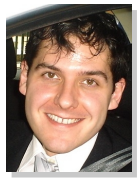
Lessons in Software Evolution Learned by Listening to Smalltalk

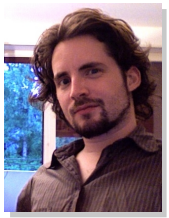
Oscar Nierstrasz

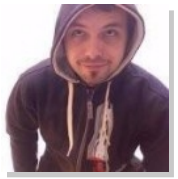
scg.unibe.ch

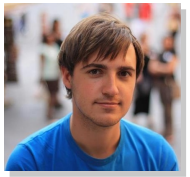
Research Team



Adrian Lienhard
Adrian Kuhn
Fabrizio Perin
Lukas Renggli
Jorge Ressia
David Roethlisberger
Niko Schwarz
Toon Verwaest
Erwann Wernli







Roadmap

The trouble with change
Smalltalk in a Nutshell
Taming Software Change
Epilogue — Bringing Models Closer to Code

Roadmap

The trouble with change

Smalltalk in a Nutshell

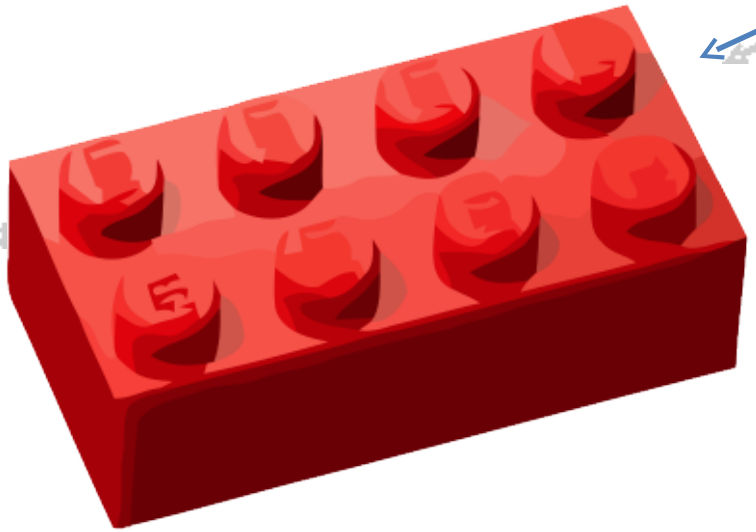
Taming Software Change

Epilogue — Bringing Models Closer to Code

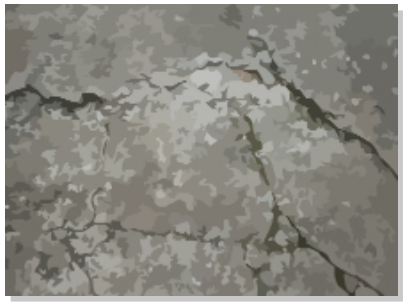
How we enable change

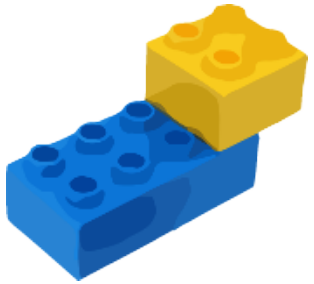
Package what is stable

Define an *interface* for configuration



A brief history of software packaging



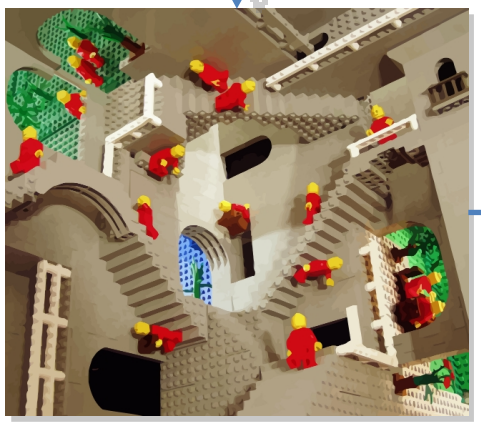


The Trouble with Software Change

Specifying change

Bridging the gap

Understanding change



Understanding the system

Managing change

Roadmap

The trouble with change

Smalltalk in a Nutshell

Taming Software Change

Epilogue — Bringing Models Closer to Code

To thine own self be true

Less is More

The object model "is as simple as possible, but no simpler"

Reify Everything

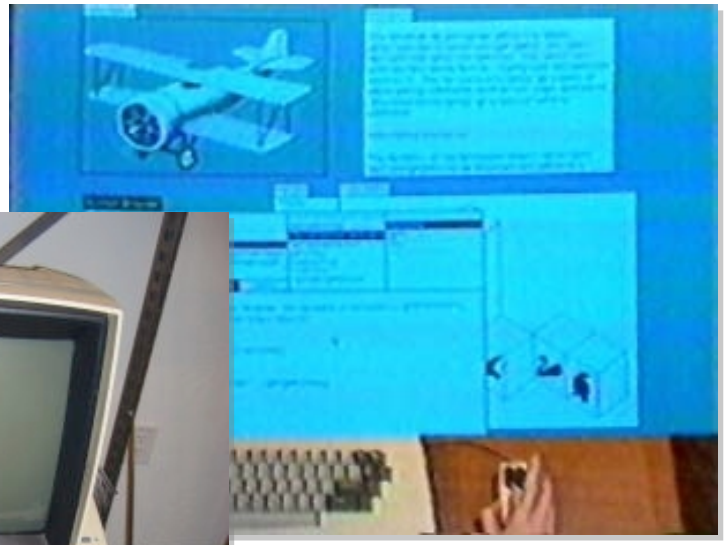
"Everything is an object" is applied consistently

You can change a Running System

Smalltalk models itself, so changing the model changes the system

The origins of Smalltalk

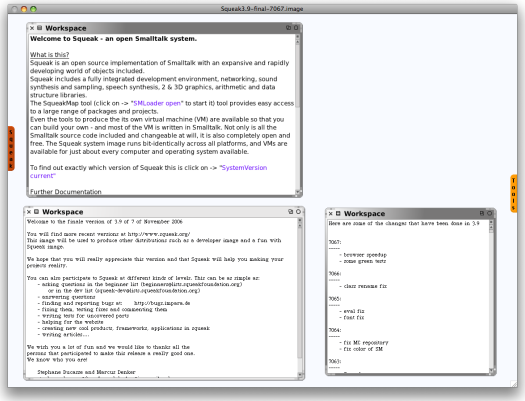
Alan Kay's Dynabook project (1968)



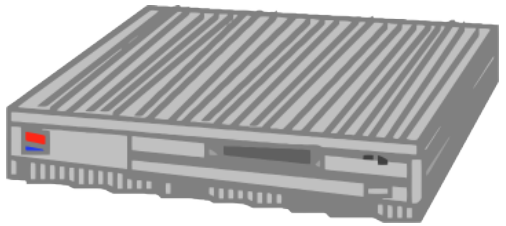
Alto — Xerox PARC (1973)

Smalltalk architecture

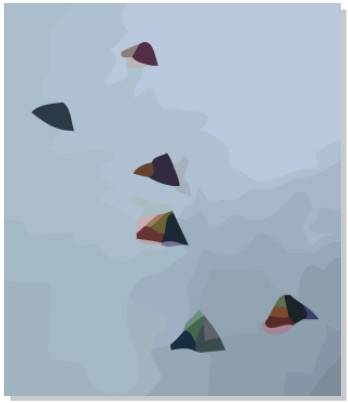
Image



Virtual machine



Changes



Sources

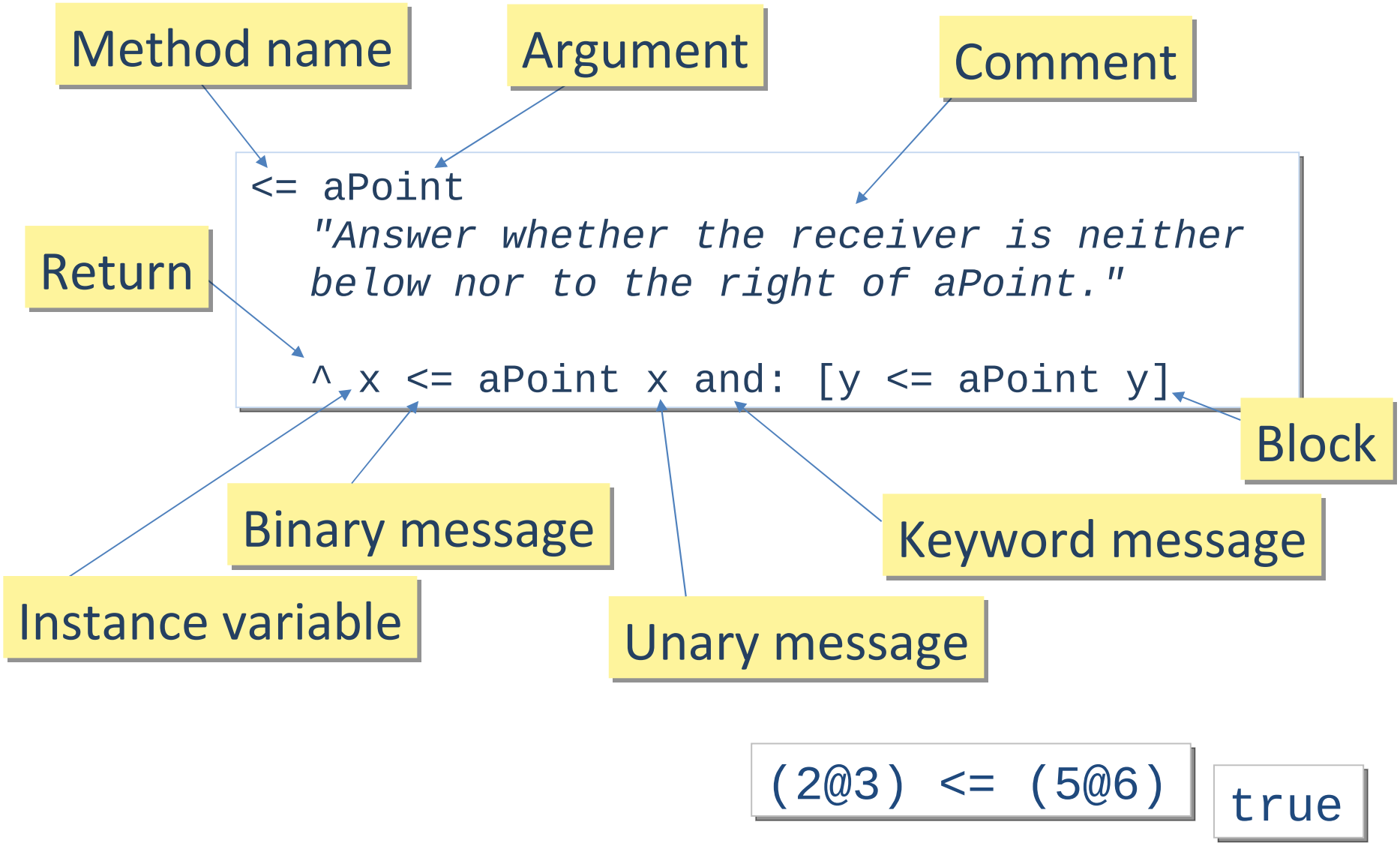
Everything happens by sending messages

First unary, then binary, then keyword:

```
2 raisedTo: 1 + 3 factorial
```

```
128
```

