10 Papers

(really fast)

Jim Weirich
Chief Scientist / EdgeCase
jim@edgecase.com
@jimweirich

EdgeCase
software artisans

10 Papers Every Programmer Should Read (At Least Twice)

-- Michael Feathers, Feb 26, 2009

It's hard to find deeply technical books and articles which stand the test of time in software: they are all Latin within 20 years.

To me, these are classic papers which contain deep "things you oughta know" about code – the material you work with.

Is denotional programming the next big thing?

The Next 700 Programming Languages

P. J. Landin

(1965)

... today ... 1,700 special programming languages used to 'communicate' in over 700 application areas.

-- Computer Software Issues, an American Mathematical Association Prospectus

Introduces the ISWIM system of languages

ISWIM -- If you See What I Mean

Not a Single Language,
But a system of languages

Defining a Language

- Abstract Syntax
- Axiomatization
  - Primitives and Combinators
- Underlaying Abstract Machine

Abstract Syntax

- Publication Language
- Reference Language
- Hardware Language

Abstract Syntax

- Publication Language

$$x(x+a)$$
$$\textbf{where } x = b + 2c$$

- Reference Language

- Hardware Language

```
x*(x+a)
where x = b + 2*c
```

Abstract Syntax

- Publication Language

$$x(x+a)$$
$$\textbf{where } x = b + 2c$$

- Reference Language

- Hardware Language

```
x*(x+a)
WHERE x = b + 2*c
```

Abstract Syntax

- Publication Language

$$x(x+a)$$
$$\textbf{where } x = b + 2c$$

- Reference Language

- Hardware Language

```
x*(x+a)
'WHERE' x = b + 2*c
```

Sample Code

$x(x+a)$
where $x = b + 2c$

$f(b + 2c)$
where $f(x) = x(x + a)$

Compared to Lisp

- Break out of the List processing paradigm
- Stack based storage (rather than GC)
- Avoid hardware dependencies
- Infix Operators, Indentation Structure
- Break compromises with tradition

Compared to Lisp

- Break out of the List processing paradigm
- Stack based storage (rather than GC)
- Avoid hardware dependencies
- Infix Operators, Indentation Structure
- Break compromises with tradition

The author believes that expressive power should be by design rather than accident

Discussion

Three points

Denotional Programs

=

No Jumping

No Assignments

Declarative Programs

=

No Jumping

No Assignments

Imperative Languages impose unneeded ordering constraints on solutions

Long discussion of issues surrounding indentation as structure

What's the best way to modularize your programs?

On the Criteria to Be Used in Decomposing Systems into Modules

D. L. Parnas

(1972)

Let's Find Out!

KWIC

Key Word In Context

The quick brown fox.

The quick brown fox quick brown fox ... The brown fox ... The quick fox ... The quick brown

brown fox ... the quick
fox ... the quick brown
quick brown fox ... the
the quick brown fox

Small System

[...] such a system could be produced by a good programmer within a week or two.

Version 1

- Input
- Circular Shift
- Alphabetizing
- Output
- Master Control

Version 2

- Line Storage
- Input
- Circular Shifter
- Alphabetizer
- Output
- Master Control

Line Storage

- CHAR(r,w,c) -- The cth charcter of the wth word of the rth row.
- SETCHAR(r,w,c,d) -- Set the char
- WORDS(r) -- number of words in r

Circular Shifter

- CSSETUP() -- Initializaer
- CSCHAR(l,w,c) -- cth char of wth word of the lth circular shift

Induced Changes

- Input format
- Non-core line storage
- Different word packing strategies
- Lazy vs non-lazy alphabetization

Version 1 Effects

- Input format
- Non-core line storage
- Different word packing strategies
- Lazy vs non-lazy alphabetization

Version 1 Effects

1 Module Effected

- Input format
- Non-core line storage
- Different word packing strategies
- Lazy vs non-lazy alphabetization

Version 1 Effects

1 Module Effected

- Input format

All Modules Effected

- Non-core line storage
- Different word packing strategies
- Lazy vs non-lazy alphabetization

Version 2 Effects

- Input format
- Non-core line storage
- Different word packing strategies
- Lazy vs non-lazy alphabetization

Version 2 Effects

- Input format
- Non-core line storage
- Different word packing strategies
- Lazy vs non-lazy alphabetization

1 Module Effected

Programs should be modularized so to hide implementation information

Information Hiding

Information Hiding

$\neq$

Encapsulation

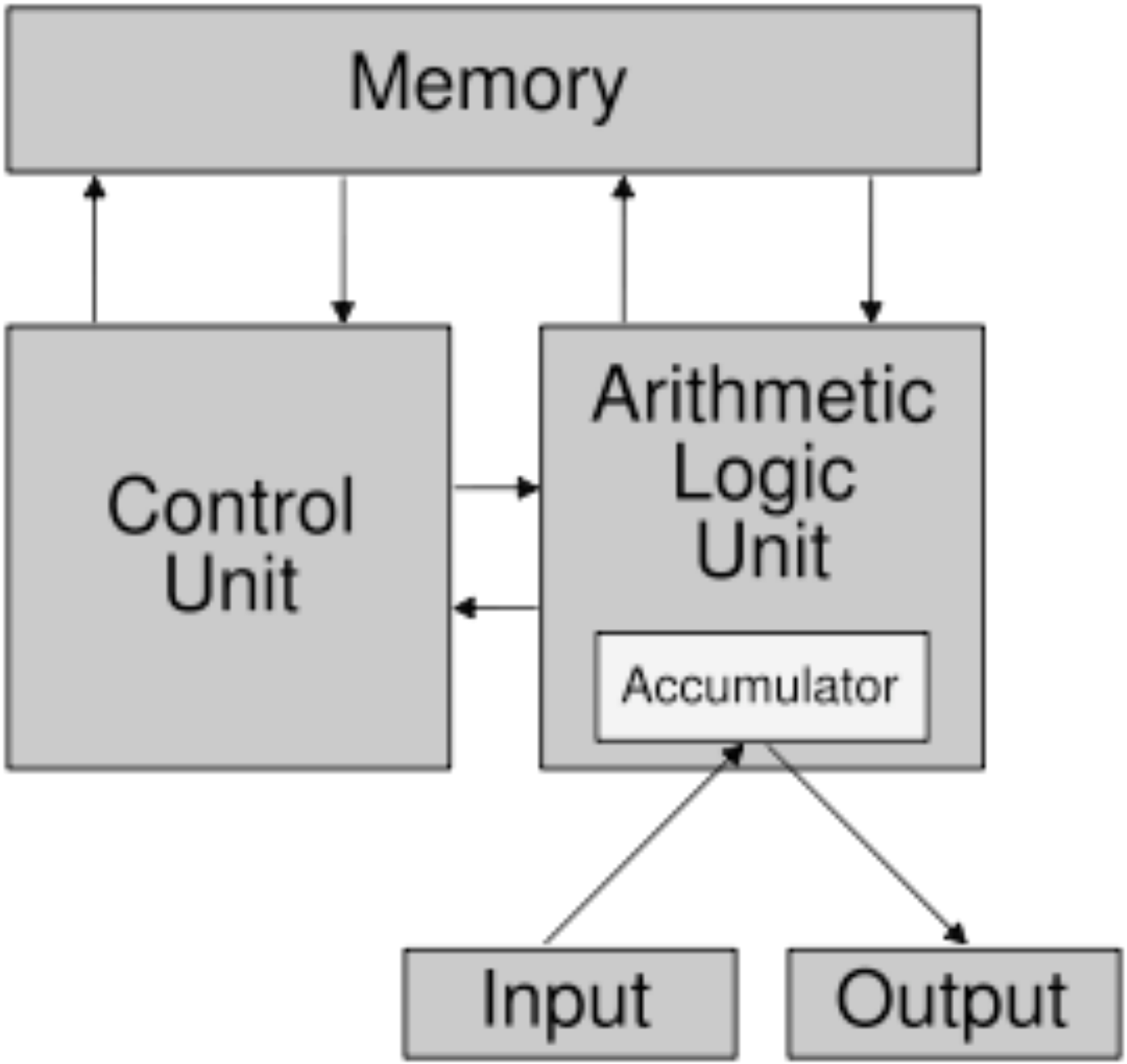
How is Programming like Algebra?

