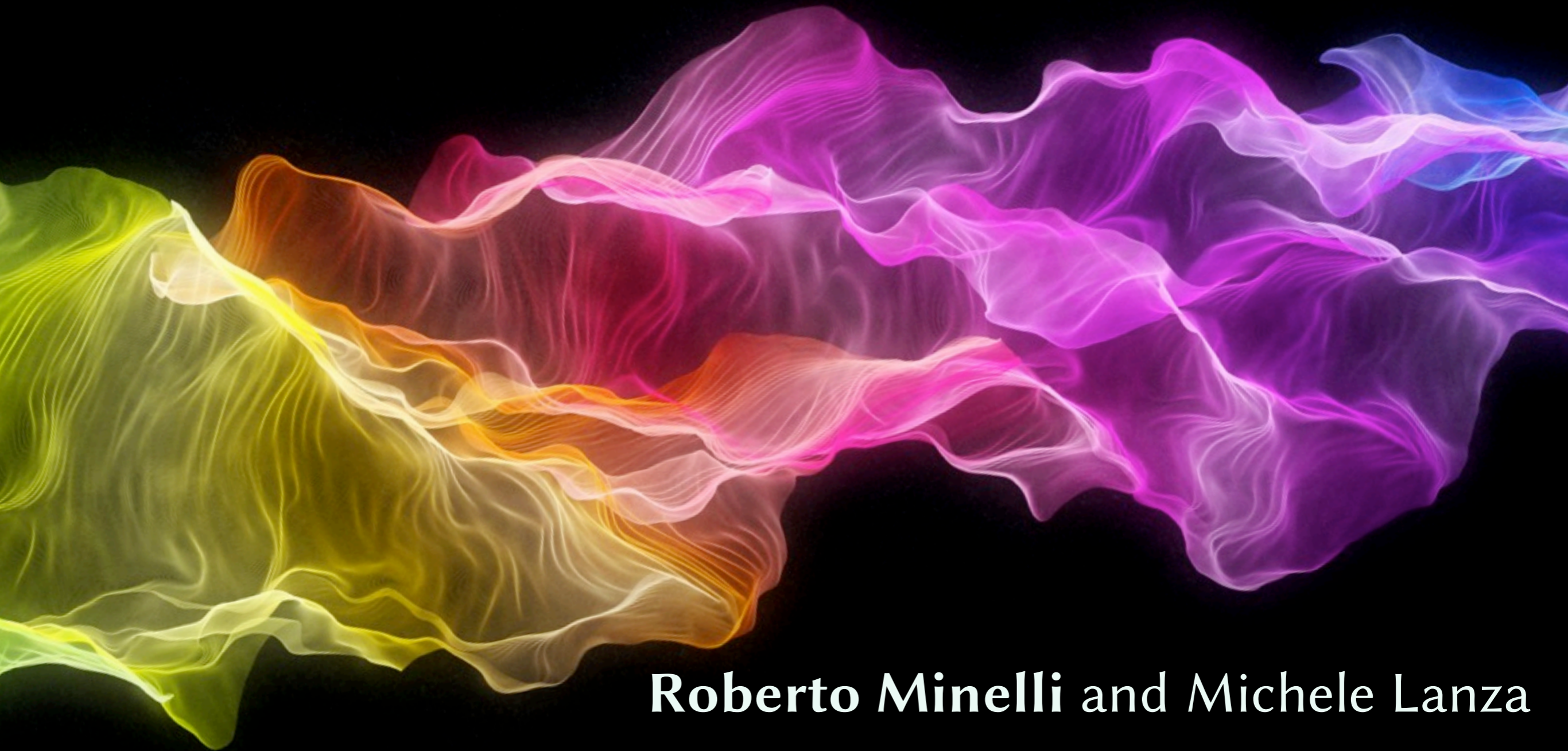


DFlow – A Platform to Profile Developers



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



VisualWorks



VA Smalltalk





Pharo

Smalltalk IDE



Writing



Writing

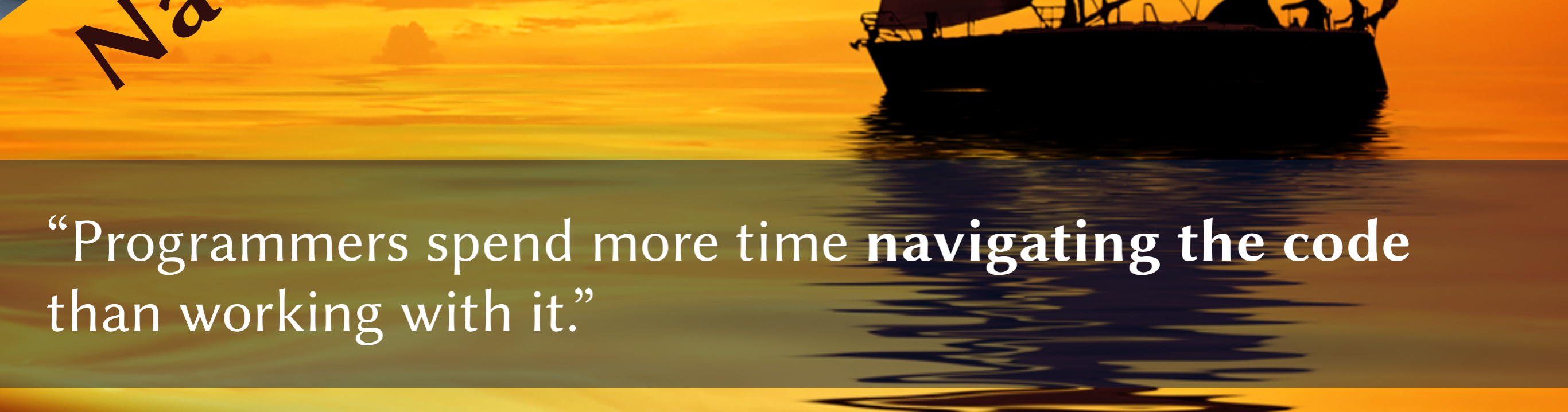


Navigating



Writing

M. Kersten and G.C. Murphy
“Mylar: a degree-of-interest
model for IDEs” AOSD 2005



Navigating

“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

The image displays a screenshot of the Pharo IDE with the DFlow extension. Three callout boxes highlight key components:

