DFlow – A Platform to Profile Developers

Roberto Minelli and Michele Lanza

REVEAL @ Faculty of Informatics

University of Lugano, Switzerland

http://dflow.inf.usi.ch

SmalltalkHub.com: **DevFlow**

Developers spend a large part of their working time using an **Integrated Development Environment**

Idea

Eclipse

Integrated Development Environment

Visual Studio

XCode

NetBeans

Squeak

Pharo

Dolphin

Cincom

VisualWorks

instantiations

VA Smalltalk

Pharo

Smalltalk IDE

Writing


Writing
Navigating


Writing
Navigating
M. Kersten and G.C. Murphy
"Mylar: a degree-of-interest model for IDEs" AOSD 2005
"Programmers spend more time **navigating the code** than working with it."

Y. Lee, N. Chen, R. Johnson
"Drag-and-drop refactoring: intuitive and efficient program transformation" ICSE 2013

"The current support for refactoring is **unintuitive and inefficient**."

"Devs are forced to open multiple **windows** (tabs). The IDE becomes a **crowded workspace**.

D. Roethlisberger, O. Nierstrasz, S. Ducasse
"Autumn leaves: Curing the window plague in IDEs" WCRE 2009

Researchers proposed various approaches to better support **browsing through software**.

J. Singer, R. Elves, and M. Storey
"Navtracks: supporting navigation in software maintenance" ICSM 2005

M. Kersten, G. C. Murphy
"Mylar: a degree-of-interest model for IDEs" AOSD 2005

To what extent does Pharo **support the navigation**?

Navigating source code with the Pharo IDE

How, **when**, **why** do developers use Pharo to **navigate the system**?

It silently **records** all the Pharo **interactions** while the developer is **programming**.

DFlow-Pharo

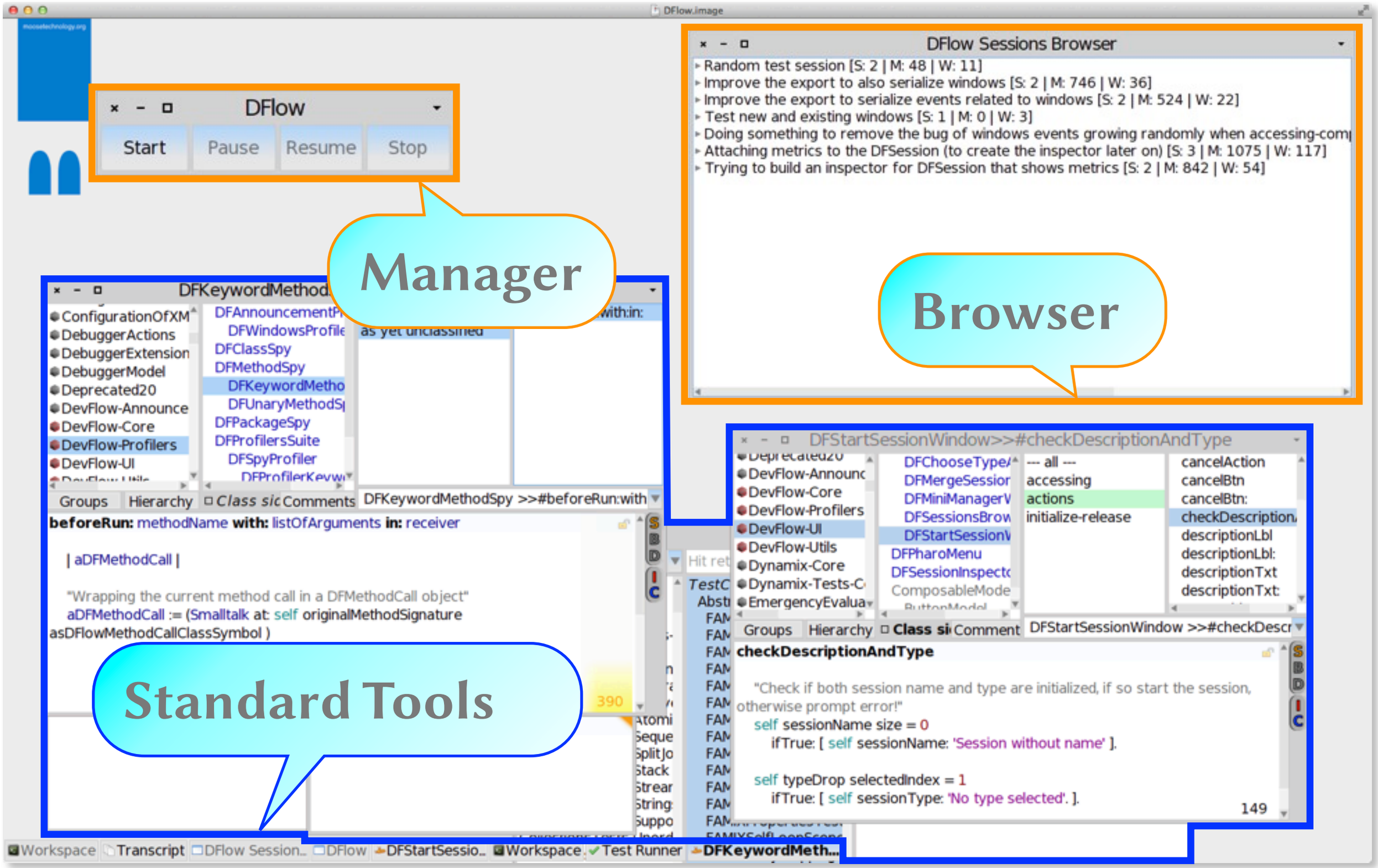
DFlow

A Platform to Profile Developers

It enables **retrospective analyses** through a web-based software **visualization** platform.

DFlow-Web

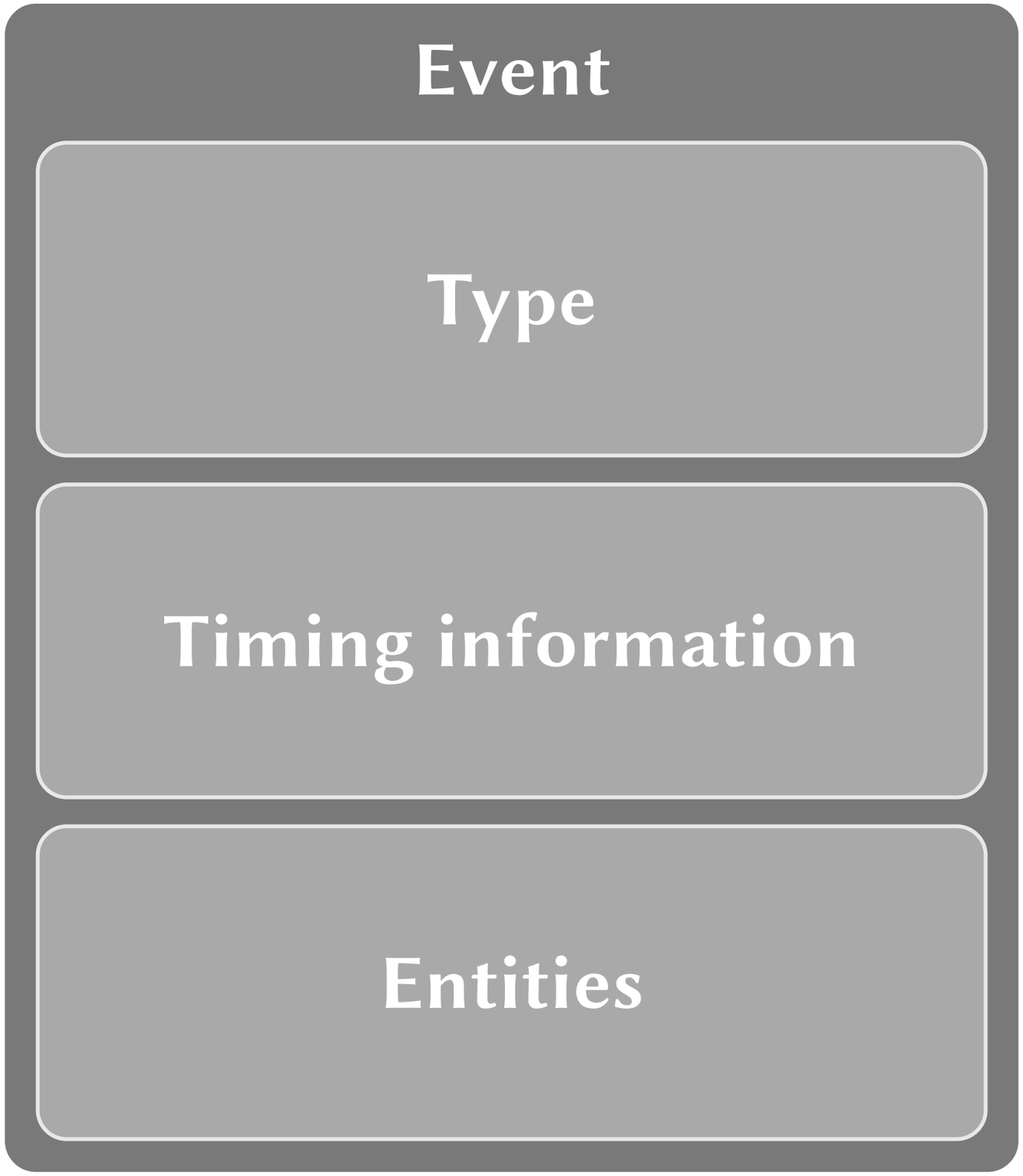
DFlow-Pharo: An Extension to the Pharo IDE