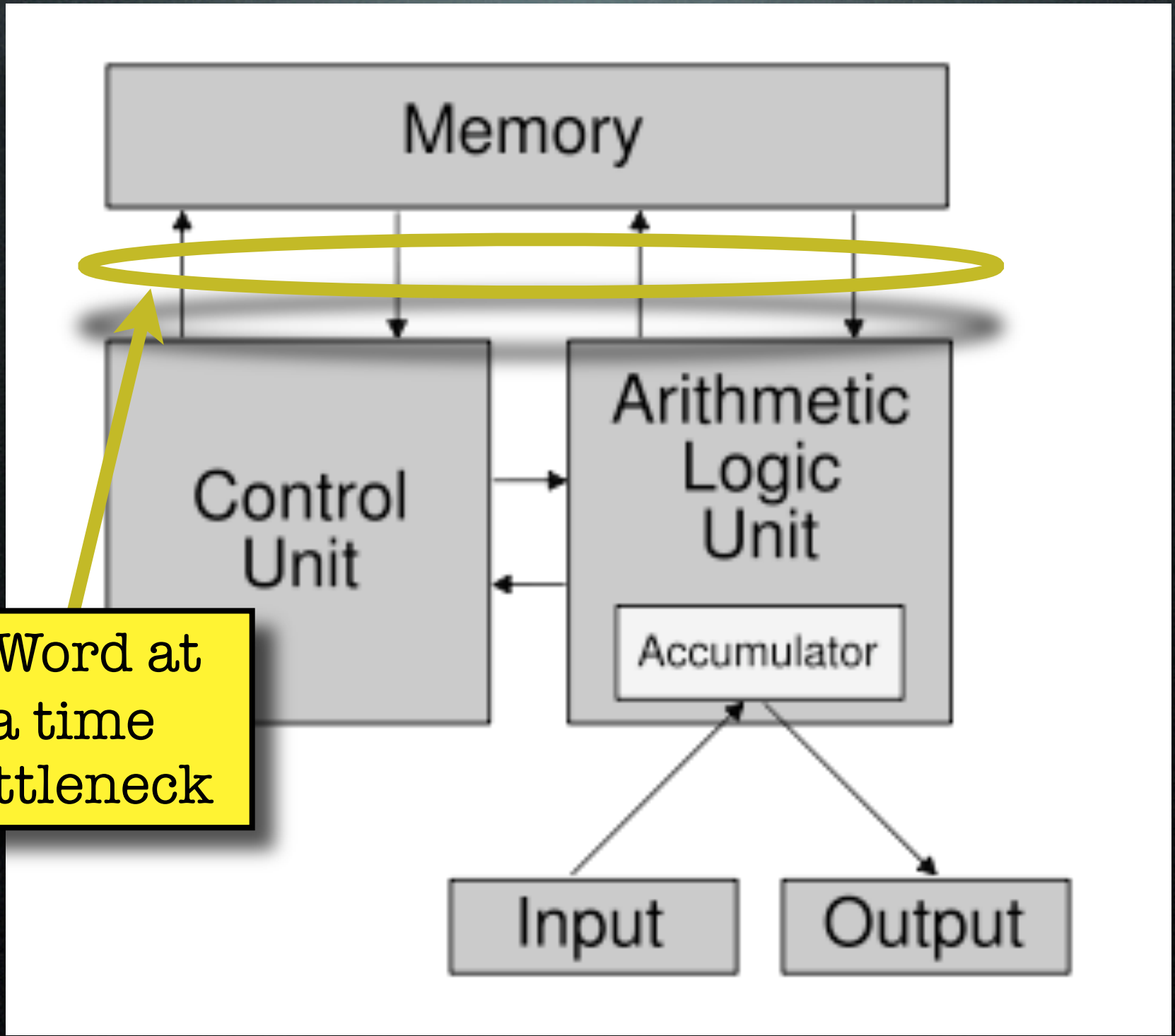
Can Programming Be Liberated from the von Neumann Style?

(A Functional Style and Its Algebra of Programs)

John Backus

(1978)


Memory
Control Unit
Arithmetic Logic Unit
Accumulator
Input
Output



1 Word at a time bottleneck

Languages are designed with this botttleneck in mind

```
c := 0
for i := 1 step 1 until n do
   c := c + a[i] * b[i]
```

Complaints

- Complex rules governing state
- Not hierarchical
- Dynamic and Repetitive
- Word at a time
- Data (i.e. n) is imbedded in the code
- Assumes arguments (only works for a/b)
- Scattered housekeeping words

$$\textbf{Def}\ \mathrm{IP} \equiv (/\ +) \circ (\alpha\ \times) \circ \mathrm{Trans}$$

Def IP ≡ (/ +)∘(α ×)∘Trans

- / -- Insert (Ruby's inject)
- α -- Apply to all
- ∘ -- Function Composition
- ×/+-- Multiplication/Addition
- Trans -- Transpose

Original Expression

IP:<<1,2,3>,<6,5,4>>

Definition of IP

IP:<<1,2,3>,<6,5,4>>

(/ +)∘(α ×)∘Trans:<<1,2,3>, <6,5,4>>

Definition of IP

IP:<<1,2,3>,<6,5,4>>

(/ +)∘(α ×)∘Trans:<<1,2,3>, <6,5,4>>

Function **Arguments**

Effect of Function Composition

IP:<<1,2,3>,<6,5,4>>

(/ +)∘(α ×)∘Trans:<<1,2,3>, <6,5,4>>

(/ +):((α ×):(Trans:<<1,2,3>, <6,5,4>>))

Effect of Function Composition

IP:<<1,2,3>,<6,5,4>>

(/ +)∘(α ×)∘Trans:<<1,2,3>, <6,5,4>>

(/ +):((α ×):(Trans:<<1,2,3>, <6,5,4>>))

Function **Arguments**

Effect of Function Composition

IP:<<1,2,3>,<6,5,4>>

(/ +)∘(α ×)∘Trans:<<1,2,3>, <6,5,4>>

(/ +):((α ×):(Trans:<<1,2,3>, <6,5,4>>))

Function

Arguments

Effect of Function Composition

IP:<<1,2,3>,<6,5,4>>

(/ +)∘(α ×)∘Trans:<<1,2,3>, <6,5,4>>

(/ +):((α ×):(Trans:<<1,2,3>, <6,5,4>>))

Function

Arguments

Applying transpose

IP:<<1,2,3>,<6,5,4>>

(/ +)∘(α ×)∘Trans:<<1,2,3>, <6,5,4>>

(/ +):((α ×):(Trans:<<1,2,3>, <6,5,4>>))

(/ +):((α ×):<<1,6>, <2,5>, <3,4>>)

Effect of “Apply to All” (α)

IP:<<1,2,3>,<6,5,4>>

(/ +)∘(α ×)∘Trans:<<1,2,3>, <6,5,4>>

(/ +):((α ×):(Trans:<<1,2,3>, <6,5,4>>))

(/ +):((α ×):<<1,6>, <2,5>, <3,4>>)