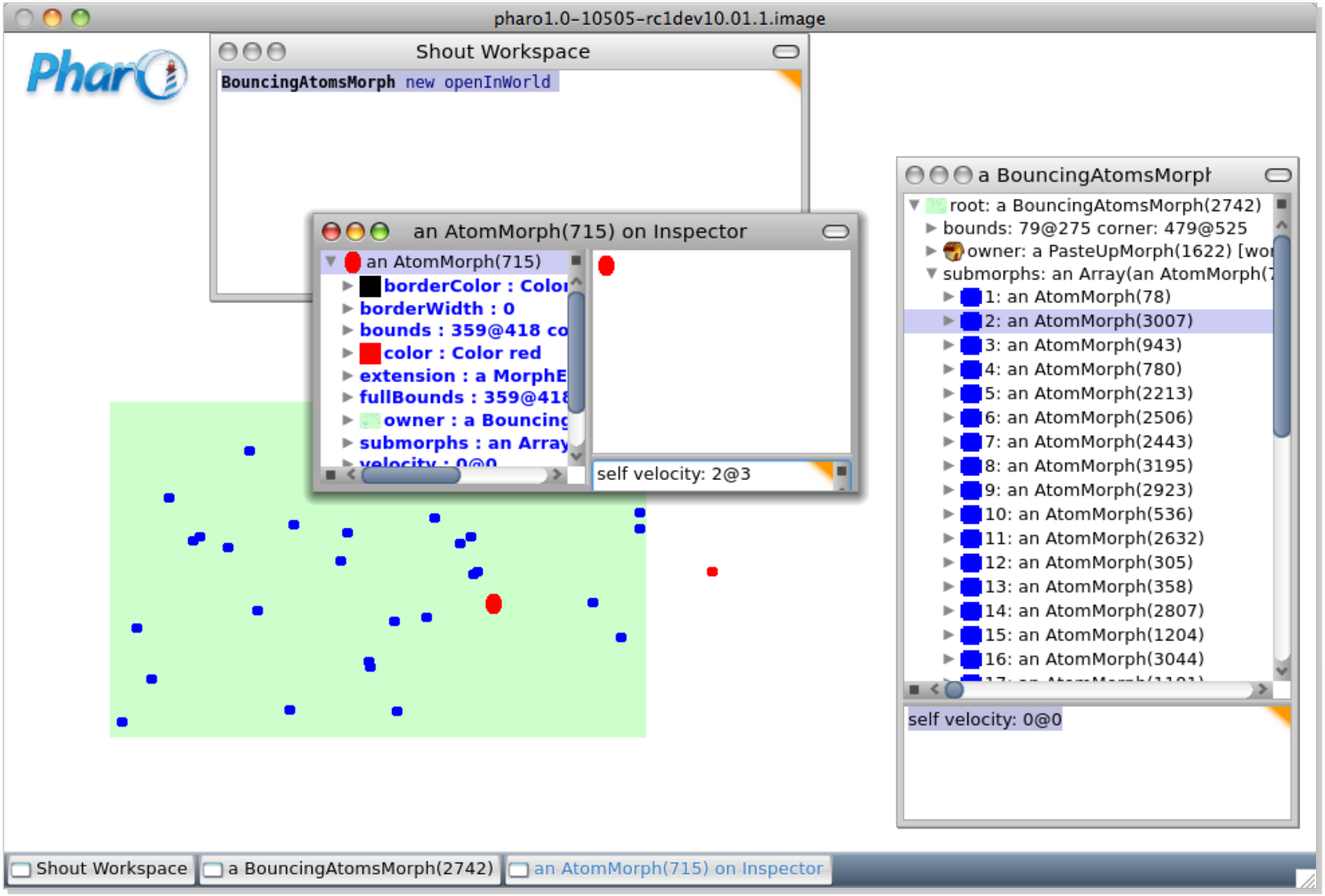
A typical method



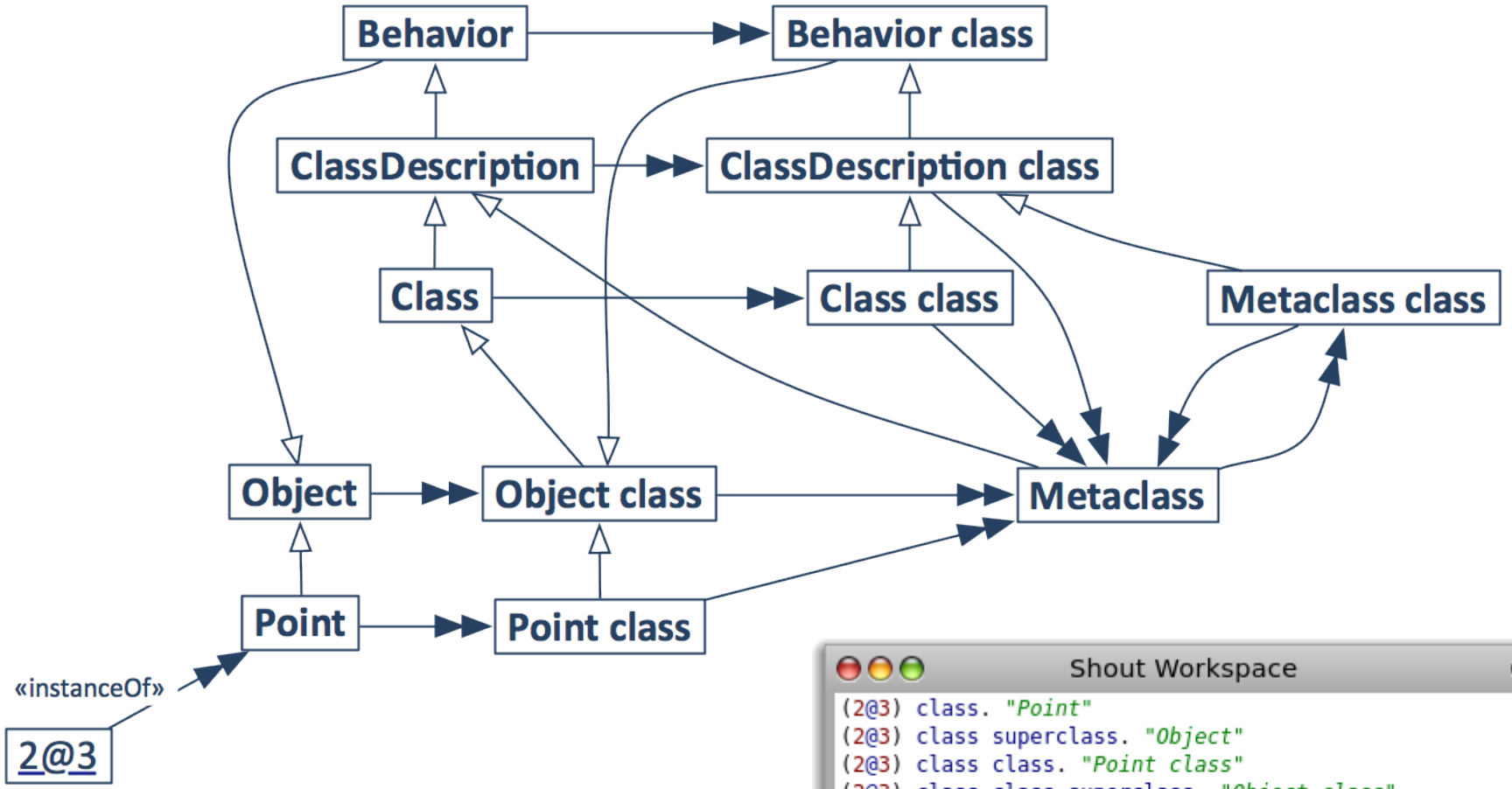

```
(2@3) <= (5@6)
```

`true`

Everything is an object



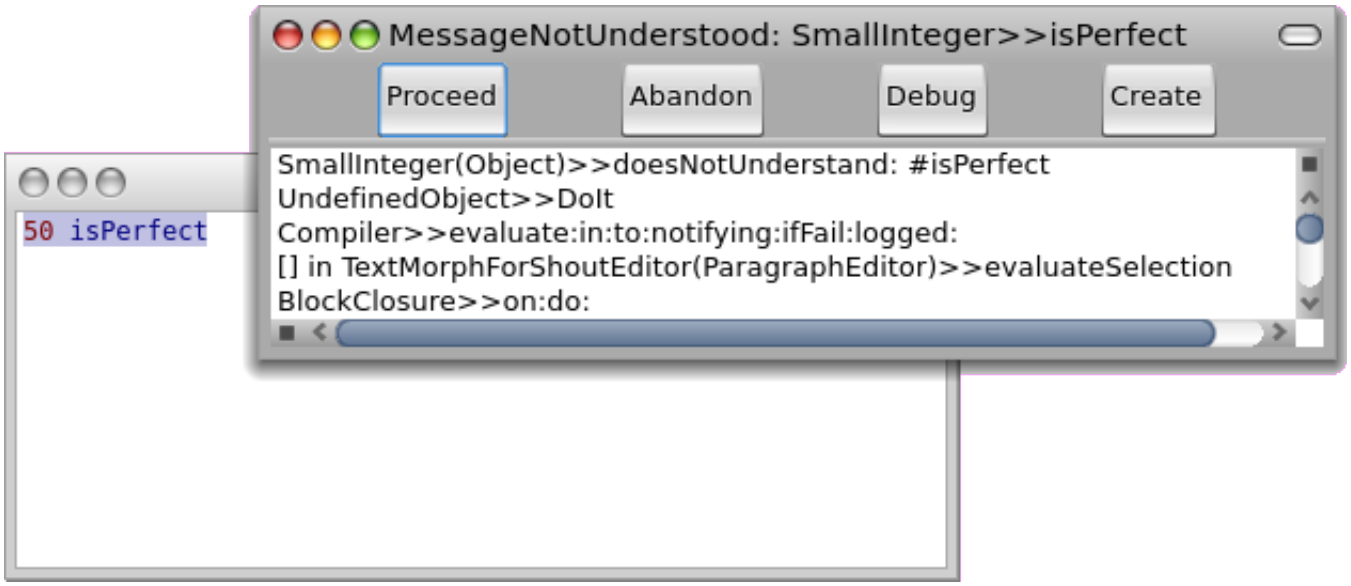
Everything is an object



Shout Workspace

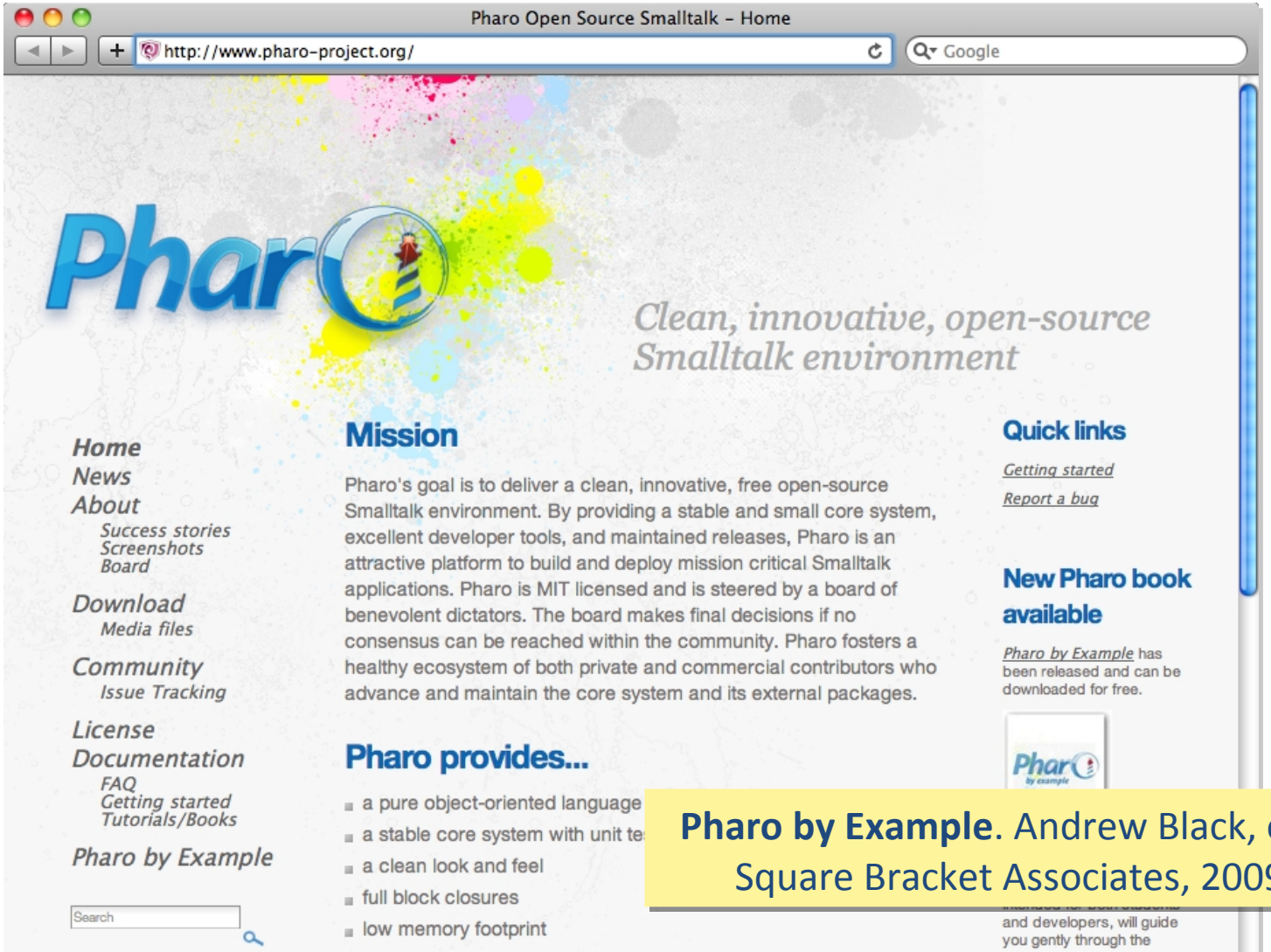
```
(2@3) class. "Point"
(2@3) class superclass. "Object"
(2@3) class class. "Point class"
(2@3) class class superclass. "Object class"
(2@3) class class class. "Metaclass"
(2@3) class class class class. "Metaclass class"
```

You *can* change a running system



AKA: "The Debugger is your Friend"

Pharo — an multi-platform, open-source Smalltalk



Pharo by Example. Andrew Black, et al. Square Bracket Associates, 2009.