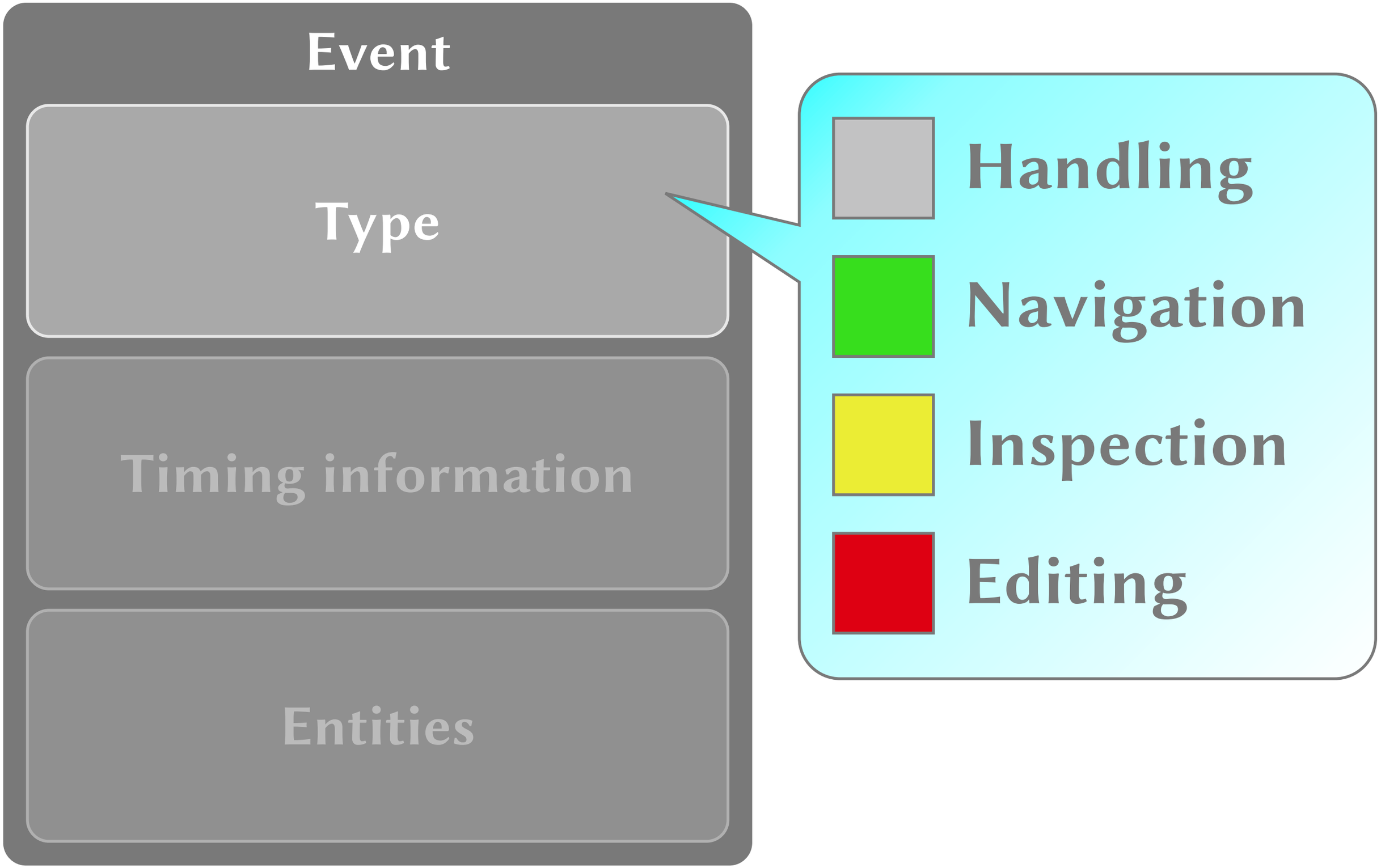


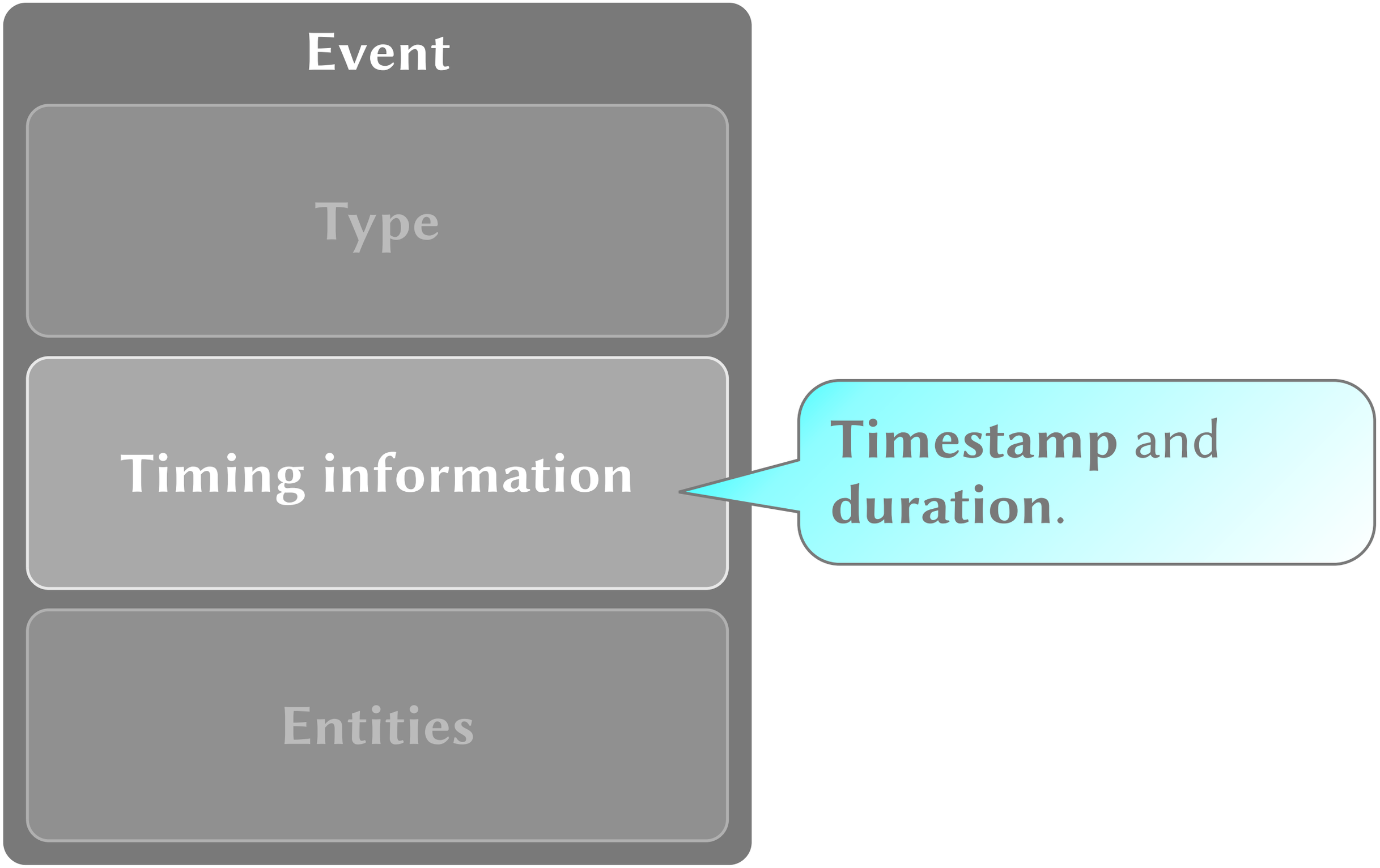
PHARO

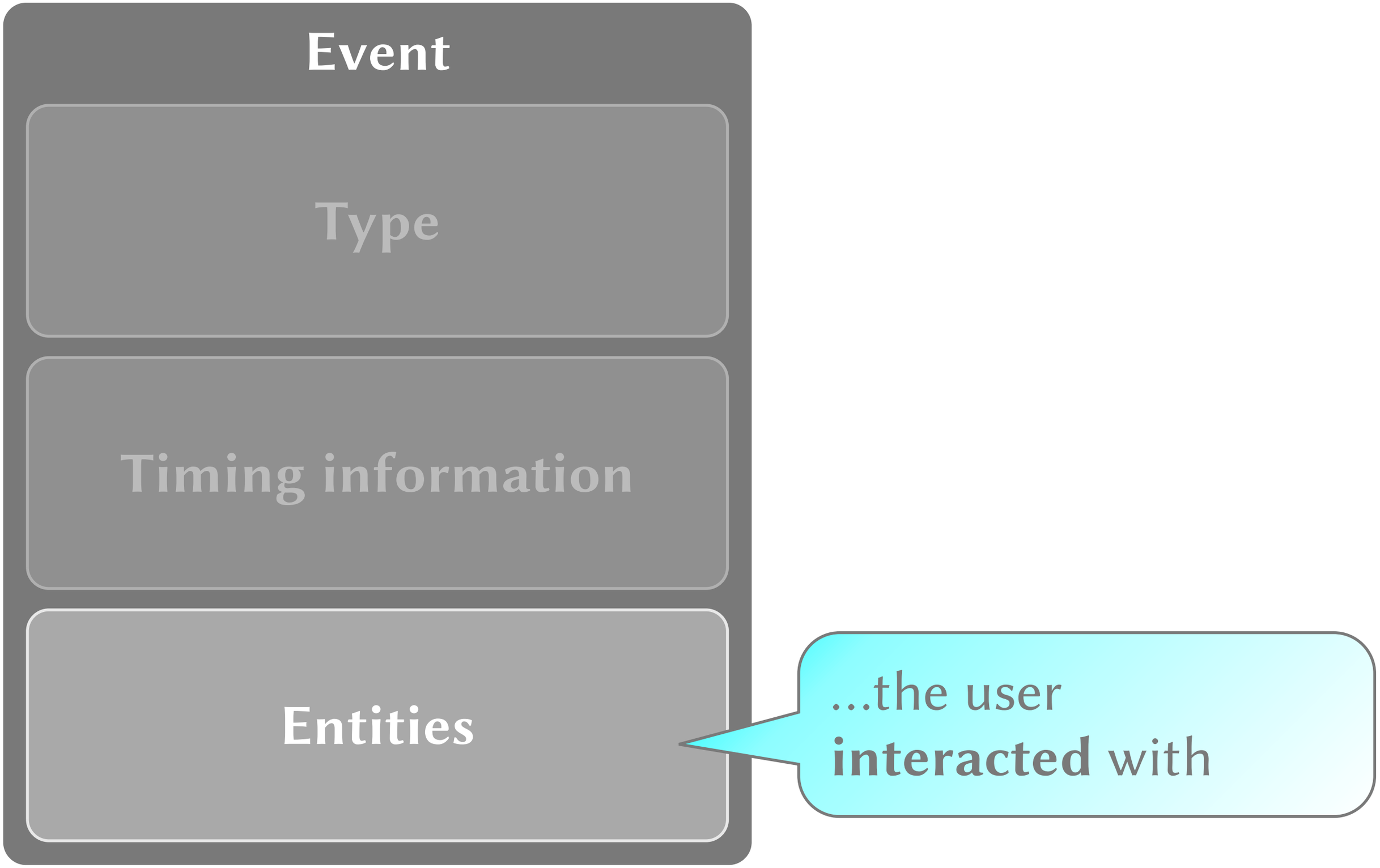
DFlow

It's demo time!


Event
Type
Timing information
Entities


Event
Type
Timing information
Entities
Handling
Navigation
Inspection
Editing


Event
Type
Timing information
Entities
Timestamp and duration.


Event
Type