- Manager:** A window titled "DFlow" containing four buttons: "Start", "Pause", "Resume", and "Stop".
- Browser:** A window titled "DFlow Sessions Browser" listing several test sessions with their respective statistics (S: | M: | W:).
 - 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-com
 - 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]
- Standard Tools:** A window titled "DFStartSessionWindow>>#checkDescriptionAndType" showing a list of actions and their descriptions. The "actions" category is selected.
 - cancelAction
 - cancelBtn
 - cancelBtn:
 - checkDescriptionAndType
 - descriptionLbl
 - descriptionLbl:
 - descriptionTxt
 - descriptionTxt:

The background shows the Pharo IDE interface with a class browser on the left, a workspace in the center, and a transcript at the bottom. The workspace contains a class definition for `DFKeywordMethodSpy` with a `beforeRun:` method.



PHARO

DFlow

It's demo time!

OFFICIAL RAILWAYS CLOCK DESIGN

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**

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

Abstract—Developers use the Integrated Development Environment (IDE) to develop a system at hand, by reading, understanding and writing its source code. They do so by exploiting the tool facilities provided by the IDE. This also allows them to build a mental model of the system to perform informed changes. It is not clear how and when developers use which facility, and to what extent the current services offered by the IDE appropriately support the navigation. We present an approach to visualize the activities of developers in the IDE, implemented in a tool: DFlow. DFlow records interactions that occur during a development session and visualizes them through a web-based visualization platform.

I. INTRODUCTION

Developers spend much time interacting with Integrated Development Environments (*i.e.*, IDEs), such as ECLIPSE¹. In order to write new code, they use IDEs to comprehend the code by building a mental model of the system. & Murphy argued that “programmers spend more time navigating the code than working with it” [1], which leads to the question whether IDEs appropriately support the navigation. For example, Lee *et al.* claimed that the current IDEs for refactoring is unintuitive and inefficient [2]. We want to investigate how, when, and why developers use IDEs to navigate the software space and to what extent IDEs support the navigation. We present an approach to visualize the workflow that happens while a developer is working on a system. We implemented the approach in DFlow, which seamlessly integrates the IDE interactions with a visualization platform.

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.

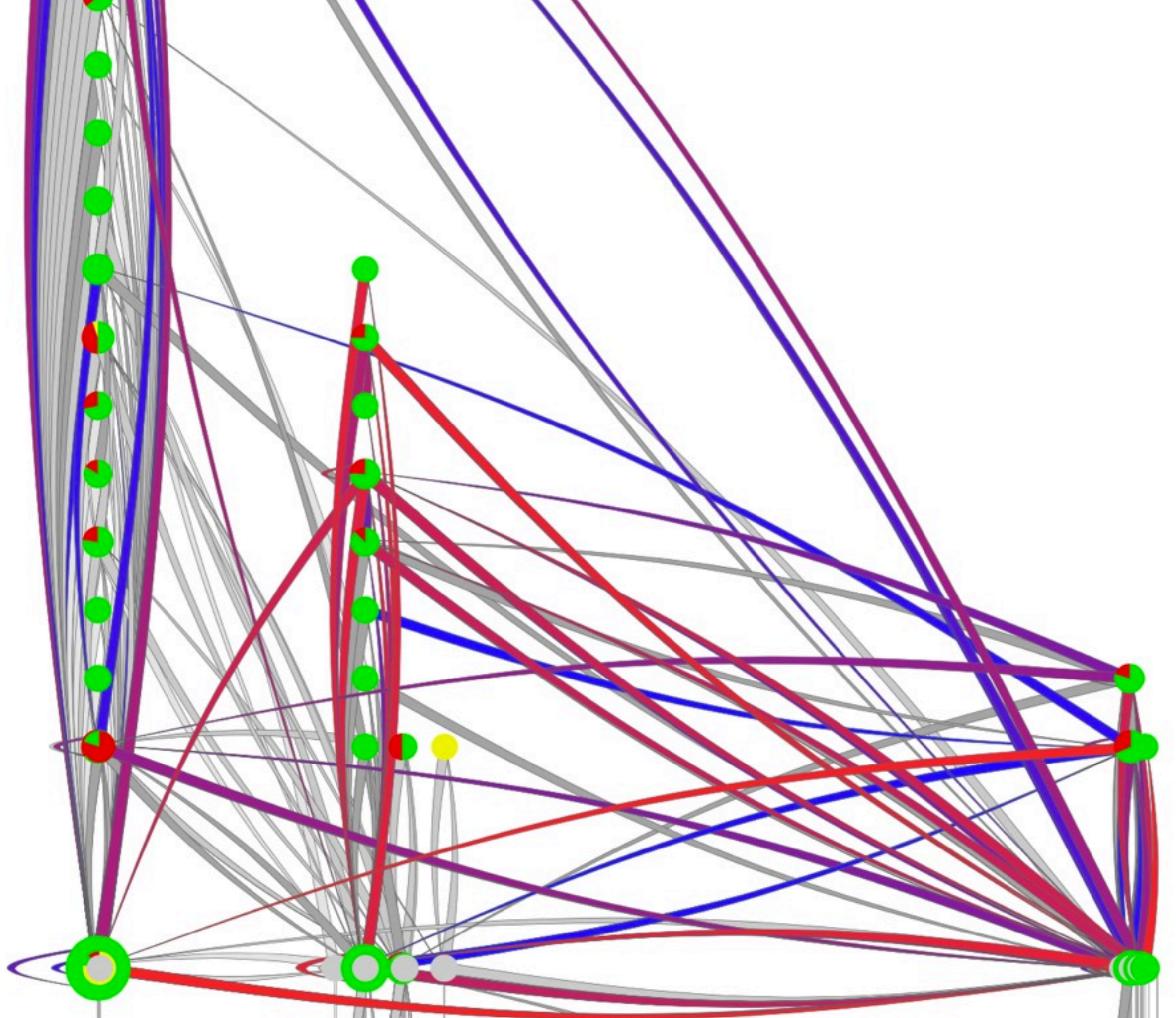
We describe DFlow, detail the custom visualization we devised, and illustrate them through scenarios.

