What are the important problems for programming languages?

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

Hamming

- Los Alamos, 1945.
- 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 you’ve 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
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

Programming Erlang

Software for a Concurrent World

Joe Armstrong
Distribution and multicore

Join Calculus

JoCaml, Polyphonic C#

\[
P \overset{\text{def}}{=} x<\nu> \\
def D \text{ in } P \\
P|P
\]

\[
J \overset{\text{def}}{=} x<\nu> \\
J|J
\]

\[
D \overset{\text{def}}{=} J \triangleright P \\
D \land D
\]
Programming the web
Programming the web
Programming the web

Links

Browser
(HTML, XML, JavaScript)

Server
(Java, Perl, PHP, Python, Ruby)

Database
(SQL, XQuery)
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

\[
\begin{array}{ccc}
T^3A & \xrightarrow{\mu TA} & T^2A \\
T\mu_A & & \downarrow \mu_A \\
T^2A & \xrightarrow{\mu_A} & TA
\end{array}
\quad
\begin{array}{ccc}
TA & \xrightarrow{\eta TA} & T^2A \\
\eta_{TA} & & \downarrow\mu_A \\
T^2A & \xleftarrow{T\eta A} & TA
\end{array}
\]
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:

\[
\begin{align*}
\text{infixl} & \ 9 \ (\langle \% \rangle) \\
\text{class} & \ \text{Idiom} \ i \ \text{where} \\
\text{idi} & \ :: \ \ x \ \rightarrow \ i \ x \\
\text{⟨⟨%⟩⟩} & \ :: \ \ i \ (s \ \rightarrow \ t) \ \rightarrow \ i \ s \ \rightarrow \ i \ t \quad \text{— pronounced ‘apply’}
\end{align*}
\]
Programming the web

The Essence of Form Abstraction*

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

School of Informatics, University of Edinburgh

module type Idiom = sig
  type α t
  val pure : α → α t
  val (⊗) : (α → β) t → α t → β t
end

module type FORMLET = sig
  include Idiom
  val xml : xml → unit t
  val text : string → unit t
  val tag : tag → attrs → α t → α t
  val input : string t
  val run : α t → xml × (env → α)
end

Fig. 4. The idiom and formlet interfaces
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 : \(\text{string} \times \text{date} \times \text{date}\) 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 WebShaper™ 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 WebShaper™ 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 WebShaper™ bindings!

---

**Formlets**

Create interactive forms with validation using type-safe code in just lines!
Static and dynamic types

![Graph showing Tiobe Programming Paradigm Index: Type system from 2002 to 2010, comparing statically typed and dynamically typed languages. The statically typed line is in red, and the dynamically typed line is in blue. The graph indicates a trend where statically typed languages have increased in popularity, while dynamically typed languages have decreased.]
Static and dynamic types

Blame for All

Amal Ahmed¹, Robert Bruce Findler², Jacob Matthews³, and Philip Wadler⁴

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

\[
\sigma \triangleright (A' \rightarrow B') \iff A \rightarrow B|^p \nu \longmapsto \sigma \triangleright \lambda x : A'. (B' \iff B)^|^p \ (\nu ((A \iff A')^\overline{p} \ x))
\]

\[
\sigma \triangleright (\Lambda X. t) A \longmapsto \sigma, X := A \triangleright t
\]

\[
\sigma \triangleright (B \iff \forall X. A)^|^p \nu \longmapsto \sigma \triangleright (B \iff A[X := \ast])^|^p \ (\nu \ast)
\]

\[
\sigma \triangleright (\forall X. B \iff A)^|^p \nu \longmapsto \sigma \triangleright \Lambda X. (B \iff A)^|^p \nu
\]
Static and dynamic types

\[ \sigma \triangleright (\forall X.t) A \mapsto \sigma, X := A \triangleright t \]
if \( X \notin \text{dom}(\sigma) \)

\[ \sigma \triangleright (B \leftarrow \forall X.A)^p \ v \mapsto \sigma \triangleright (B \leftarrow A[X := *)^p) \ (v *) \]
if \( X \notin \text{dom}(\sigma) \)

\[ \sigma \triangleright (\forall X.B \leftarrow A)^p \ v \mapsto \sigma \triangleright \forall X. (B \leftarrow A)^p \ v \]
if \( X \notin \text{dom}(\sigma) \)

\[ \sigma \triangleright (*) \leftarrow G)^p \ v \text{ is }^q G \mapsto \sigma \triangleright \text{true} \]
if \( G \neq X \)

\[ \sigma \triangleright (*) \leftarrow G)^p \ v \text{ is }^q H \mapsto \sigma \triangleright \text{false} \]
if \( G \neq X \text{ and } G \neq H \)

\[ \sigma \triangleright (*) \leftarrow X)^p \ v \text{ is }^q H \mapsto \text{blame } q \]