Timing information
Entities
...the user
interacted with

How do we profile the developer?

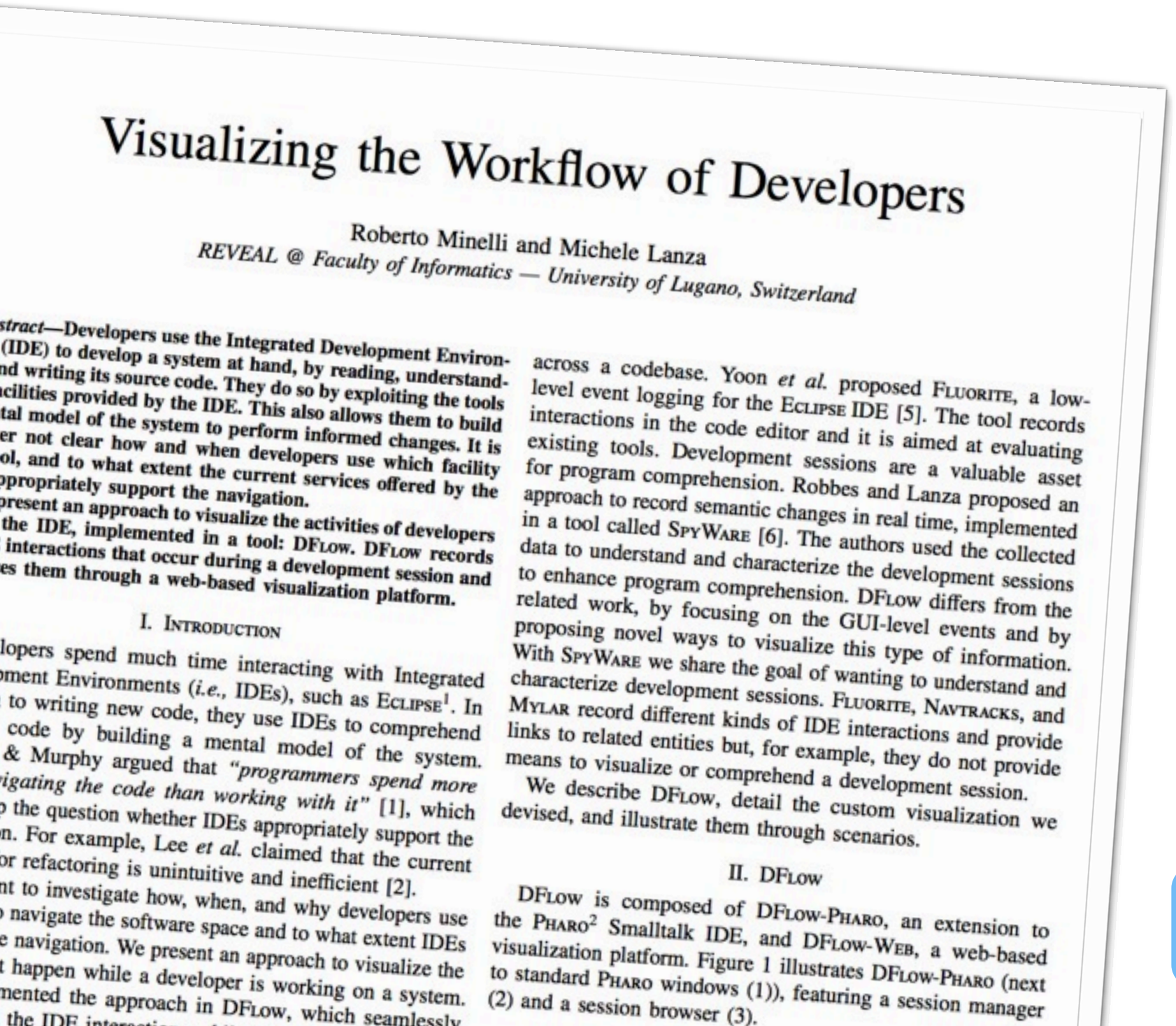
Code
Instrumentation

Custom-made
Profilers

A. Bergel, Bañados, Robbes, Röthlisberger
"**SPY: A Flexible Code Profiling Framework**"
Computer Languages, Systems and Structures 2012


What to do with this
large amount of data?