II. DFlow

DFlow is composed of DFlow-PHARO, an extension to the PHARO² 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).

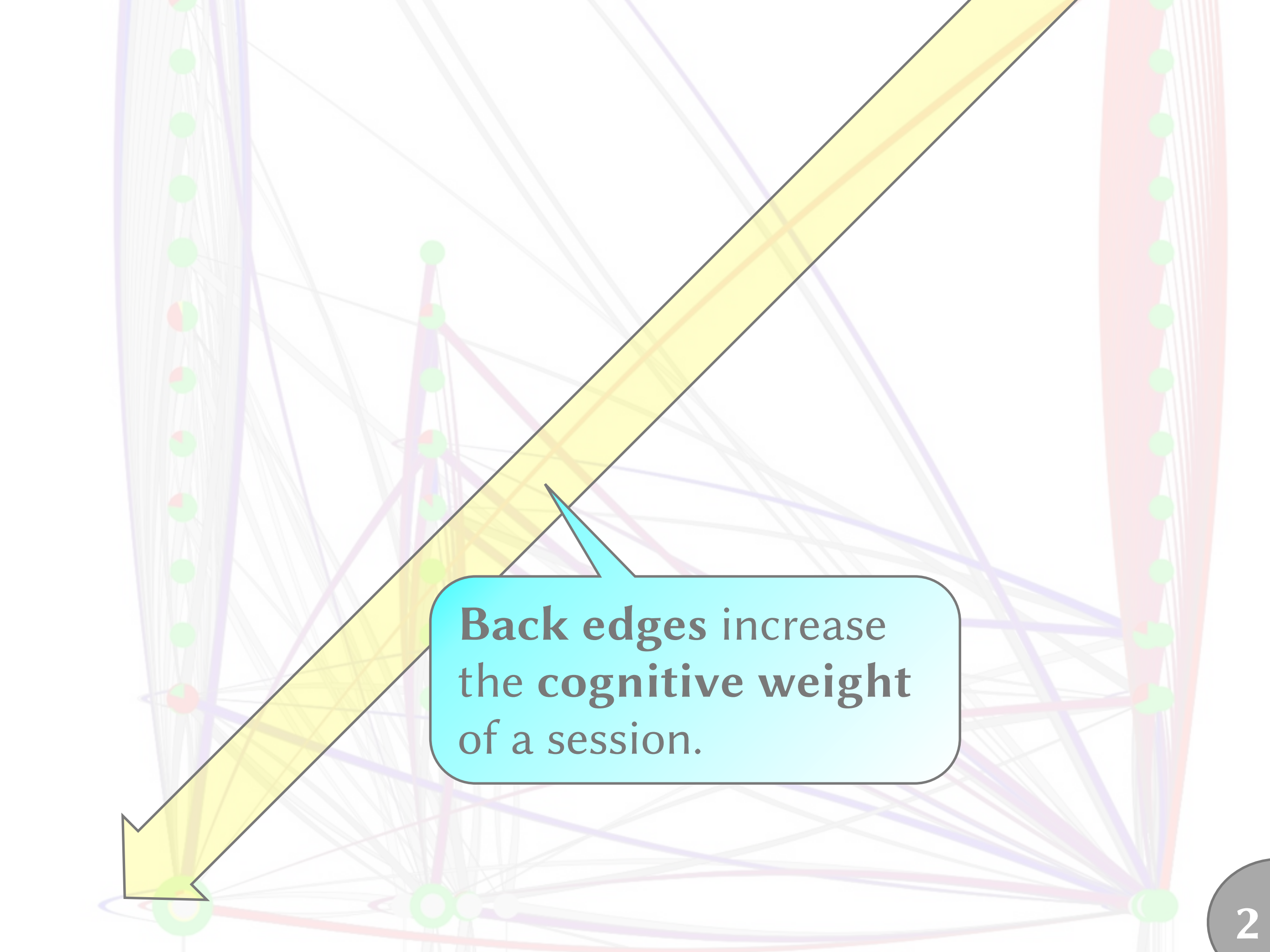


R. Minelli and M. Lanza
“Visualizing the Workflow of Developers.”
VISSOFT 2013





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



The screenshot displays the DFlow IDE interface. At the top left, there is a 'DFlow' control panel with buttons for 'Start', 'Pause', 'Resume', and 'Stop'. Below this is a 'CI-Loader' window with a sidebar containing 'Most Viewed Classes', 'Last Modified Classes', and 'Work'. The main area of 'CI-Loader' shows a class definition for 'Object subclass: #NameOfSubclass' with instance and class variable names, pool dictionaries, and a category of 'CI-Loader'. To the right, the 'DFlow Sessions Browser' window lists several sessions with their respective statistics (S, M, W). The bottom status bar shows the current workspace and active windows: 'Workspace', 'DFlow', 'DFlow Sessions Browser', and 'CI-Loader'.

