From Abstracting to Projectional Editors

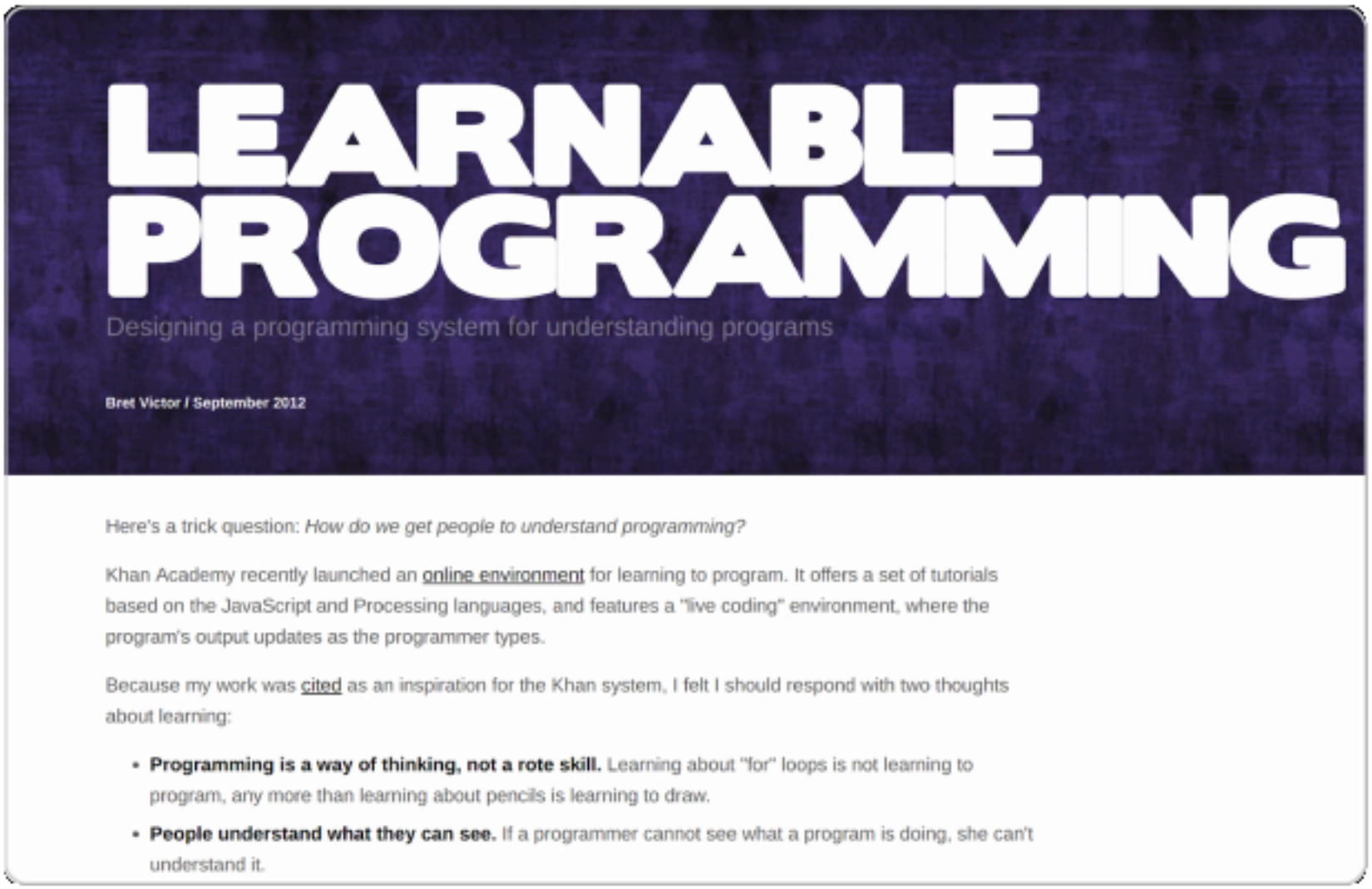
Seminar Programming Experience

Tom Beckmann, Carl Gödecken

Prof. Dr. Robert Hirschfeld, Jens Lincke, Stefan Ramson

04. July 2018 - Summer term 2018 - Hasso Plattner Institute - Software Architectures Group

„Learnable Programming“



LEARNABLE PROGRAMMING

Designing a programming system for understanding programs

Bret Victor / September 2012

Here's a trick question: *How do we get people to understand programming?*

Khan Academy recently launched an online environment for learning to program. It offers a set of tutorials based on the JavaScript and Processing languages, and features a "live coding" environment, where the program's output updates as the programmer types.

