

MICROPRINTS

A PIXEL-BASED, SEMANTICALLY RICH
VISUALIZATION OF METHODS

ROMAIN ROBBES, STÉPHANE DUCASSE
AND MICHELE LANZA

OUTLINE

- Visualization techniques
- Microprints
- Microprints at work
- Customizing Microprints
- Conclusion

VISUALIZATION TECHNIQUES

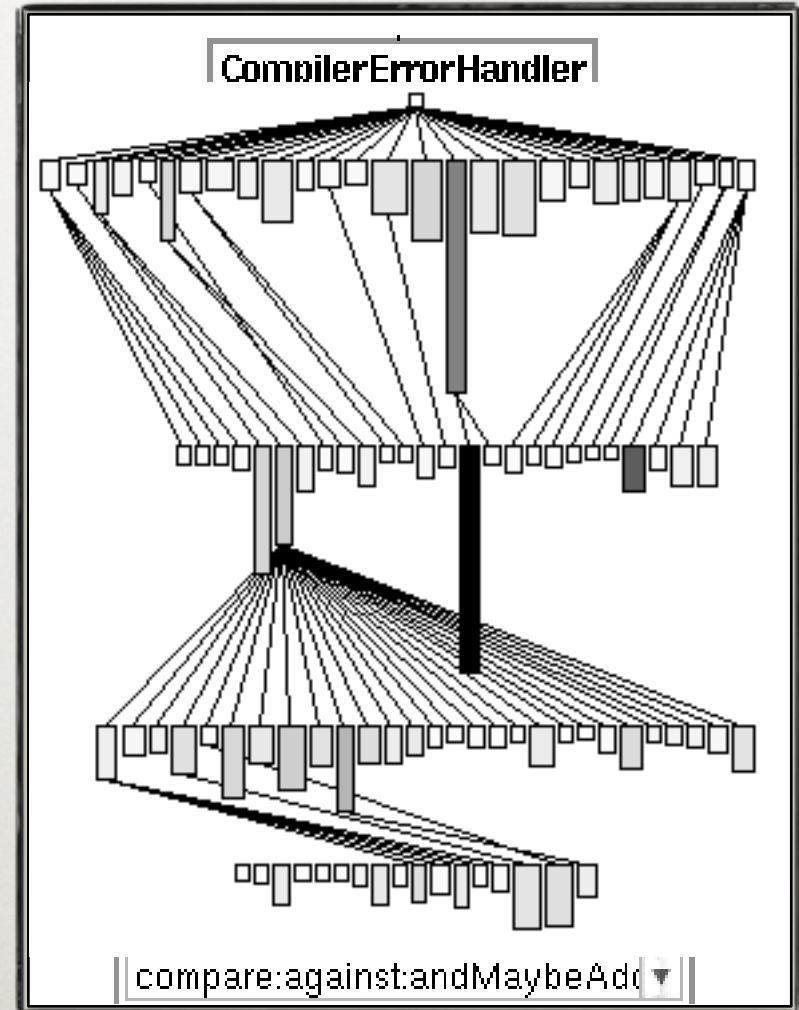
AND SOME (HUMAN) LIMITATIONS

WHY VISUALIZING CODE?

- A way to:
 - Represent a lot of information in a small space
 - Choose starting points in an unknown code base
 - Facilitate code comprehension

VISUALISATION TECHNIQUES

- Some examples:
 - UML
 - Syntax Highlighting
 - Polymetric Views



LIMITATIONS

- Visualizations are interpreted by humans
- Unfortunately we are the limiting factor
 - Limited color choice
 - Limited information density

LIMITED COLOR CHOICE

someMethod: anObject

"Type a new method here and see how it formats as you type"

```
| temp |  
#(a b c) do: [:tmp :temp | Transcript show: temp; cr; show: tmp]. "temp is redeclared"  
self isNil and: [temp isNil] ifTrue: [self foo]. "and:ifTrue: and #foo are not understood"  
temp := ('string' size + #symbol hash).  
(Array with: self with: super with: true with: thisContext) do: [:each | each halt].  
^undeclaredVariable value + 4 + $a asInteger
```

- We must avoid:
 - Colors too close to each other
 - Too many colors (7 max)

INFORMATION DENSITY

someMethod: anObject

"Type a new method here and see how it formats as you type"

```
| temp |  
#(a b c) do: [:tmp :temp | Transcript show: temp, cr; show: tmp].    "temp is redeclared"  
self isNil and: [temp isNil] ifTrue: [self foo].    "and:ifTrue: and #foo are not understood"  
temp := ('string' size + #symbol hash).  
(Array with: self with: super with: true with: thisContext) do: [:each | each halt].  
^undeclaredVariable value + 4 + $a asInteger|
```

- Too much information renders it useless
- Harder to locate a precise information...
 - Like finding a needle in a haystack

MICROPRINTS

ASSUMPTION, PRINCIPLE AND ADVANTAGES

MICROPRINT ASSUMPTION

- It is easier to merge information than to sort it
- It is **easier** to **merge information** than to **sort** it
- **It is easier to merge information than to sort it**
- It is **easier** to **merge information** **than** to **sort it**

BUT WE DO NOT HAVE THAT MUCH SPACE!