DFlow Sessions Browser

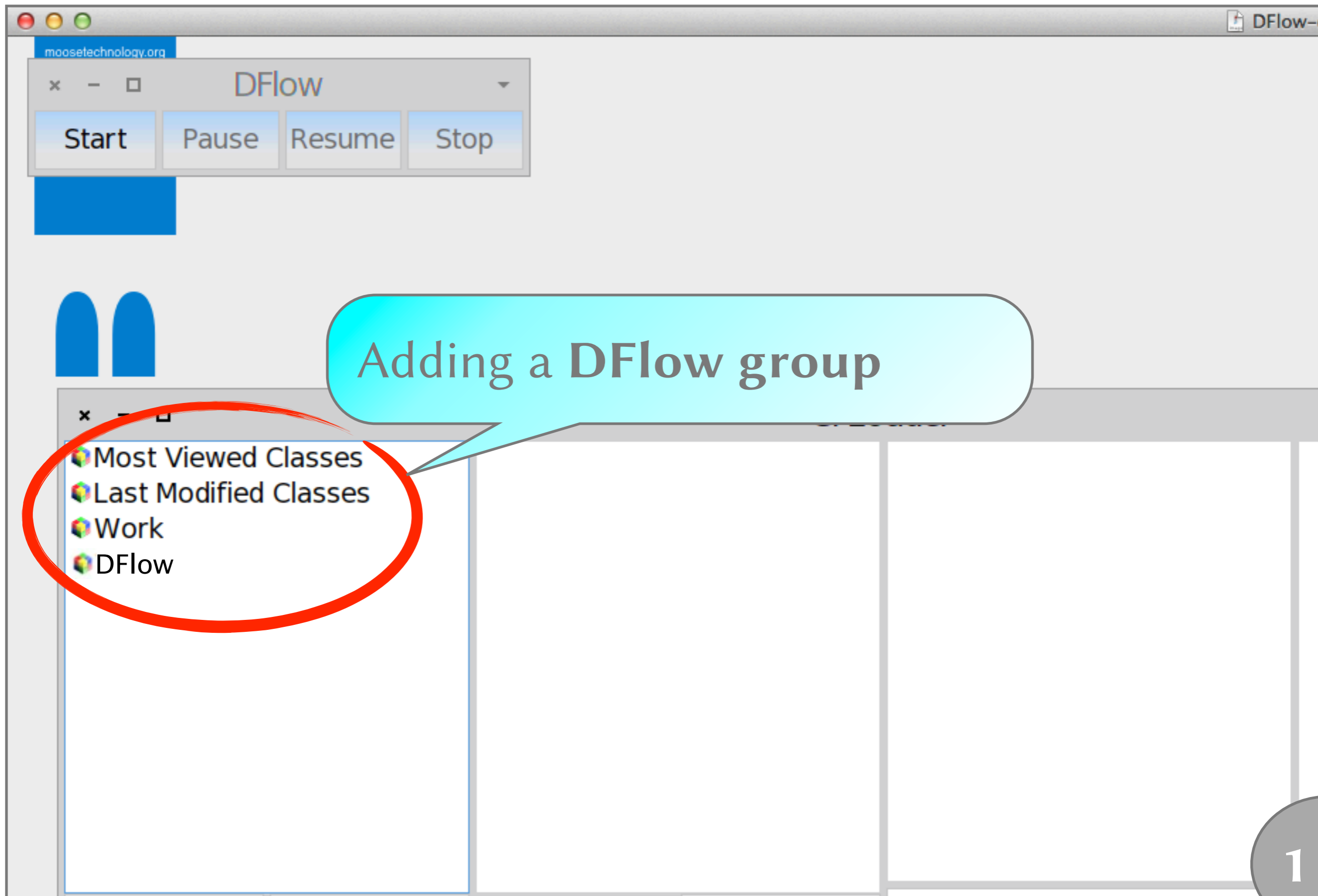
- Random test session [S: 2 | M: 48 | W: 11]
- Improve the export to also serialize windows [S: 2 | M: 746 | W: 36]
- Improve the export to serialize events related to windows [S: 2 | M: 524 | W: 22]
- Test new and existing windows [S: 1 | M: 0 | W: 3]
- Doing something to remove the bug of windows events 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]
- add also WindowActivated and WindowLabelled to the profiled announce
- 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

Object subclass: #NameOfSubclass
instanceVariableNames: "
classVariableNames: "
poolDictionaries: "
category: 'CI-Loader'

Workspace DFlow DFlow Sessions Browser CI-Loader

Forward



Adding a DFlow group

- Most Viewed Classes
- Last Modified Classes
- Work
- DFlow

Forward

The screenshot shows the ZeroDivide debugger interface. At the top, there is a window title bar with 'ZeroDivide' and standard window controls. Below the title bar is a list of stack frames. The selected frame is ' [...] in SmalltalkEditor class', which is highlighted in blue. Below the stack frames is a toolbar with buttons: 'Proceed', 'Restart', 'Into', 'Over', 'Through', 'Full Stack', 'Run to Here', 'Where', and 'Create'. The main area of the debugger displays code snippets, including a 'do:' block and several '(aBuilder shortcut: #inspectIt)', '(aBuilder shortcut: #implementorsOfIt)', and '(aBuilder shortcut: #sendersOfIt)' blocks. At the bottom, there are three panels: 'self' (showing instance variables like 'all inst vars', 'superclass', 'methodDict', 'format', 'layout', 'instanceVariables'), 'thisContext' (showing 'stack top', 'all temp vars', 'aBuilder', 'morph'), and a third panel showing the current execution context: '[:morph | morph doIt] in SmalltalkEditor class>>buildSmalltalkEditorKeymappingsOn:'.