Roadmap

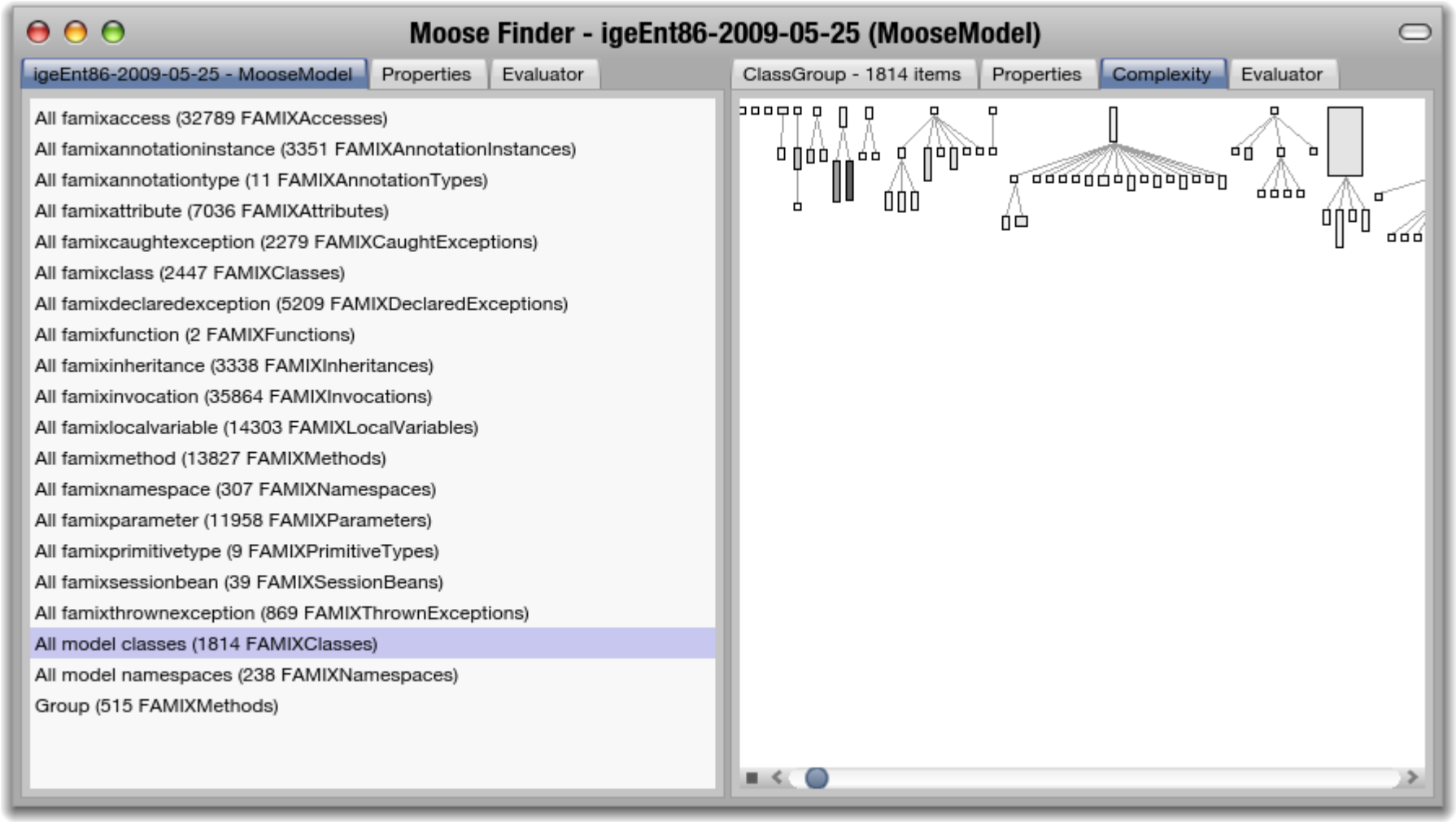
The trouble with change

Smalltalk in a Nutshell

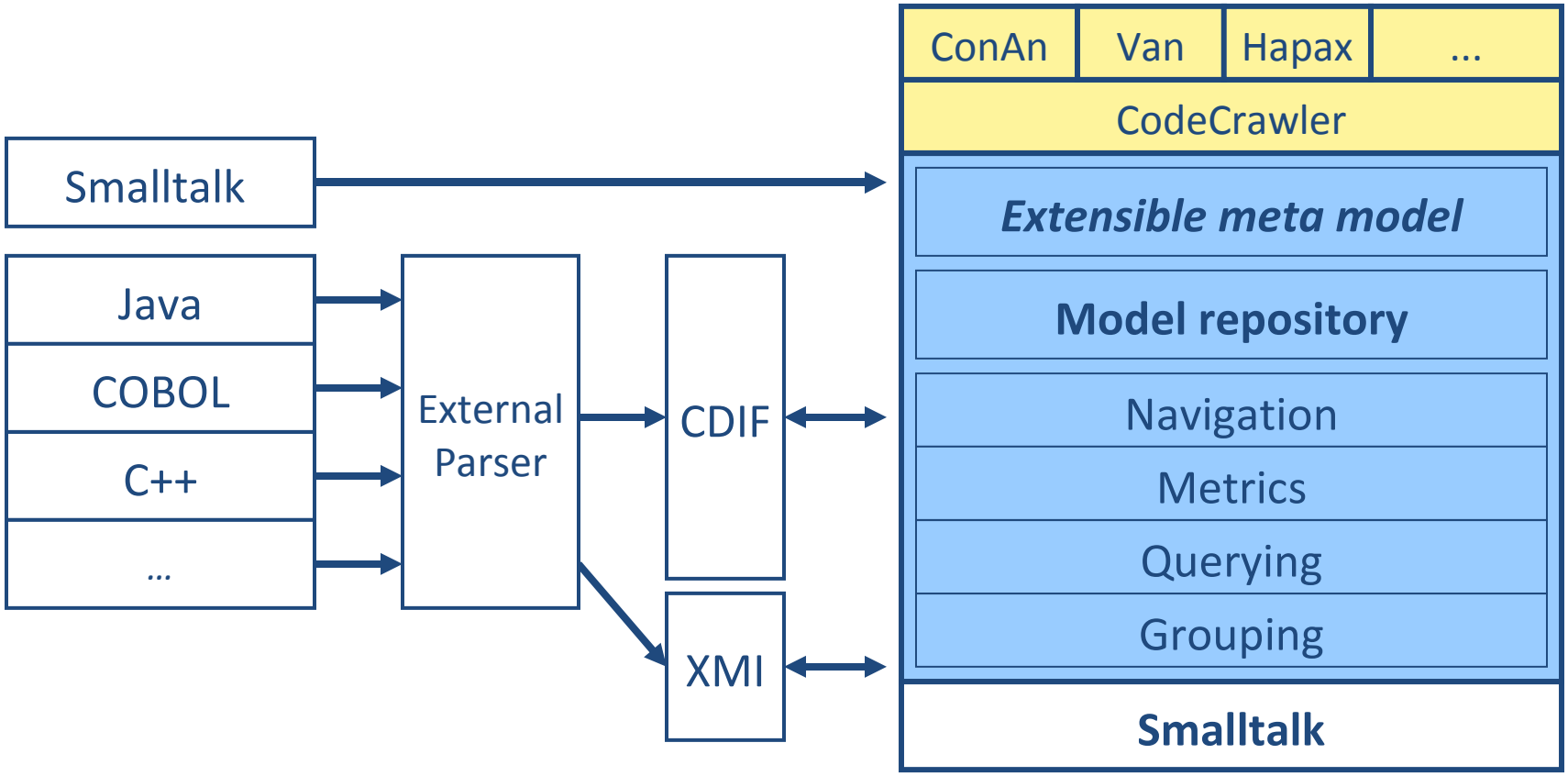
Taming Software Change

Epilogue — Bringing Models Closer to Code

Moose: a platform for reverse engineering

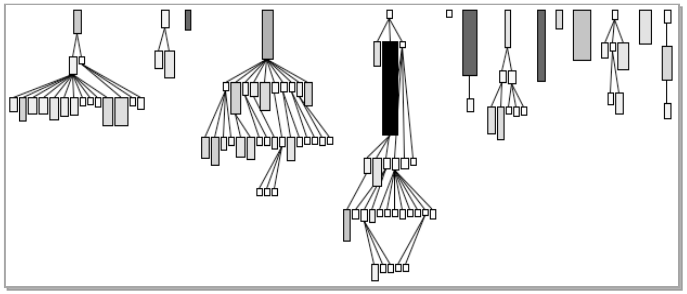


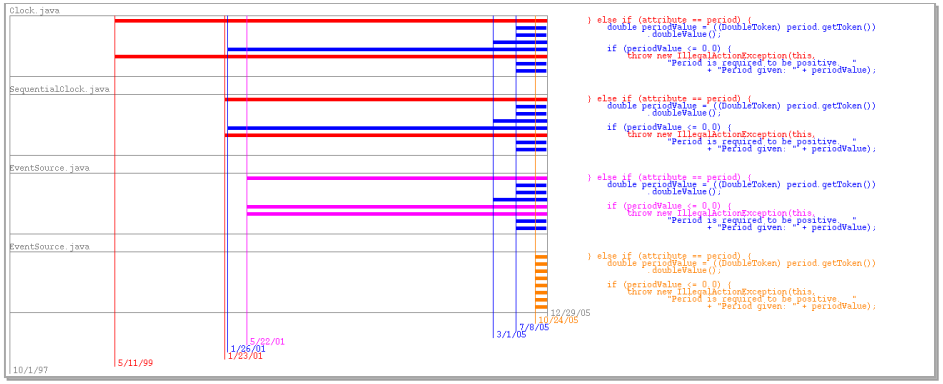
Explicit metamodels enable change

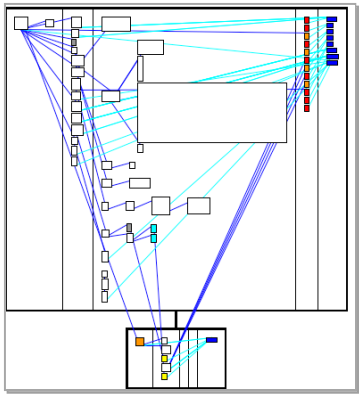


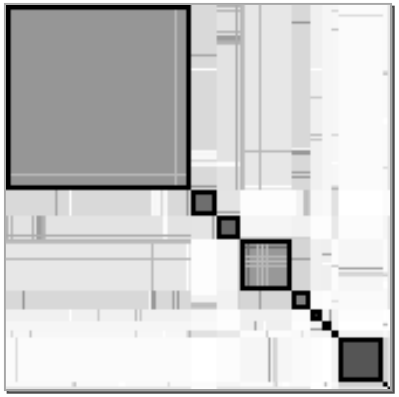
The Story of Moose: an Agile Reengineering Environment. Oscar Nierstrasz, Stéphane Ducasse, and Tudor Gîrba. ESEC/FSE 2005