- So we reduce and stack each view:

It is easier to merge
information than to
sort it

It is **easier** to **merge**

information than to

sort it

It is easier to **merge**

information than to

sort it

It is **easier** to **merge**

information than to

sort it

MICROPRINT PRINCIPLE

- Several views of a method via a variant of code highlighting:
 - Variable access
 - Control flow
 - Object interactions
 - Code and frequency coverage

ADVANTAGES OF THE APPROACH

- Several focused views
- Few information in each view
- Few colors as well
- Colors can be reused in several views

MICROPRINTS AT WORK

SAMPLE PATTERNS, ALTERNATIVE USES

MICROPRINTS IN PICTURES

The screenshot shows an IDE window with the following components:

- Top Bar:** Two tabs labeled "RBASTToSourceMapper>>pointForIndex:".
- Left Panel:** A "Package Hierarchy" tree showing a structure with folders like "Local Image", "Base VisualWorks *", "CodeCrawlerDevelo", and "Deprecation +".
- Center Panel:** A class browser showing "PseudoVarMarkerTes", "RBASTEvaluator", and two instances of "RBASTToSourceMap".
- Right Panel:** An "Instance" view showing methods like "initialize-release", "accessing", "converting", and "querying".
- Bottom Panel:** A "Shared Variable" view showing "nodeForIndex:", "pointForIndex:", "rectangleFor:point:ind", and "rectangleForNode:". Below this are tabs for "Source", "Rewrite", "Code Critic", and "MicroPrint".

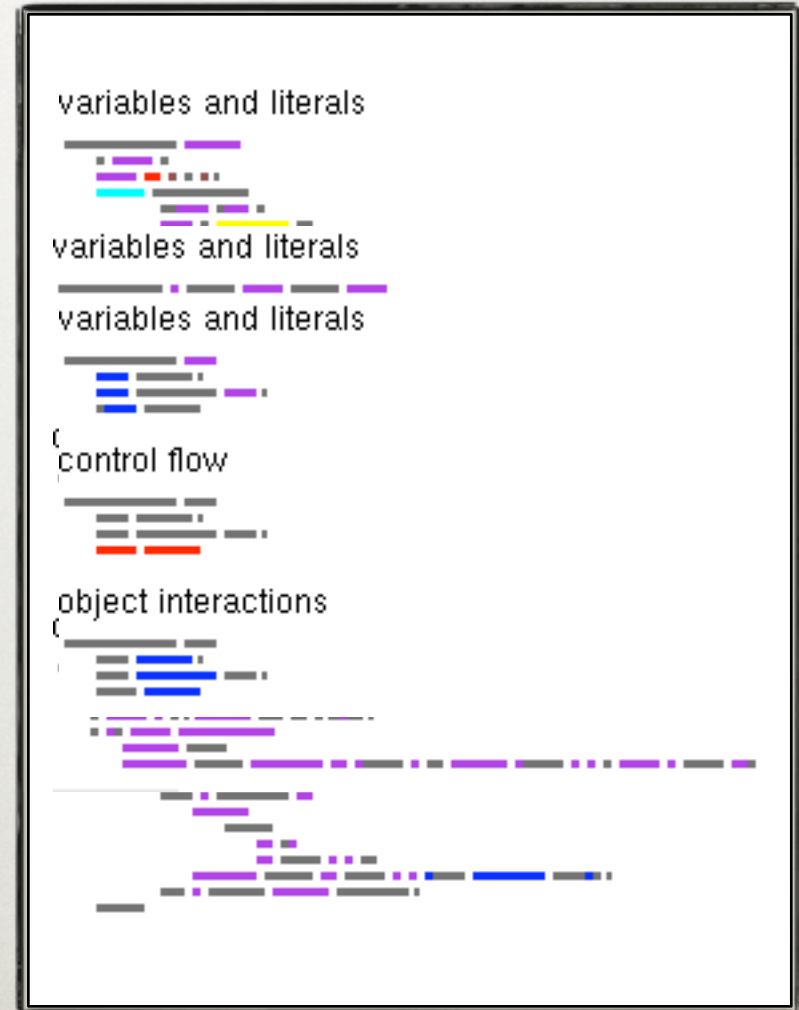
The **MicroPrint** view displays the following code:

```
pointForIndex: integer
| point |
point := 0 @ 1.
source doWithIndex:
  [:char :idx |
  char = Character cr
  ifTrue:
    [point
     x: 0;
     y: point y + 1]
  ifFalse: [point x: point x + (self lengthOf: char)].
  idx = integer ifTrue: [^point].
^point
```

