What are the important problems for programming languages?

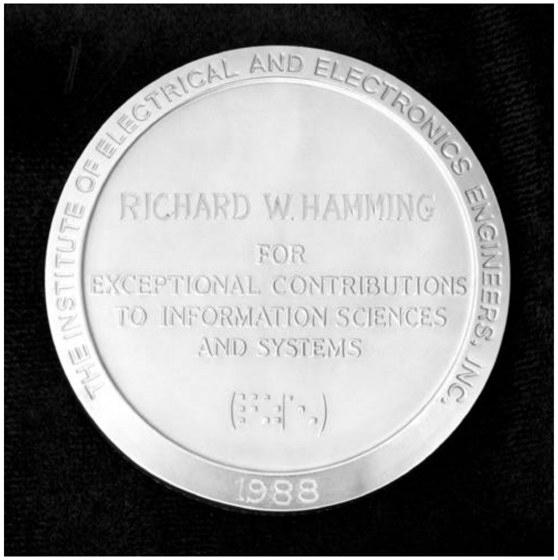
Philip Wadler, University of Edinburgh
wadler@inf.ed.ac.uk

Part I

Hamming

Richard W. Hamming, 1915–1998

- Los Alamos, 1945.
- Bell Labs, 1946–1976.
- Naval Postgraduate School, 1976–1998.
- Turing Award, 1968. (Third time given.)
- IEEE Hamming Medal, 1987. (First time given.)



What are the important problems?

Hamming started to eat at the Chemistry table.

"I started asking, 'What are the important problems of your field?' And after a week or so, 'What important problems are you working on?' And after some more time I came in one day and said, 'If what you are doing is not important, why are you working on it?' I wasn't welcomed after that.

"In the fall, Dave McCall stopped me in the hall and said, 'Hamming, that remark of yours got underneath my skin. I thought about it all summer. I haven't changed my research, but I think it was well worthwhile.' I noticed a couple of months later he was made the head of the department. I noticed the other day he was a Member of the National Academy of Engineering. I have never again heard the names of any of the other fellows."

— Hamming

You need an attack

"If you do not work on an important problem, it's unlikely you'll do important work. It's perfectly obvious. . . .

"Let me warn you, 'important problem' must be phrased carefully. The three outstanding problems in physics, in a certain sense, were never worked on while I was at Bell Labs. By important I mean guaranteed a Nobel Prize and any sum of money you want to mention. We didn't work on (1) time travel, (2) teleportation, and (3) antigravity. They are not important problems because we do not have an attack. It's not the consequence that makes a problem important, it is that you have a reasonable attack."

— Hamming

Keep many problems in mind

"Most great scientists know many important problems. They have something between 10 and 20 important problems for which they are looking for an attack. And when they see a new idea come up, one hears them say 'Well that bears on this problem.' They drop all the other things and get after it.

"Now I can tell you a horror story that was told to me but I can't vouch for the truth of it. I was sitting in an airport talking to a friend of mine from Los Alamos about how it was lucky that the fission experiment occurred over in Europe when it did because that got us working on the atomic bomb here in the US. He said 'No; at Berkeley we had gathered a bunch of data; we didn't get around to reducing it because we were building some more equipment, but if we had reduced that data we would have found fission.' They had it in their hands and they didn't pursue it. They came in second!"

— Hamming

Ambiguity

"Great scientists tolerate ambiguity very well. They believe the theory enough to go ahead; they doubt it enough to notice the errors and faults so they can step forward and create the new replacement theory. If you believe too much you'll never notice the flaws; if you doubt too much you won't get started. It requires a lovely balance. . . . Darwin writes in his autobiography that he found it necessary to write down every piece of evidence which appeared to contradict his beliefs because otherwise they would disappear from his mind. When you find apparent flaws you've got to be sensitive and keep track of those things, and keep an eye out for how they can be explained or how the theory can be changed to fit them."

— Hamming

Great thoughts

“I finally adopted what I called ‘Great Thoughts Time.’ When I went to lunch Friday noon, I would only discuss great thoughts after that. By great thoughts I mean ones like: ‘What will be the role of computers in all of AT&T?’, ‘How will computers change science?’

“For example, I came up with the observation at that time that nine out of ten experiments were done in the lab and one in ten on the computer. I made a remark to the vice presidents one time, that it would be reversed, i.e. nine out of ten experiments would be done on the computer and one in ten in the lab. They knew I was a crazy mathematician and had no sense of reality. I knew they were wrong and they’ve been proved wrong while I have been proved right.”

— Hamming

Keep your door open

> "I notice that if you have the door to your office closed, you get more work done today and tomorrow, and you are more productive than most. But 10 years later somehow you don't know quite know what problems are worth working on; all the hard work you do is sort of tangential in importance. He who works with the door open gets all kinds of interruptions, but he also occasionally gets clues as to what the world is and what might be important."
>
> — Hamming

Generalize

"When using the machine up in the attic in the early days, I was solving one problem after another after another; a fair number were successful and there were a few failures. I went home one Friday after finishing a problem, and curiously enough I wasn't happy; I was depressed. I could see life being a long sequence of one problem after another after another. After quite a while of thinking I decided, 'No, I should be in the mass production of a variable product. I should be concerned with all of next year's problems, not just the one in front of my face.' By changing the question I still got the same kind of results or better, but I changed things and did important work. I attacked the major problem—How do I conquer machines and do all of next year's problems when I don't know what they are going to be?"

— Hamming

If I have seen further than others . . .

"How do I do this one so I'll be on top of it? How do I obey Newton's rule? He said, 'If I have seen further than others, it is because I've stood on the shoulders of giants.' These days we stand on each other's feet!

"I suggest that by altering the problem, by looking at the thing differently, you can make a great deal of difference in your final productivity because you can either do it in such a fashion that people can indeed build on what you've done, or you can do it in such a fashion that the next person has to essentially duplicate again what you've done."

— Hamming

Sell yourself

"I have now come down to a topic which is very distasteful; it is not sufficient to do a job, you have to sell it. 'Selling' to a scientist is an awkward thing to do. It's very ugly; you shouldn't have to do it. The world is supposed to be waiting, and when you do something great, they should rush out and welcome it. But the fact is everyone is busy with their own work. You must present it so well that they will set aside what they are doing, look at what youve done, read it, and come back and say, 'Yes, that was good.'

"While going to meetings I had already been studying why some papers are remembered and most are not. The technical person wants to give a highly limited technical talk. Most of the time the audience wants a broad general talk and wants much more survey and background than the speaker is willing to give. As a result, many talks are ineffective."