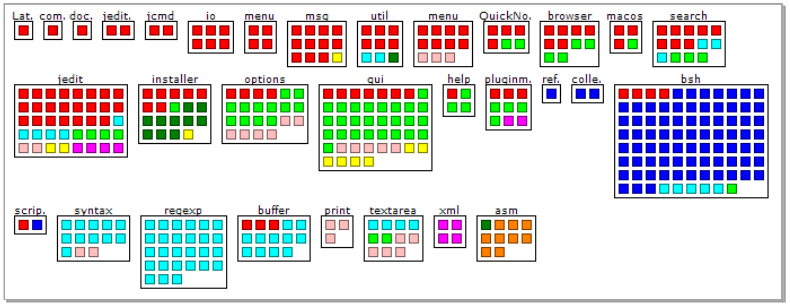
Moose visualizations

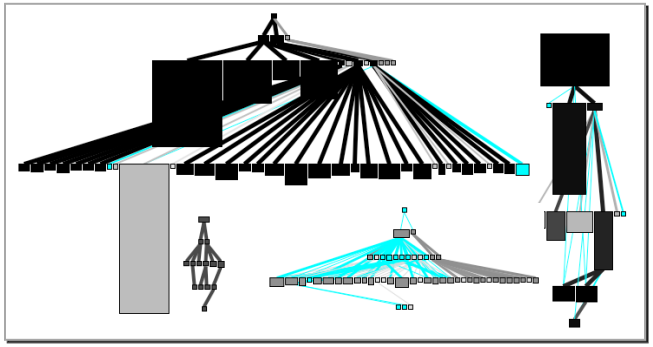


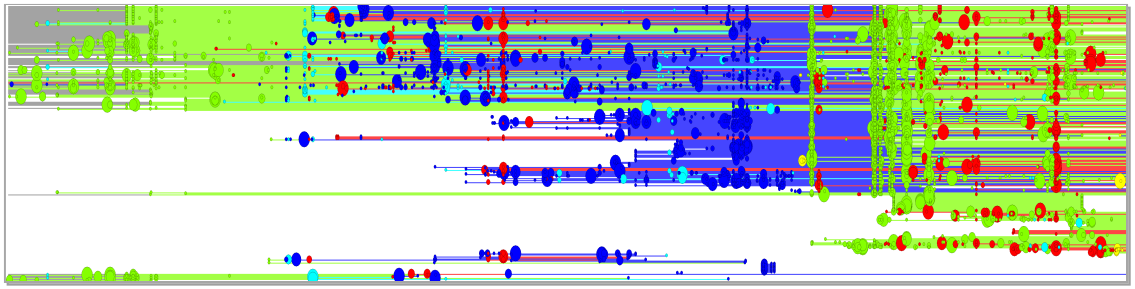




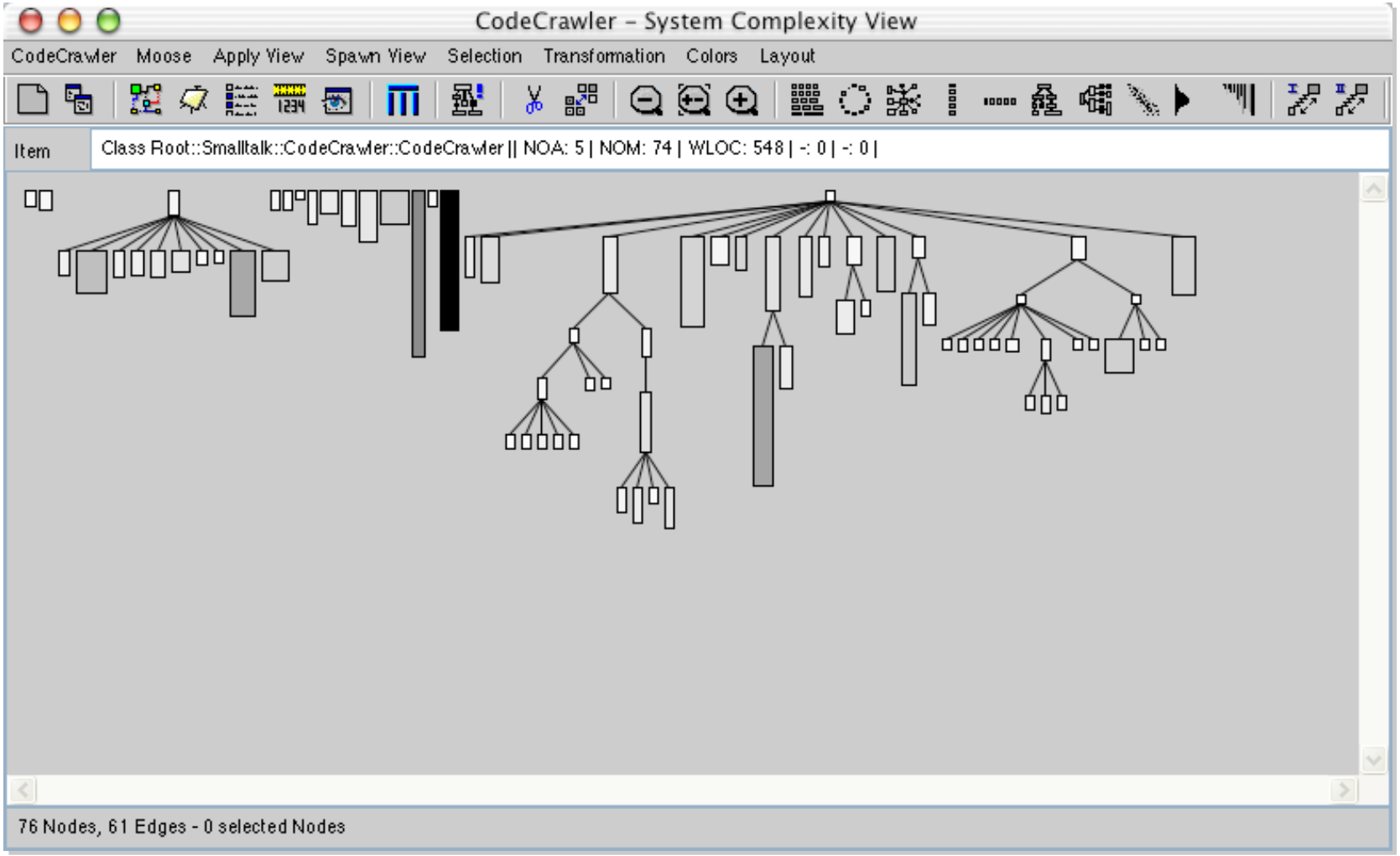




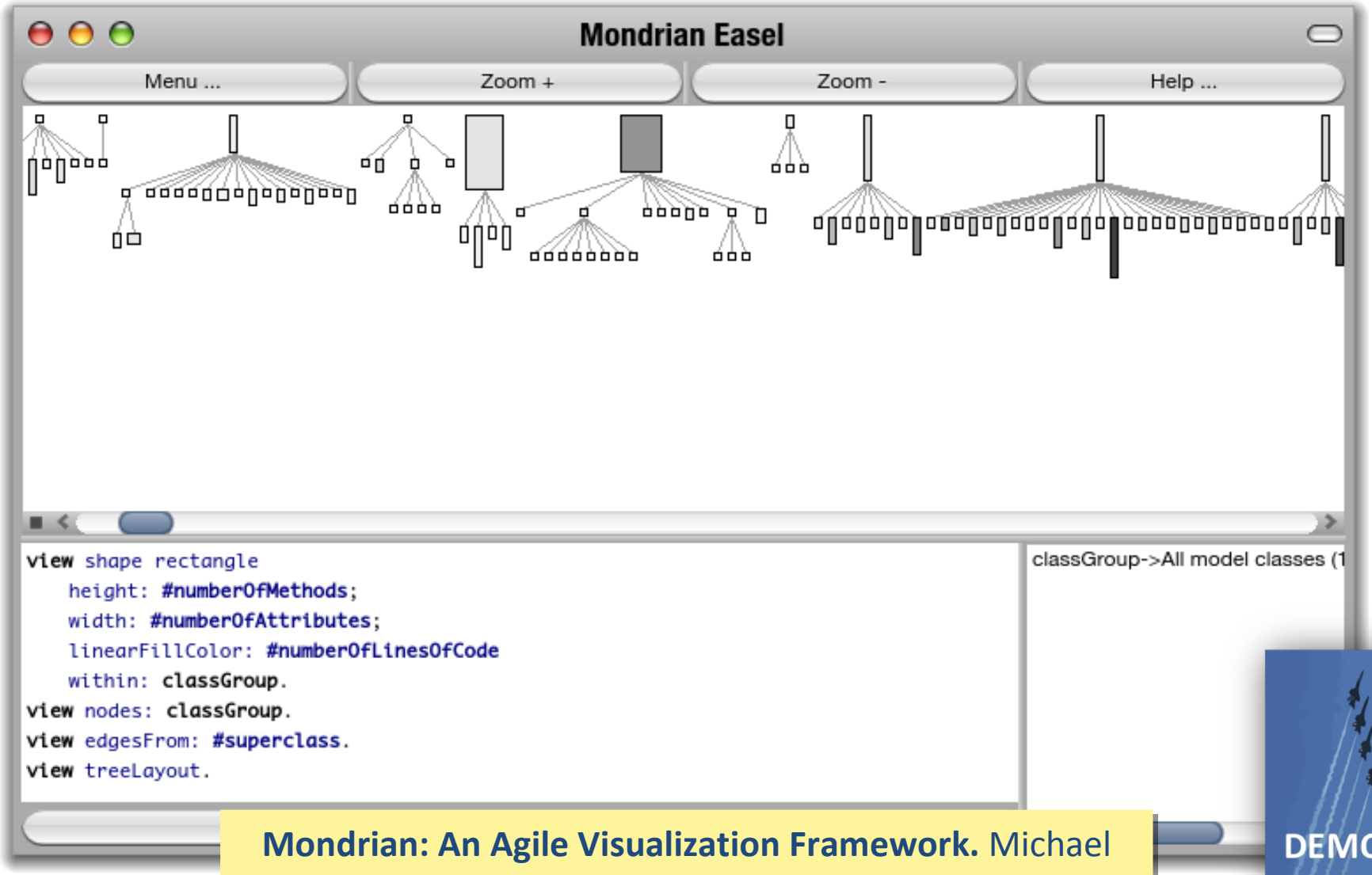




Programming visualizations with CodeCrawler

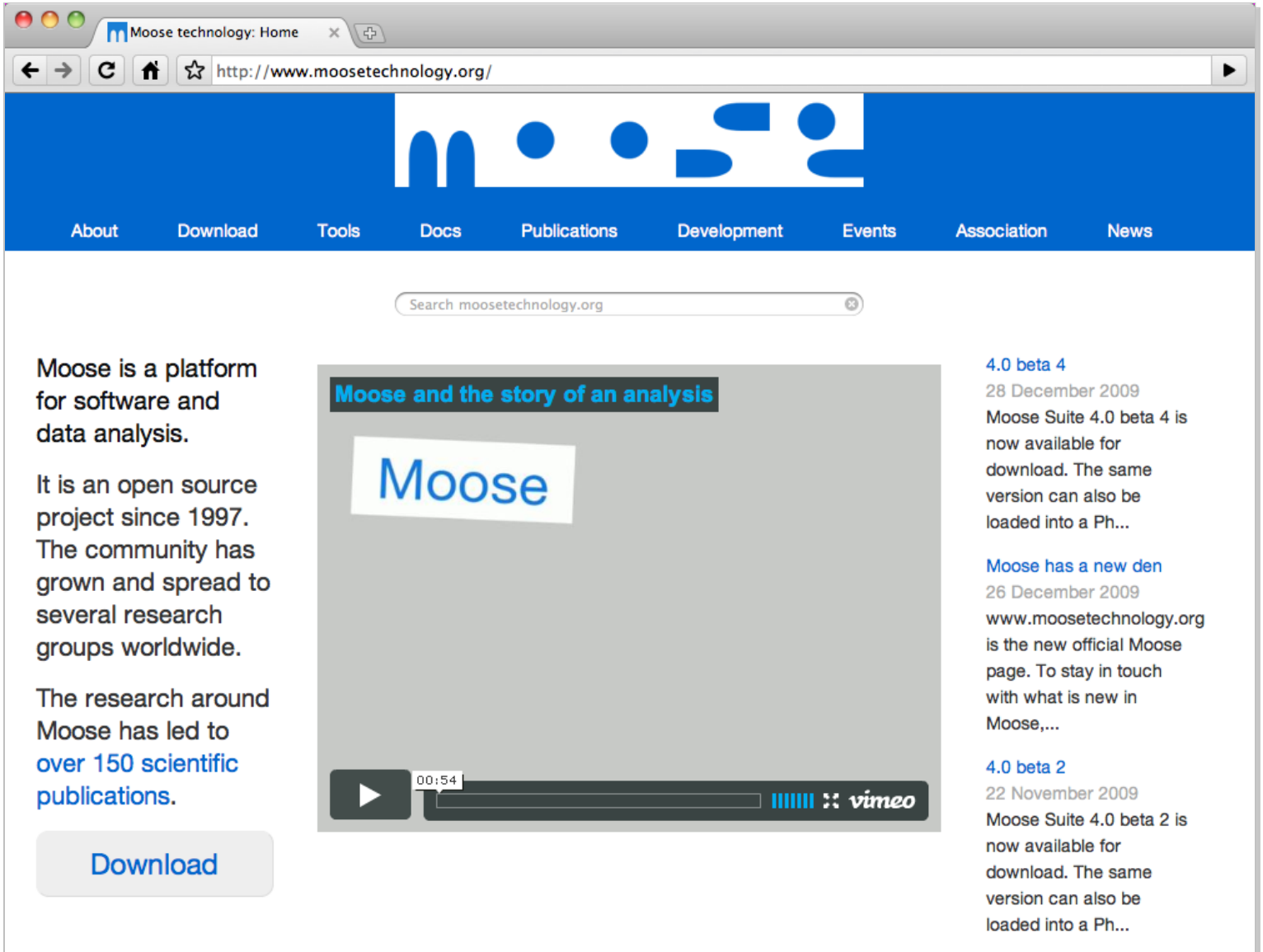


Scripting visualizations with Mondrian

