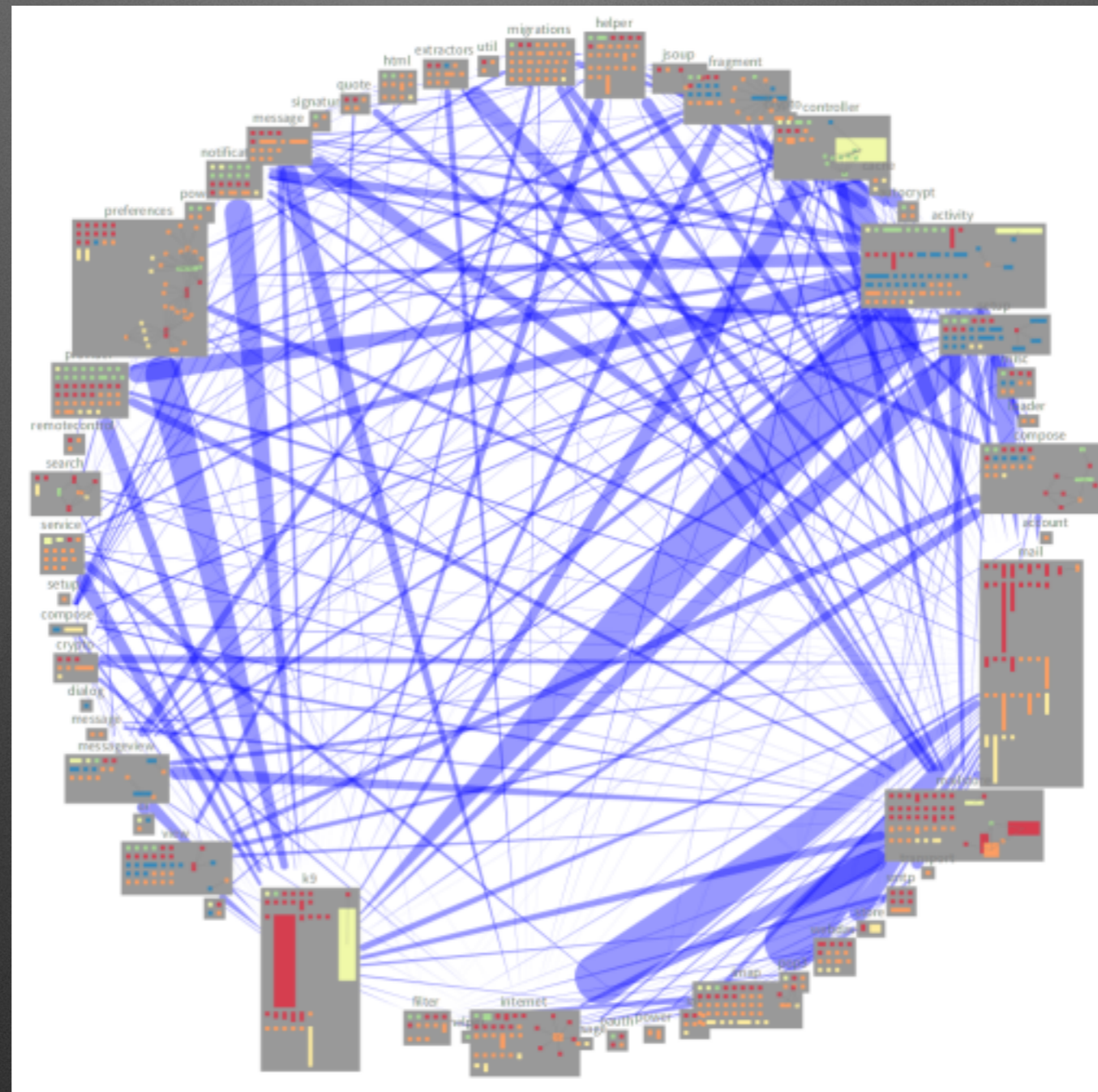
<https://pharoweekly.wordpress.com/2020/05/24/roassal-1-hour-xp-assembly-code-of-gwbasic/>