It's demo time!



Visualizing the Workflow of Developers

Roberto Minelli and Michele Lanza
REVEAL @ Faculty of Informatics — University of Lugano, Switzerland

stract—Developers use the Integrated Development Environ- (IDE) to develop a system at hand, by reading, understand- nd writing its source code. They do so by exploiting the tools cilities provided by the IDE. This also allows them to build tal model of the system to perform informed changes. It is er not clear how and when developers use which facility ol, and to what extent the current services offered by the ppropriately support the navigation.

present an approach to visualize the activities of developers the IDE, implemented in a tool: DFlow. DFlow records interactions that occur during a development session and es them through a web-based visualization platform.

I. Introduction

lopers spend much time interacting with Integrated ment Environments (*i.e.*, IDEs), such as Eclipse[1]. In to writing new code, they use IDEs to comprehend code by building a mental model of the system. & Murphy argued that *"programmers spend more igating the code than working with it"* [1], which the question whether IDEs appropriately support the n. For example, Lee *et al.* claimed that the current or refactoring is unintuitive and inefficient [2]. nt to investigate how, when, and why developers use navigate the software space and to what extent IDEs e navigation. We present an approach to visualize the t happen while a developer is working on a system. mented the approach in DFlow, which seamlessly the IDE interaction

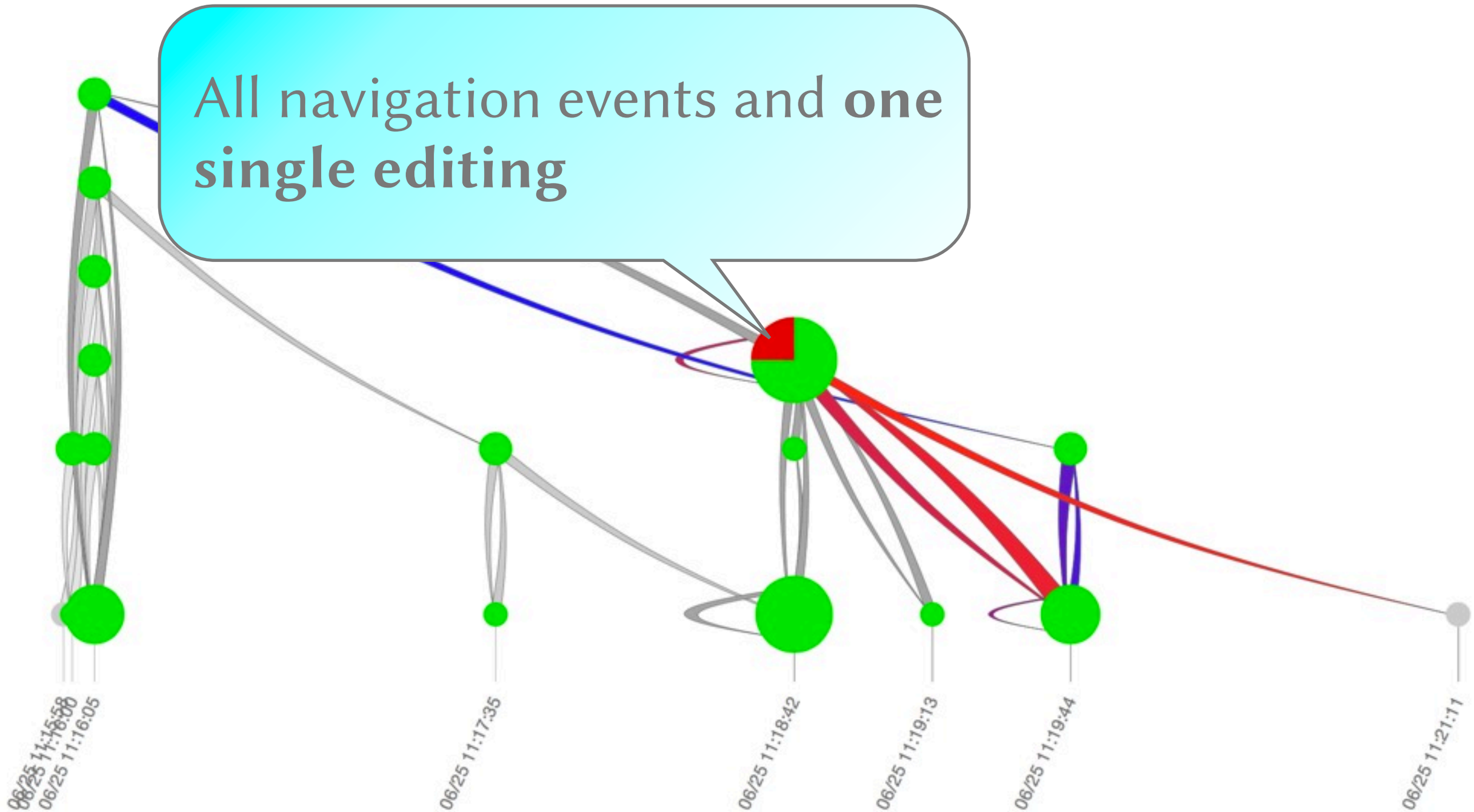
across a codebase. Yoon *et al.* proposed Fluorite, a low-level event logging for the Eclipse IDE [5]. The tool records interactions in the code editor and it is aimed at evaluating existing tools. Development sessions are a valuable asset for program comprehension. Robbes and Lanza proposed an approach to record semantic changes in real time, implemented in a tool called SpyWare [6]. The authors used the collected data to understand and characterize the development sessions to enhance program comprehension. DFlow differs from the related work, by focusing on the GUI-level events and by proposing novel ways to visualize this type of information. With SpyWare we share the goal of wanting to understand and characterize development sessions. Fluorite, Navtracks, and Mylar record different kinds of IDE interactions and provide links to related entities but, for example, they do not provide means to visualize or comprehend a development session.

We describe DFlow, detail the custom visualization we devised, and illustrate them through scenarios.