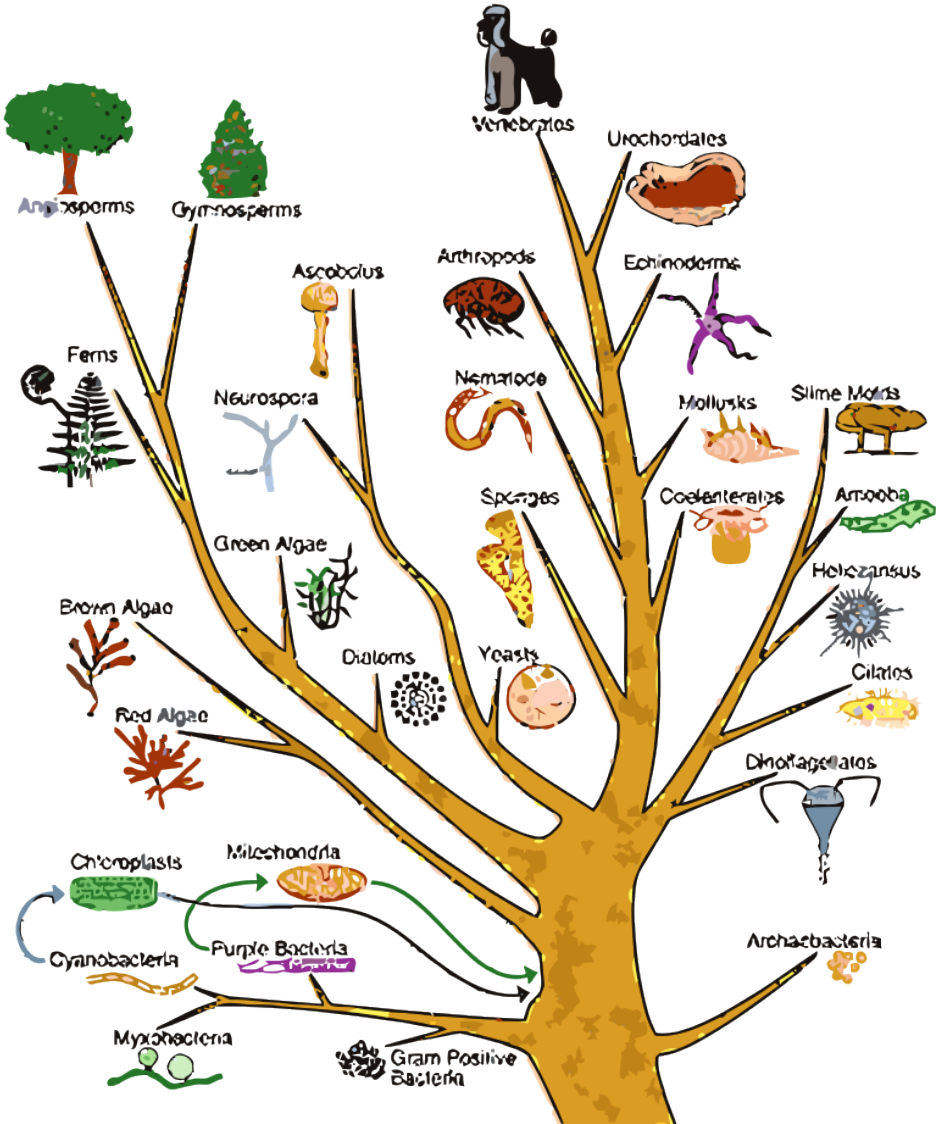


Mondrian: An Agile Visualization Framework. Michael Meyer, Tudor Gîrba, and Mircea Lungu. SoftVis 2006

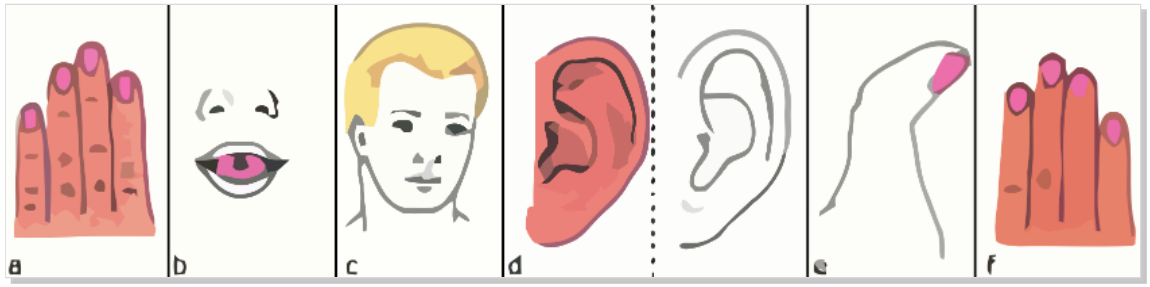
Moose — a platform for collaborative research



Feature inheritance ≠ specialization

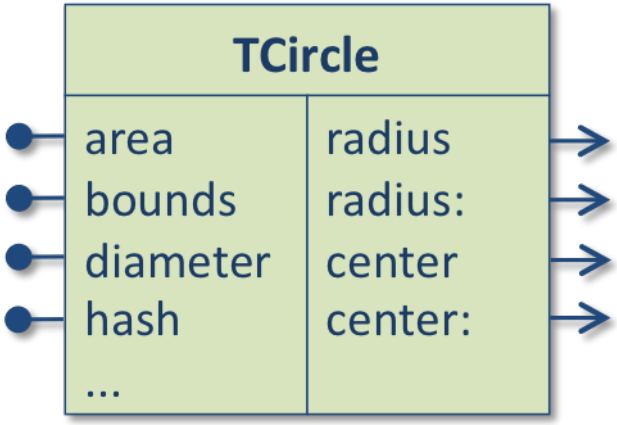


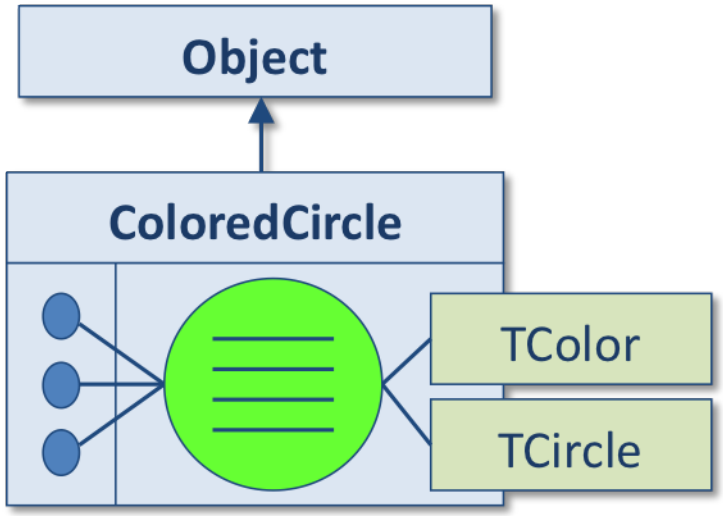
First-class traits enable fine-grained reuse