Execution of IA generating tests



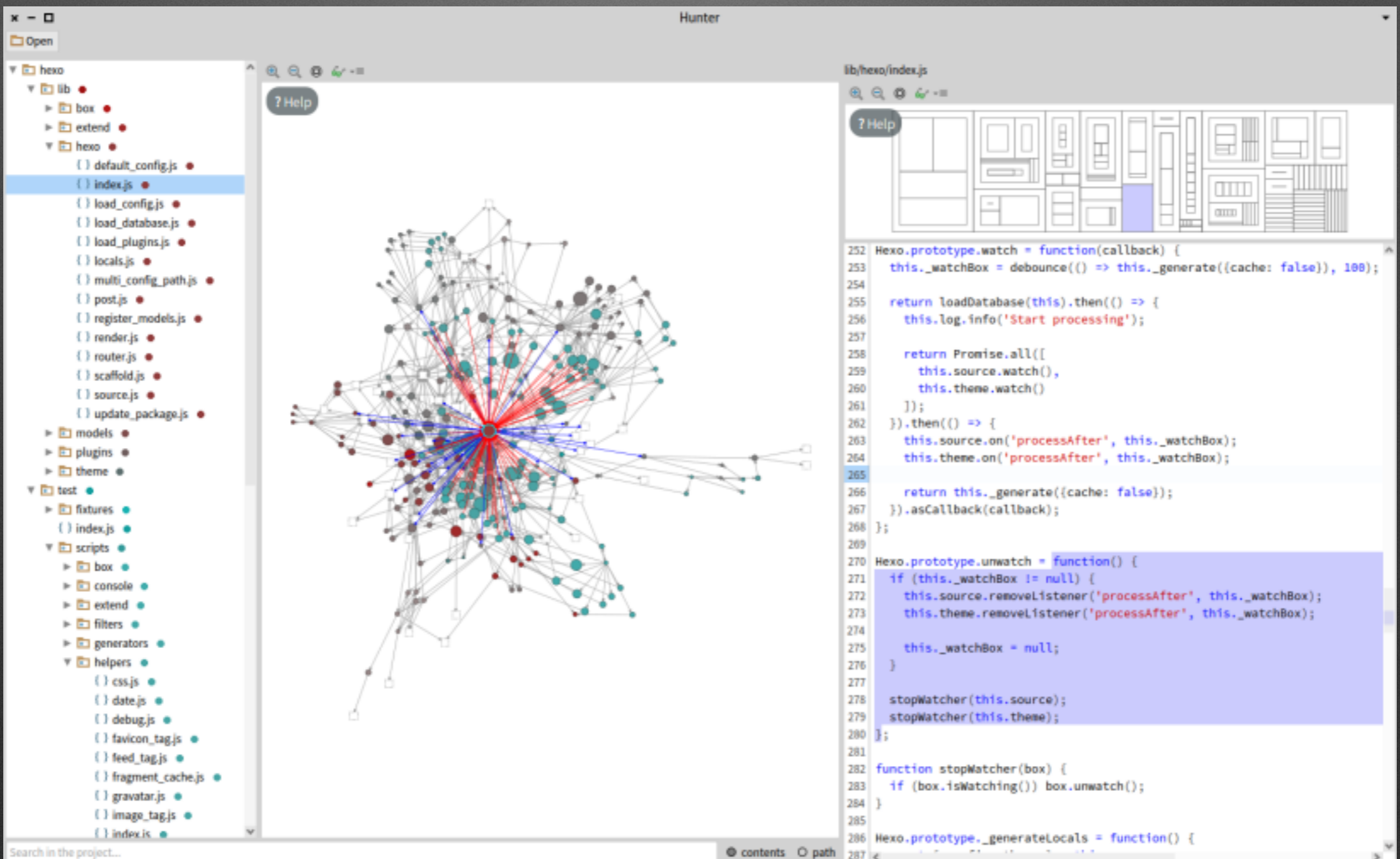
Analysis Android application



**Often developers
write their own tools**

Building your own tool

- Example Javascript analysis



The screenshot displays the Hunter tool interface, which is used for JavaScript analysis. The interface is divided into three main sections:

