

Write your own tiny programming system(s)

Tiny systems as a methodology

Tomas Petricek, Charles University

✉ tomas@tomasp.net

✈ [@tomasp.net](https://tomasp.net)

🌐 <https://tomasp.net>

🌐 <https://d3s.mff.cuni.cz/teaching/nprg077>



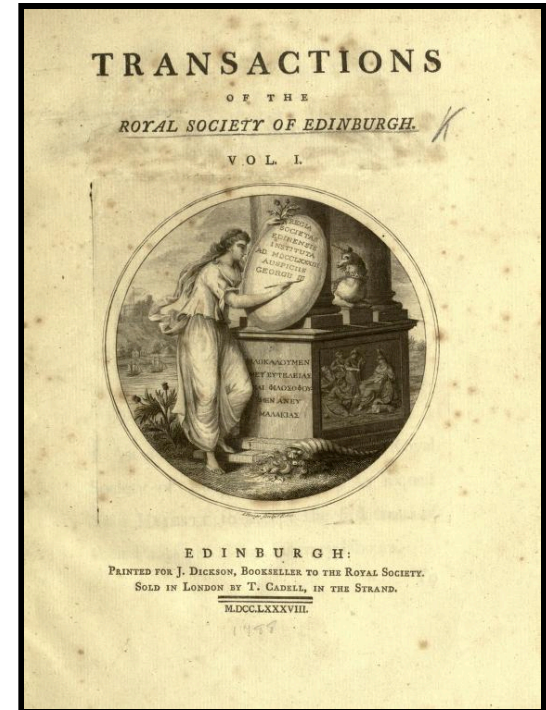
Academic research

What are we trying to study?

- Basic essential principles
- In isolation from other factors
- You have to ignore a lot!

What to ignore in programming?

- Efficient implementation?
- Wide-spread user adoption?
- User interface of editor tools?



Programming language theory

Ignore implementation and practical features

Prove that the core idea is formally sound

$$\boxed{\Gamma @ R \vdash e : \tau}$$

$$\text{(const)} \frac{}{() @ \perp \vdash c : \iota} \quad \text{(var)} \frac{}{(x : \tau) @ \langle \text{use} \rangle \vdash x : \tau}$$

$$\text{(abs)} \frac{\Gamma, x : \sigma @ R \times \langle s \rangle \vdash e : \tau}{\Gamma @ R \vdash \lambda x. e : \sigma \xrightarrow{s} \tau}$$

$$\text{(app)} \frac{\Gamma_1 @ R \vdash e_1 : \sigma \xrightarrow{t} \tau \quad \Gamma_2 @ S \vdash e_2 : \sigma}{\Gamma_1, \Gamma_2 @ R \times (t \oplus S) \vdash e_1 e_2 : \tau}$$

$$\text{(let)} \frac{\Gamma_1 @ S \vdash e_1 : \sigma \quad \Gamma_2, x : \sigma @ R \times \langle t \rangle \vdash e_2 : \tau}{\Gamma_1, \Gamma_2 @ R \times (t \oplus S) \vdash \text{let } x = e_1 \text{ in } e_2 : \tau}$$

$$\text{(ctx)} \frac{\Gamma @ R \vdash e : \tau \quad \Gamma' @ R' \rightsquigarrow \Gamma @ R, \theta}{\Gamma' @ R' \vdash \theta e : \tau}$$

$$\boxed{\Gamma' @ R' \rightsquigarrow \Gamma @ R, \theta}$$

$$\text{(weak)} \quad \Gamma, x : \tau @ R \times \langle \text{ign} \rangle \rightsquigarrow \Gamma @ R, \emptyset$$