(/ +):<×:<1,6>, ×:<2,5>, ×:<3,4>>

Applying multiplication

IP:<<1,2,3>,<6,5,4>>

(/ +)∘(α ×)∘Trans:<<1,2,3>, <6,5,4>>

(/ +):((α ×):(Trans:<<1,2,3>, <6,5,4>>))

(/ +):((α ×):<<1,6>, <2,5>, <3,4>>)

(/ +):<×:<1,6>, ×:<2,5>, ×:<3,4>>

(/ +):<6, 10, 12>

Effect of Insert (/)

IP:<<1,2,3>,<6,5,4>>

(/ +)∘(α ×)∘Trans:<<1,2,3>, <6,5,4>>

(/ +):((α ×):(Trans:<<1,2,3>, <6,5,4>>))

(/ +):((α ×):<<1,6>, <2,5>, <3,4>>)

(/ +):<×:<1,6>, ×:<2,5>, ×:<3,4>>

(/ +):<6, 10, 12>

+:<6, +:<10, 12>>

Applying addition

IP:<<1,2,3>,<6,5,4>>

(/ +)∘(α ×)∘Trans:<<1,2,3>, <6,5,4>>

(/ +):((α ×):(Trans:<<1,2,3>, <6,5,4>>))

(/ +):((α ×):<<1,6>, <2,5>, <3,4>>)

(/ +):<×:<1,6>, ×:<2,5>, ×:<3,4>>

(/ +):<6, 10, 12>

+:<6, +:<10, 12>>

+:<6, 22>

Applying addition again

IP:<<1,2,3>,<6,5,4>>

(/ +)∘(α ×)∘Trans:<<1,2,3>, <6,5,4>>

(/ +):((α ×):(Trans:<<1,2,3>, <6,5,4>>))

(/ +):((α ×):<<1,6>, <2,5>, <3,4>>)

(/ +):<×:<1,6>, ×:<2,5>, ×:<3,4>>

(/ +):<6, 10, 12>

+:<6, +:<10, 12>>

+:<6, 22>

28

Compare

- No hidden state, work only on args
- Hierarchical, built from simpler elements
- Static, no need to execute it in your head
- Works on conceptual units, not words
- Works on any data of any size
- No named args, works on any vectors
- Employs general reusable housekeeping

(developes language primitives and data types)

(we are skipping this part)

Algebra of Programs

$$(a + b) \times c \equiv (a \times c + b \times c)$$

$$[f, g] \circ h \equiv [f \circ h, g \circ h]$$

I Composition and construction

I.1 $[f_1, \ldots, f_n] \circ g \equiv [f_1 \circ g, \ldots, f_n \circ g]$

I.2 $\alpha f \circ [g_1, \ldots, g_n] \equiv [f \circ g_1, \ldots, f \circ g_n]$

I.3 $/f \circ [g_1, \ldots, g_n]$

$\equiv f \circ [g_1, /f \circ [g_2, \ldots, g_n]]$ when $n \geq 2$

$\equiv f \circ [g_1, f \circ [g_2, \ldots, f \circ [g_{n-1}, g_n] \ldots]]$

$/f \circ [g] \equiv g$

I.4 $f \circ [\bar{x}, g] \equiv (\mathrm{bu}\ f\ x) \circ g$

I.5 $1 \circ [f_1, \ldots, f_n] \leq f_1$

$s \circ [f_1, \ldots, f_s, \ldots, f_n] \leq f_s$ for any selector s, $s \leq n$

defined$\circ f_i$ (for all $i \neq s$, $1 \leq i \leq n$) $\rightarrow\rightarrow$

$s \circ [f_1, \ldots, f_n] \equiv f_s$

Why is that important?

Optimization

Side Note:

$\perp$

$\perp \equiv$ **Bottom**

$$\langle A, \perp \rangle \equiv \perp$$

$$f{:}\langle A,\bot\rangle \equiv \bot$$

$$f{:}\bot \equiv \bot$$

Bottom Preserving

Mocking

FlexMock.undefined
is a bottom object

What works (and doesn't work) to make programs more reliable?

An Experimental Evaluation of the Assumption of Independence in Multi-Version Programming

John C. Knight & Nancy G Leveson

(1986)

By having multiple groups implement the same program independently, you get more reliable

Airbus A310 used dual programming for flap and slat control

The Experiment

- 27 Programs (+ one Gold Program)
- Anti-Missile system
- Highly debugged requirements
- Students got 15 test cases
- Acceptance test: 200 cases
- 1 Million test cases for analysis

Quality

- Quality of the programs was high
  - 6 - No Failures
  - 23 - Better than 99.9%
  - all - Better than 99%

Fault: Any program text that caused a failure

Some faults were non-correlated

E.g.
Not initializating a var,
Wrong array index

Some Faults **were** Correlated

E.g.
Comparing cosines of angles is not the same as comparing angle

E.g.
Incomplete case analysis of points in a straight line

The Correlations were statistically analyzed

99% Confidence that independent devel does not prevent common faults

How can we learn to given up global thinking while programming?

A Laboratory For Teaching Object-Oriented Thinking

Kent Beck & Ward Cunningham

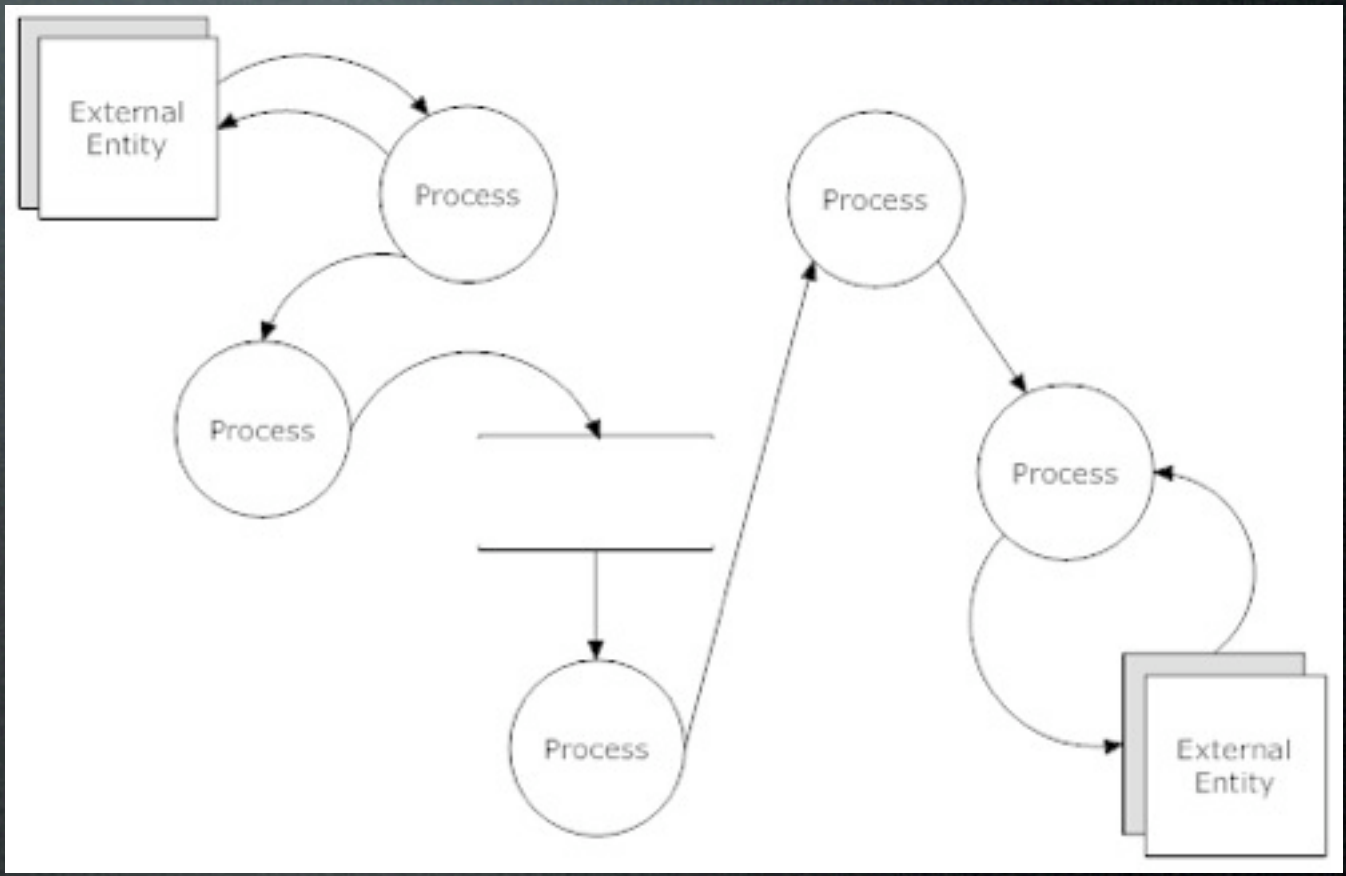
(1989)