- File Explorer (Left):** Shows a hierarchical view of the project files. The 'hexo' directory is expanded, showing sub-directories like 'lib', 'test', and 'helpers', along with various JavaScript files such as 'index.js', 'load_config.js', and 'update_package.js'.
- Network Graph (Center):** A complex network graph representing the relationships between the analyzed JavaScript code. Nodes are represented by colored circles (red, blue, green, grey) and are interconnected by lines, forming a dense, star-like structure with many connections radiating from a central point.
- Code Editor (Right):** Displays the source code of the selected file, 'lib/hexo/index.js'. The code is shown in a dark theme with syntax highlighting. The function `Hexo.prototype.unwatch` is highlighted in blue, showing its implementation which removes listeners and stops watchers.

```
252 Hexo.prototype.watch = function(callback) {
253   this._watchBox = debounce(() => this._generate({cache: false}), 100);
254
255   return loadDatabase(this).then(() => {
256     this.log.info('Start processing');
257
258     return Promise.all([
259       this.source.watch(),
260       this.theme.watch()
261     ]);
262   }).then(() => {
263     this.source.on('processAfter', this._watchBox);
264     this.theme.on('processAfter', this._watchBox);
265
266     return this._generate({cache: false});
267   }).asCallback(callback);
268 };
269
270 Hexo.prototype.unwatch = function() {
271   if (this._watchBox != null) {
272     this.source.removeListener('processAfter', this._watchBox);
273     this.theme.removeListener('processAfter', this._watchBox);
274
275     this._watchBox = null;
276   }
277
278   stopWatcher(this.source);
279   stopWatcher(this.theme);
280 };
281
282 function stopWatcher(box) {
283   if (box.isWatching()) box.unwatch();
284 }
285
286 Hexo.prototype._generateLocals = function() {
287
```

Probabilistic Data Structure

- <https://github.com/osoco/PharoPDS>
- Defined new data structure
- And the **analysis tools**

The screenshot displays the 'PDS Algorithms Viewer' interface. The main window is titled 'PDS Algorithms Viewer' and features a menu bar with options: '+ New Bloom Filter', 'Analysis', 'Profiling', 'Benchmarking', and 'Reset'. The left sidebar shows 'PDS Algorithms' with 'Bloom Filter' selected. The main content area is divided into several sections:

- Add elements:** A text input field contains 'e.g. Madrid', followed by an 'Add' button. Below it, a blue notification reads 'london added to filter!'.
- Membership test:** A text input field contains 'E.g. London', followed by a 'Test' button.
- Bloom Filter BitSet:** A 15x15 grid of green squares representing the bit set. Some squares are blue, indicating they are set to 1. The grid is labeled 'In Out Center' at the top.
- Parameters:** A table titled 'a PDSBloomFilter' shows the following parameters:

Name	Value
'Target Elements (n)'	100
'Target FPP'	0.03
'Number of hashes (k)'	6
'Current Elements'	0
'Current FPP'	0.0
- False-Positive Probability Curve:** A graph showing the False Positive Probability (FPP) on the y-axis (ranging from 0.0 to 0.15) versus Elements Added on the x-axis (ranging from 0.0 to 150.0). A blue curve represents the 'FPP curve', and a red dot at the origin represents the 'Current FPP'.

HTTP traffic analysis

- <http://youtu.be/rIBbeMdFCys>

The screenshot displays the Pharo IDE interface. At the top left is the Pharo logo. A Monticello Browser window is open, showing a repository at `http://smalltalkhub.com/mc/SvenVanCaekenberghe/f`. It lists several packages, including `HP35-Calculator-SvenVanCaekenberghe.17.mcz` through `.7.mcz`. Below the browser is a Playground window. The left pane shows a log of system events, with the following entries:

Time	Announcement
2014-10-15T15:27:46.89322+0	2014-10-15 15:27:46 030 Connection Closed 128.93.162.53:80
2014-10-15T15:27:46.882202+	2014-10-15 15:27:46 029 GET /mc/SvenVanCaekenberghe/HP35/main/
2014-10-15T15:27:46.873225+	2014-10-15 15:27:46 028 Response Read a ZnResponse(200 OK text/
2014-10-15T15:27:46.764834+	2014-10-15 15:27:46 027 Request Written a ZnRequest(GET /mc/Sven
2014-10-15T15:27:46.749283+	2014-10-15 15:27:46 026 Connection Established smalltalkhub.com:80

The right pane of the playground shows the state of a `ZnResponseReadEvent` object:

Variable	Value
self	2014-10-15 15:27:46 028 Response Read a ZnResponse(200 OK text/
clientid	nil
duration	0
id	28
response	a ZnResponse(200 OK text/plain 1197B)
timestamp	2014-10-15T15:27:46.873225+02:00

Below the variable table, a message box shows the message sent to `self`: `"2014-10-15 15:27:46 028 Response Read a ZnResponse(200 OK text/plain 1197B) 0ms"`.