Class = superclass + state + traits + glue

Traits provide and require methods

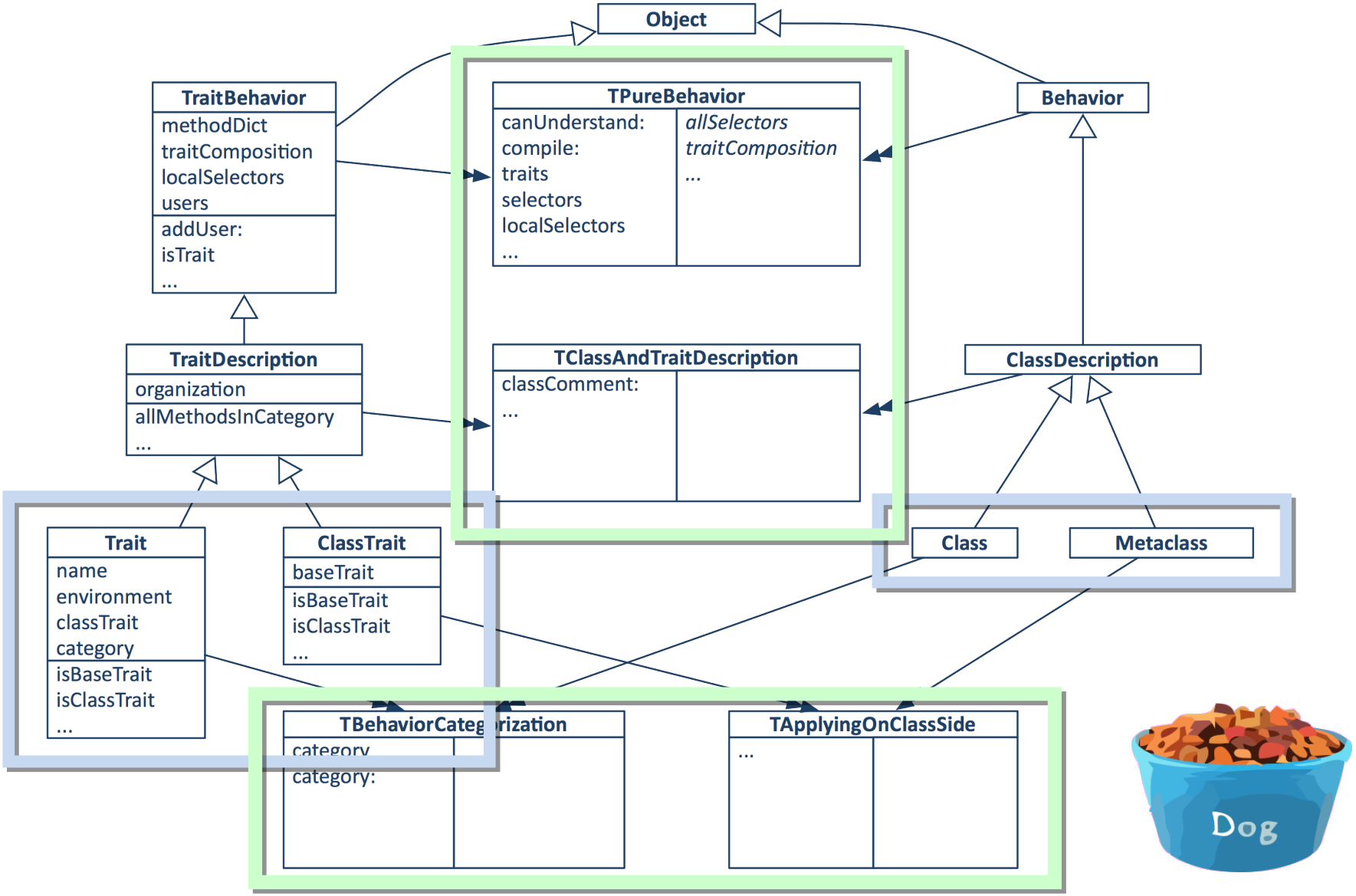




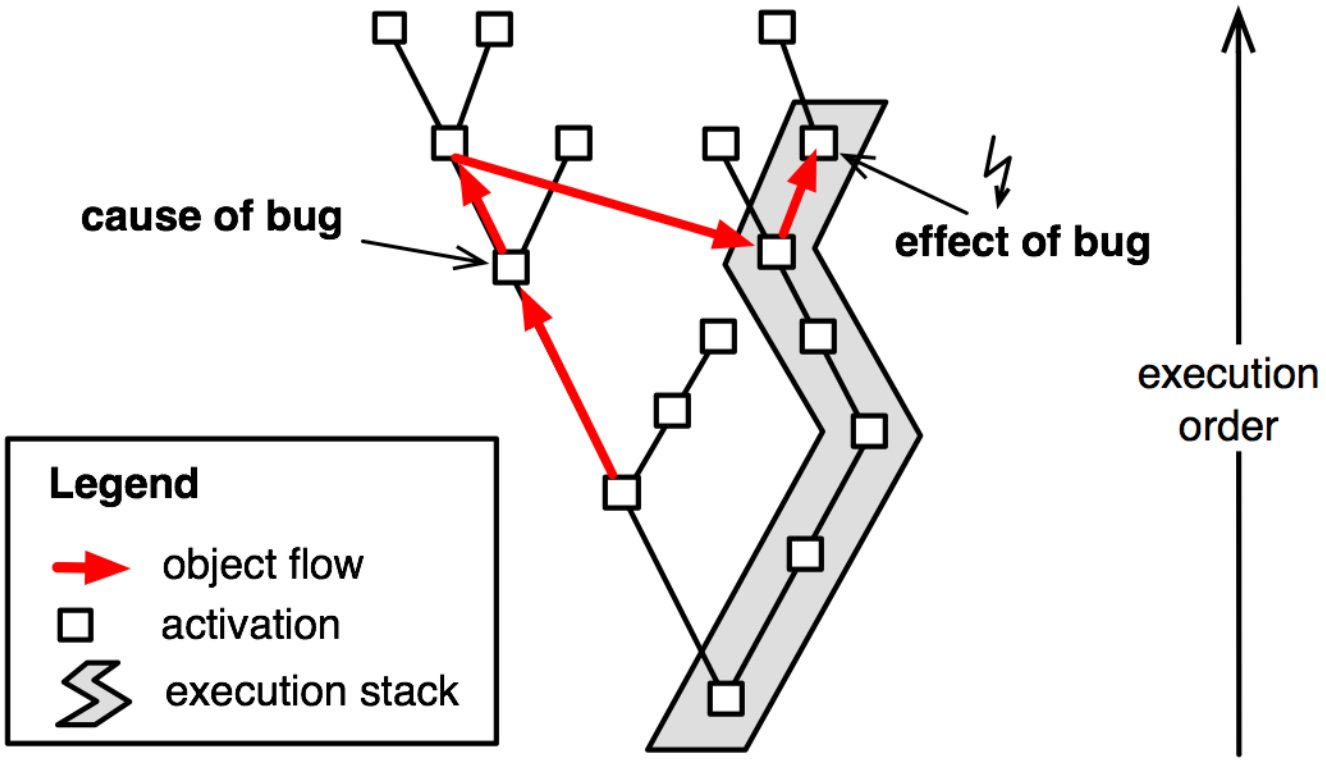
The composing class retains control

Traits: A Mechanism for fine-grained Reuse. Stéphane Ducasse, Oscar Nierstrasz, Nathanael Schärli, Roel Wuyts and Andrew Black. ACM TOPLAS, March 2006