Into

Over

Through

Forward

Into

The screenshot shows the ZeroDivide IDE interface. At the top, there's a title bar with 'ZeroDivide'. Below it is a list of objects and their methods, with '[...] in SmalltalkEditor class' selected. The main area is a code editor containing Smalltalk code. A toolbar with buttons like 'Proceed', 'Restart', 'Into', 'Over', 'Through', 'Full Stack', 'Run to Here', 'Where', and 'Create' is visible. The code includes comments for shortcuts like #inspectIt, #implementorsOfIt, and #sendersOfIt. At the bottom, there are three panels: 'self' (showing instance variables), 'thisContext' (showing stack top and variables), and a third panel showing the current execution context.

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

Into

Over

Through

Forward

The screenshot shows the ZeroDivide debugger interface. At the top, a list of methods is displayed, with "[...] in SmalltalkEditor class" selected. Below this is a toolbar with buttons: Proceed, Restart, Into, Over, Through, Full Stack, Run to Here, Where, and Create. The main area contains code with the line "do: [:morph | morph dolt]." highlighted. Below the code are three sections: "(aBuilder shortcut: #inspectIt)", "(aBuilder shortcut: #implementorsOfIt)", and "(aBuilder shortcut: #sendersOfIt)". At the bottom, there are three panes: "self" (listing instance variables like all inst vars, superclass, methodDict, format, layout, instanceVariables), "thisContext" (listing stack top, all temp vars, aBuilder, morph), and a third pane showing the current execution context: "[:morph | morph dolt] in SmalltalkEditor class>>buildSmalltalkEditorKeymappingsOn:". The "Into" button in the toolbar is highlighted.

Into

Over

Into

Over

Through

Forward

The screenshot shows the ZeroDivide IDE interface. At the top, there's a title bar with 'ZeroDivide'. Below it is a list of objects and their methods, with '[...] in SmalltalkEditor class' selected. The main area is a code editor containing Smalltalk code. A toolbar with buttons like 'Proceed', 'Restart', 'Into', 'Over', 'Through', 'Full Stack', 'Run to Here', 'Where', and 'Create' is visible. The code includes several (aBuilder shortcut: #inspectIt) blocks and (aBuilder shortcut: #implementorsOfIt) blocks. At the bottom, there are three debugger windows: 'self', 'thisContext', and a stack trace window.

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

self
all inst vars
superclass
methodDict
format
layout
instanceVariables

thisContext
stack top
all temp vars
aBuilder
morph

[:morph | morph doIt] in SmalltalkEditor
class>>buildSmalltalkEditorKeymappingsOn:
n:

Into

Over

Over

Into

Over

Through

Forward

The screenshot shows a Smalltalk IDE window titled "ZeroDivide". At the top, there is a list of objects with their corresponding methods. The object "[...] in SmalltalkEditor class" is selected, and its method "buildSmalltalkEditorKeymappingsOn:" is highlighted. Below the list is a toolbar with buttons: "Proceed", "Restart", "Into", "Over", "Through", "Full Stack", "Run to Here", "Where", and "Create". The main area contains code snippets, including "do: [:morph | morph doIt]." and several "(aBuilder shortcut: #inspectIt)", "(aBuilder shortcut: #implementorsOfIt)", and "(aBuilder shortcut: #sendersOfIt)" blocks. At the bottom, there are three panes: "self" (listing instance variables like "all inst vars", "superclass", "methodDict", "format", "layout", "instanceVariables"), "thisContext" (listing "stack top", "all temp vars", "aBuilder", "morph"), and a code snippet "[:morph | morph doIt] in SmalltalkEditor class>>buildSmalltalkEditorKeymappingsOn:".

Into

Over

Over

Into

Into

Over

Through

Forward

```
TextMorphForEditView(TextMorph) handleEdit:
PluggableTextMorph handleEdit:
PluggableTextMorph dolt
[...] in SmalltalkEditor class buildSmalltalkEditorKeymappingsOn:
BlockClosure cull:
BlockClosure cull:cull:
BlockClosure cull:cull:cull:
KMCategoryTarget completeMatch:buffer:
[...] in KMKeyman notifyCompleteMatchToBuffer:
```