Because my work was cited as an inspiration for the Khan system, I felt I should respond with two thoughts about learning:

- **Programming is a way of thinking, not a rote skill.** Learning about "for" loops is not learning to program, any more than learning about pencils is learning to draw.
- **People understand what they can see.** If a programmer cannot see what a program is doing, she can't understand it.

http://worrydream.com/LearnableProgramming/

Learnable Programming

Create By Abstracting

Abstract

var
function

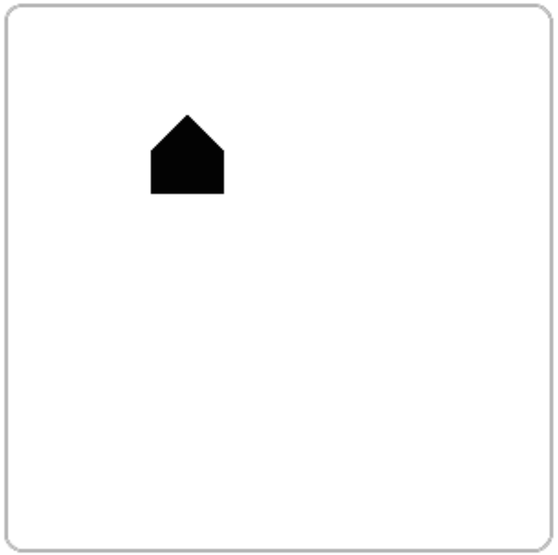
Flow

if
for
while

```
rect(80, 80, 40, 25);
triangle(80, 80, 100, 60, 120, 80);
```

- start concrete
- experiment
- abstract


drawOn aCanvas
aCanvas fillRectangle 80 @ 80 extent 40 @ 24 fillStyle
aCanvas drawTriangleFrom 80 @ 80 to 120 @ 80 and 100 @ 60 color



Our Requirements

- **deep integration** between language concepts and editor
- syntactical correctness as often as possible -> **live reload**
- every syntactic entity needs to be **tangible**