— Hamming

Part II

Great Problems of The Past

Turing awards in Programming Languages

1966 Alan Perlis—Algol

1971 John McCarthy—Lisp

1972 Edsger Dijkstra—Algol, Structured Programming

1974 Donald Knuth—“The Art of Computer Programming”

1976 Michael Rabin and Dana Scott—“Finite Automata and their Decision Problem”

1977 John Backus—Fortran, BNF

1978 Robert Floyd—Parsing, semantics, program verification

1979 Kenneth Iverson—APL

1980 C.A.R. Hoare—Algol, Hoare Logic, CSP

Turing awards in Programming Languages

1983 Dennis M. Ritchie and Kenneth Lane Thompson—C, Unix

1984 Niklaus E. Wirth—Pascal

1991 Robin Milner—LCF, ML, CCS

1996 Amir Pnueli—temporal logic

2001 Ole-Johan Dahl and Kristen Nygarrd—Simula

2003 Alan Kay—Smalltalk

2005 Peter Naur—Algol, BNF

2006 Frances Allen—compilers

2007 Edmund Clarke, E. Allen Emerson, Joseph Sifakis—model checking

2008 Barbara Liskov—CLU

Programming Language Achievement Award

1997 Guy Steele—Scheme, Common Lisp, HPF, Java

1998 Frances Allen—compilers

1999 Ken Kennedy—compilers, parallel computing

2000 Susan Graham

2001 Robin Milner—LCF, ML, CCS

2002 John McCarthy—Lisp

2003 John Reynolds—Gedanken
definitional interpreters, continuations, second-order lambda calculus

Programming Language Achievement Award

2004 John Backus—Fortran, FP

2005 Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides—design patterns

2006 Ron Cyton, Jeanne Ferrante, Barry Rosen, Mark Wegman, Kenneth Zadeck—single assignment

2007 Niklaus Wirth—Pascal, Modula 2

2008 Barbara Liskov—CLU

2009 Rod Burstall—Hope
algebraic types, structural induction, dependent types for modules

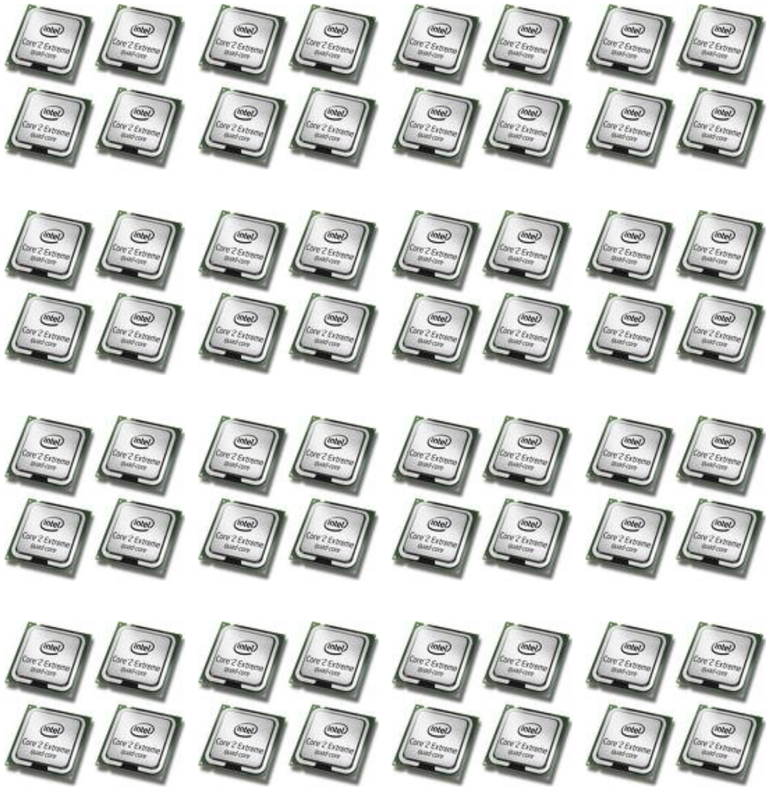
Part III

Great Problems of Today

Distribution and multicore

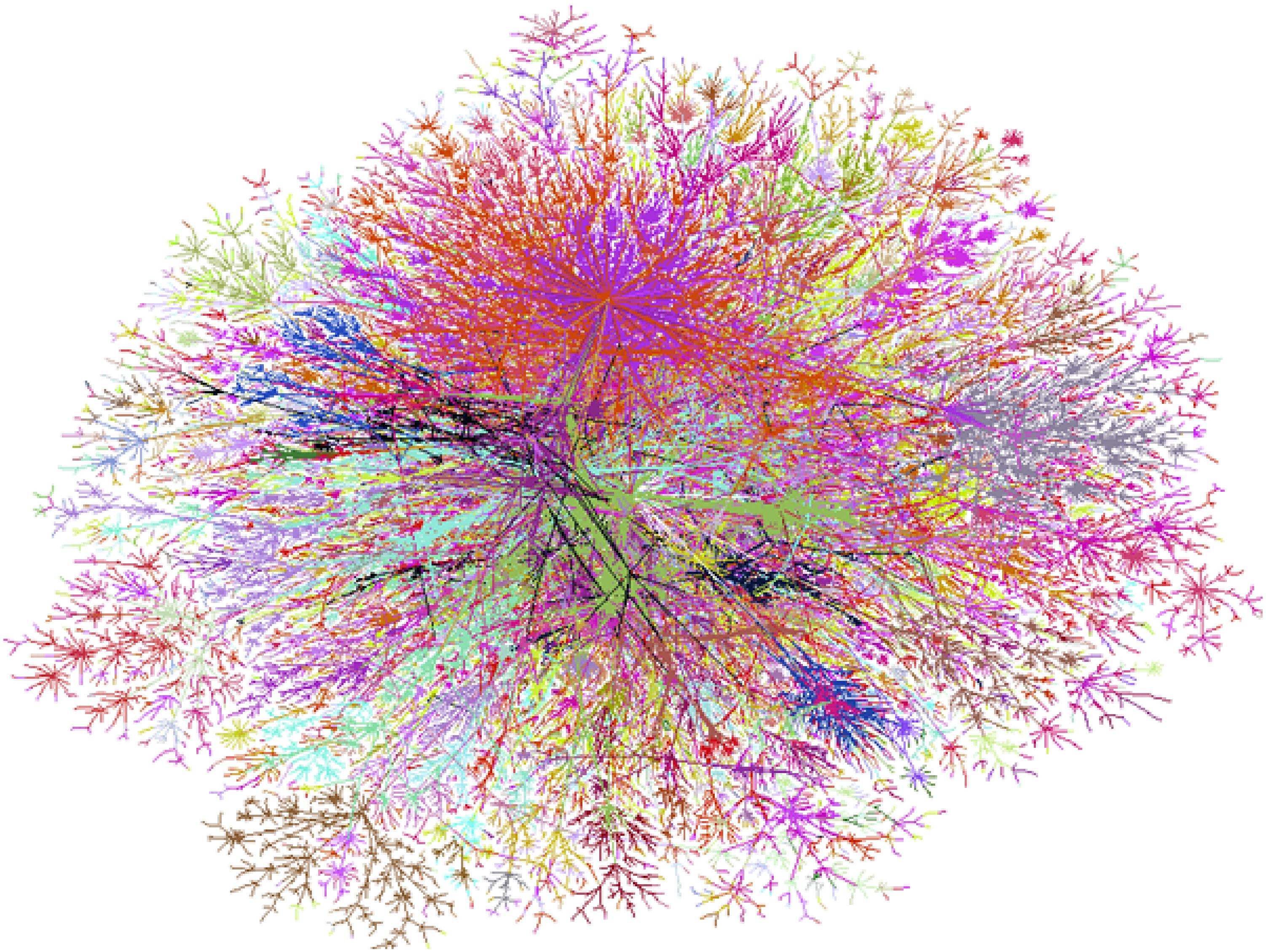
Distribution and multicore

Distribution and multicore



Distribution and multicore

Distribution and multicore



Distribution and multicore

Join Calculus

JoCaml, Polyphonic C#

$$P \stackrel{\text{def}}{=} x\langle\tilde{v}\rangle \qquad J \stackrel{\text{def}}{=} x\langle\tilde{v}\rangle \qquad D \stackrel{\text{def}}{=} J \triangleright P$$

$$\texttt{def}\ D\ \texttt{in}\ P \qquad J|J \qquad D \wedge D$$

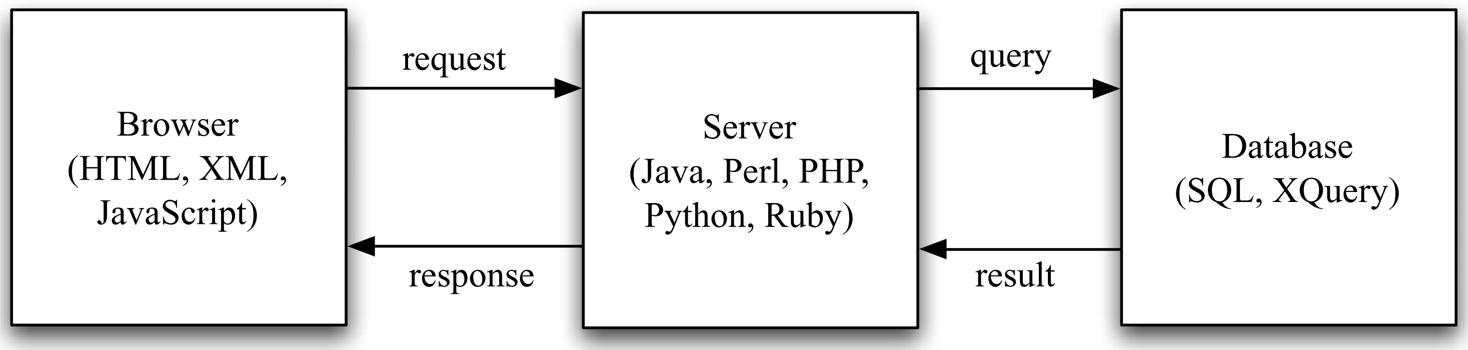
$$P|P$$

Programming the web

Programming the web

Programming the web

Links



Programming the web

iData For The World Wide Web

Programming Interconnected Web Forms

Rinus Plasmeijer and Peter Achten

Software Technology, Nijmegen Institute for Computing and Information Sciences,
Radboud University Nijmegen, Toernooiveld 1, 6525ED Nijmegen, Netherlands