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

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

(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
do: [ :morph | morph sendersOfIt ].
```

self
all inst vars
superclass
methodDict
format
layout
instanceVariables

thisContext
stack top
all temp vars
aBuilder
morph

[:morph | morph dolt] in SmalltalkEditor
class>>buildSmalltalkEditorKeymappingsOn:

Into

Over

Over

Into

Into

Into

Over

Through

Forward

The screenshot shows a Smalltalk IDE window titled "ZeroDivide". The main window displays a stack trace with the following entries:

- TextMorphForEditView(TextMorph) handleEdit:
- PluggableTextMorph handleEdit:
- PluggableTextMorph dolt
- [...] in SmalltalkEditor class buildSmalltalkEditorKeymappingsOn:
- BlockClosure cull:
- BlockClosure cull:cull:
- BlockClosure cull:cull:cull:
- KMCategoryTarget completeMatch:buffer:
- [...] in KMKeyman notifyCompleteMatchToBuffer:

Below the stack trace is a toolbar with buttons: Proceed, Restart, Into, Over, Through, Full Stack, Run to Here, Where, Create. The "Into" button is highlighted.

The main window contains the following code:

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

Below the code are three (aBuilder shortcut: #inspectIt) blocks:

- category: #SmalltalkEditor
default: \$i command mac | \$i ctrl win
do: [:morph | morph inspectIt].
- category: #SmalltalkEditor
default: \$m command mac | \$m ctrl w
do: [:morph | morph implementorsOfI
- category: #SmalltalkEditor
default: \$n command mac | \$n ctrl wi

An inspect window titled "a PluggableTextMorp" is open, showing the following instance variables:

- self
- all inst vars
- bounds
- owner
- submorphs
- fullBounds
- color
- extension
- borderWidth

The inspect window also shows a reference to "a PluggableTextMorph(31 798067 2)".

At the bottom left of the IDE window, a small window shows the following instance variables:

- self
- all inst vars
- superclass
- methodDict
- format
- layout
- instanceVariables

Into

Over

Over

Into

Into

#inspect

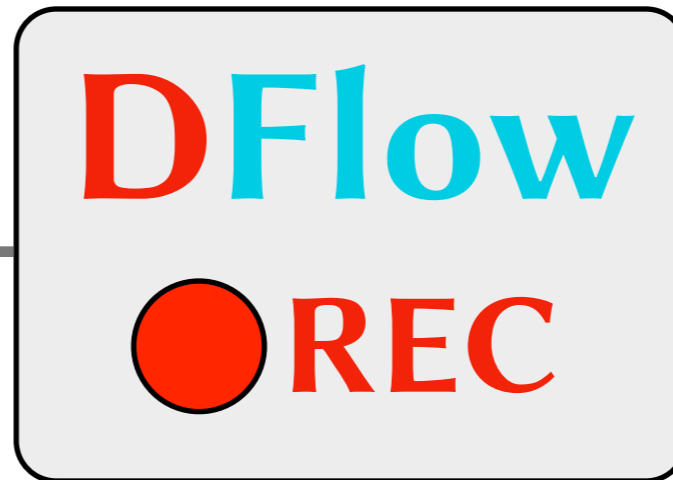
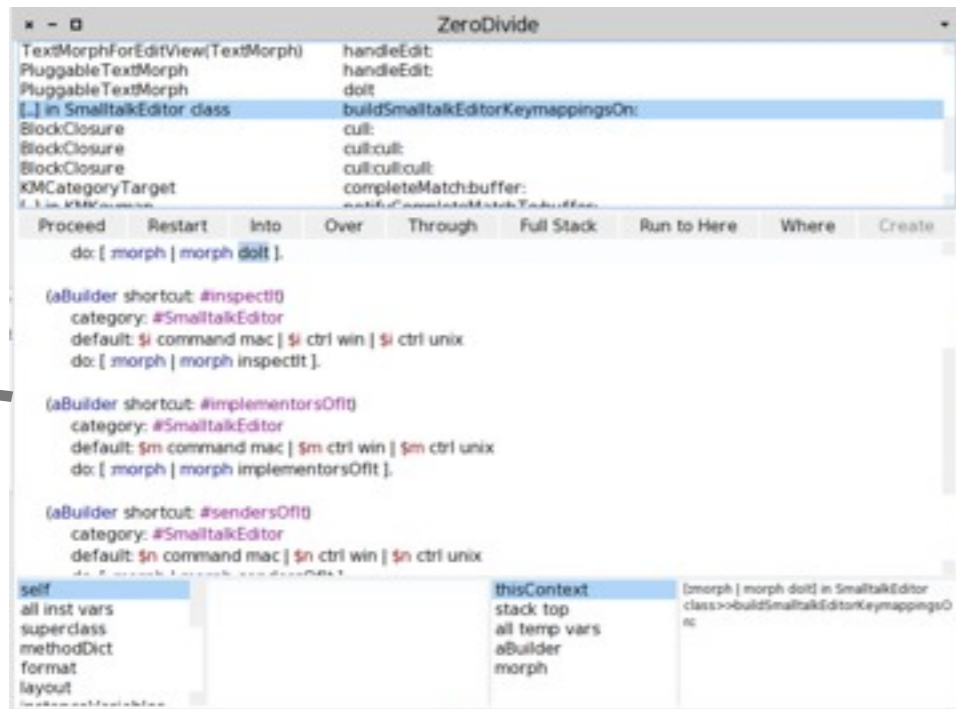
Into

Over

Through

Forward

...next time



Into

Over

Over

Into

Into

#inspect

DFlow
data

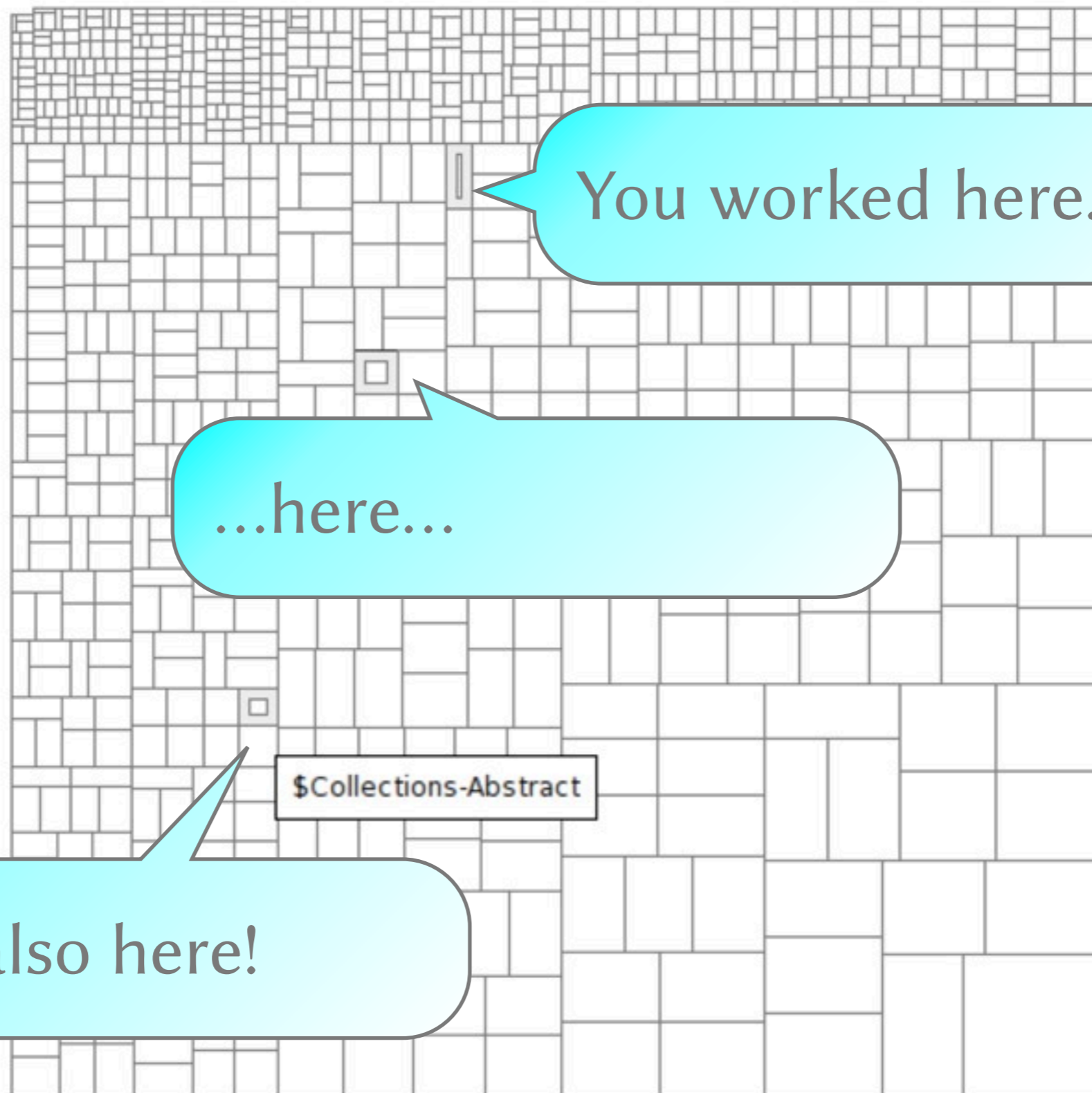
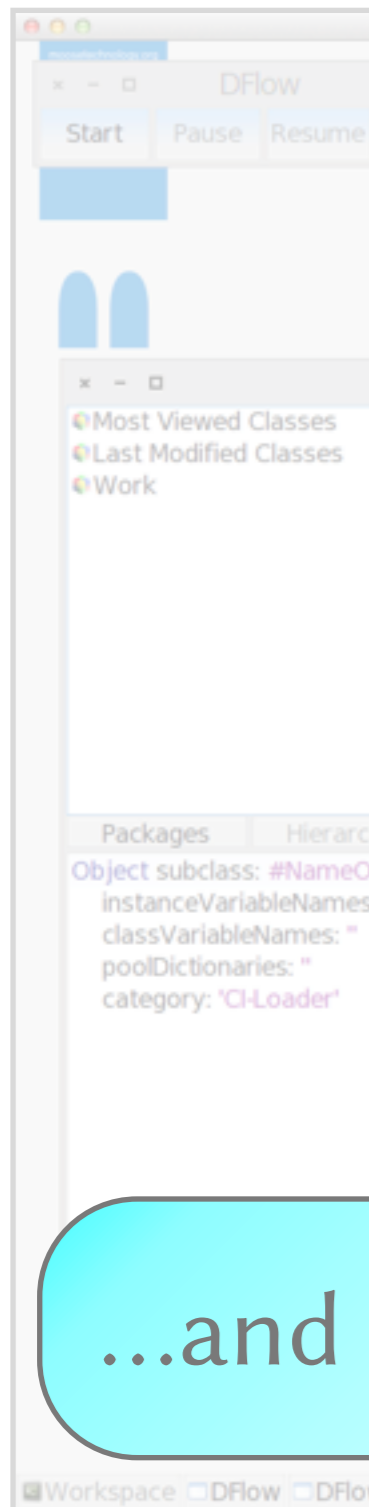
+ context

suggest

Forward