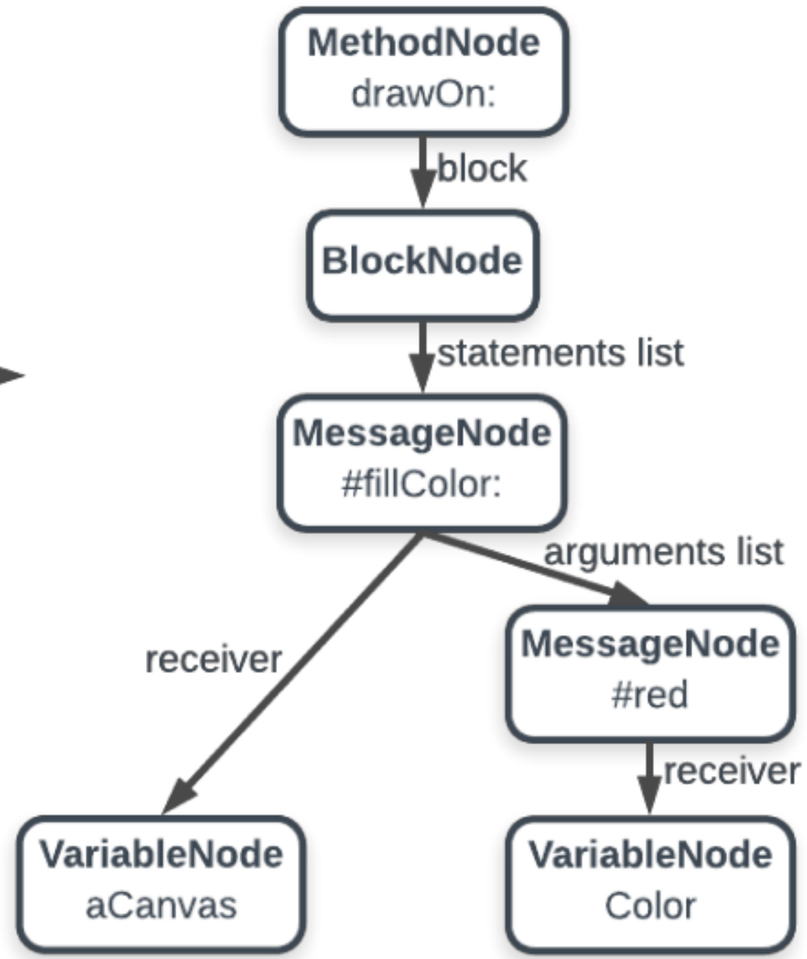
Syntactical Entity vs Token

Our First Prototype

Abstract Syntax Tree

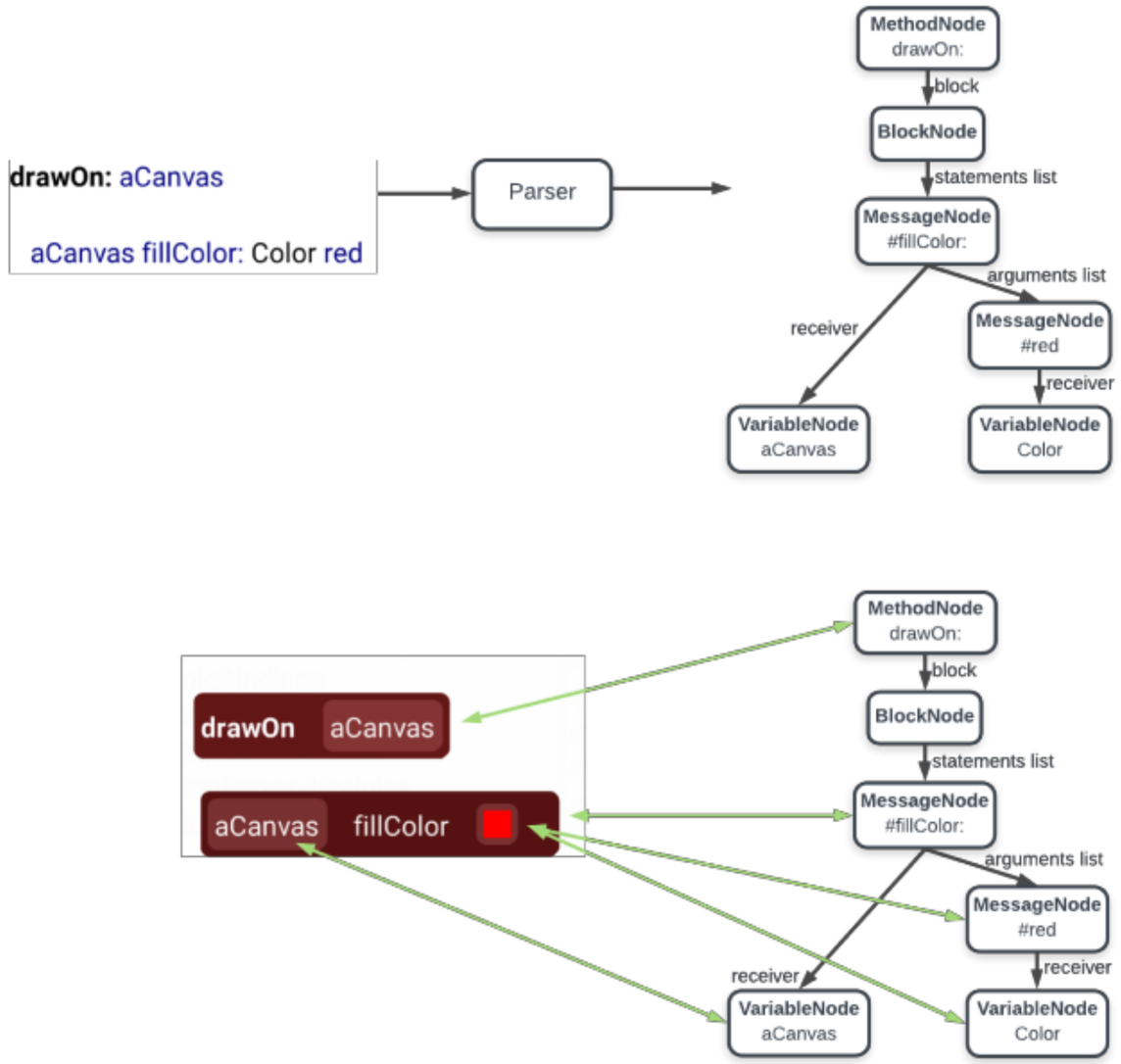
```
drawOn: aCanvas

    aCanvas fillColor: Color red
```



Projectional Editing - Our Approach

- edits happen on **projections** of the existing AST, modify it directly
  - (hypothetically) always have a syntactically correct program



Projections

42 42

'(\D+\.\D+\d*@hpi\.de'

regular editor:

- need to change **characters** to change meaning

projectional editor:

- meaning is **inherent**, tool projects **value**
- different projections with the **same meaning**
  (decimal, hexadecimal, slider, ...)
- can replace **entire objects** (e.g. tables, color pickers, ...)