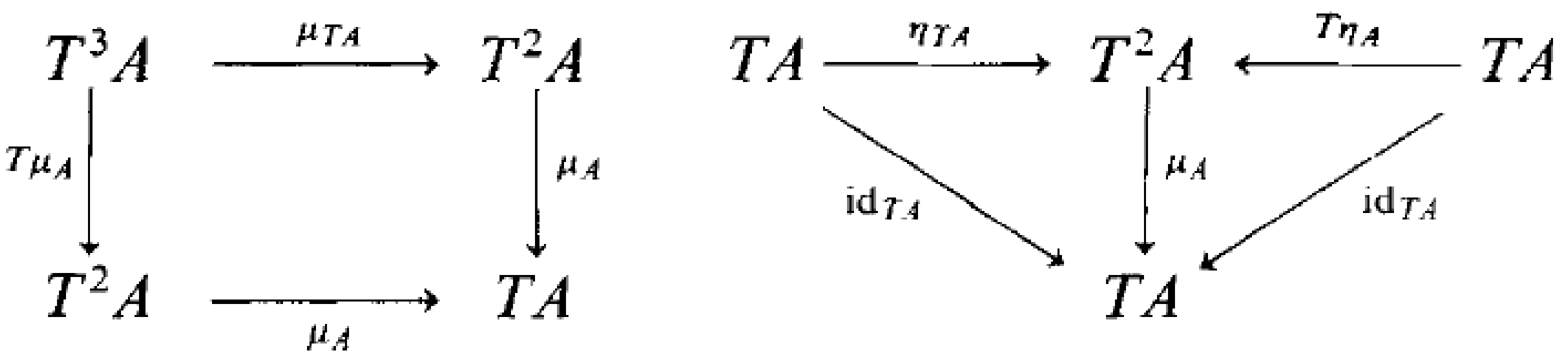
```
counterIData :: IDataId Int → IDataFun Int
counterIData iDataId i         = mkIData iDataId i ibm
where ibm        = { toView    = λn v → useOldView (n,down,up) v
                   , updView   = λ_ v → updCounter v
                   , fromView  = λ_ (n,_,_) → n
                   , resetView = Nothing }
      (up,down) = (LButton (defpixel / 6) "+",LButton (defpixel / 6) "-")

      updCounter :: Counter → Counter
      updCounter (n,Pressed,_) = (n - 1,down,up)
      updCounter (n,_,Pressed) = (n + 1,down,up)
      updCounter noPresses     = noPresses

      useOldView new (Just old)= old
      useOldView new Nothing   = new
```

Programming the web

Notions of Computation and Monads

EUGENIO MOGGI*

Department of Computer Science, University of Edinburgh, Edinburgh EH9 3JZ, UK



Programming the web

FUNCTIONAL PEARLS

[ABORTED] A trail told by an idiom

Conor McBride

1 Introduction

Nobody likes their programs to be full of sound and fury, signifying nothing. Abstraction is the weapon of choice in the war on wanton waffle. This paper is about an abstraction which I find rather handy. It's a weaker variation on the theme of a monad, but it has a more *functional* feel. I call it an *idiom*:

```
infixl 9 ⟨%⟩
class Idiom i where
    idi     ::  x → i x
    (⟨%⟩)   ::  i (s → t) → i s → i t        — pronounced 'apply'
```

Programming the web

The Essence of Form Abstraction⋆

Ezra Cooper, Sam Lindley, Philip Wadler, and Jeremy Yallop

School of Informatics, University of Edinburgh

$$\begin{array}{l}
\textsf{module type } \mathit{Idiom} = \textsf{sig} \\
\quad \textsf{type } \alpha\ t \\
\quad \textsf{val } \mathit{pure}\ : \alpha \to \alpha\ t \\
\quad \textsf{val } (\otimes) : (\alpha \to \beta)\ t \to \alpha\ t \to \beta\ t \\
\textsf{end}
\end{array}$$

$$\begin{array}{l}
\textsf{module type } \mathit{FORMLET} = \textsf{sig} \\
\quad \textsf{include } \mathit{Idiom} \\
\quad \textsf{val } \mathit{xml}\ : \mathit{xml} \to \mathit{unit}\ t \\
\quad \textsf{val } \mathit{text}\ : \mathit{string} \to \mathit{unit}\ t \\
\quad \textsf{val } \mathit{tag}\ : \mathit{tag} \to \mathit{attrs} \to \alpha\ t \to \alpha\ t \\
\quad \textsf{val } \mathit{input} : \mathit{string}\ t \\
\quad \textsf{val } \mathit{run}\ : \alpha\ t \to \mathit{xml} \times (\mathit{env} \to \alpha) \\
\textsf{end}
\end{array}$$

Fig. 4. The idiom and formlet interfaces

Programming the web

let $date\_formlet : date\ formlet =$ formlet
<div>
Month: $\{input\_int \Rightarrow month\}$
Day: $\{input\_int \Rightarrow day\}$
</div>
yields $make\_date\ month\ day$

let $travel\_formlet : (string \times date \times date)\ formlet =$
formlet
<#>
Name: $\{input \Rightarrow name\}$
<div>
Arrive: $\{date\_formlet \Rightarrow arrive\}$
Depart: $\{date\_formlet \Rightarrow depart\}$
</div>
$\{submit$ "Submit"$\}$
</#>
yields $(name, arrive, depart)$

Programming the web

The IntelliFactory WebSharper™ Platform

Writing good web applications is not an easy task today. It requires a mastery of numerous languages (JavaScript, HTML, CSS), and an acute awareness of existing standards and browser implementation quirks. Poor debugging tools, and the lack of compositionality and component reuse in the multi-tiered, multi-language web environment compound the problem even more.

Seamless ASP.NET Integration

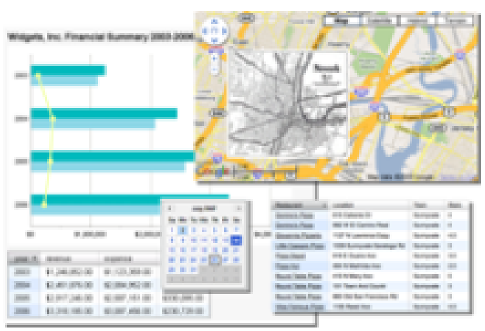
Plug your WebSharper™ applications into existing ASP.NET sites and deploy via IIS!

Functional Reactive Coding

Use powerful F# asynchronous constructs and first-class events with your client applications!

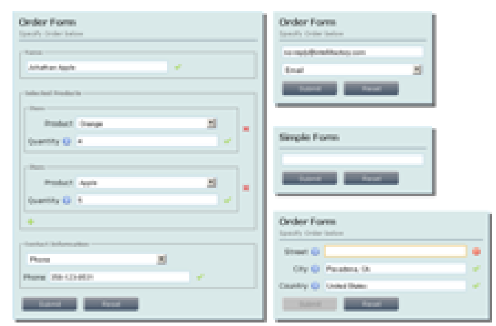
Extensions

Develop applications that use any JavaScript-based technology via WebSharper™ bindings!