$$\text{(exch)} \quad \frac{\Gamma_1, y : \sigma, x : \tau, \Gamma_2 @ R \times \langle t \rangle \times \langle s \rangle \times Q \rightsquigarrow \Gamma_1, x : \tau, y : \sigma, \Gamma_2 @ R \times \langle s \rangle \times \langle t \rangle \times Q, \emptyset}{\Gamma_1, y : \sigma, x : \tau, \Gamma_2 @ R \times \langle t \rangle \times \langle s \rangle \times Q \rightsquigarrow \Gamma_1, x : \tau, y : \sigma, \Gamma_2 @ R \times \langle s \rangle \times \langle t \rangle \times Q, \emptyset}$$

$$\text{(contr)} \quad \frac{\Gamma_1, x : \tau, \Gamma_2 @ R \times \langle s \oplus t \rangle \times Q \rightsquigarrow \Gamma_1, y : \tau, z : \tau, \Gamma_2 @ R \times \langle s \rangle \times \langle t \rangle \times Q, [y, z \mapsto x]}{\Gamma_1, x : \tau, \Gamma_2 @ R \times \langle s \oplus t \rangle \times Q \rightsquigarrow \Gamma_1, y : \tau, z : \tau, \Gamma_2 @ R \times \langle s \rangle \times \langle t \rangle \times Q, [y, z \mapsto x]}$$

$$\text{(sub)} \quad \frac{\Gamma_1, x : \tau, \Gamma_2 @ R \times \langle s' \rangle \times T \rightsquigarrow \Gamma_1, x : \tau, \Gamma_2 @ R \times \langle s \rangle \times T, \emptyset}{\Gamma_1, x : \tau, \Gamma_2 @ R \times \langle s' \rangle \times T \rightsquigarrow \Gamma_1, x : \tau, \Gamma_2 @ R \times \langle s \rangle \times T, \emptyset} \quad (s \leq s')$$

Human-computer interaction (HCI)

Ignore inner working and implementation

Show that users can actually use it and how

The screenshot shows a web application interface with a text editor for code, a dropdown menu for selection, and a table of results. The code in the text editor is:

```
olympics.'filter data'. 'Games is'. 'Rio (2016)'. then  
  .'group data'. 'by Team'. 'sum Gold'. 'sum Silver'. then  
  .'sort data'.
```

The dropdown menu is open, showing the following options:

- by Gold
- by Gold descending
- by Silver
- by Silver descending
- by Team
- by Team descending
- then

The table of results is:

Team	Gold	Silver
United States	141	55
United Kingdom (Great Britain)	68	55
Germany	53	

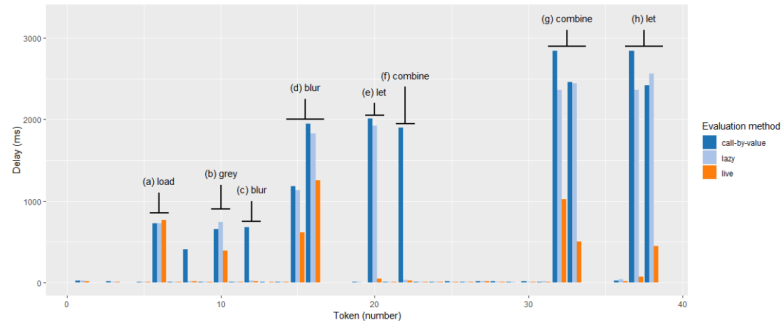
Annotations on the screenshot:

- 1: Full source code in a text editor
- 2: Programming via iterative prompting
- 3: Instantaneous preview of results

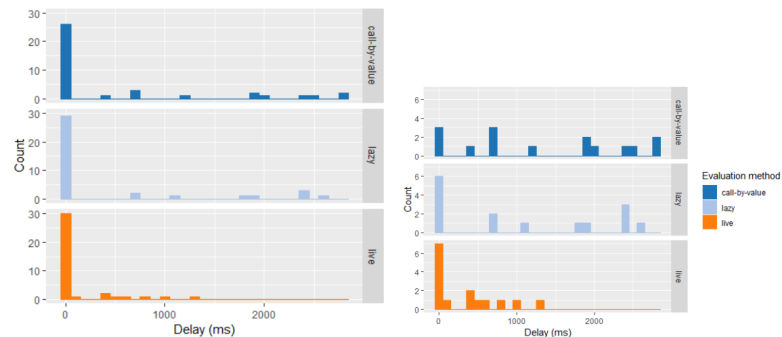
Performance evaluation

Ignore usability and design implications

Show that you can do better than a baseline



■ **Figure 11** Time required to recompute the results of a sample program after individual tokens are added or modified for three different evaluations strategies.