On the right side of the MicroPrint view, there are three diagrams illustrating state changes/accesses, control flow, and object interactions, each represented by a series of horizontal lines with colored segments (purple, yellow, red, blue, green) indicating specific events or states over time.

SAMPLE VISUAL PATTERNS

- Lazy initialisation
- Complex / trivial logic
- Template / framework method



ALTERNATIVE USES OF MICROPRINTS

- Visualizing:
 - Method protocols
 - Classes
 - Hierarchies
 - Or more?

CUSTOMIZING MICROPRINTS

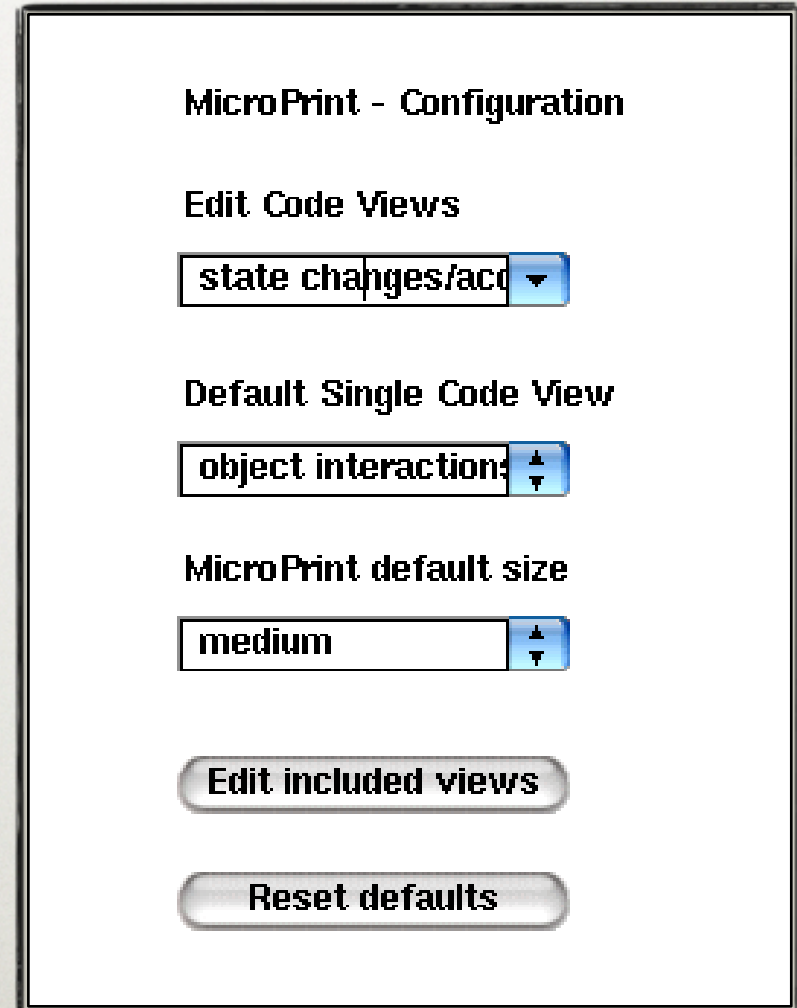
ROLL YOUR OWN MICROPRINTS IN TWO EASY
LESSONS

FIRST: SOME IMPLEMENTATION DETAILS

- Microprints use the Refactoring Browser:
 - RB parser, AST and visitor
- Microprint = color to marker mapping
 - Markers = visitors of RB ASTs
 - Looking for “interesting” nodes

COMPOSE YOUR OWN MICROPRINT

- Use the GUI to:
 - Add markers
 - Map them to colors
 - Set their priorities



CREATE YOUR OWN MARKER

- Define a new visitor
- When an AST node is interesting:
 - self mark: aNode (boolean)
 - self mark: aNode value: v (integer)
 - If you want to store a metric

CONCLUSION

CONCLUSION

- Merging information...
 - Is easier than sorting it
- Microprints:
 - Several semantic views of methods
 - Integrated in VW and CodeCrawler
 - Customizable and extensible

ANY QUESTIONS?

- Here's one:
- Which microprints would you need?