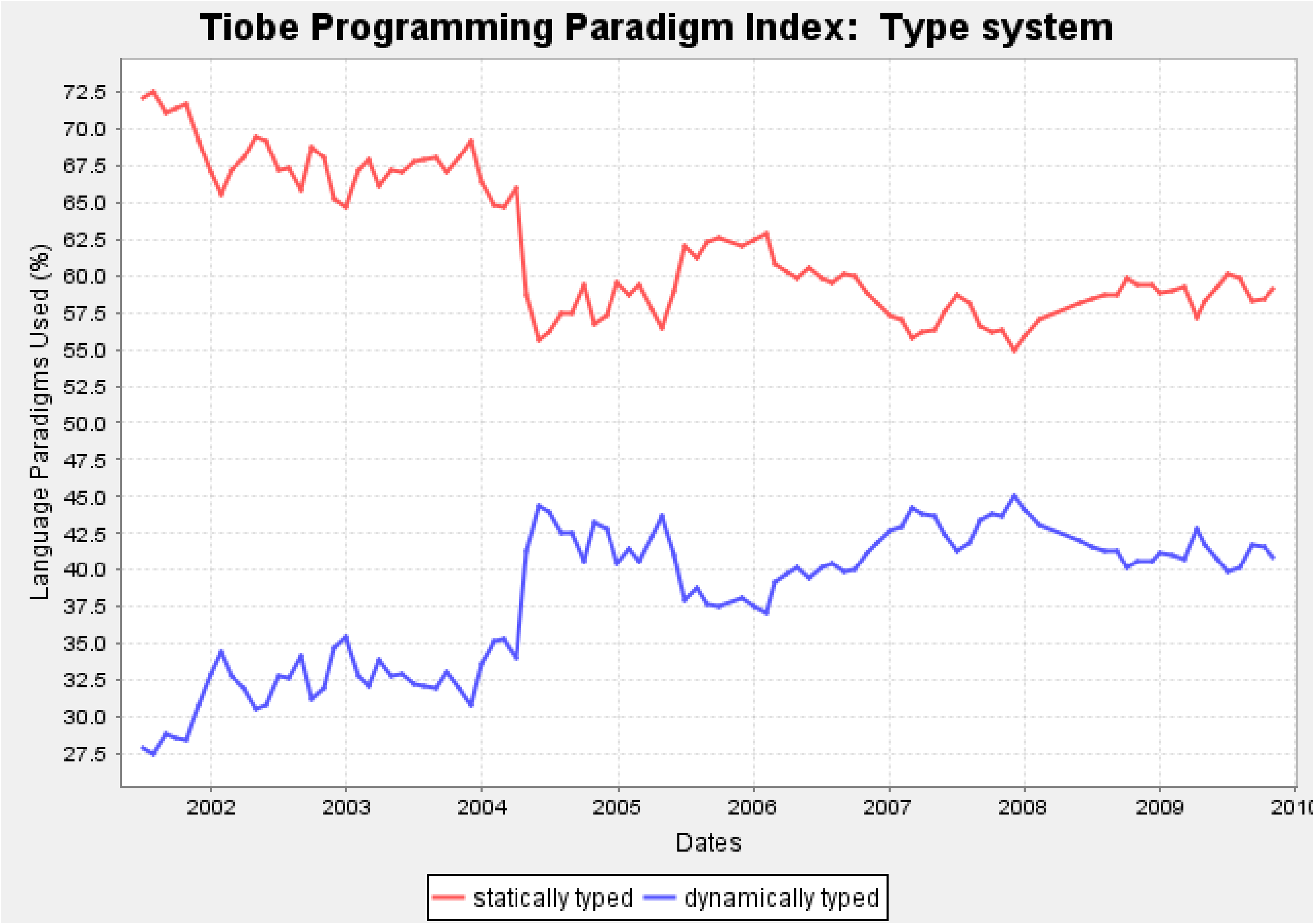


Formlets

Create interactive forms with validation using type-safe code in just lines!



Static and dynamic types



Static and dynamic types

Blame for All

Amal Ahmed[1], Robert Bruce Findler[2], Jacob Matthews[3], and Philip Wadler[4]

[1] Indiana University
[2] Northwestern University
[3] Google
[4] University of Edinburgh

$$\sigma \triangleright \langle A' \to B' \Leftarrow A \to B \rangle^p \; v \longmapsto \sigma \triangleright \lambda x : A' . \langle B' \Leftarrow B \rangle^p \; (v \; (\langle A \Leftarrow A' \rangle^{\overline{p}} \; x))$$

$$\sigma \triangleright (\Lambda X . t) \; A \longmapsto \sigma, X := A \triangleright t$$

$$\sigma \triangleright \langle B \Leftarrow \forall X . A \rangle^p \; v \longmapsto \sigma \triangleright \langle B \Leftarrow A[X := \star] \rangle^p \; (v \; \star)$$

$$\sigma \triangleright \langle \forall X . B \Leftarrow A \rangle^p \; v \longmapsto \sigma \triangleright \Lambda X . \langle B \Leftarrow A \rangle^p \; v$$

Static and dynamic types

$$
\begin{array}{rcll}
\sigma \triangleright (\Lambda X.\, t)\; A & \longmapsto & \sigma, X := A \triangleright t & \text{if } X \notin \mathrm{dom}(\sigma) \\
\sigma \triangleright \langle B \Leftarrow \forall X.\, A\rangle^{p}\; v & \longmapsto & \sigma \triangleright \langle B \Leftarrow A[X := \star]\rangle^{p}\; (v\; \star) & \\
\sigma \triangleright \langle \forall X.\, B \Leftarrow A\rangle^{p}\; v & \longmapsto & \sigma \triangleright \Lambda X.\, \langle B \Leftarrow A\rangle^{p}\; v & \text{if } X \notin \mathrm{dom}(\sigma) \\
\sigma \triangleright \langle \star \Leftarrow G\rangle^{p}\; v\; \mathtt{is}^{q}\; G & \longmapsto & \sigma \triangleright \mathtt{true} & \text{if } G \neq X \\
\sigma \triangleright \langle \star \Leftarrow G\rangle^{p}\; v\; \mathtt{is}^{q}\; H & \longmapsto & \sigma \triangleright \mathtt{false} & \text{if } G \neq X \text{ and } G \neq H \\
\sigma \triangleright \langle \star \Leftarrow X\rangle^{p}\; v\; \mathtt{is}^{q}\; H & \longmapsto & \mathtt{blame}\; q & \\
\sigma \triangleright (\lambda x{:}A.\, t)\; v & \longmapsto & \sigma \triangleright t[x := v] & \\
\sigma \triangleright \langle A' \to B' \Leftarrow A \to B\rangle^{p}\; v & \longmapsto & \sigma \triangleright \lambda x{:}A'.\, \langle B' \Leftarrow B\rangle^{p}\; (v\; (\langle A \Leftarrow A'\rangle^{\overline{p}}\; x)) & \\
\sigma \triangleright \langle \star \Leftarrow \star\rangle^{p}\; v & \longmapsto & \sigma \triangleright v & \\
\sigma \triangleright \langle \iota \Leftarrow \iota\rangle^{p}\; v & \longmapsto & \sigma \triangleright v & \\
\sigma \triangleright \langle X \Leftarrow X\rangle^{p}\; v & \longmapsto & \sigma \triangleright v & \\
\sigma \triangleright \langle \star \Leftarrow A \to B\rangle^{p}\; v & \longmapsto & \sigma \triangleright \langle \star \Leftarrow \star \to \star\rangle^{p}\; \langle \star \to \star \Leftarrow A \to B\rangle^{p}\; v & \text{if } A \to B \neq \star \to \star \\
\sigma \triangleright \langle A \to B \Leftarrow \star\rangle^{p}\; v & \longmapsto & \sigma \triangleright \langle A \to B \Leftarrow \star \to \star\rangle^{p}\; \langle \star \to \star \Leftarrow \star\rangle^{p}\; v & \text{if } A \to B \neq \star \to \star \\
\sigma \triangleright \langle G \Leftarrow \star\rangle^{q}\; \langle \star \Leftarrow G\rangle^{p}\; v & \longmapsto & \sigma \triangleright v & \\
\sigma \triangleright \langle H \Leftarrow \star\rangle^{q}\; \langle \star \Leftarrow G\rangle^{p}\; v & \longmapsto & \mathtt{blame}\; q & \text{if } G \neq H
\end{array}
$$

Static and dynamic types—Equality in Haskell

```
class  Eq a  where
  (==) :: a -> a -> Bool

instance  Eq Int  where
  (==)  =  eqInt

instance  Eq Char  where
  x == y             =  ord x == ord y

instance  (Eq a, Eq b) => Eq (a,b)  where
  (u,v) == (x,y)     =  (u == x) && (v == y)

instance  Eq a => Eq [a]  where
  [] == []           =  True
  [] == y:ys         =  False
  x:xs  ==  []       =  False
  x:xs  ==  y:ys     =  (x == y) && (xs == ys)
```

Static and dynamic types—Equality in Lisp

```
(defun (equal x y)
  (or
    (eq x y)
    (and
      (consp x)
      (consp y)
      (equal (car x) (car y))
      (equal (cdr x) (cdr y)))))
```