The most difficult problem in teaching object- oriented programming is getting the learner to give up the global knowledge of control that is possible with procedural programs, and rely on the local knowledge of objects to accomplish their tasks.

Novice designs are littered with regressions to global thinking: gratuitous global variables, unnecessary pointers, and inappropriate reliance on the implementation of other objects.

Elements of Procedural Design

- Processes
- Data Flows
- Data Stores

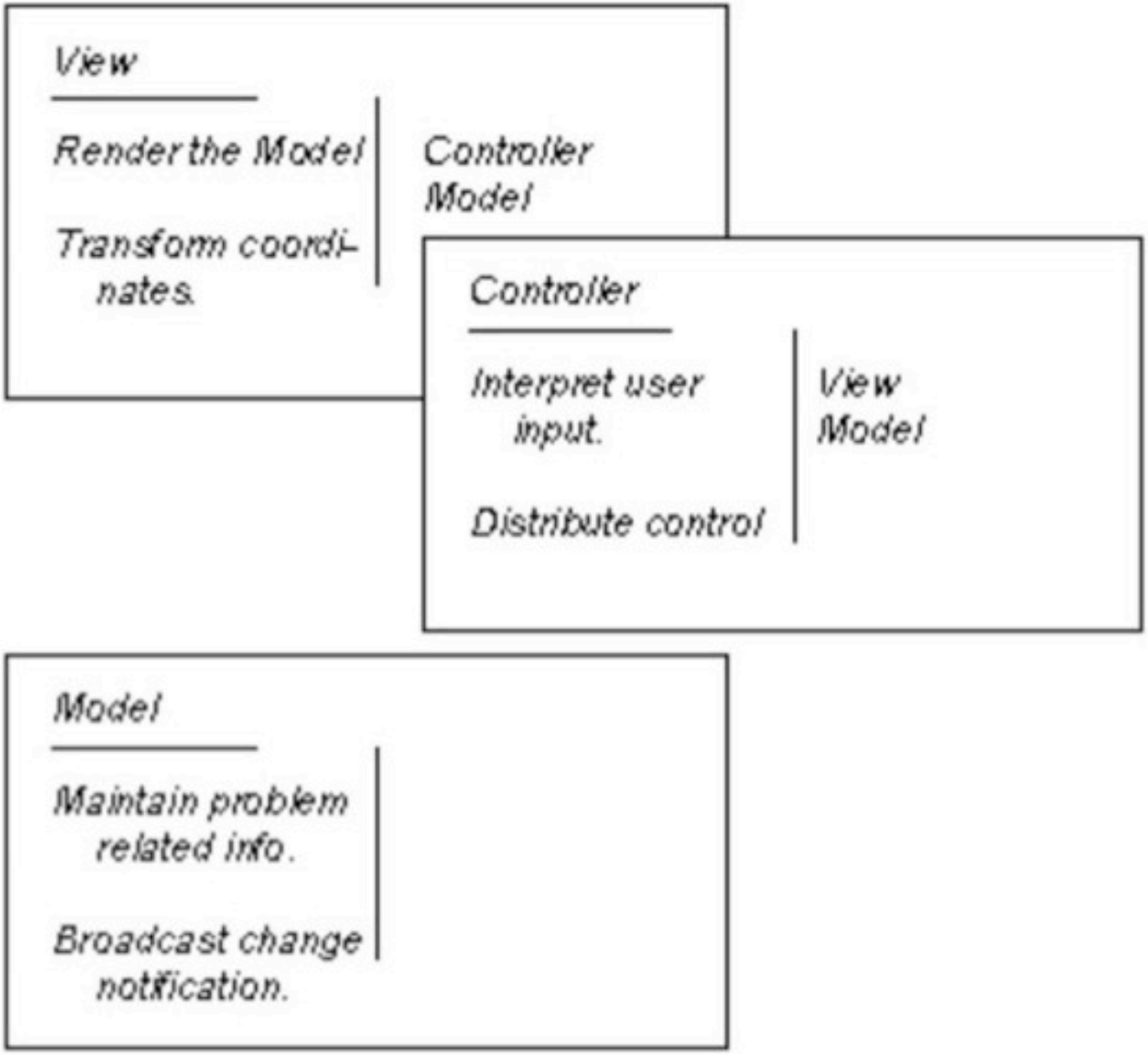


Elements of OO Design

- Class names -- Provides a vocabulary
- Responsibilities -- Problems to be solved
- Colaborators -- No Object is an Island

Class Name

Responsibilities	Colaborators
...	...


View
Render the Model
Transform coordi-
nates.
Controller
Model
Controller
Interpret user
input.
Distribute control
View
Model
Model
Maintain problem
related info.
Broadcast change
notification.

Problem:

School with processes for teaching and administration

Rephrased Problem:

Teacher, Janitor, Principal roles, colaborating to get things done.

We were surprised at the value of physically moving the cards around.

Reviewers watching a videotape of a design session...

Viewers had no trouble following the discussion ...

(even though they couldn't read the cards)

The most telling moments came when a viewer's explanation required that he point to a blurry card in the frozen image on the screen.

... in the switch to objects strong programmers become stronger, but weaker programmers are left behind.

-- Ragu Raghavan

Using cards ... even weaker programmers ... could contribute to object designs.

What are the ingredients for successful software?

Lisp: Good News, Bad News, How to Win Big

Richard P. Gabriel

(1991)

Richard Gabriel

- Noted Lisp Expert
- Lucid, Park Place Systems, Sun, IBM
- Author (4 tech books, 2 poetry books in the works)
- He is the lead guitarist in a rock 'n' roll band and a poet.

Status Report on Lisp

Standard is Progressing (Common Lisp)

Performance (comparable to C)

Environment

Integration (FORTRAN, C, Pascal)

Good Things

Most powerful, comprehensive, and pervasively object-oriented extension of any language

Ability to Deliver Solutions

So ...

Why isn't it accepted in the industry?

Two Design Approaches

Design Criteria

Simplicity

Correctness

Consistency

Completeness

MIT/Standford Approach

MIT/Standford Approach

Simplicity

Correctness

Consistency

Completeness

MIT/Standford Approach

Simplicity (interface)

Simplicity (impl)

Correctness

Consistency

Completeness

Do It **Right**

New Jersey Approach

New Jersey Approach

Simplicity (interface)

Simplicity (impl)

Correctness

Consistency

Completeness

Worse Is Better

Lisp & CLOS is
MIT / Stanford

Unix & C is
New Jersey

PC-Loser-ing

Worse is Better because ...