Stepping ARM 64 bits asm

The screenshot shows a debugger window titled "Untitled window" with a split view. The left pane displays ARM assembly code with the instruction at address 16r418, `ldr x12, #40`, highlighted. The right pane shows a memory dump with addresses from 16r151E40 to 16r151ED8. A "Step" button is visible in the center, and a "Disassemble at PC" button is at the bottom.

Address	Instruction	Comment	Register	Value
16r400	<code>mov x0, x29</code>	<code>#[224 3 29 170]</code>	lr	'16r0'
16r404	<code>ret</code>	<code>#[192 3 95 214]</code>	pc	'16r0'
16r408	<code>mov x0, x28</code>	<code>#[224 3 28 170]</code>	sp	'16r0'
16r40C	<code>ret</code>	<code>#[192 3 95 214]</code>	fp	'16r151E50'
16r410	<code>str x10, [x28, #-8]!</code>	<code>#[138 143 31 248]</code>	x0	'16r0'
16r414	<code>ldr x10, #36</code>	<code>#[42 1 0 88]</code>	x1	'16r0'
16r418	<code>ldr x12, #40</code>	<code>#[76 1 0 88]</code>	x2	classRegister '16r0'
16r41C	<code>str x29, [x12]</code>	<code>#[157 1 0 249]</code>	x3	'16r0'
16r420	<code>mov x1, x28</code>	<code>#[225 3 28 170]</code>	x4	'16r0'
16r424	<code>adds x1, x1, #8</code>	<code>#[33 32 0 177]</code>	x5	receiverRegister '16r0'
16r428	<code>ldr x12, #32</code>	<code>#[12 1 0 88]</code>		
16r42C	<code>str x1, [x12]</code>	<code>#[129 1 0 249]</code>		
16r430	<code>ldr x10, [x28], #8</code>	<code>#[138 135 64 248]</code>		
16r434	<code>ret</code>	<code>#[192 3 95 214]</code>		

Address	Value
16r151E40	16r0
16r151E48	16r0
16r151E50	16r0
16r151E58	16r0
16r151E60	16r0
16r151E68	16r0
16r151E70	16r151E50
16r151E78	16r151E40
16r151E80	16r0
16r151E88	16r0
16r151E90	16r0
16r151E98	16r0
16r151EA0	16r0
16r151EA8	16r0
16r151EB0	16r0
16r151EB8	16r0
16r151EC0	16r0
16r151EC8	16r0
16r151ED0	16r0
16r151ED8	16r0

Moments of grace...

**I want my halt to only
stop when called from
THAT test called testMe!**

mycode

self haltlf: #testMe

...

Use stack reification

Walk it

Halt if needed

(in 5 lines)

```
haltIf: aSelector  
| cntxt |  
cntxt := thisContext.  
[ cntxt isNil ] whileFalse: [  
    cntxt selector = aSelector  
    ifTrue: [ self halt ].  
    cntxt := cntxt sender ]
```

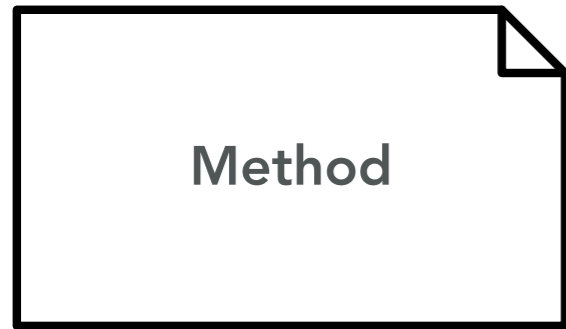
**Dynamically rewriting
deprecated calls @
runtime**

How to support migration to new versions

We deprecate API

How to help our users to migrate?

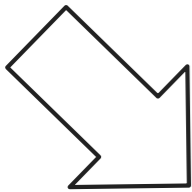
Deprewriter



Annotation (1)

Declaration

Execution

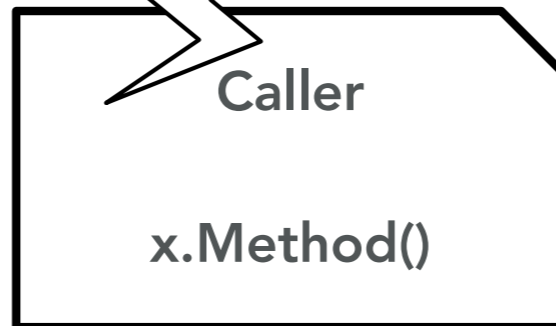
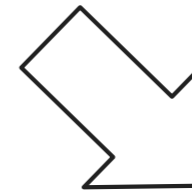