\[ \sigma \triangleright (\lambda x:A.t) \ v \mapsto \sigma \triangleright t[x := v] \]

\[ \sigma \triangleright (A' \rightarrow B') \leftarrow A \rightarrow B)^p \ v \mapsto \sigma \triangleright \lambda x:A'. (B' \leftarrow B)^p \ (v (\langle A \leftarrow A'\rangle^p x)) \]

\[ \sigma \triangleright (*) \leftarrow *)^p \ v \mapsto \sigma \triangleright v \]

\[ \sigma \triangleright (1 \leftarrow 1)^p \ v \mapsto \sigma \triangleright v \]

\[ \sigma \triangleright (X \leftarrow X)^p \ v \mapsto \sigma \triangleright v \]

\[ \sigma \triangleright (*) \leftarrow A \rightarrow B)^p \ v \mapsto \sigma \triangleright (* \leftarrow *)^p \ (* \leftarrow * \leftarrow A \rightarrow B)^p \ v \]
if \( A \rightarrow B \neq * \rightarrow * \)

\[ \sigma \triangleright (A \rightarrow B \leftarrow *)^p \ v \mapsto \sigma \triangleright (A \rightarrow B \leftarrow * \rightarrow *)^p \ (\langle * \rightarrow * \leftarrow * \rangle^p \ v \] if \( A \rightarrow B \neq * \rightarrow * \)

\[ \sigma \triangleright (G \leftarrow *)^q \ (\langle * \leftarrow G)^p \ v \mapsto \sigma \triangleright v \]

\[ \sigma \triangleright (H \leftarrow *)^q \ (\langle * \leftarrow G)^p \ v \mapsto \text{blame } q \]
if \( G \neq H \)
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)
(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

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

<table>
<thead>
<tr>
<th></th>
<th>Language</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>ActionScript</td>
<td>0.519%</td>
</tr>
<tr>
<td>22</td>
<td>COBOL</td>
<td>0.430%</td>
</tr>
<tr>
<td>23</td>
<td>Transact-SQL</td>
<td>0.412%</td>
</tr>
<tr>
<td>24</td>
<td>FoxPro/xBase</td>
<td>0.379%</td>
</tr>
<tr>
<td>25</td>
<td>Fortran</td>
<td>0.376%</td>
</tr>
<tr>
<td>26</td>
<td>Logo</td>
<td>0.361%</td>
</tr>
<tr>
<td>27</td>
<td>Scratch</td>
<td>0.346%</td>
</tr>
<tr>
<td>28</td>
<td>Alice</td>
<td>0.329%</td>
</tr>
<tr>
<td>29</td>
<td>Ada</td>
<td>0.305%</td>
</tr>
<tr>
<td>30</td>
<td>S-lang</td>
<td>0.292%</td>
</tr>
<tr>
<td>31</td>
<td>RPG (OS/400)</td>
<td>0.285%</td>
</tr>
<tr>
<td>32</td>
<td>Erlang</td>
<td>0.261%</td>
</tr>
<tr>
<td>33</td>
<td>PowerShell</td>
<td>0.259%</td>
</tr>
<tr>
<td>34</td>
<td>Scala</td>
<td>0.244%</td>
</tr>
<tr>
<td>35</td>
<td>Awk</td>
<td>0.243%</td>
</tr>
<tr>
<td>36</td>
<td>Prolog</td>
<td>0.234%</td>
</tr>
<tr>
<td>37</td>
<td>NXT-G</td>
<td>0.234%</td>
</tr>
<tr>
<td>38</td>
<td>Tcl/Tk</td>
<td>0.221%</td>
</tr>
<tr>
<td>39</td>
<td>Focus</td>
<td>0.209%</td>
</tr>
<tr>
<td>40</td>
<td>LabWindows/CVI</td>
<td>0.193%</td>
</tr>
<tr>
<td>41</td>
<td>Haskell</td>
<td>0.183%</td>
</tr>
<tr>
<td>42</td>
<td>PL/I</td>
<td>0.177%</td>
</tr>
<tr>
<td>43</td>
<td>JavaFX Script</td>
<td>0.176%</td>
</tr>
<tr>
<td>44</td>
<td>MAX/MSP</td>
<td>0.161%</td>
</tr>
<tr>
<td>45</td>
<td>LabVIEW</td>
<td>0.157%</td>
</tr>
<tr>
<td>46</td>
<td>Falcon</td>
<td>0.156%</td>
</tr>
<tr>
<td>47</td>
<td>Groovy</td>
<td>0.152%</td>
</tr>
<tr>
<td>48</td>
<td>Modula-3</td>
<td>0.146%</td>
</tr>
<tr>
<td>49</td>
<td>Forth</td>
<td>0.144%</td>
</tr>
<tr>
<td>50</td>
<td>Smalltalk</td>
<td>0.135%</td>
</tr>
</tbody>
</table>
Empiricism
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

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.
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.

The Impact Project: Publications

2008


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