- Get a “Good Enough” solution early
- Get adopted everywhere
- Incrementally improve as needed

If you want to get adopted ...

Unix / C had it right

A wrong lesson is to take the parable literally and to conclude that C is the right vehicle for AI software. The 50% solution has to be basically right, and in this case it isn't.

Do distributed systems really work?

A Note on Distributed Computing

Jim Waldo, Geoff Wyant,
Ann Wollrath & Sam Kendall

(1994)

SOA? REST?
Corba? RPC? DCOM?

Objects
all the way down

Assumptions

- There is a single natural OO design for an app independent of how it is deployed.
- Failure and Performance issues are an implementation detail
- The interface of an object is independent of the context where it is used.

Four Issues

(1)
Latency

Ignoring the difference between the performance of local and remote invocations can lead to designs whose implementations are virtually assured of having performance problems ...

... because the design requires a large amount of communication between components that are in different address spaces and on different machines.

(2)
Memory Access

(3)
Partial Failure

A distributed system is a system where I can't get my work done because a computer has failed that I've never even heard of.

-- Leslie Lamport

Remote objects can simply “disappear”

(A) “Can you make remote method invocation look like local method invocation”

VS

(B) “What is the price of making remote method invocation identical to local method invocation?”

(A) Treats all objects as local ...

But can't handle partial failures

(B) Treats all objects as remote ...

But has additional (unneeded) overhead

(4)
Concurrency

No central locking/resource management

NFS

NFS

- Distributed File System built on a non-distributed API
- Soft-mounts
  - Endless tuning
- Hard-mounts
  - Programs hang until server is back
  - One server crashes and many workstations hang

Must think about the distribution model

It's not an implementation detail

Can dynamic languages really be fast?

Programming as an Experience

(the Inspiration for Self)

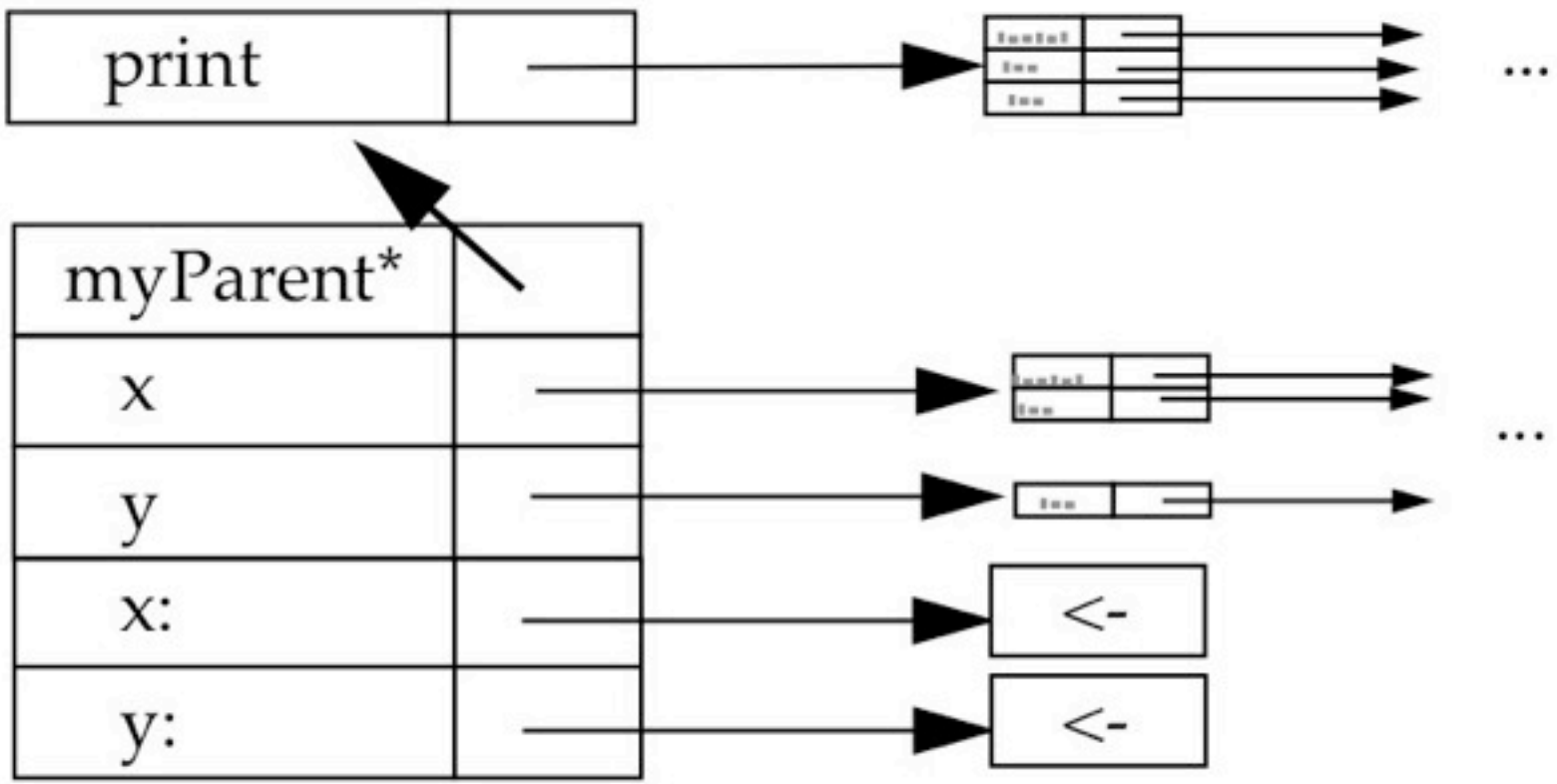
Randall B. Smith & David Ungar

(1995)

Summary of 10 years of Self Development

Prototyped Based

Slot Based



Programmers noticed that this static declarative information could be of great value in making a program more understandable.

[...]

a movement was born; using declarations both to benefit human readers and compilers.

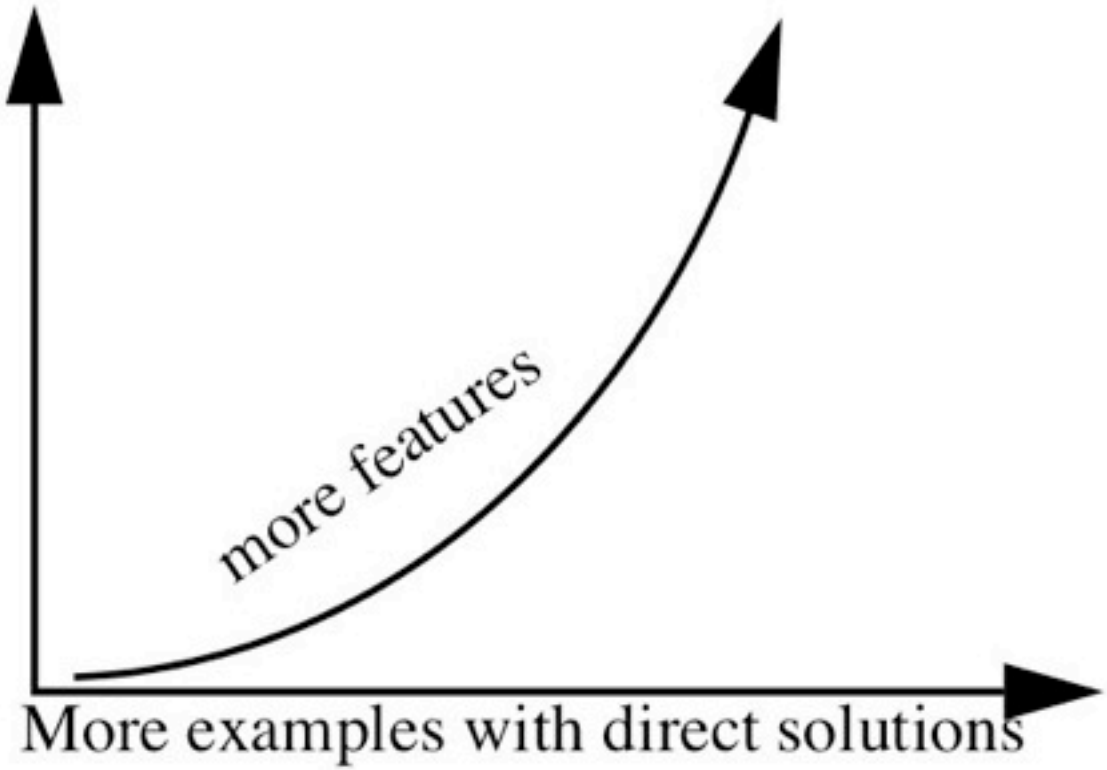
In our opinion, this trend has been a mixed blessing [...]