Exception handling (3)

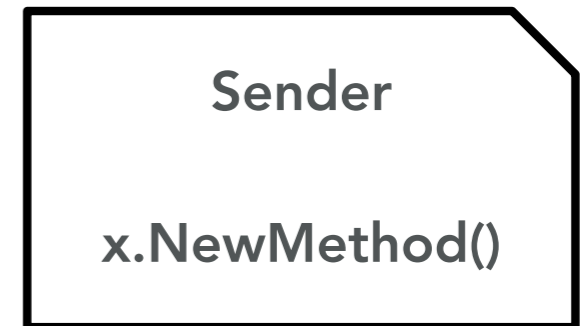
Warning

Caller rewriting (4)

Execution continues (5)



Execution (2)



Rewriting deprecation

crLog: aString

self

deprecated: 'Please use trace* methods instead.'

transformWith:

'@receiver crLog: `@statements1'

-> '@receiver crTrace: `@statements1'.

self crTrace: aString

**Run your tests.
Your code and your
tests use the new
API!**

At notification time:

Walk the stack

Get caller AST

If should be rewritten

Rewrite it and proceed

execution

transform

| node rewriteRule aMethod |

aMethod := self contextOfSender method.

node := self contextOfSender sourceNodeExecuted.

rewriteRule := self rewriterClass new

replace: rule key with: rule value.

(rewriteRule executeTree: node)

ifFalse: [^ self signal].

node replaceWith: rewriteRule tree.

**Pharo is
research friendly**

International Research Groups

Lafhis (AR)

SCG (CH)

CAR (FR)

RMOD (FR)

Ummisco (IRD)

Reveal (CH)

Lysic (FR)

ENSTA-Bretagne (FR)

CEA-List (FR)

Ryerson (CAN)

OC (FR)

CCMI-FIT (CZ)

ASERG (BR)

Pleiad (CL)

Macau (UNO)

Cirad (FR)

USTH (Vietnam)

Soft-Qual (Serbia)

Uni. Quilmes (AR)

ENIT (FR)

CS (Bo)

Maroua (CAM)

ETS (CAN)

**We are ready
to help you
validate
your ideas**

We are interested in

Tools, tests, refactorings, program transformation, visualisation, merge, code review, debugging, test selection, migration, navigation, IDE, code browsing, DSL, recommender, profiler, specific datastructure, type inference, code optimizers, ...

**We can be
your guinea
pigs,...**

(well kind of.... we have real users)

Actively supporting research eg. SCG from Uni. Berne

Spotter

Search

#Menu 15/62 →

- System Browser ✓ →
- Playground
- Test Runner
- Spotter
- Iceberg
- Monticello Browser

No diagnostic and usage data is being sent. Would you like to send diagnostic and usage data to help us improve Pharo? [Go to settings](#)

NumberParser

Scoped Variables History Navigator

Type: Pkg1|^Pkg2|Pk.*Co

- Last Modified Metho
- Configurations
- Work
- AST-Core
- AST-Tests-Core
- Alien

- ASTCache
- ASTCacheReset
- ManifestASTCore
- NumberParser
- RBComment
- RBParseTreeRule

Hier. Class Com.

```
Object subclass: #NumberParser
  instanceVariableNames: 'sourceStream base neg integerPart fraction
nDigits lastNonZero requestor failBlock'
  classVariableNames: ''
  package: 'AST-Core-Parser'
```

1/5 [1]

No diagnostic and usage data is being sent. Would you like to send diagnostic and usage data to help improve Pharo?

1/9 [1]