■ **Figure 12** Distribution of delays incurred when updating previews. We show a histogram computed from all delays (left) and only from delays larger than 15 ms (right).

Tiny systems

What is not covered?

-  Syntax choices and writing parsers
-  Compilation and JIT-based runtimes
-  Formal semantics and correctness
-  Supporting real-world use cases

Tiny systems

What can we study?

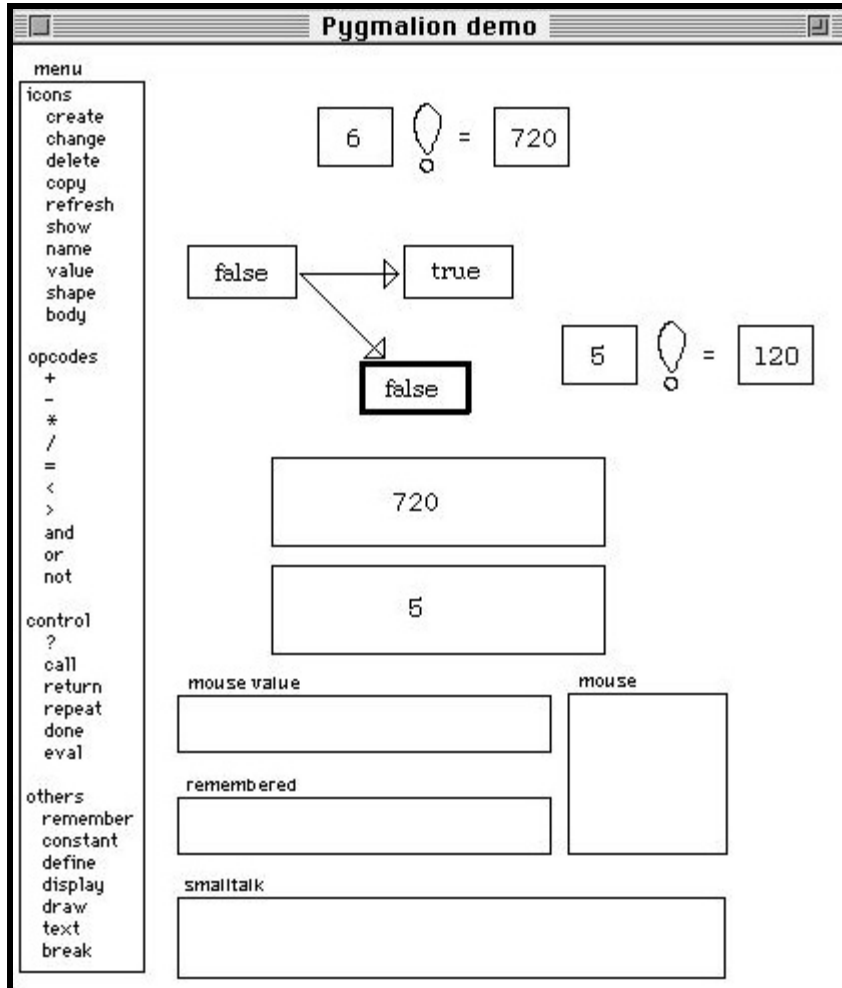
- Can talk about stateful interactive systems
- ⚙ Implement key aspects of inner working
- 💾 Reconstruct interesting past systems
- 📎 But cannot be printed on 12 pages of A4

Demo Pygmalion

Why study 1908s
programming systems?