Most languages ... confuse these two issues [i.e. programmer readability and compile optimization]

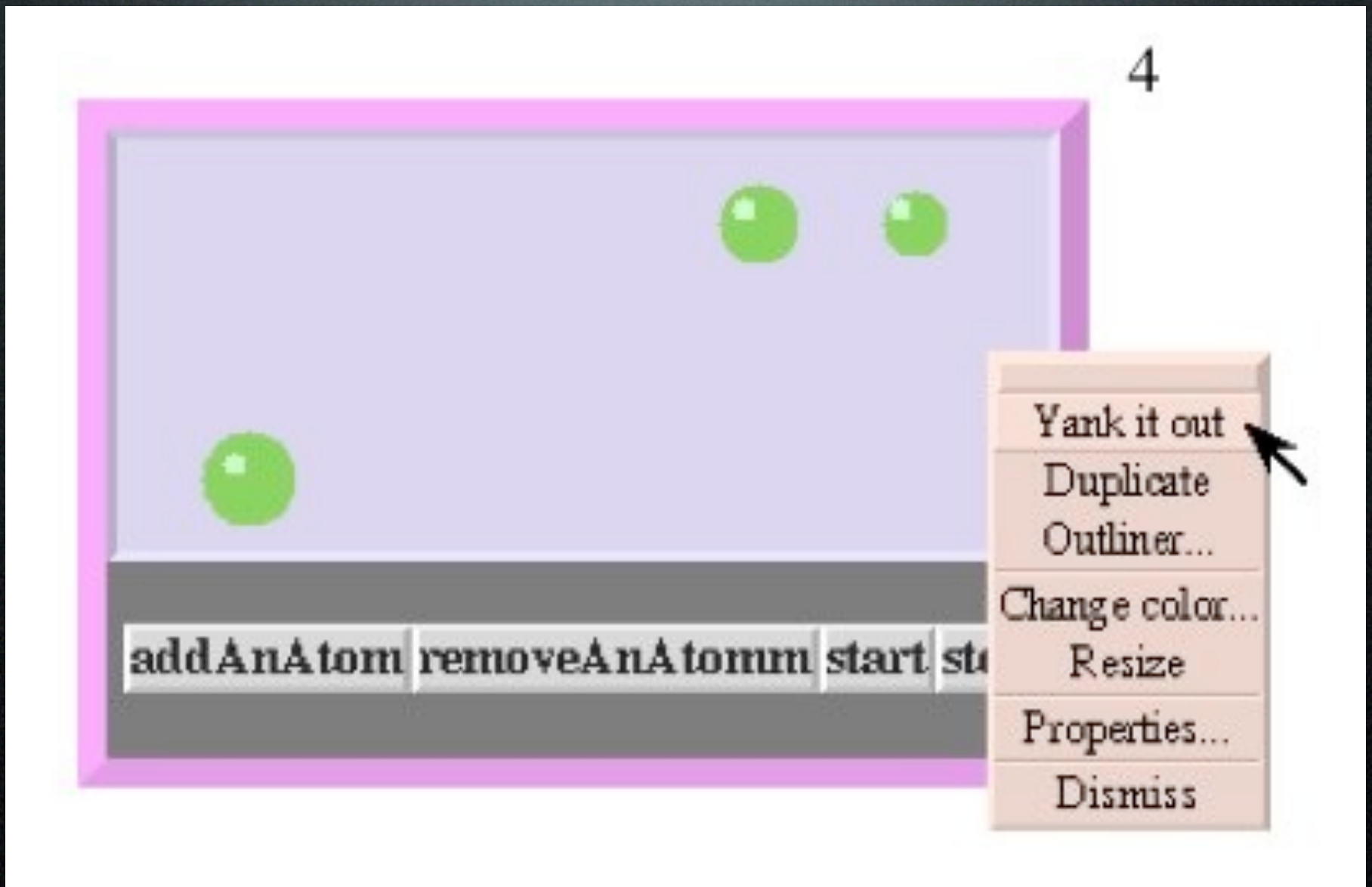
Minimalistic Language

Greater learning time

More policy decisions to make and revisit



Direct Manipulation


4
addAnAtom
removeAnAtomm
start
Yank it out
Duplicate
Outliner...
Change color...
Resize
Properties...
Dismiss

Fast Code

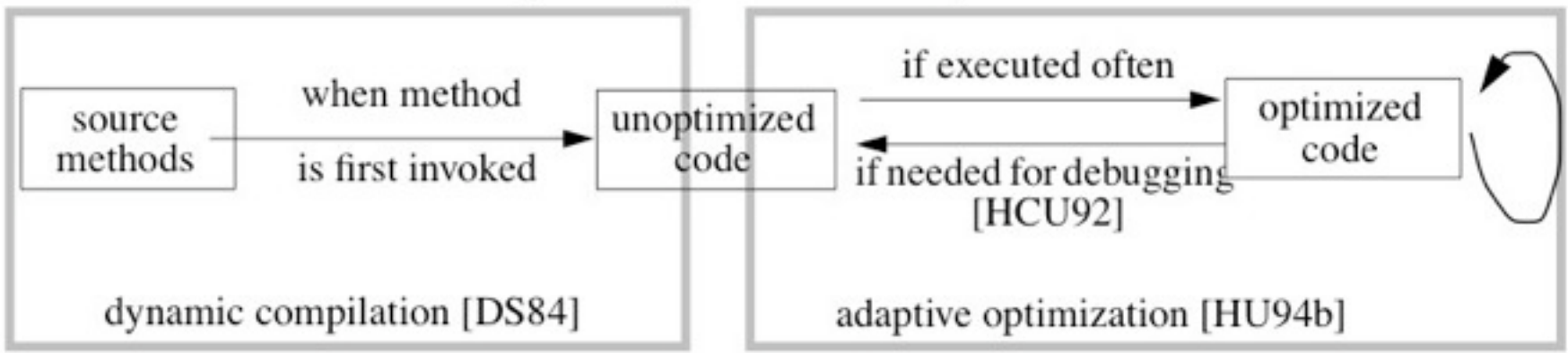


Figure 12. Compilation in the Self system (from [HU94b]).

Automatic Inlining

Effects of Self

- Javascript
- Java (Hot-Spot)
- Morphic (Squeak Smalltalk)

How can we improve the design of our object protocols?

Arguments and Results

James Noble

(1997)

6 Patterns for Method Arguments and Return values

How can you simplify a complex protocl that has a regular argument structures?