WorkingTreeMorph

Emergency Reset!

```
drawTrunkOn aCanvas from aPoint length aLength angle anAngle
| endPoint heading |
heading := anAngle sin @ anAngle cos negated
endPoint := aPoint + heading * aLength
aCanvas line aPoint to endPoint width 3 color ■
^ endPoint
```

```
initialize
super initialize
```

```
drawOn aCanvas
self drawTreeOn aCanvas from self bounds center x @ self bounds bottom - 10 length 40 angle 0
```

```
testing
drawTreeOn aCanvas from basePoint length length angle angle
length > 10 ifTrue
| newBasePoint |
newBasePoint := self drawTrunkOn aCanvas from basePoint length length angle angle
self drawTreeOn aCanvas from newBasePoint length length * 0.7 angle angle + 0.3
self drawTreeOn aCanvas from newBasePoint length length * 0.7 angle angle - 0.3
```

Discussion

Speed of Usage

- graphical programming generally perceived to be **slower**
- early projectional editors required editing of tree structures

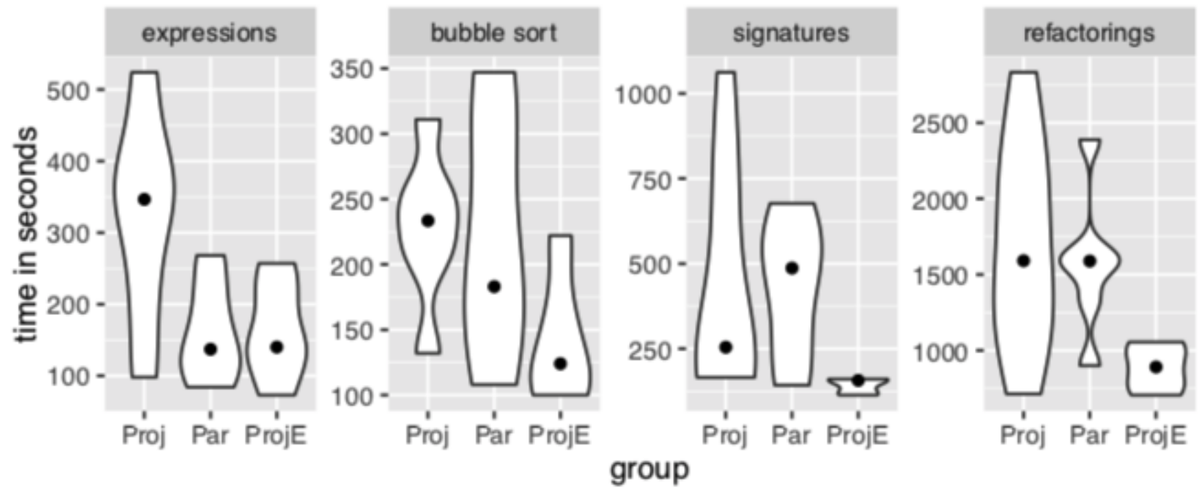


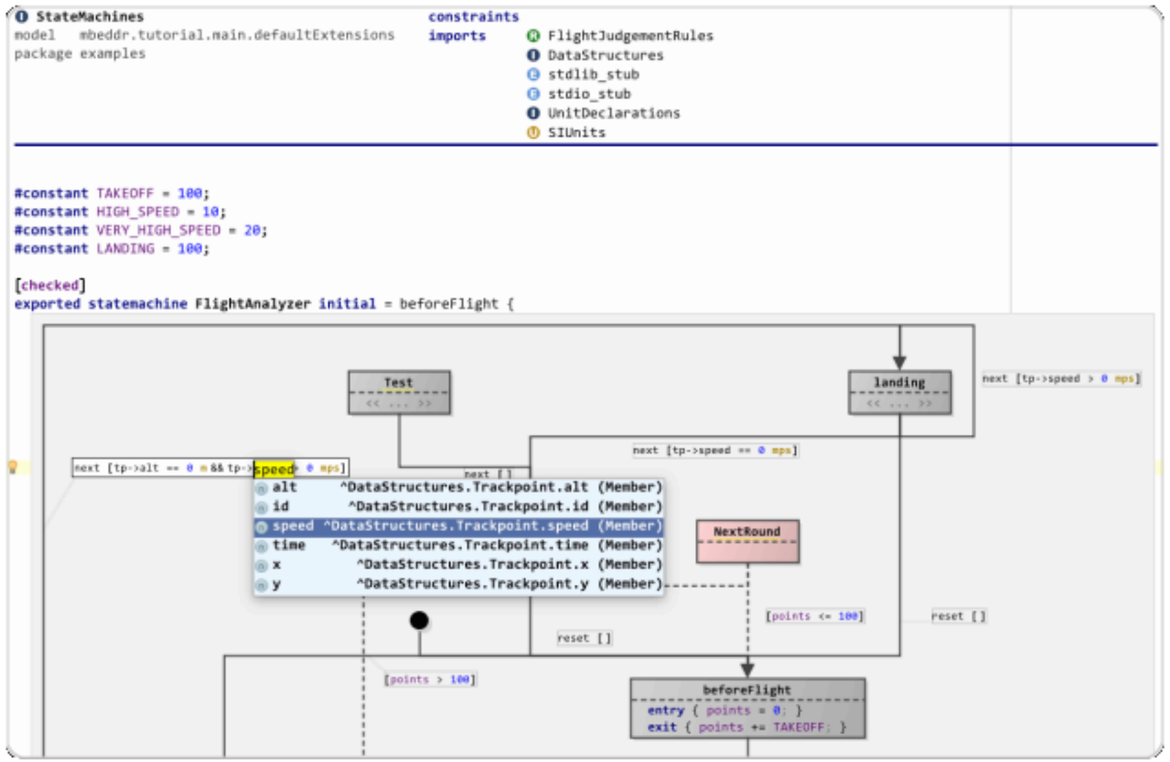
Figure 2: Task completion times in seconds (violin plots)

BERGER, Thorsten, et al. Efficiency of projectional editing: A controlled experiment.
In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2016. S. 763-774.

Jetbrain's MPS

uses **"guessing", autocompletion** and a **text-editor-like layout**, among other things

Implementation below: **mbeddr**, MPS-based editor for **embedded C**



Souce: http://mbeddr.com/2015/03/05/graphicalSM.html

Guessing: space on a number means **sending a message**, **+** in a message name means **binary message**

42

Expressing Intent

- vim: generally perceived to **boost editing speed**