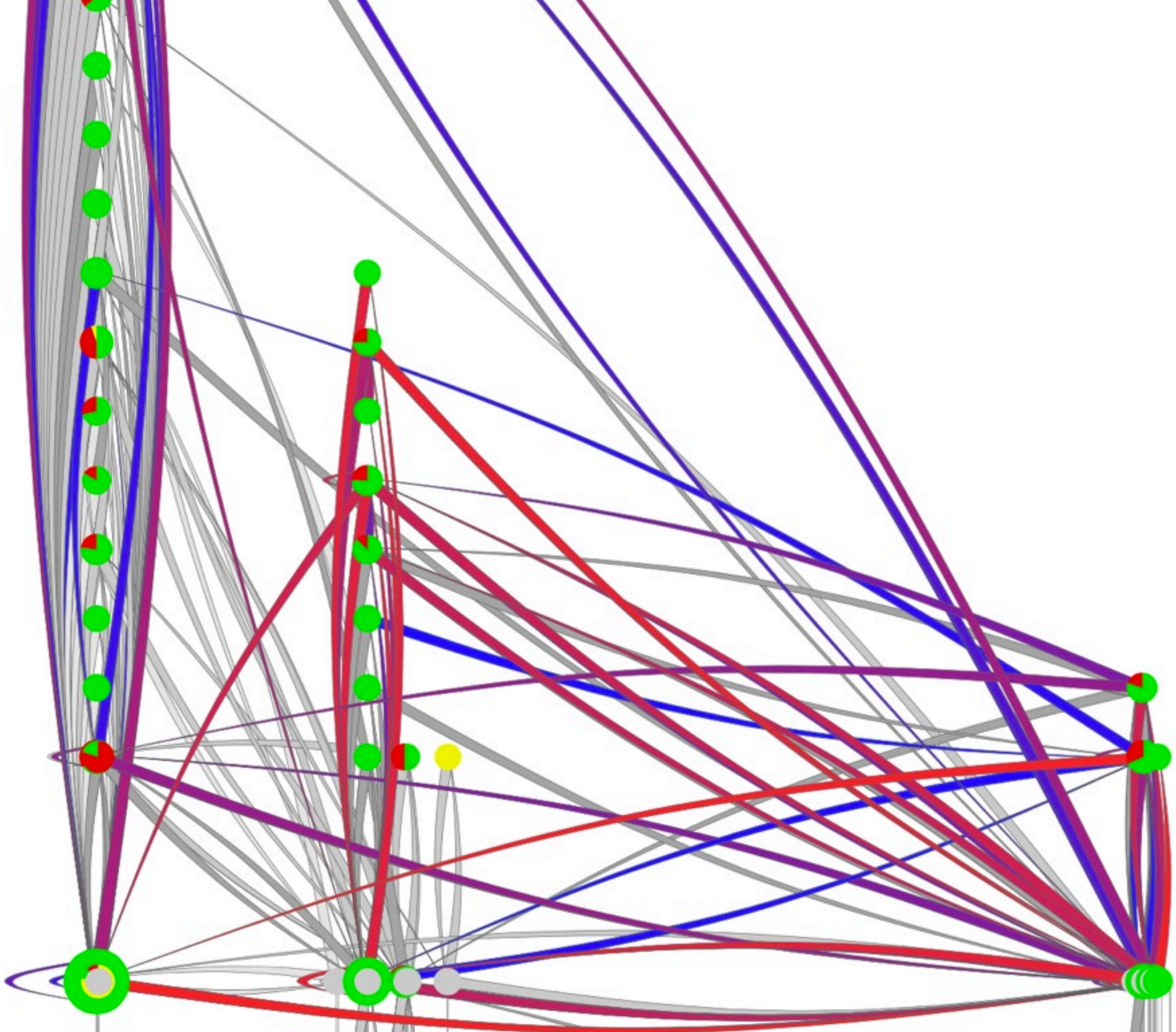
II. DFlow

DFlow is composed of DFlow-Pharo, an extension to the Pharo[2] Smalltalk IDE, and DFlow-Web, a web-based visualization platform. Figure 1 illustrates DFlow-Pharo (next to standard Pharo windows (1)), featuring a session manager (2) and a session browser (3).

R. Minelli and M. Lanza
"Visualizing the Workflow of Developers."
VISSOFT 2013



A **debugging** session...




Green navigation **stacks**
(browsing the API of a class)

Back edges increase the **cognitive weight** of a session.


What's
next?

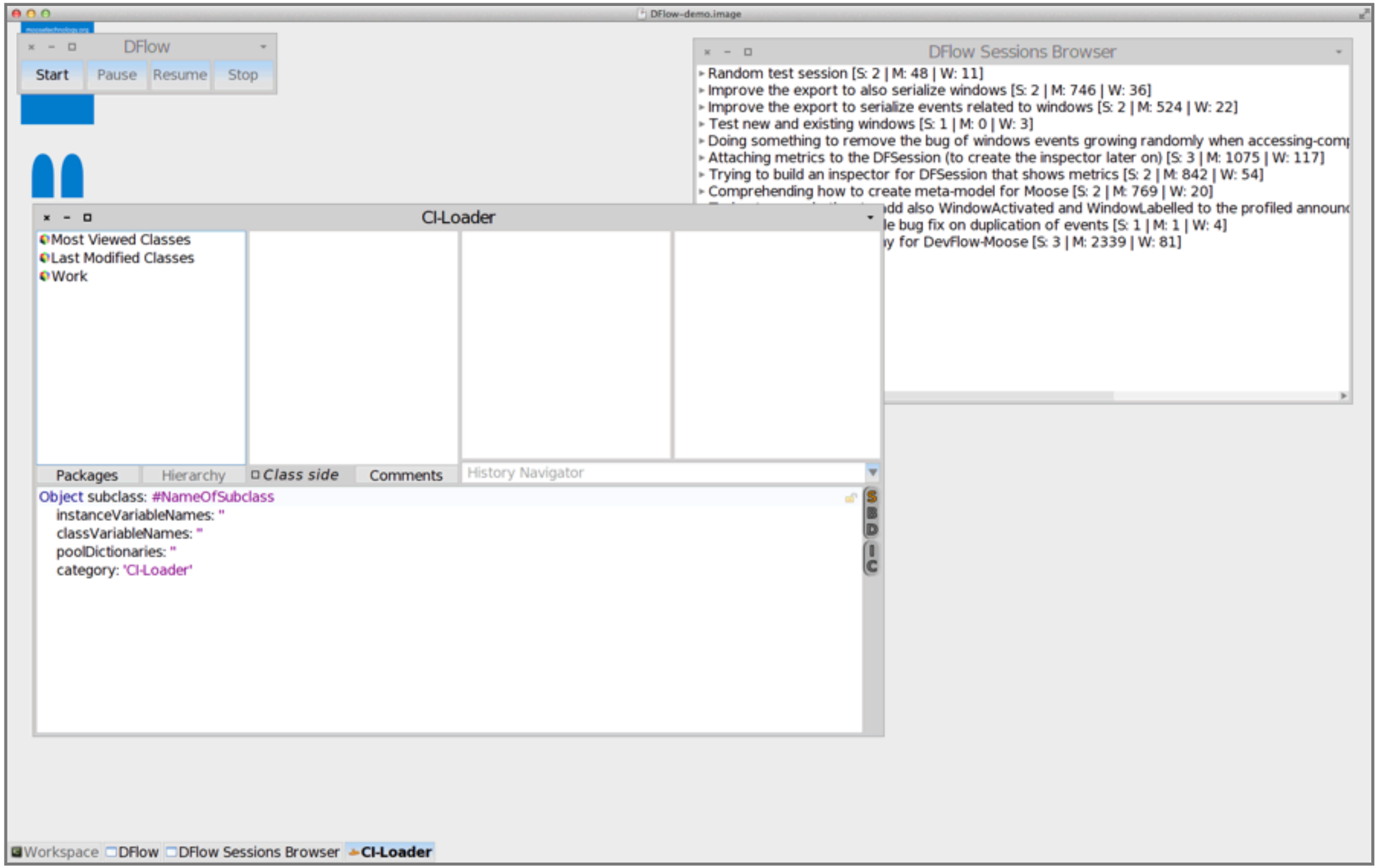
Forward

- **Understand** and **characterize** development **sessions**
- Identify development **workflow bottlenecks** in terms of IDE usage

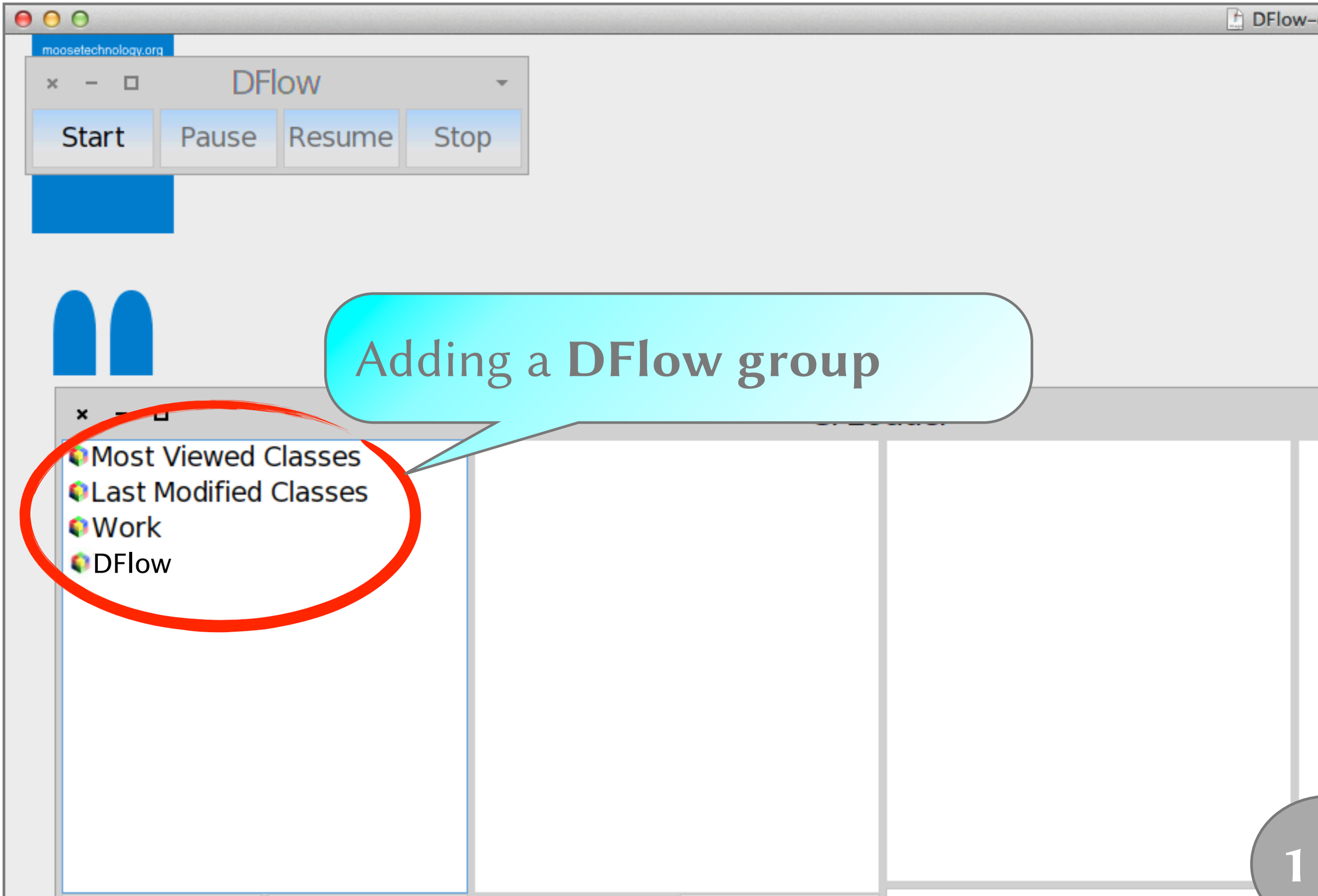
- Use **DFlow data** in a **run-time** context to enhance the IDE