No code programming!

First us of an "icon"!



Two uses of tiny systems

Education



Best way to learn?

Write it on your own!



Understand principles

As well as subtle details



I hope you'll have fun!

Little may be enough...

Research



Imagine new paradigms

End-user programming?



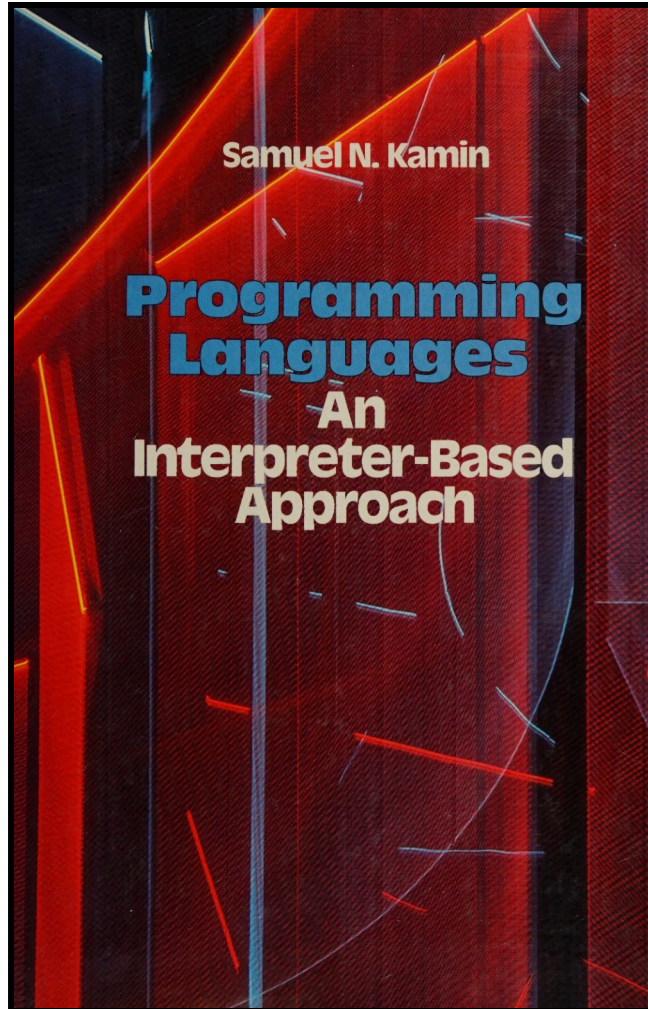
Focus on interaction

How exactly did it work



Ignore practical details

They can often wait!



Teaching tiny systems

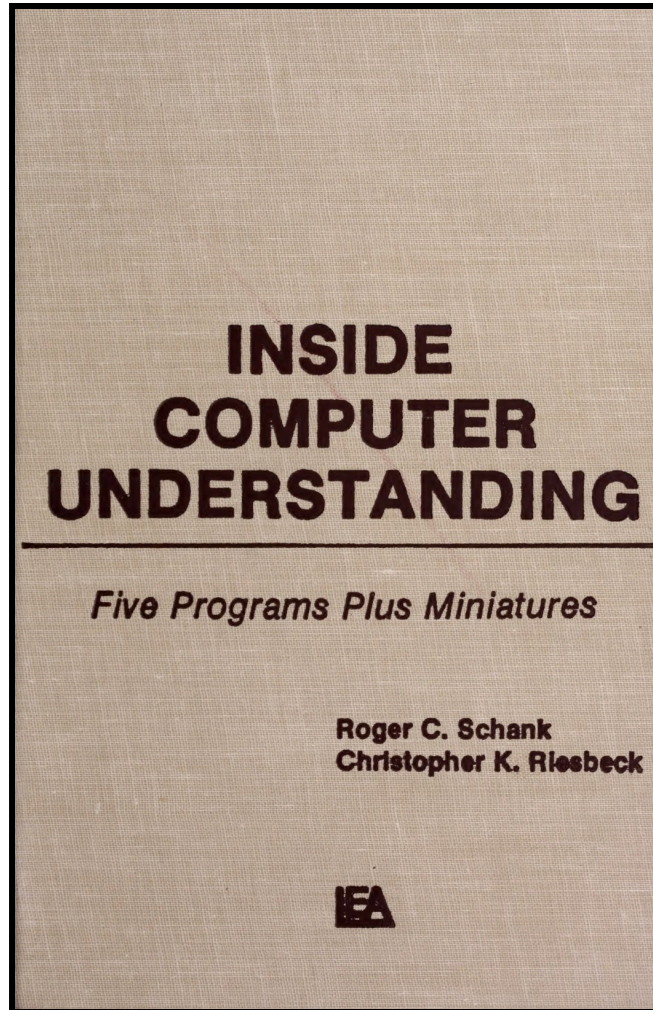
(Kamin, 1990)