Traits and Classes share common behaviour

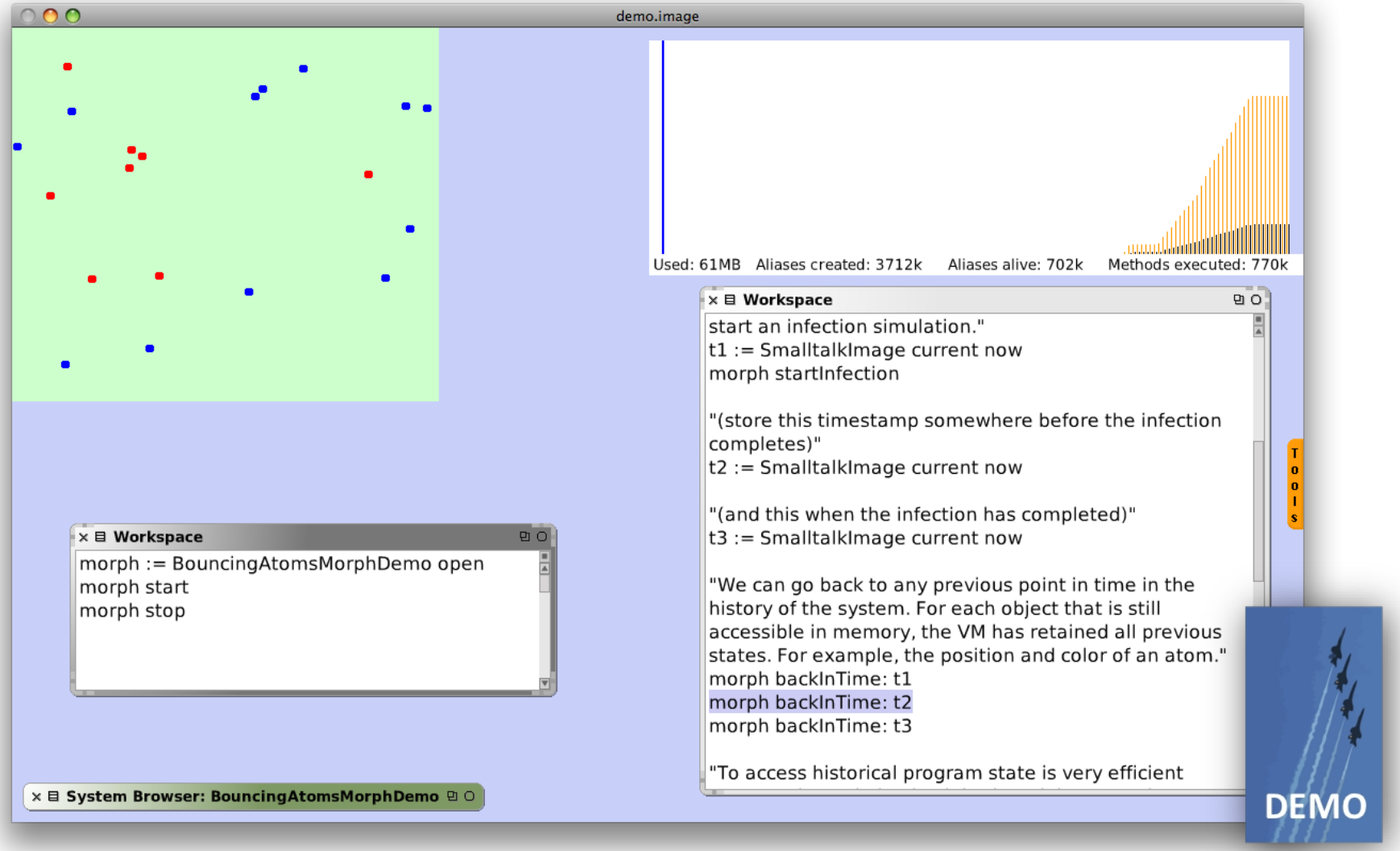


First-class aliases enable object flow analysis

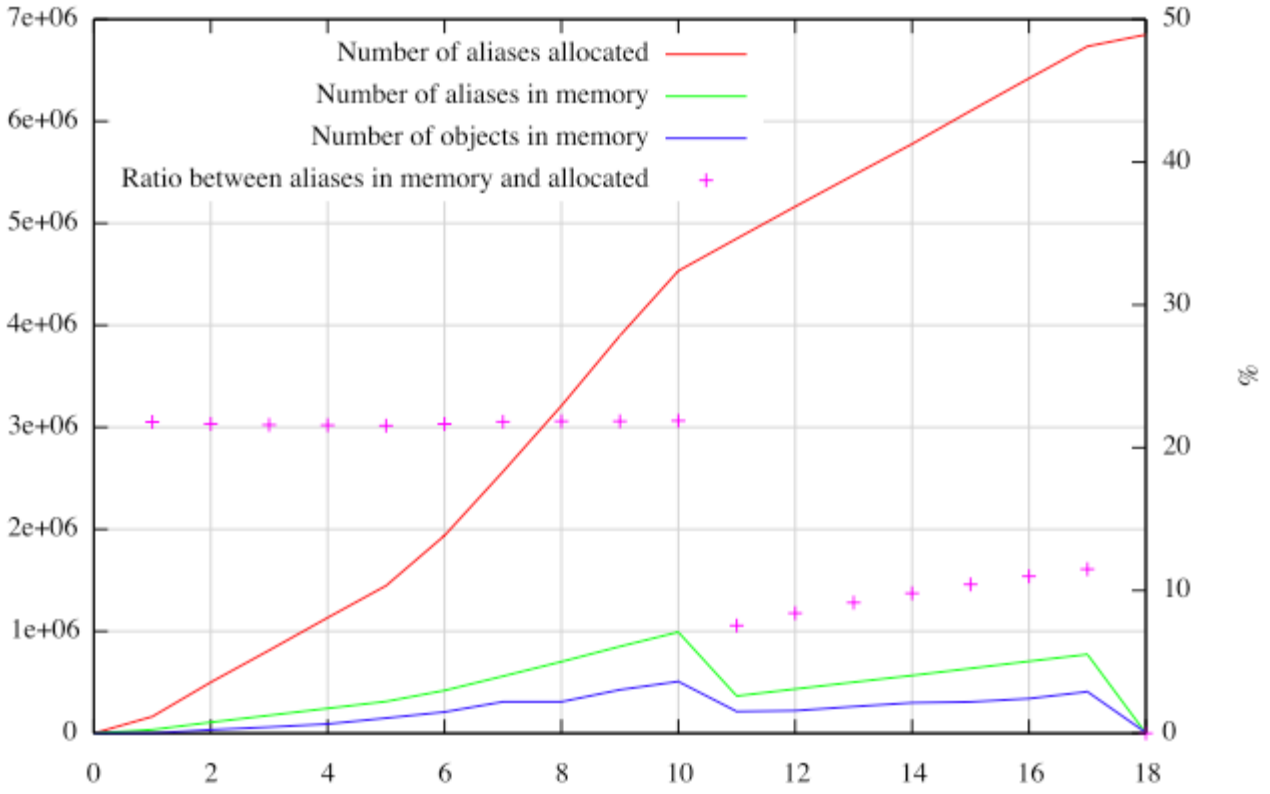


Practical Object-Oriented Back-in-Time Debugging. Adrian Lienhard, Tudor Gîrba, and Oscar Nierstrasz. ECOOP 2008.