Reverse

Forward


DFlow-demo.image
DFlow
Start
Pause
Resume
Stop
DFlow Sessions Browser
Random test session [S: 2 | M: 48 | W: 11]
Improve the export to also serialize windows [S: 2 | M: 746 | W: 36]
Improve the export to serialize events related to windows [S: 2 | M: 524 | W: 22]
Test new and existing windows [S: 1 | M: 0 | W: 3]
Doing something to remove the bug of windows events growing randomly when accessing-comp
Attaching metrics to the DFSession (to create the inspector later on) [S: 3 | M: 1075 | W: 117]
Trying to build an inspector for DFSession that shows metrics [S: 2 | M: 842 | W: 54]
Comprehending how to create meta-model for Moose [S: 2 | M: 769 | W: 20]
dd also WindowActivated and WindowLabelled to the profiled announc
le bug fix on duplication of events [S: 1 | M: 1 | W: 4]
y for DevFlow-Moose [S: 3 | M: 2339 | W: 81]
CI-Loader
Most Viewed Classes
Last Modified Classes
Work
Packages
Hierarchy
Class side
Comments
History Navigator
Object subclass: #NameOfSubclass
instanceVariableNames: ''
classVariableNames: ''
poolDictionaries: ''
category: 'CI-Loader'
Workspace
DFlow
DFlow Sessions Browser
CI-Loader

Forward



Adding a **DFlow group**

Forward

ZeroDivide

TextMorphForEditView(TextMorph)	handleEdit:
PluggableTextMorph	handleEdit:
PluggableTextMorph	doIt
[...] in SmalltalkEditor class	buildSmalltalkEditorKeymappingsOn:
BlockClosure	cull:
BlockClosure	cull:cull:
BlockClosure	cull:cull:cull:
KMCategoryTarget	completeMatch:buffer:

Proceed | Restart | Into | Over | Through | Full Stack | Run to Here | Where | Create

```
    do: [ :morph | morph doIt ].

  (aBuilder shortcut: #inspectIt)
    category: #SmalltalkEditor
    default: $i command mac | $i ctrl win | $i ctrl unix
    do: [ :morph | morph inspectIt ].

  (aBuilder shortcut: #implementorsOfIt)
    category: #SmalltalkEditor
    default: $m command mac | $m ctrl win | $m ctrl unix
    do: [ :morph | morph implementorsOfIt ].

  (aBuilder shortcut: #sendersOfIt)
    category: #SmalltalkEditor
    default: $n command mac | $n ctrl win | $n ctrl unix
```

self	thisContext	[:morph | morph doIt] in SmalltalkEditor class>>buildSmalltalkEditorKeymappingsOn:
all inst vars	stack top	
superclass	all temp vars	
methodDict	aBuilder	
format	morph	
layout		

Into | Over | Through

Forward

ZeroDivide

TextMorphForEditView(TextMorph)	handleEdit:
PluggableTextMorph	handleEdit:
PluggableTextMorph	doIt
[...] in SmalltalkEditor class	buildSmalltalkEditorKeymappingsOn:
BlockClosure	cull:
BlockClosure	cull:cull:
BlockClosure	cull:cull:cull:
KMCategoryTarget	completeMatch:buffer:

Proceed | Restart | Into | Over | Through | Full Stack | Run to Here | Where | Create

```
    do: [ :morph | morph doIt ].

  (aBuilder shortcut: #inspectIt)
    category: #SmalltalkEditor
    default: $i command mac | $i ctrl win | $i ctrl unix
    do: [ :morph | morph inspectIt ].

  (aBuilder shortcut: #implementorsOfIt)
    category: #SmalltalkEditor
    default: $m command mac | $m ctrl win | $m ctrl unix
    do: [ :morph | morph implementorsOfIt ].

  (aBuilder shortcut: #sendersOfIt)
    category: #SmalltalkEditor
    default: $n command mac | $n ctrl win | $n ctrl unix
```

self	thisContext	[:morph | morph doIt] in SmalltalkEditor class>>buildSmalltalkEditorKeymappingsOn:
all inst vars	stack top	
superclass	all temp vars	
methodDict	aBuilder	
format	morph	
layout		

Into

Into | Over | Through

Forward

ZeroDivide

TextMorphForEditView(TextMorph)	handleEdit:
PluggableTextMorph	handleEdit:
PluggableTextMorph	doIt
[...] in SmalltalkEditor class	buildSmalltalkEditorKeymappingsOn:
BlockClosure	cull:
BlockClosure	cull:cull:
BlockClosure	cull:cull:cull:
KMCategoryTarget	completeMatch:buffer:

Proceed | Restart | Into | Over | Through | Full Stack | Run to Here | Where | Create

```
    do: [ :morph | morph doIt ].

  (aBuilder shortcut: #inspectIt)
    category: #SmalltalkEditor
    default: $i command mac | $i ctrl win | $i ctrl unix
    do: [ :morph | morph inspectIt ].

  (aBuilder shortcut: #implementorsOfIt)
    category: #SmalltalkEditor
    default: $m command mac | $m ctrl win | $m ctrl unix
    do: [ :morph | morph implementorsOfIt ].

  (aBuilder shortcut: #sendersOfIt)
    category: #SmalltalkEditor
    default: $n command mac | $n ctrl win | $n ctrl unix
```

self	thisContext	[:morph | morph doIt] in SmalltalkEditor class>>buildSmalltalkEditorKeymappingsOn:
all inst vars	stack top	
superclass	all temp vars	
methodDict	aBuilder	
format	morph	
layout		

Into

Over

Into | Over | Through

Forward

ZeroDivide

TextMorphForEditView(TextMorph)	handleEdit:
PluggableTextMorph	handleEdit:
PluggableTextMorph	doIt
[...] in SmalltalkEditor class	buildSmalltalkEditorKeymappingsOn:
BlockClosure	cull:
BlockClosure	cull:cull:
BlockClosure	cull:cull:cull:
KMCategoryTarget	completeMatch:buffer:

Proceed | Restart | Into | Over | Through | Full Stack | Run to Here | Where | Create

```
    do: [ :morph | morph doIt ].

  (aBuilder shortcut: #inspectIt)
    category: #SmalltalkEditor
    default: $i command mac | $i ctrl win | $i ctrl unix
    do: [ :morph | morph inspectIt ].

  (aBuilder shortcut: #implementorsOfIt)
    category: #SmalltalkEditor
    default: $m command mac | $m ctrl win | $m ctrl unix
    do: [ :morph | morph implementorsOfIt ].

  (aBuilder shortcut: #sendersOfIt)
    category: #SmalltalkEditor
    default: $n command mac | $n ctrl win | $n ctrl unix
```

self	thisContext	[:morph | morph doIt] in SmalltalkEditor class>>buildSmalltalkEditorKeymappingsOn:
all inst vars	stack top	
superclass	all temp vars	
methodDict	aBuilder	
format	morph	
layout		

Into

Over

Over

Into | Over | Through

Forward

ZeroDivide

TextMorphForEditView(TextMorph)	handleEdit:
PluggableTextMorph	handleEdit:
PluggableTextMorph	doIt
[...] in SmalltalkEditor class	buildSmalltalkEditorKeymappingsOn:
BlockClosure	cull:
BlockClosure	cull:cull:
BlockClosure	cull:cull:cull:
KMCategoryTarget	completeMatch:buffer:

Proceed | Restart | Into | Over | Through | Full Stack | Run to Here | Where | Create

```
    do: [ :morph | morph doIt ].

  (aBuilder shortcut: #inspectIt)
    category: #SmalltalkEditor
    default: $i command mac | $i ctrl win | $i ctrl unix
    do: [ :morph | morph inspectIt ].

  (aBuilder shortcut: #implementorsOfIt)
    category: #SmalltalkEditor
    default: $m command mac | $m ctrl win | $m ctrl unix
    do: [ :morph | morph implementorsOfIt ].

  (aBuilder shortcut: #sendersOfIt)
    category: #SmalltalkEditor
    default: $n command mac | $n ctrl win | $n ctrl unix
```

self	thisContext	[:morph | morph doIt] in SmalltalkEditor class>>buildSmalltalkEditorKeymappingsOn:
all inst vars	stack top	
superclass	all temp vars	
methodDict	aBuilder	
format	morph	
layout		

Into

Over

Over

Into

Into

Over

Through

Forward

ZeroDivide

TextMorphForEditView(TextMorph)	handleEdit:
PluggableTextMorph	handleEdit:
PluggableTextMorph	doIt
[...] in SmalltalkEditor class	buildSmalltalkEditorKeymappingsOn:
BlockClosure	cull:
BlockClosure	cull:cull:
BlockClosure	cull:cull:cull:
KMCategoryTarget	completeMatch:buffer:

Proceed | Restart | Into | Over | Through | Full Stack | Run to Here | Where | Create

```
    do: [ :morph | morph doIt ].

  (aBuilder shortcut: #inspectIt)
    category: #SmalltalkEditor
    default: $i command mac | $i ctrl win | $i ctrl unix
    do: [ :morph | morph inspectIt ].

  (aBuilder shortcut: #implementorsOfIt)
    category: #SmalltalkEditor
    default: $m command mac | $m ctrl win | $m ctrl unix
    do: [ :morph | morph implementorsOfIt ].

  (aBuilder shortcut: #sendersOfIt)
    category: #SmalltalkEditor
    default: $n command mac | $n ctrl win | $n ctrl unix
```

self	thisContext	[:morph | morph doIt] in SmalltalkEditor class>>buildSmalltalkEditorKeymappingsOn:
all inst vars	stack top	
superclass	all temp vars	
methodDict	aBuilder	
format	morph	
layout		