- modes enable **minimal number of keystrokes** to express actions
- grammar to express **intent** to editor

[quantifier]	**verb**	**[modifier]**	**object**
	d (delete)	a (around)	p (paragraph)
1	c (change)	i (inside)	w (word)
20	i (insert)	t ('till)	b (block)
	y (copy)		s (sentence)
			" (quoted)

Expressing Intent

- instead of acting on a **text buffer**, we can act on **smalltalk code**
  --> create grammar that **expresses this clearly**

doIt

42 squared

Things We Learned

- **remove all syntax**
- first class **undo**
- **keyboard only** vs **mouse**
- even simple Smalltalk language has an **abundance of possible actions**

Limitations

- **performance:** bubbles are costly, zooming not easy
- **detour** via generated codestring -> parser -> compiler
  - easier compilation
  - tool compatibility
  - can't attach extra info for projections (e.g. regex example)
- how to display **multiple or long methods**
- some aspects **mocked** (autocompletion), occasional crashes
- editor still feels **fragile**
- currently **requires basic smalltalk knowledge**

Future Work

- advanced **refactoring tools**
- **more** projections, **more advanced** projections (replace entire blocks with UI)
- **storing** projection data
- adapt **tool support** (VCS, debugging)
- **performance optimizations**
- extended **test suite** (editor should always feel perfectly safe)

```
drawOn aCanvas
aCanvas fillRectangle 17 @ 42 extent 234 @ 241 fillStyle Color r 0.723 g 0.175 b 0.331
aCanvas fillRectangle 82 @ 15 extent 205 @ 212 fillStyle TranslucentColor r 1 g 0.6 b 0.0 alpha 0.761
aCanvas drawString 'HPI' from 1 to 3 at 98 @ 35 font PHStyleContext fontForSize 86 color
```