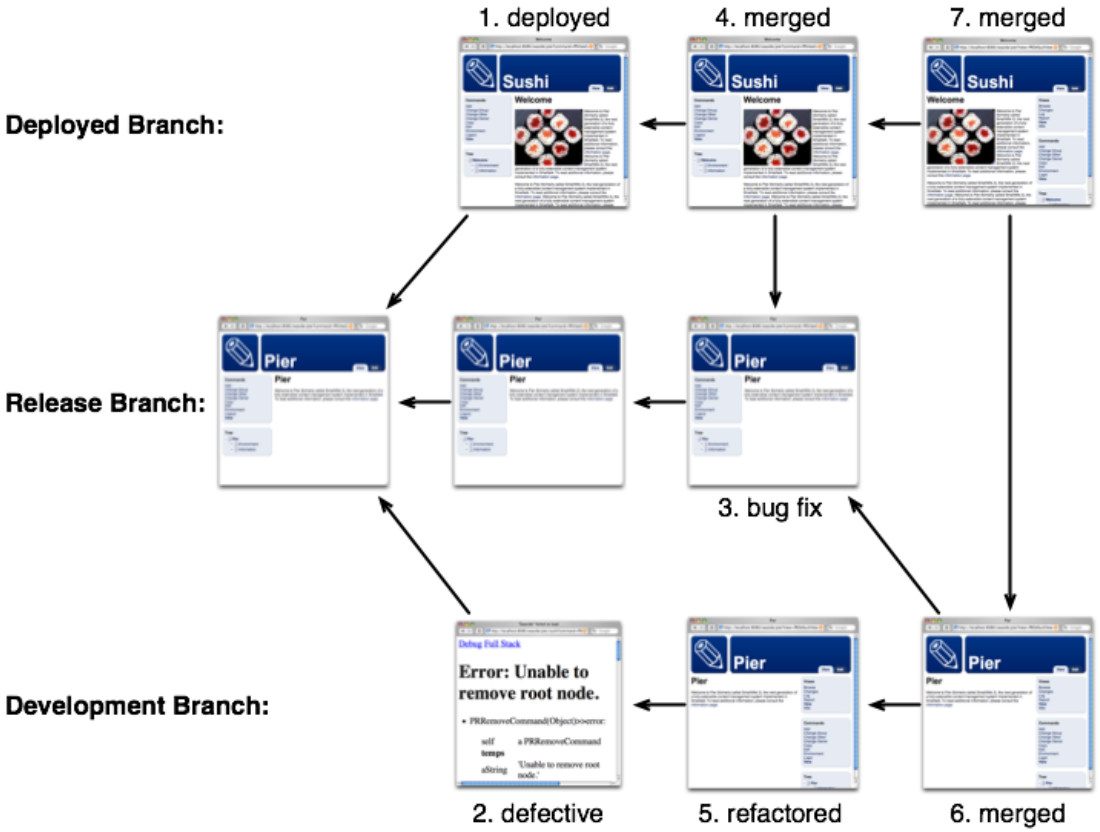
A back-in-time VM with object flow analysis



GC remembers only what is needed

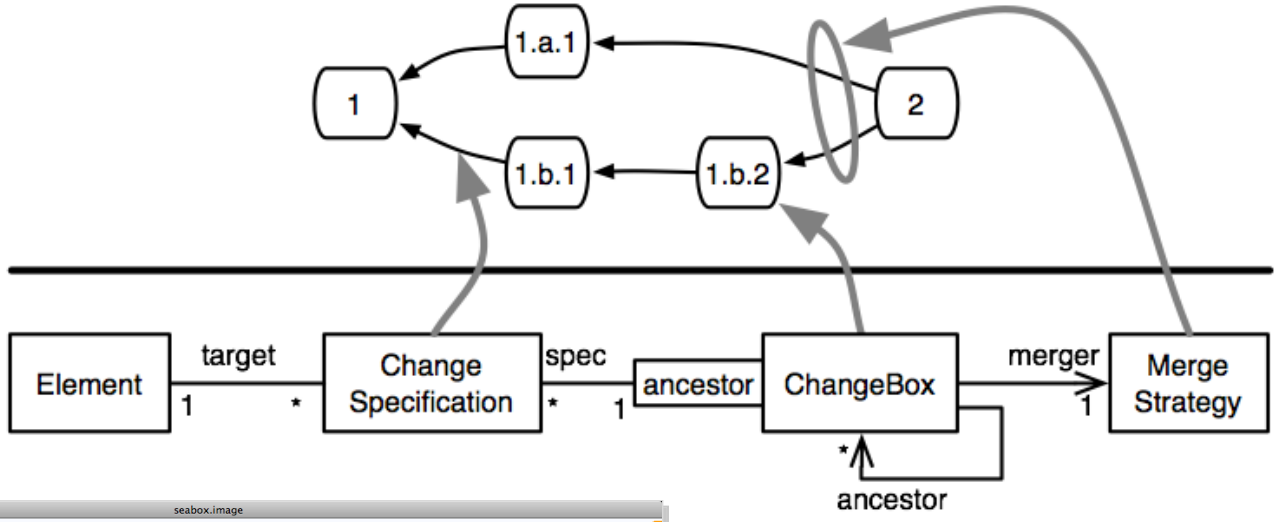


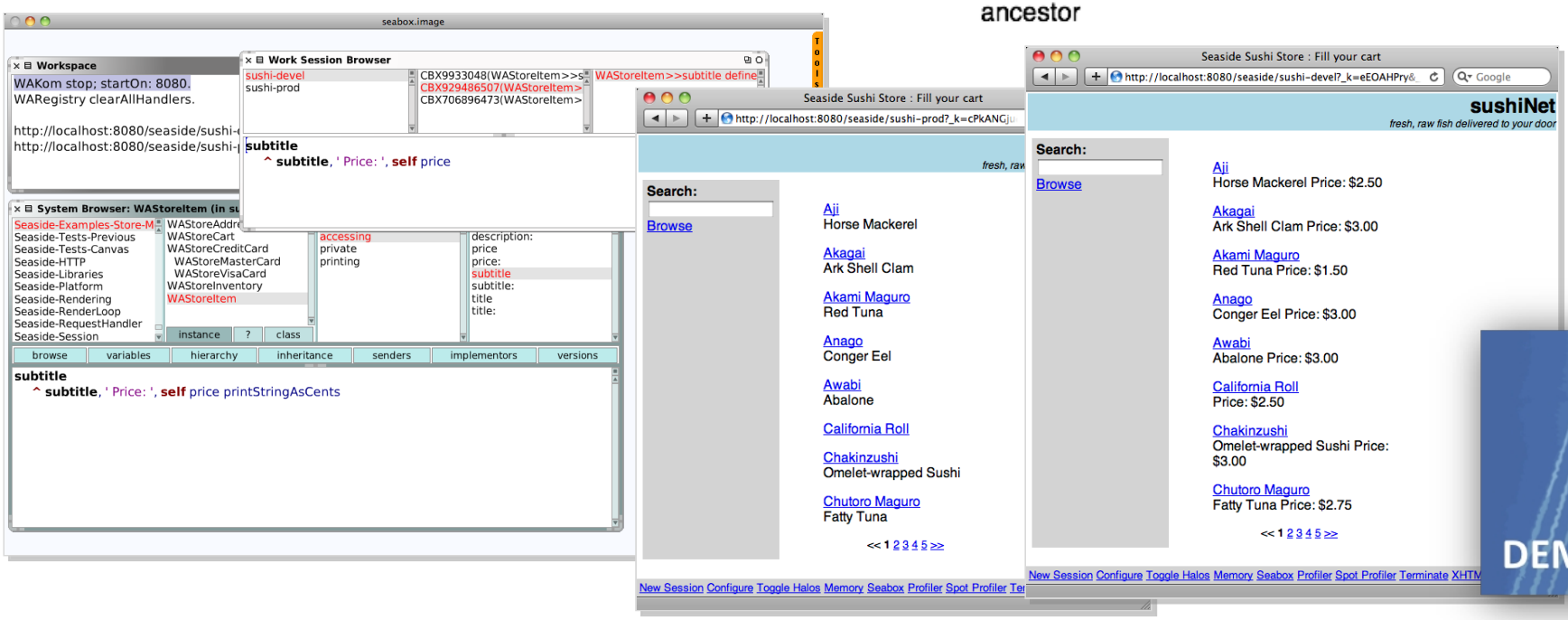
How to manage change in a running system?



Encapsulating and Exploiting Change with Changeboxes. Marcus Denker, et al. ICDL 2007.

First-class changes enable scoped evolution





Roadmap

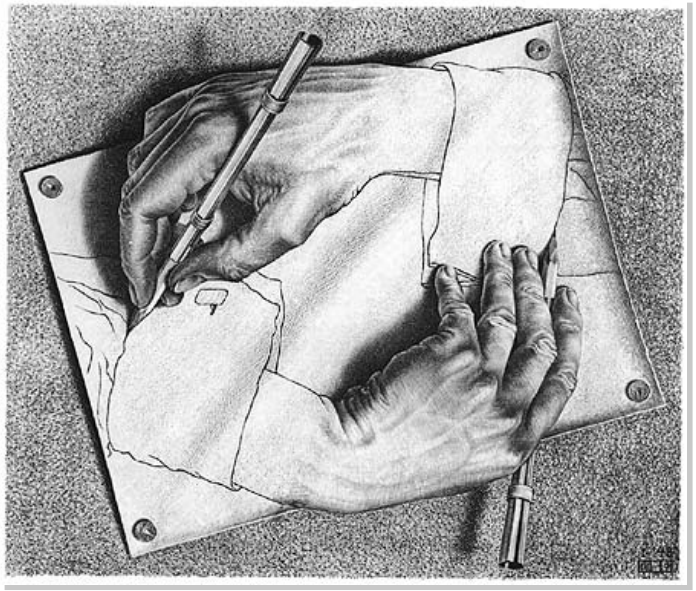
The trouble with change

Smalltalk in a Nutshell

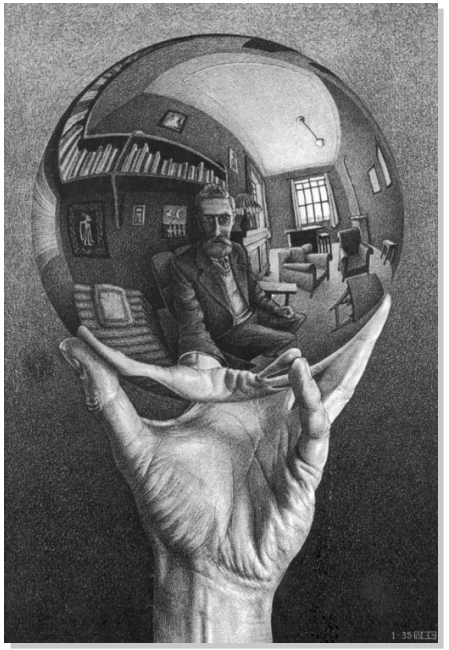
Taming Software Change

Epilogue — Bringing Models Closer to Code

Systems that support change need to be *model-centric* and *context-aware*



Model-centric systems are self-describing and can be adapted dynamically



Context-aware systems control the scope of change to static or dynamic contexts