Into

Over

Over

Into

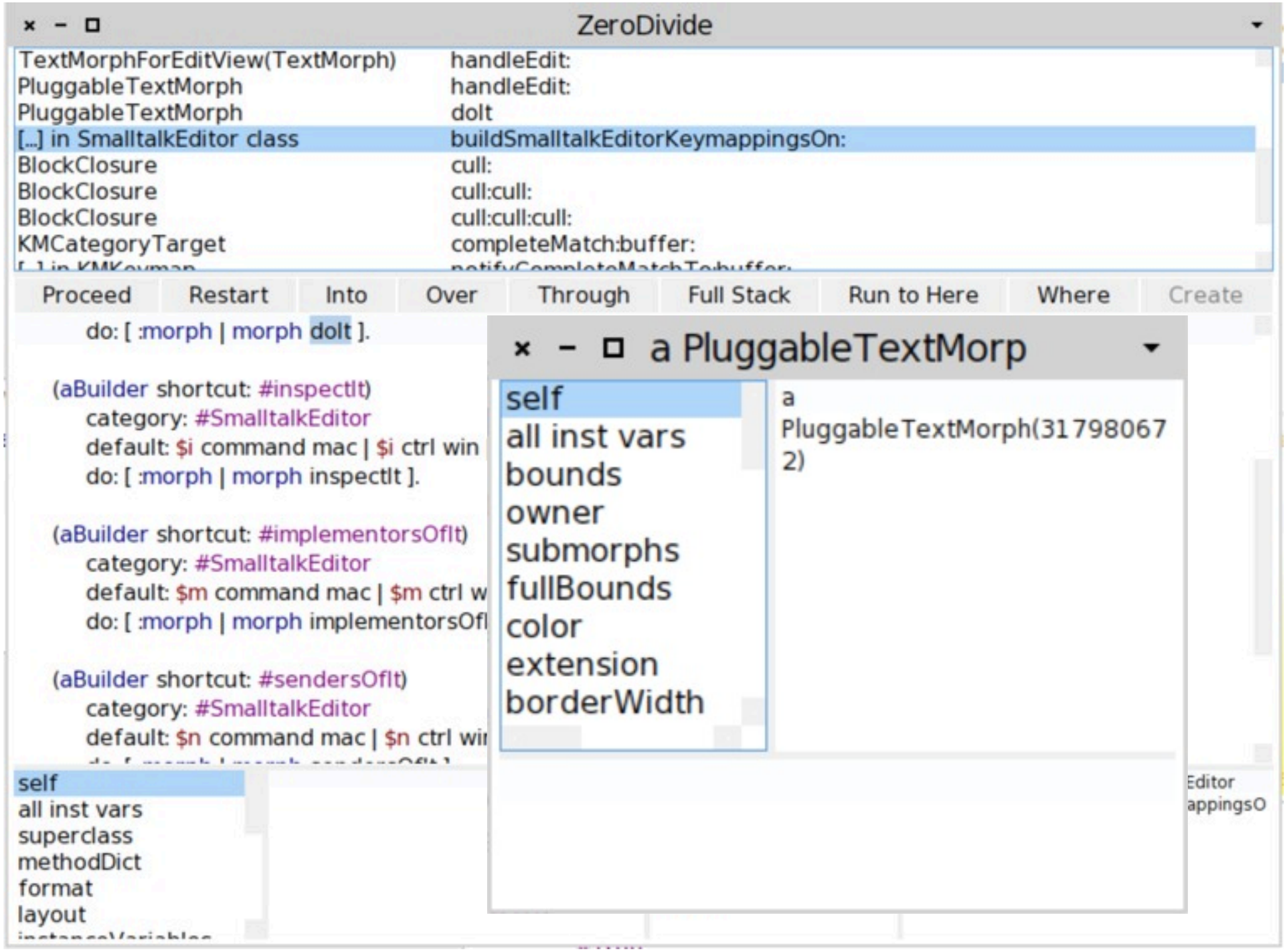
Into

Into

Over

Through

Forward



Into

Over

Over

Into

Into

#inspect

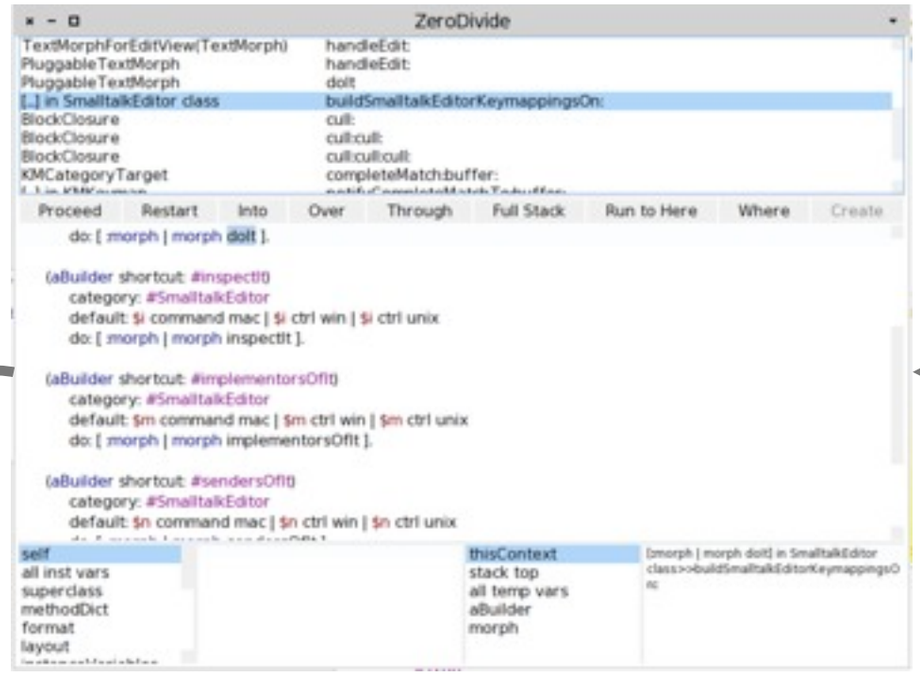
Into

Over

Through

Forward

...next time



DFlow

REC

Into

Over

Over

Into

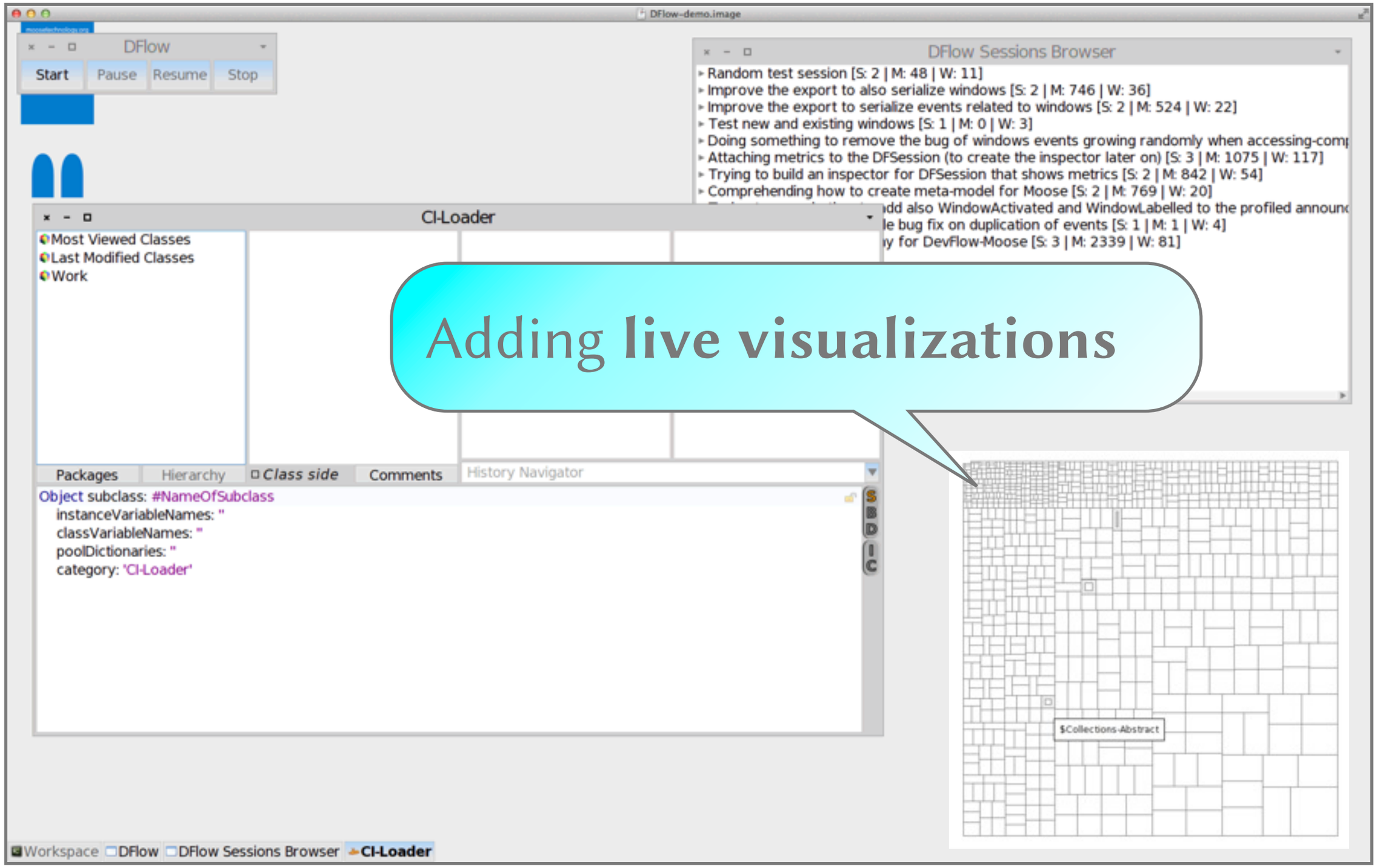
Into

#inspect

DFlow data + **context**

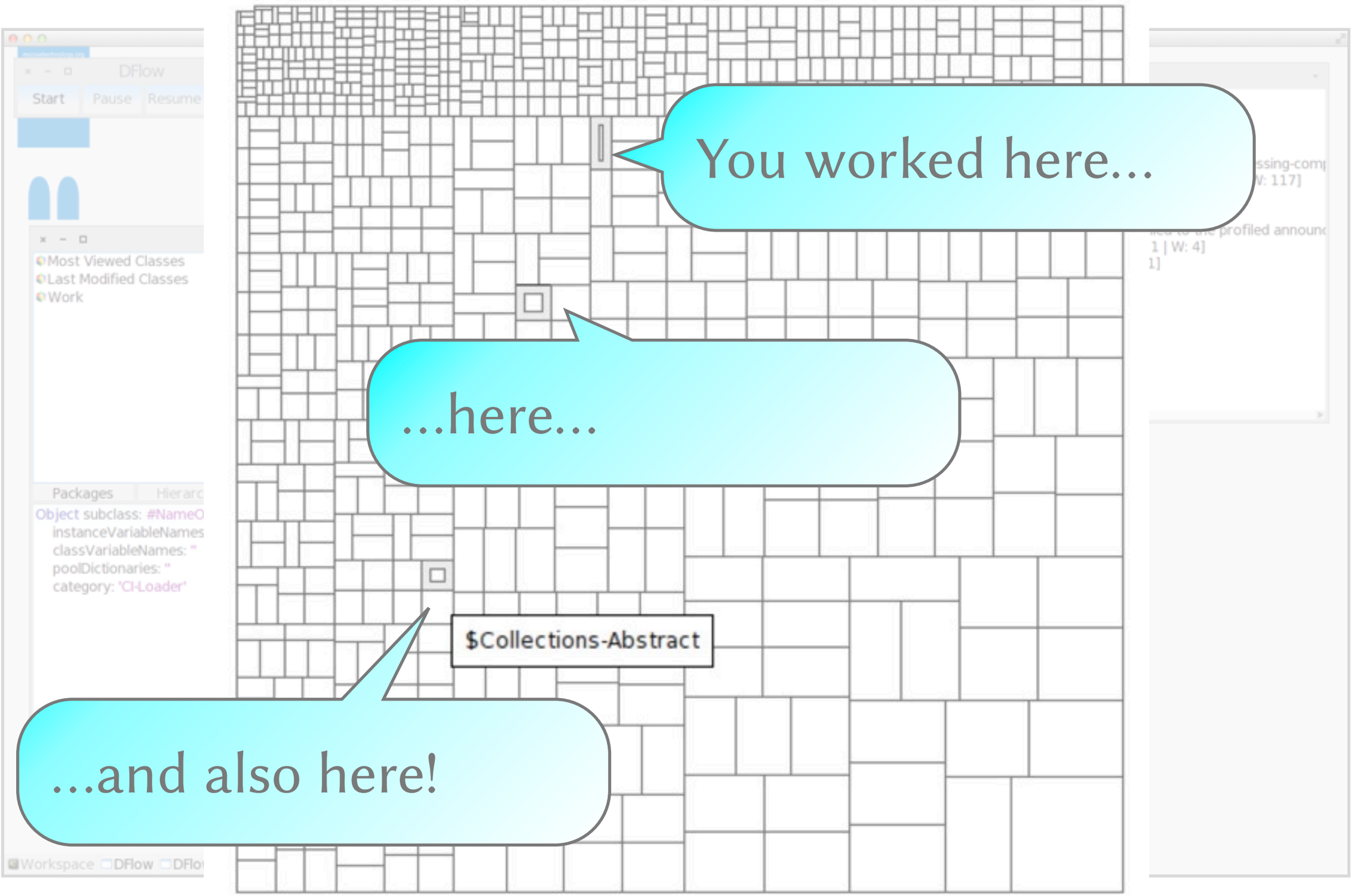
suggest

Forward



Adding **live visualizations**

Forward


DFlow
Start
Pause
Resume
Most Viewed Classes
Last Modified Classes
Work
Packages
Object subclass: #NameO
instanceVariableNames
classVariableNames: ''
poolDictionaries: ''
category: 'Cl-Loader'
Workspace
DFlow
You worked here...
...here...
$Collections-Abstract
...and also here!

To what extent does Pharo **support the navigation**?

Navigating source code with the Pharo IDE

How, **when**, **why** do developers use Pharo to **navigate the system**?

- **Understand** and **characterize** development **sessions**
- Identify development **workflow bottlenecks** in terms of IDE usage

Reverse

Forward

- Use **DFlow data** in a **run-time** context to enhance the IDE

Any other idea?

Talk to me (or drop me an email)

roberto.minelli@usi.ch

robertominelli