Arguments Object

```
class View
  def draw_rectangle(top_left, bottom_right, color)
  def fill_rectangle(top_left, bottom_right, color)
  def draw_oval(top_left, bottom_right, color)
end
```

Arguments Object

```
class Graphic
  def initialize(top_left, bottom_right, color)
end

class View
  def draw_rectangle(graphic)
  def fill_rectangle(graphic)
  def draw_oval(graphic)
end
```

How can you simplify a protocol where several messags differ mainly in their names?

Selector Object

```
class Graphic
  def initialize(top_left, bottom_right, color)
end

class View
  def draw_rectangle(graphic)
  def fill_rectangle(graphic)
  def draw_oval(graphic)
end
```

Selector Object

```
class Graphic
  def initialize(shape, top_left, bottom_right, color)
end

class View
  def draw(graphic)
end
```

How can you simplify an extremely complicated protocol?

Curried Object

```
font = Font.new('Times')
offset = font.height

view.draw_string("Hello", origin, font)
view.draw_string("World", origin+offset, font)
view.draw_string("!!!!", origin + 2*offset, font)
```

Curried Object

```
font = Font.new('Times')
pen = Pen.new(view, origin, font)

pen.draw_string("Hello")
pen.draw_string("World")
pen.draw_string("!!!!")
```

How can you store and manage a difficult answer to a difficult problem?

Result Object

```
m = MetricCalculator.new(an_object)

m.compute_size_of_interface
m.compute_number_of_inherited_methods
m.compute_number_of_ancestors
```

Result Object

```
metric_report =
 MetricCalculator.new(an_object).compute_metrics

metric_report.size_of_interface
metric_report.number_of_inherited_methods
metric_report.number_of_ancestors
```

How do you answer a question while you think about something else?

Future Object

```
metric_report =
  MetricCalculator.new(an_object).compute_metrics

# Go work on something else for a while ...

metric_report.size_of_interface
metric_report.number_of_inherited_methods
metric_report.number_of_ancestors
```

How can you answer a question that is easy to answer now, but tat may never be asked.

Lazy Object

```
metric_report =
  MetricCalculator.new(an_object).compute_metrics

if show_results?
  # defer calculation until the results are needed
  metric_report.size_of_interface
  metric_report.number_of_inherited_methods
  metric_report.number_of_ancestors
end
```

What does it mean to trust your software?

Reflections on Trusing Trust

Ken Thompson

(1984)

Stage I

Quine

```
_="_=%p;puts _%%_";puts _%_
```

Writing programs that write other programs is pretty straight forward

Writing programs that write (**themselves** and) other programs is pretty straight forward

Stage II

Learning Compilers

```
c = next();
if (c != '\\')
    return(c);
c = next();
if (c == '\\')
    return ('\\');
if (c == 'n')
    return('\n');
```

```
c = next();
if (c != '\\')
   return(c);
c = next();
if (c == '\\')
   return ('\\');
if (c == 'n')
   return('\n');
if (c == 'v')
   return ('\v');
```

```
c = next();
if (c != '\\')
    return(c);
c = next();
if (c == '\\')
    return ('\\');
if (c == 'n')
    return('\n');
if (c == 'v')
    return ('\v');
```

ERROR

```
c = next();
if (c != '\\')
    return(c);
c = next();
if (c == '\\')
    return ('\\');
if (c == 'n')
    return('\n');
if (c == 'v')
    return (11);
```

```
c = next();
if (c != '\\')
    return(c);
c = next();
if (c == '\\')
    return ('\\');
if (c == 'n')
    return('\n');
if (c == 'v')
    return ('\v');
```

```
c = next();
if (c != '\\')
    return(c);
c = next();
if (c == '\\')
    return ('\\');
if (c == 'n')
    return('\n');
if (c == 'v')
    return ('\v');
```

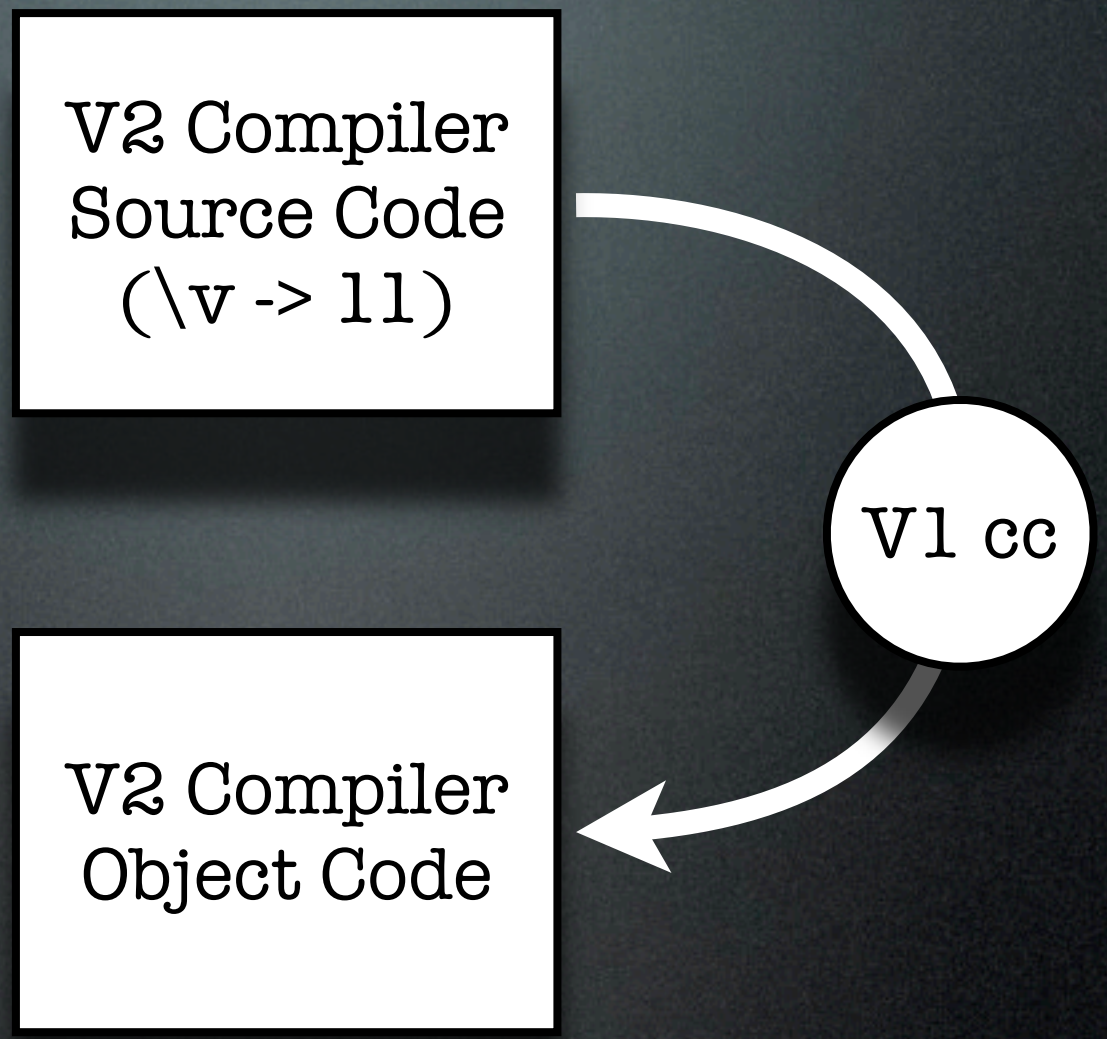
OK

V1 Compiler
Source Code
(no \v)

V1 cc

V2 Compiler
Source Code
(\v -> 11)