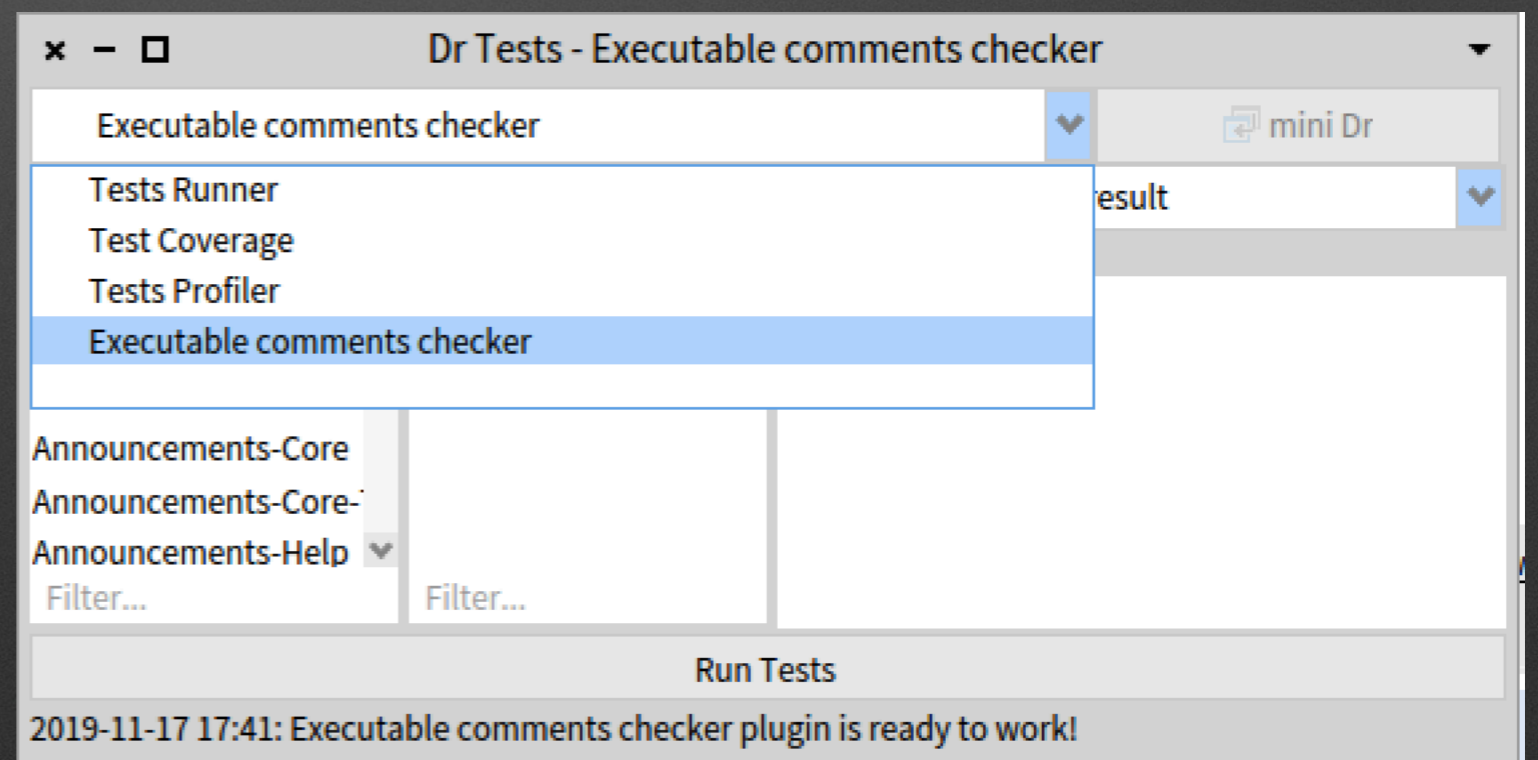
Utility methods ? X

Helpful? 👍 🗑️

Best Paper Award @ ICPC'19

**Kubelka, Bergel, Robbes,
“Live Programming and Software Evolution: Questions
during a Programming Change Task”**

DrTests: a plugin-based architecture to plug test analyses



**We validated our 27000
tests for Rotten Green
Tests (ICSE'19)**

Test Amplification

**by S. Demeyer, H. Rocha, M. Abdi of Antwerp
Universiteit**

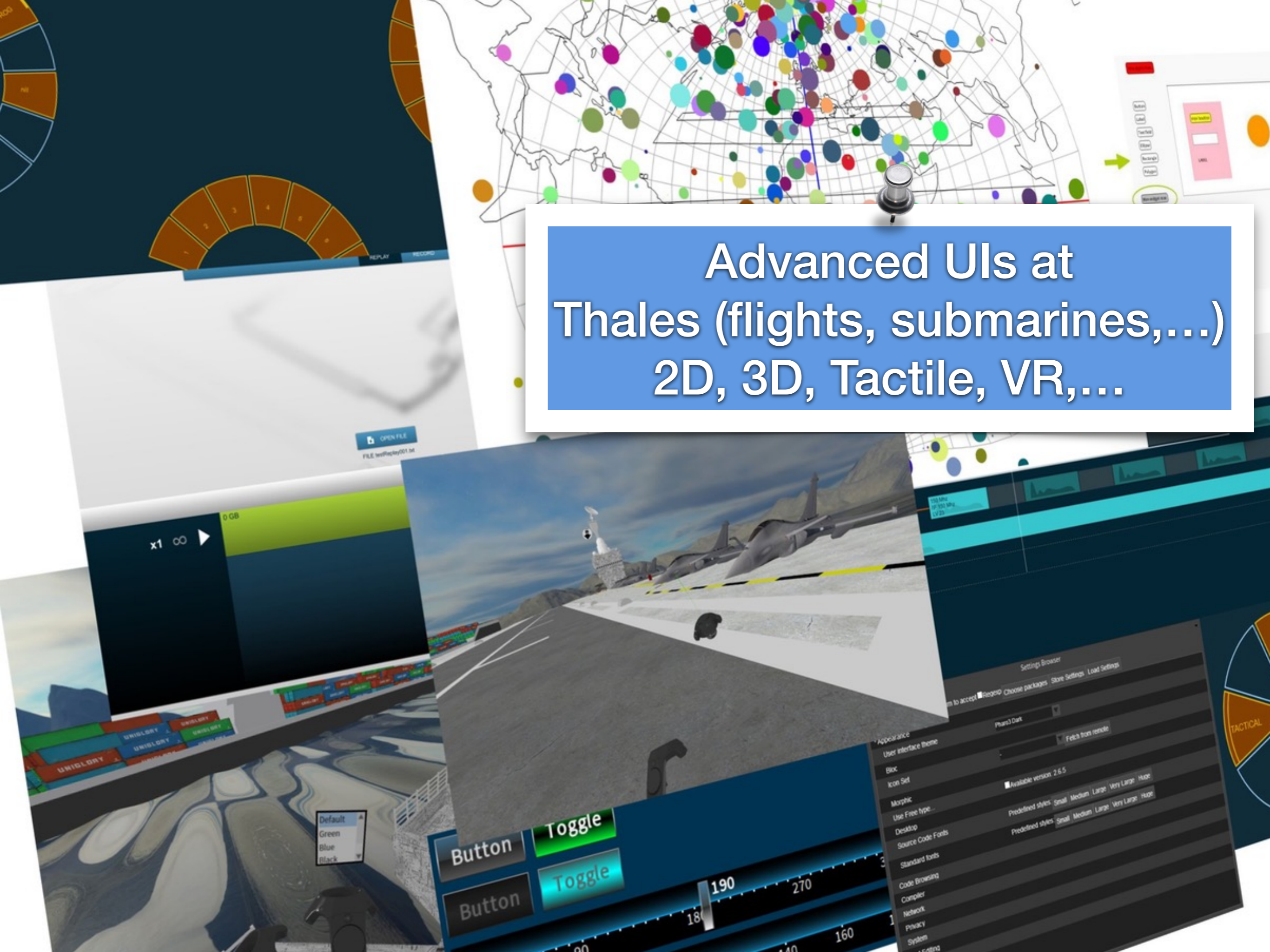
Map reduce debugging

by M. Marra and Prof. E. Gonzales Boix from Vrije
Universiteit Brussel

Code Review

by A. Bachelli, A. Bergel / ObjectProfile

Advanced UIs at Thales (flights, submarines,...) 2D, 3D, Tactile, VR,...



Empowering is the right word

The immersive programming experience

Pharo is a pure object-oriented programming language *and* a powerful environment, focused on simplicity and immediate feedback (think IDE and OS rolled into one).

- Pharo is an energizing and creative environment
- Moldable tools are powerful
- Tried to share my feeling
- But “The idea of experience does not replace experience.” Alain

Discover

Learn more about Pharo's key features and elegant design

Download

Download latest version (8.0)!
Read more about [here](#)

Learn

Access the Pharo Moot!
3000 people registered and follow the Pharo Moot. You can find it [here](#).

A scenic view of a lighthouse on a cliff overlooking the ocean. The lighthouse is white with a black top section. The cliff is covered in green trees. The ocean is blue with white waves crashing against the shore. The sky is a clear, light blue.

Fun, simple

Pure & elegant

Productive

Empowering

Addictive

Full access