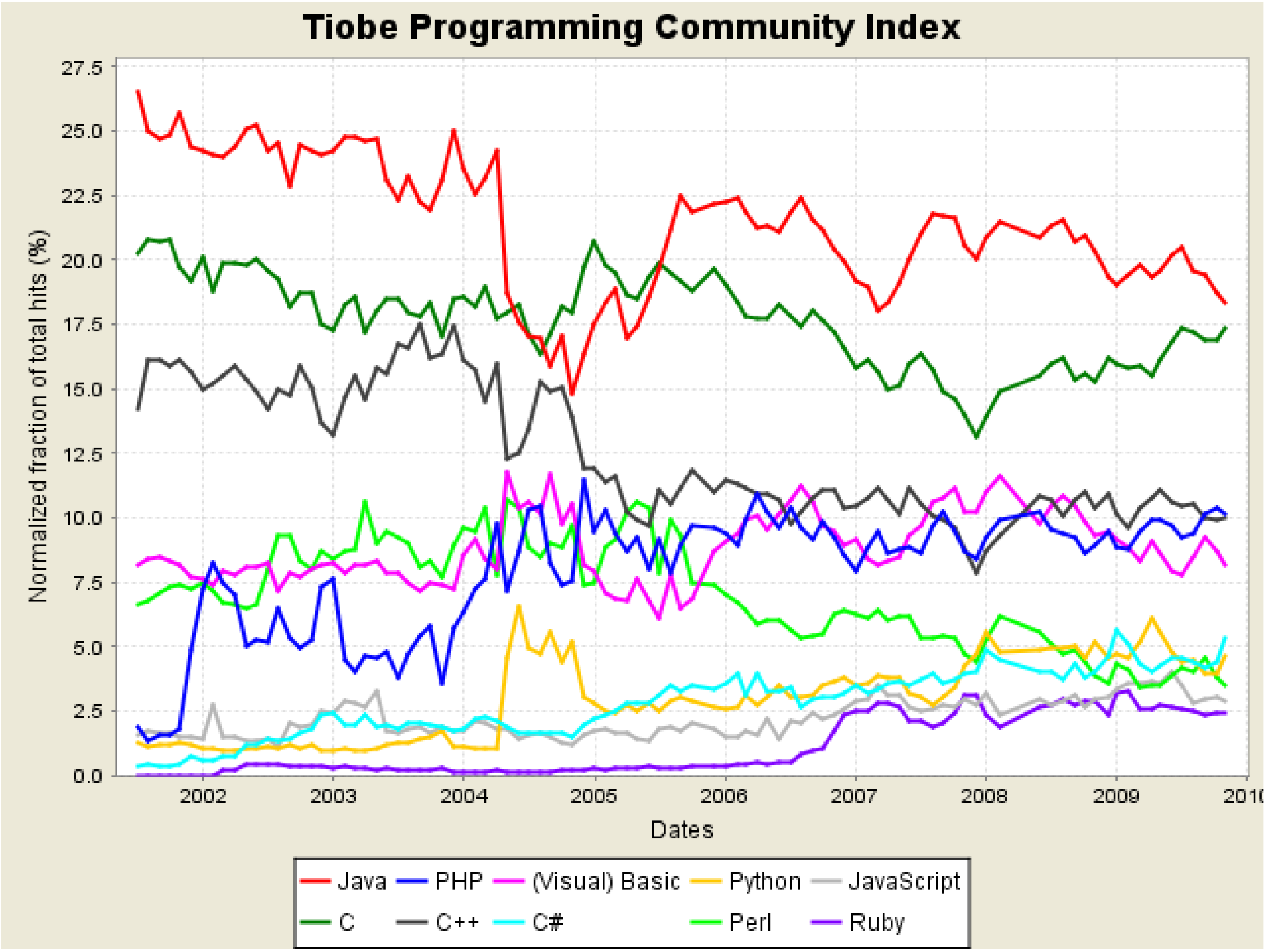
The next order of magnitude

"When I got interested in the field, the mainstream was probably Fortran and COBOL and even C was fairly new. The functional programming pioneers spoke of an order of magnitude improvement in productivity and I think functional programming has delivered that.

"If you compare Haskell programs to C code or even C++ often, they are about an order of magnitude smaller and simpler. The same is true for Erlang, those results are being validated in the industry. Where is the next order of magnitude coming from? I wish I had an answer to that question because it's hard to see almost. When you look at a beautiful Haskell program, how could this be 10 times shorter? But I think we need to be asking ourselves that kind of question. If I had a good idea there, I would spend the rest of my career working on it."

— John Hughes 

The Popularity Contest



The Popularity Contest

Position Nov 2009	Position Nov 2008	Delta in Position	Programming Language	Ratings Nov 2009	Delta Nov 2008	Status
1	1	=	Java	18.373%	-1.93%	A
2	2	=	C	17.315%	+2.04%	A
3	5	↑↑	PHP	10.176%	+1.24%	A
4	3	↓	C++	10.002%	-0.36%	A
5	4	↓	(Visual) Basic	8.171%	-1.10%	A
6	7	↑	C#	5.346%	+1.32%	A
7	6	↓	Python	4.672%	-0.47%	A
8	9	↑	Perl	3.490%	-0.39%	A
9	10	↑	JavaScript	2.916%	-0.01%	A
10	11	↑	Ruby	2.404%	-0.47%	A
11	8	↓↓↓	Delphi	2.127%	-1.88%	A
12	13	↑	PL/SQL	0.908%	-0.03%	A
13	38	↑↑↑↑↑↑↑↑↑↑	Objective-C	0.893%	+0.74%	A-
14	14	=	SAS	0.816%	+0.09%	A
15	16	↑	Pascal	0.654%	+0.14%	A–
16	15	↓	ABAP	0.643%	+0.07%	A–
17	21	↑↑↑↑	Lisp/Scheme	0.618%	+0.15%	B
18	22	↑↑↑↑	MATLAB	0.611%	+0.15%	B
19	20	↑	Lua	0.544%	+0.07%	B
20	12	↓↓↓↓↓↓↓↓	D	0.543%	-0.90%	B

The Popularity Contest

21	ActionScript	0.519%
22	COBOL	0.430%
23	Transact-SQL	0.412%
24	FoxPro/xBase	0.379%
25	Fortran	0.376%
26	Logo	0.361%
27	Scratch	0.346%
28	Alice	0.329%
29	Ada	0.305%
30	S-lang	0.292%
31	RPG (OS/400)	0.285%
32	Erlang	0.261%
33	PowerShell	0.259%
34	Scala	0.244%
35	Awk	0.243%

36	Prolog	0.234%
37	NXT-G	0.234%
38	Tcl/Tk	0.221%
39	Focus	0.209%
40	LabWindows/CVI	0.193%
41	Haskell	0.183%
42	PL/I	0.177%
43	JavaFX Script	0.176%
44	MAX/MSP	0.161%
45	LabVIEW	0.157%
46	Falcon	0.156%
47	Groovy	0.152%
48	Modula-3	0.146%
49	Forth	0.144%
50	Smalltalk	0.135%

Empiricism

Click to LOOK INSIDE!

Java Generics and Collections (Paperback)

by M Naftalin (Author), P Wadler (Author)

(3 customer reviews)

RRP: ~~£26.99~~

Price: £14.43 & this item **Delivered FREE in the UK** with Super Saver Delivery. See details and conditions

You Save: £12.56 (47%)

In stock.

Dispatched from and sold by **Amazon.co.uk**. Gift-wrap available.

Want guaranteed delivery by 1pm Wednesday, November 25? Order it in the next 0 hours and 43 minutes, and choose **Express Delivery** at checkout. See Details

29 new from £12.85 **9 used** from £13.69

Share your own customer images

Search inside this book

Other Editions:	RRP:	Our Price:	Other Offers:
Paperback			2 used & new from £30.98

Quantity: 1

Add to Shopping Basket

or

Sign in to turn on 1-Click ordering.

Add to Wish List

More Buying Choices

38 used & new from £12.85

Have one to sell? Sell yours here

Sale

12 Days of Christmas Sale in Books

Get up to 65% off some of our top titles. Shop now

› See more product promotions

Special Offers and Product Promotions

- Illuminate your book with the innovative Philips LED reading light--exclusive to Amazon.co.uk. Shop now.

Frequently Bought Together

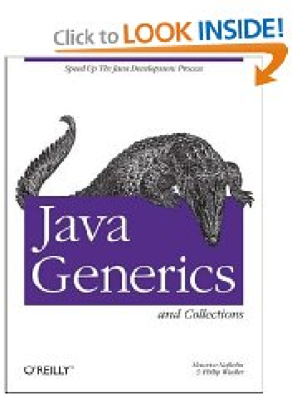
Price For All Three: £48.88

Add all three to Basket

Show availability and delivery details

- **This item:** Java Generics and Collections by M Naftalin
- Java Concurrency in Practice by Brian Goetz
- Effective Java: Second Edition by Joshua Bloch

Empiricism



Share your own customer images

Search inside this book

Java Generics and Collections (Paperback)

~ Maurice Naftalin (Author), Philip Wadler (Author)

Key Phrases: collections framework, substitution principle, generic client, Principle of Truth, Joshua Bloch, Principle of Indecent Exposure (more...)

(23 customer reviews)

List Price: ~~$34.99~~

Price: $23.09 & eligible for **FREE Super Saver Shipping** on orders over $25. Details

You Save: $11.90 (34%)

In Stock.

Ships from and sold by **Amazon.com**. Gift-wrap available.

Want it delivered Wednesday, November 25? Order it in the next 6 hours and 13 minutes, and choose **One-Day Shipping** at checkout. Details

33 new from $19.08 **17 used** from $16.93

Formats	Amazon Price	New from	Used from
Kindle Edition, February 9, 2009	$20.00	–	–
Paperback, September 30, 2006	$23.09	$19.08	$16.93
Unknown Binding, December 31, 2006	–	–	–

Quantity: 1

Add to Shopping Cart

or

Sign in to turn on 1-Click ordering.

or

Add to Cart with FREE Two-Day Shipping

Amazon Prime Free Trial required. Sign up when you check out. Learn More

Add to Wish List

Express Checkout with PayPhrase

Loving Familiarity

What's this? | Create PayPhrase

More Buying Choices

50 used & new from $16.93

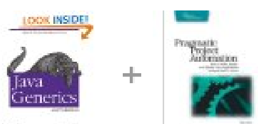
Have one to sell? Sell yours here

Share with Friends

Like this book? Find similar titles from O'Reilly and Partners in our O'Reilly Bookstore.