The screenshot displays the DFlow IDE interface. At the top left, there is a 'DFlow' control panel with 'Start', 'Pause', 'Resume', and 'Stop' buttons. Below it is a 'CI-Loader' window with a sidebar for 'Most Viewed Classes', 'Last Modified Classes', and 'Work'. The main editor area shows the 'Class side' of a class, with details for 'Object subclass: #NameOfSubclass' and instance/class variable names. To the right, the 'DFlow Sessions Browser' lists various sessions with their statistics (S, M, W). A callout bubble with the text 'Adding live visualizations' points to a visualization of class relationships, showing a grid of boxes representing classes and their connections. One box is labeled '\$Collections-Abstract'. The bottom status bar shows the current workspace and active windows: 'Workspace', 'DFlow', 'DFlow Sessions Browser', and 'CI-Loader'.

Forward

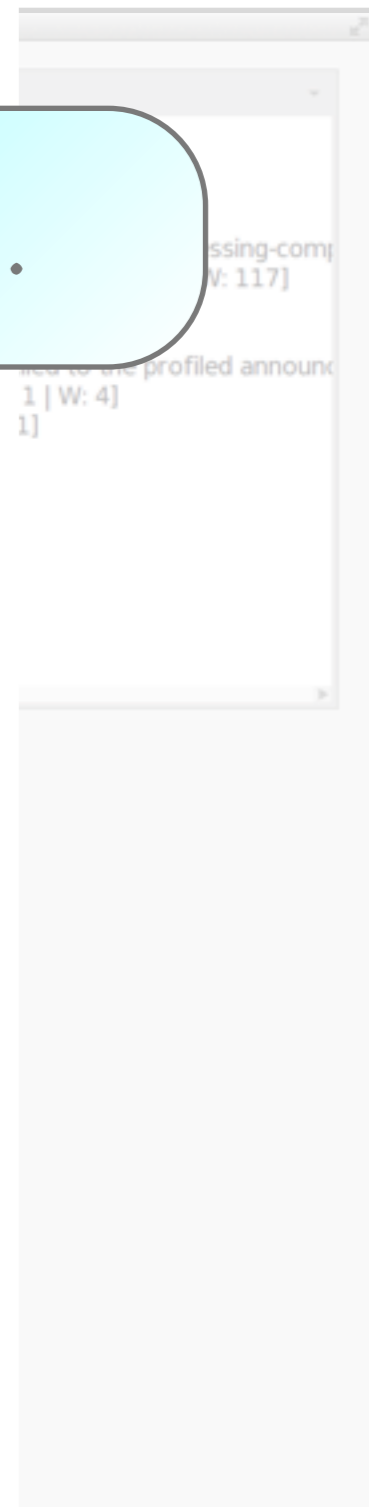


You worked here...

...here...

`$Collections-Abstract`

...and also here!

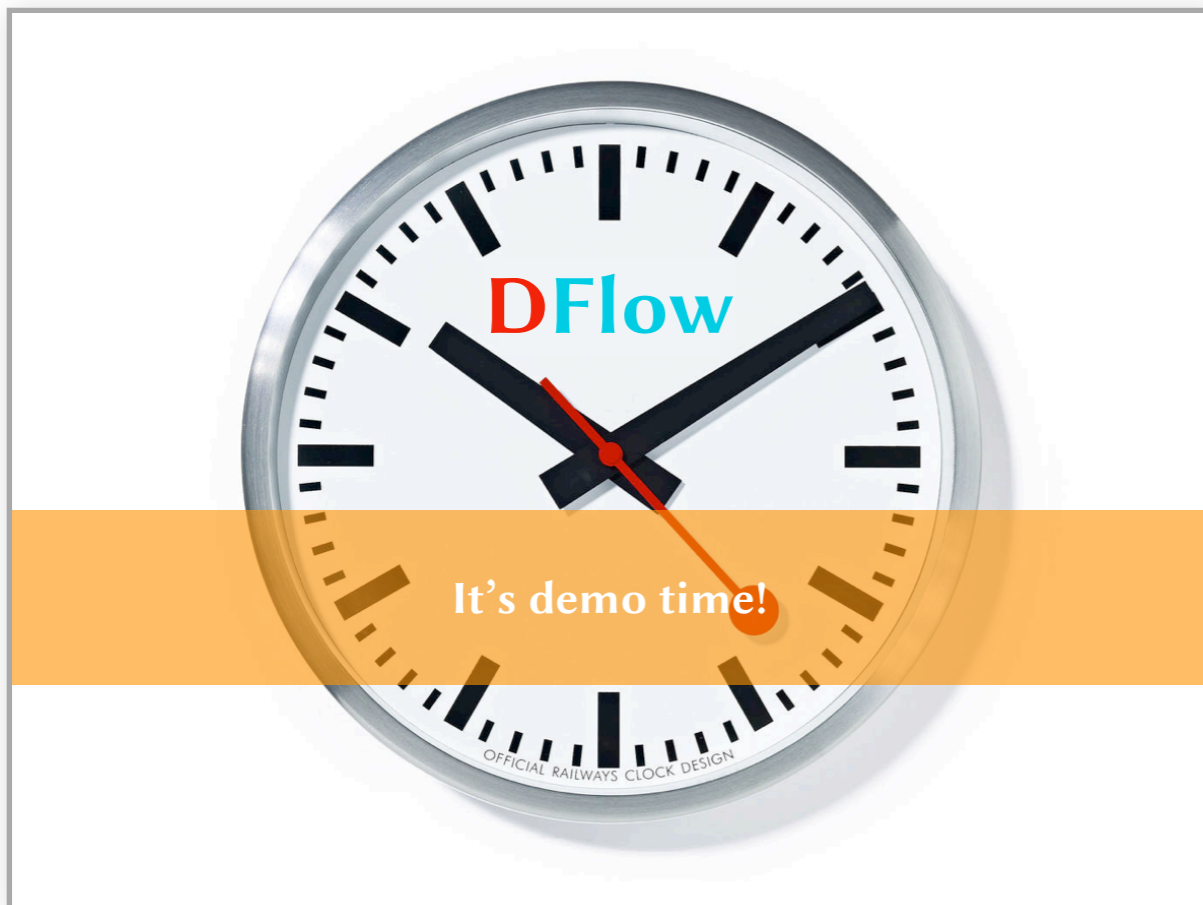




To what extent does Pharo support the navigation?

Navigating source code with the *Pharo* IDE

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



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

Any other idea?

Talk to me (or drop me an email)

 roberto.minelli@usi.ch

 [robertominelli](https://twitter.com/robertominelli)