Used in multiple
courses worldwide

Examples in Pascal

Languages covered are
APL, Clu, LISP, Prolog,
Smalltalk, Scheme, SASL

Not always focused
on the key aspect



Tiny systems and AI

(Schank, Riesbeck, 1981)

Miniature implementations
of 5 Yale AI lab programs

Faster, more efficient,
easier to understand,
modify and extend

"Miniatures, demos and
artworks" by Warren Sack

Tiny systems and ML

(Distill, 2016-2021)

Five affordances of
interactive articles

Connecting people & data

Making systems playful

Prompting self-reflection

Personalizing reading

Reducing cognitive load

 Distill

ABOUTPRIZESUBMIT

Communicating with Interactive Articles

Examining the design of interactive articles by synthesizing theory from disciplines such as education, journalism, and visualization.

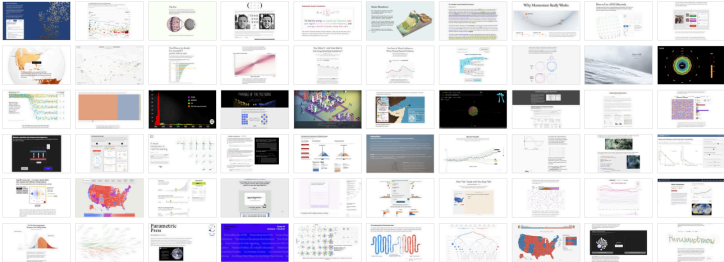


FIGURE 1: Exemplary Interactive Articles From Around The Web. Select an article for more information.

AUTHORS	AFFILIATIONS	PUBLISHED	DOI
Fred Hohman	Georgia Tech	Sept. 11, 2020	10.23915/distill.00028
Matthew Conlen	University of Washington		
Jeffrey Heer	University of Washington		
Duen Horng (Polo) Chau	Georgia Tech		

Contents

Introduction

Interactive Articles: Theory & Practice

- Connecting People and Data
- Making Systems Playful
- Prompting Self-Reflection
- Personalizing Reading
- Reducing Cognitive Load

Challenges for Authoring Interactives

Critical Reflections

Looking Forward

Computing has changed how people communicate. The transmission of news, messages, and ideas is instant. Anyone's voice can be heard. In fact, access to digital communication technologies such as the Internet is so fundamental to daily life that their disruption by government is condemned by the United Nations Human Rights Council [1]. But while the technology to distribute our ideas has grown in leaps and bounds, the interfaces have remained largely the same.

Parallel to the development of the internet, researchers like Alan Kay and Douglas Engelbart worked to build technology that would empower individuals and enhance cognition. Kay imagined the Dynabook [2] in the hands of children across the world. Engelbart, while best remembered for his "mother of all demos," was more interested in the ability of computation to augment human intellect [3]. Neal Stephenson wrote speculative fiction that imagined interactive paper that could display videos and interfaces, and books that could teach and respond to their readers [4].