Best Value

Buy **Java Generics and Collections** and get **Pragmatic Project Automation: How to Build, Deploy, and Monitor Java Apps** at an **additional 5% off** Amazon.com's everyday low price.

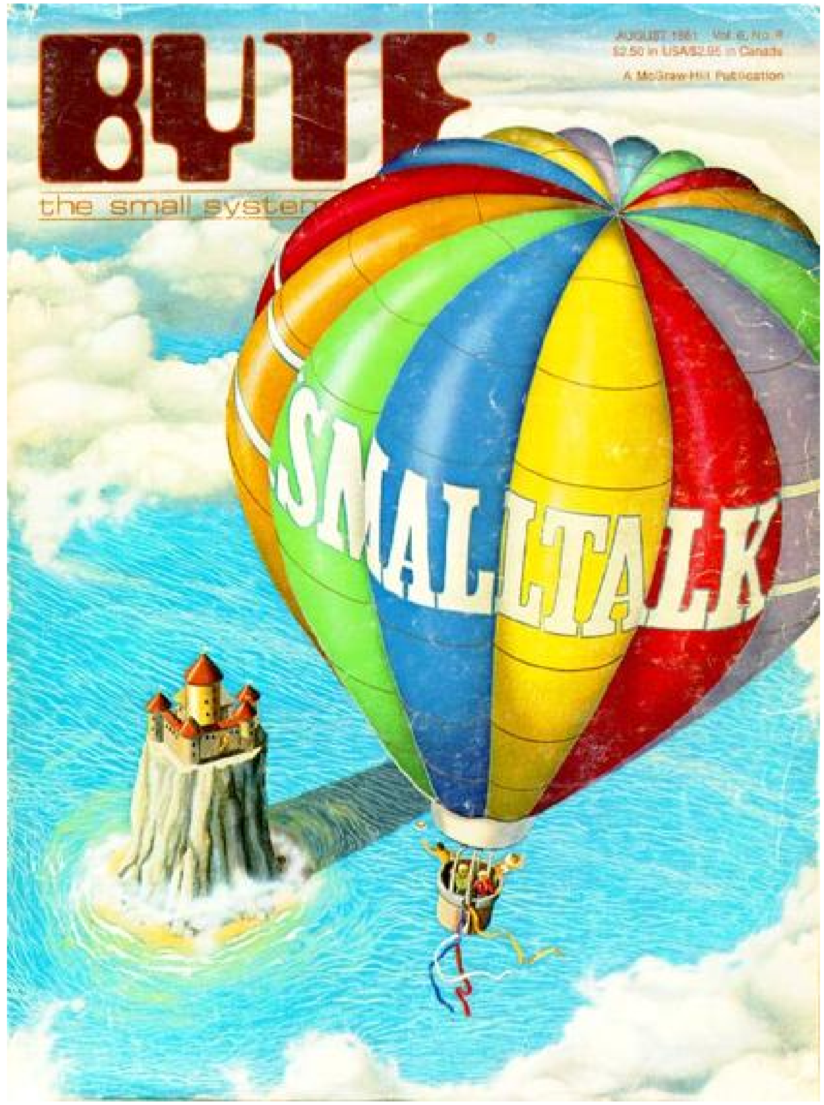


Buy Together Today: $41.87

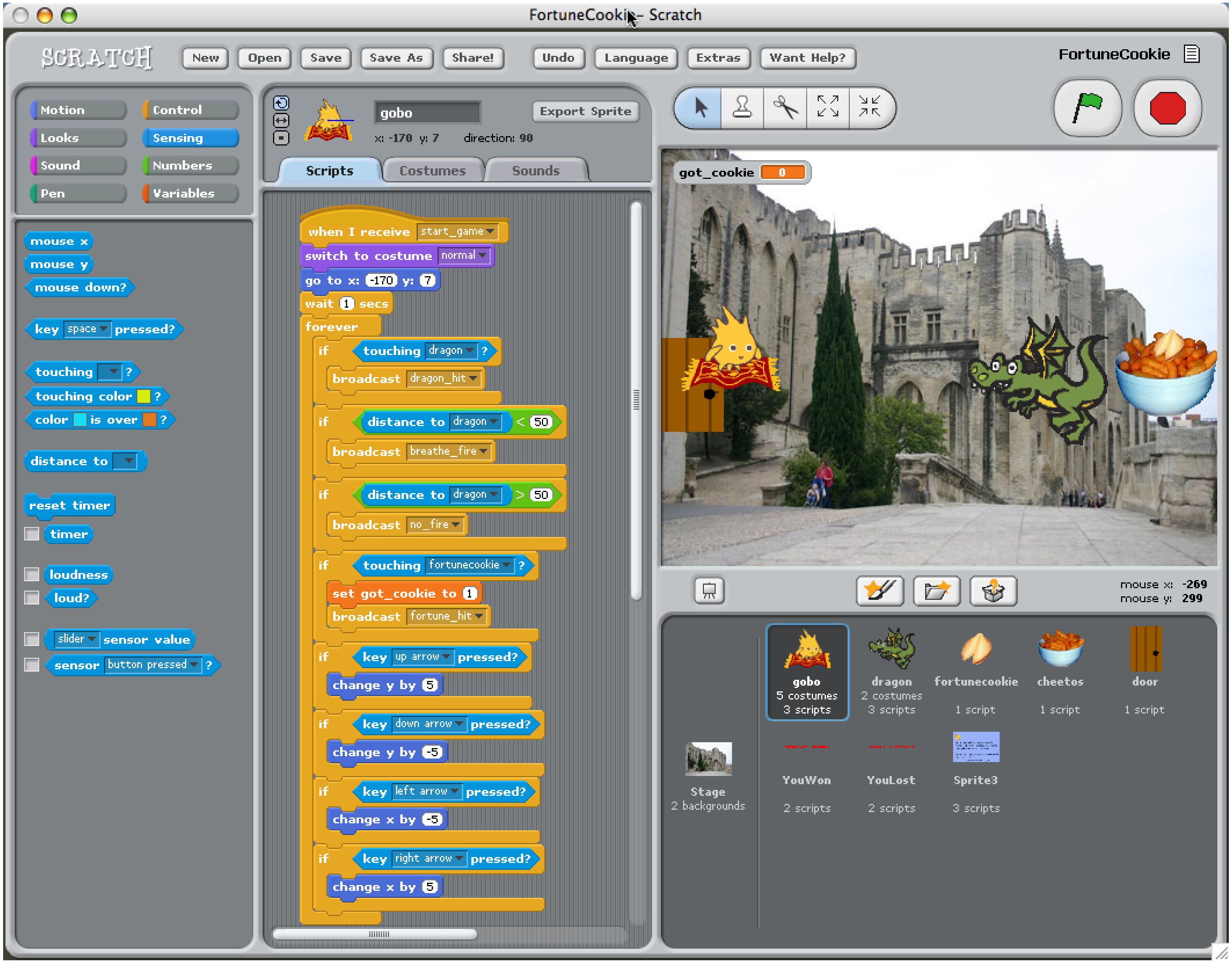
Add both to Cart

Show availability and shipping details

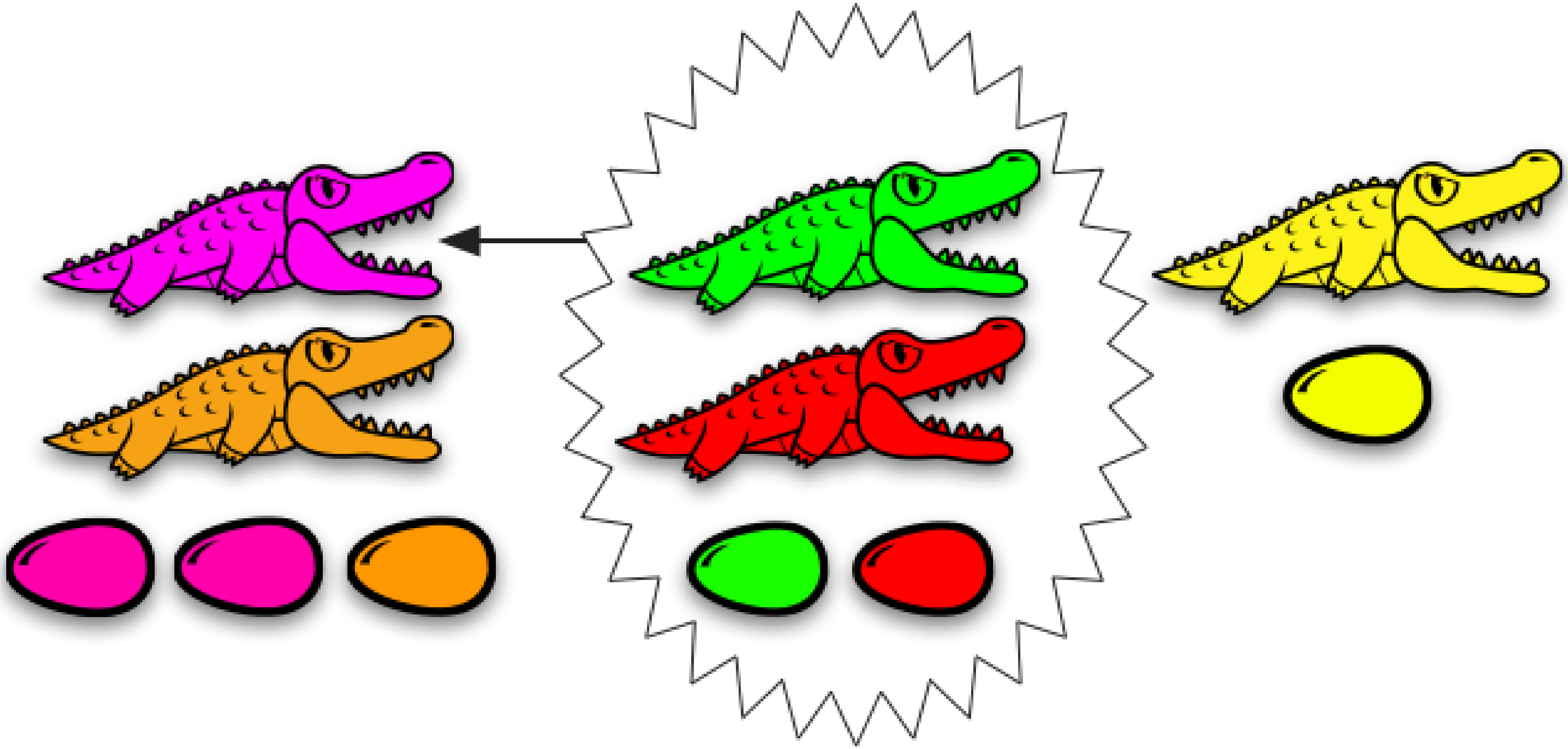
Child's Play—Smalltalk



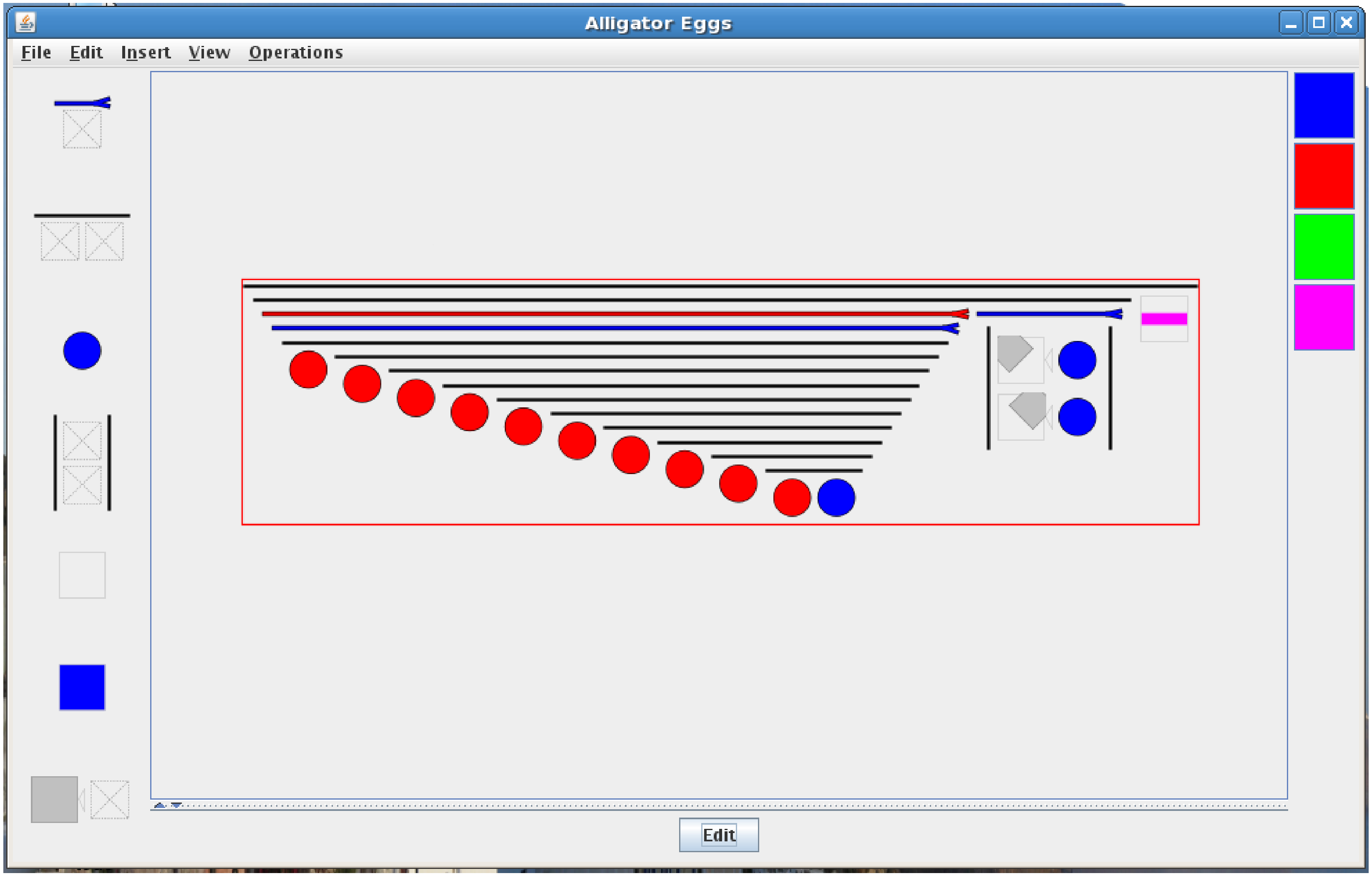
Child’s Play—Scatch



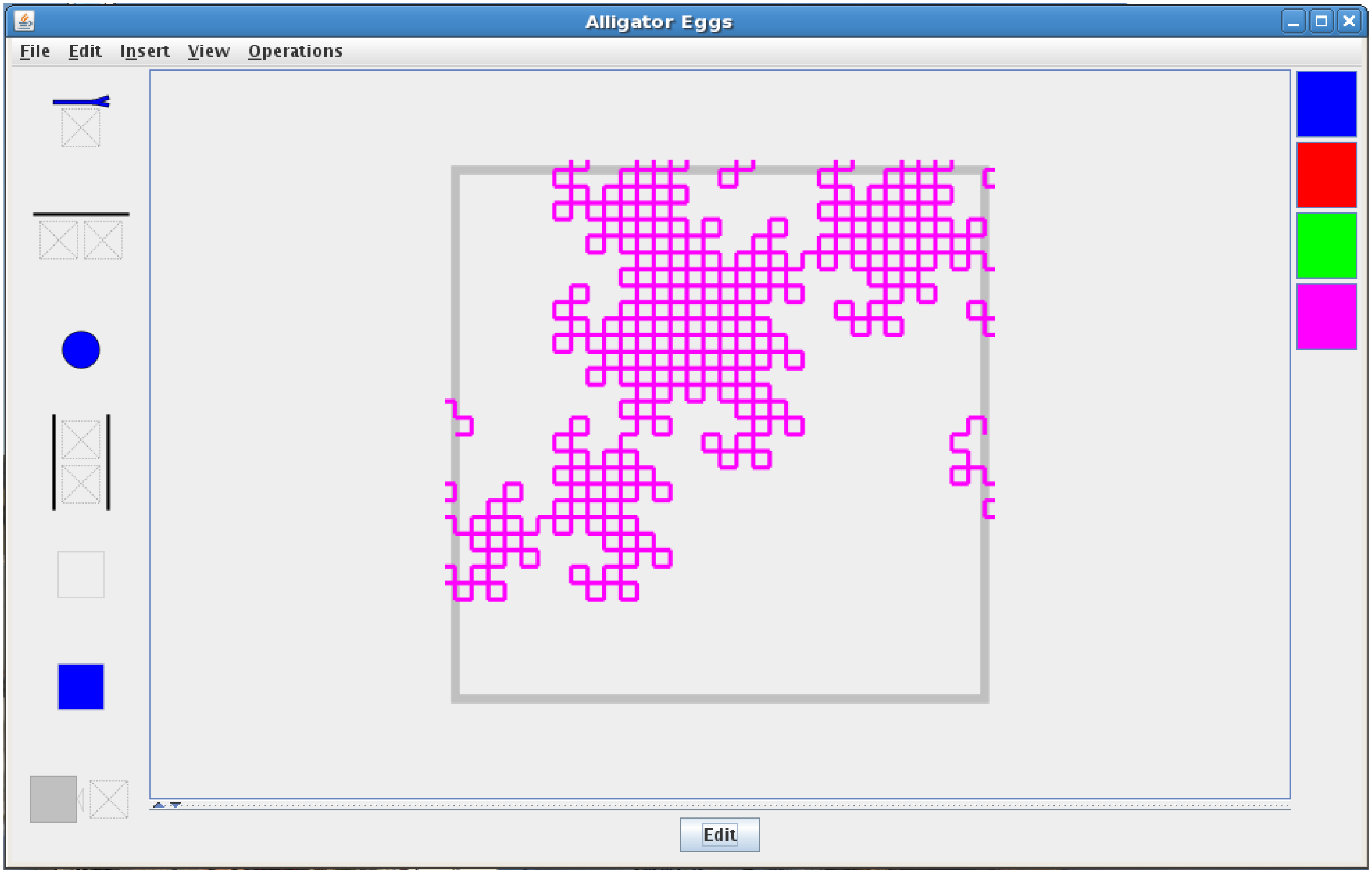
Child's Play—Bret Victor's Alligator Eggs



Child's Play—Alligator Eggs App



Child's Play—Alligator Eggs App

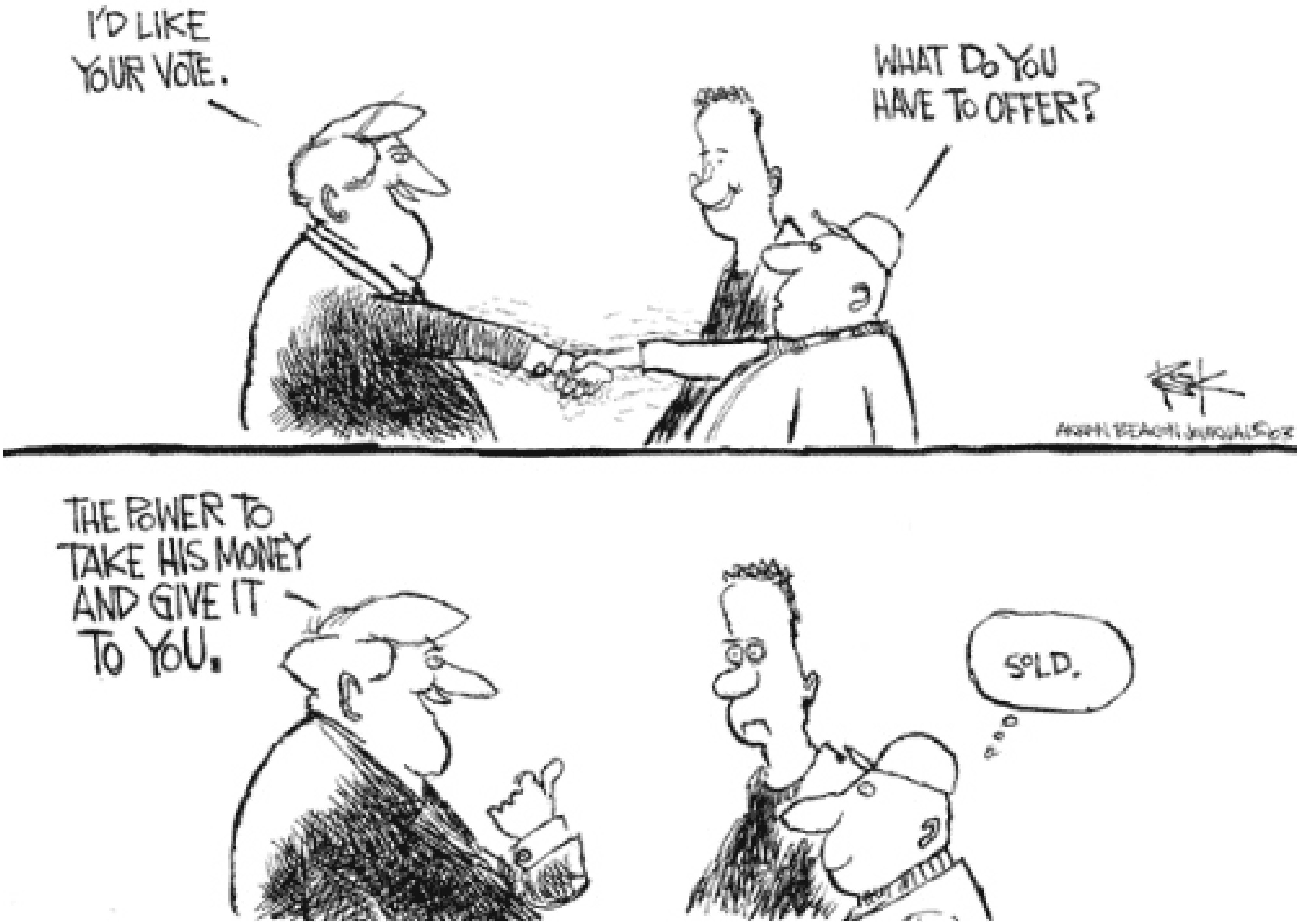


Child's Play—Alligator Eggs Video

Alligators, revisited

An implementation of Bret Victor's Oh no! Alligators! by Torsten Strobl.

Politics


I'D LIKE YOUR VOTE.
WHAT DO YOU HAVE TO OFFER?
THE POWER TO TAKE HIS MONEY AND GIVE IT TO YOU.
SOLD.

Politics

SIGPLAN

"To explore programming language concepts and tools focusing on design, implementation and efficient use."

Home | Membership | Conferences | Awards | What's New

Resources | Hot Links | Conference Calendar | Student Information | Contact Us

SIGPLAN Executive Committee 2009-2012

The Executive Committee is elected every 3 years by the members of ACM SIGPLAN.

Elected members

- Chair: Philip Wadler
- Vice Chair: Graham Hutton
- Secretary: Andrew P. Black
- Treasurer: Cristina Cifuentes
- Members at Large:
  - Matthew Flatt
  - Dan Grossman
  - Tony Hosking
  - Erez Petrank
  - Benjamin Zorn
- Past Chair: Kathleen Fisher

Ex-officio members

- ACM Program Director: Ginger Ignatoff
- Editors of SIGPLAN Newsletters:
  - SIGPLAN Information Director: Jack Davidson
  - SIGPLAN Notices: Mark Bailey
  - FORTRAN Forum: Ian Chivers
- ACM TOPLAS Editors:
  - Kathryn McKinley
  - Keshav Pingali
- Steering Committee Chairs
  - FOOL
    Christopher Stone
  - Haskell
    Daan Leijen
  - ICFP
    Ralf Hinze

Politics

The ACM Special Interest Group on Software Engineering provides a forum for computing professionals from industry, government and academia to examine principles, practices, and new research results in software engineering.

Join Us | Contact Us | Search

The Impact Project: Publications

2008

- Dieter Rombach, Marcus Ciolkowski, Ross Jeffery, Oliver Laitenberger, Frank McGarry, Forrest Shull. *Impact of research on practice in the field of inspections, reviews and walkthroughs: learning from successful industrial uses*. In *ACM SIGSOFT Software Engineering Notes* Volume 33, Issue 6 (November 2008). 26-35.
- Wolfgang Emmerich, Mikio Aoyama, Joe Sventek. *The Impact of Research on the Development of Middleware Technology*. *ACM Trans. Softw. Eng. Methodol.* 17, 4 (Aug. 2008), 1-48.
- Leon J. Osterweil, Carlo Ghezzi, Jeff Kramer, Alexander L. Wolf. *Determining the Impact of Software Engineering Research on Practice* . *IEEE Computer*, Volume 41, Issue 3, March 2008 Page(s):39 - 49.

2003

- Barbara G. Ryder, Mary Lou Soffa, "*Influences on the design of exception handling ACM SIGSOFT project on the impact of software engineering research on programming language design*". ACM SIGSOFT Software Engineering Notes, Volume 28 , Issue 4 (July 2003) - *pdf*