V1 cc

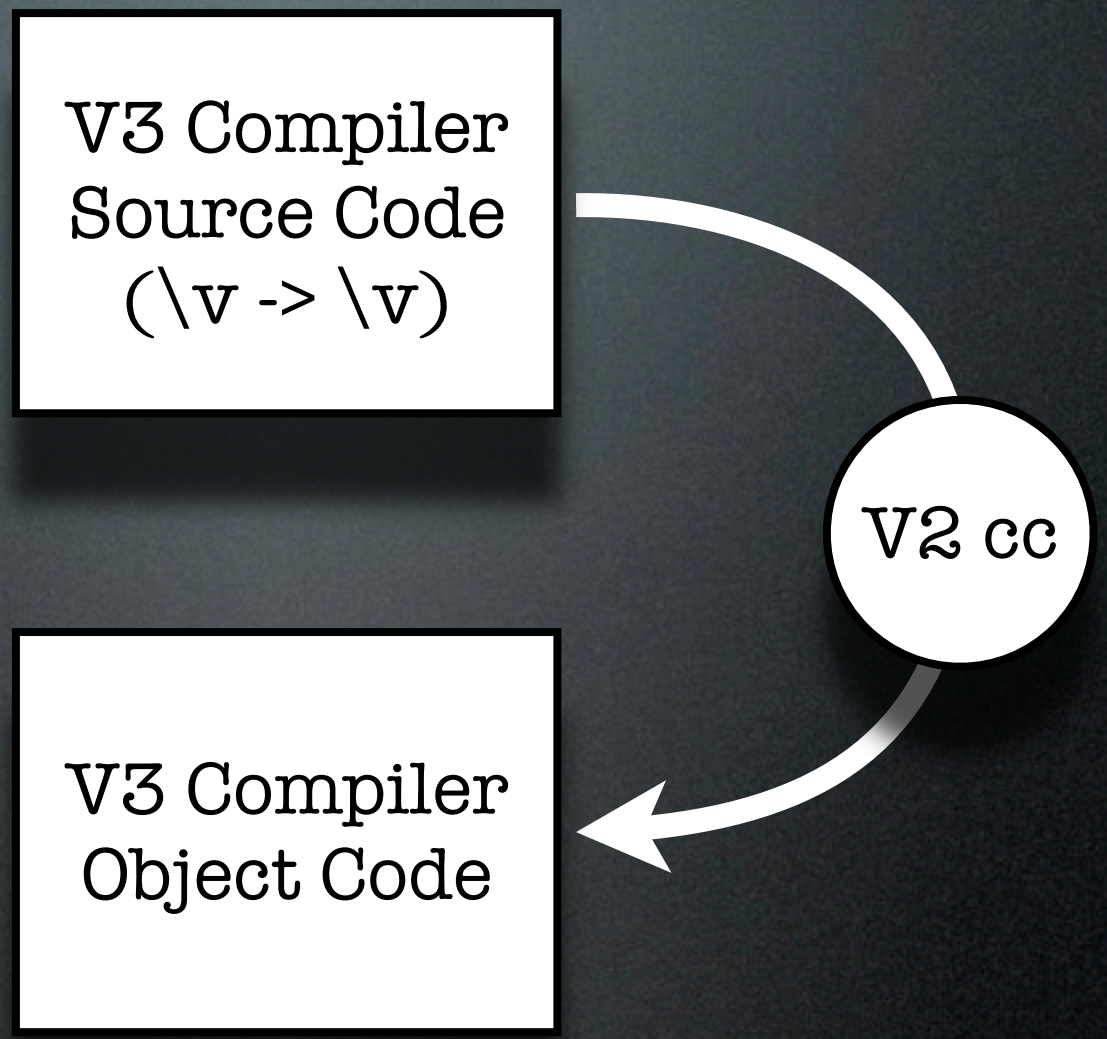

V2 Compiler
Source Code
(\v -> 11)
V1 cc
V2 Compiler
Object Code

V2 Compiler
Source Code
(\v -> 11)

V2 cc

V3 Compiler
Source Code
(\v -> \v)

V2 cc


V3 Compiler
Source Code
(\v -> \v)
V2 cc
V3 Compiler
Object Code

V3 Compiler
Source Code
(\v -> \v)

V3 cc

Stage III

Hacking Trust

```
compile(s)
char *s;
{
   if (match(s, “pattern”)) {
      compile(“bug”);
   }
   ...
}
```

```
compile(s)
char *s;
{
    if (match(s, "pattern")) {
        compile("bug");
    }
    ...
}
```

Matches Unix login()

```
compile(s)
char *s;
{
   if (match(s, "pattern")) {
      compile("bug");
   }
   ...
}
```

Matches Unix login()

Insert back door

```
compile(s)
char *s;
{
   if (match(s, “pattern1”)) {
      compile(“bug1”);
   }
   if (match(s, “pattern2”)) {
      compile(“bug2”);
   }
   ...
}
```

Insert backdoor into login()

```
compile(s)
char *s;
{
    if (match(s, "pattern1")) {
        compile("bug1");
    }
    if (match(s, "pattern2")) {
        compile("bug2");
    }
    ...
}
```

```
compile(s)
char *s;
{
   if (match(s, "pattern1")) {
      compile("bug1");
   }
   if (match(s, "pattern2")) {
      compile("bug2");
   }
   ...
}
```

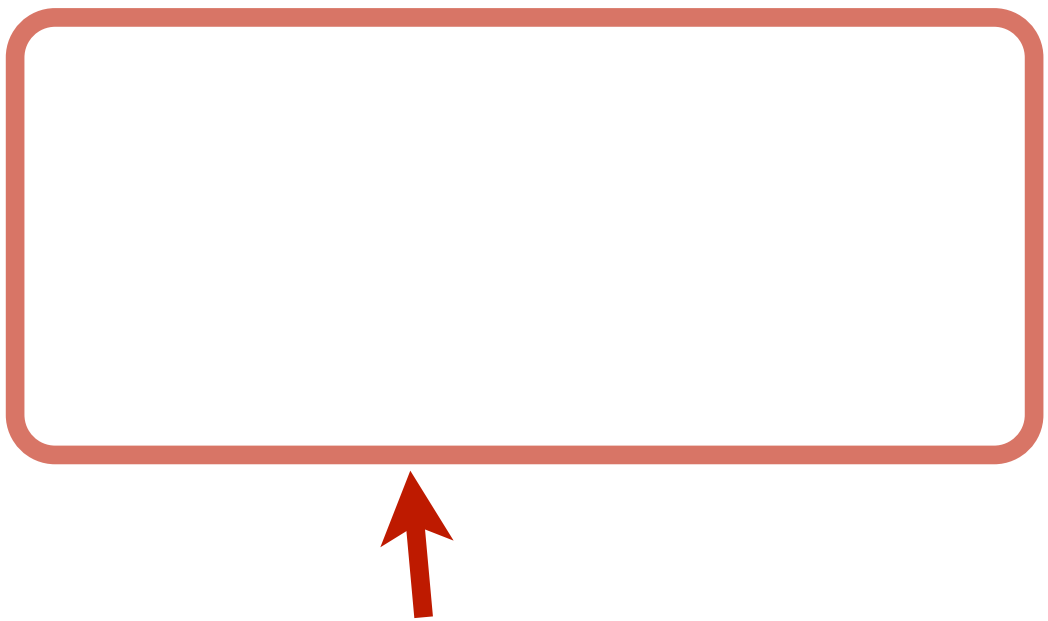
Insert backdoor into login()

Insert bug1 & bug2 into compiler

```
compile(s)
char *s;
{

    ...
}
```

```
compile(s)
char *s;
{
    ...
}
```



Source is gone
But existing compiler
continues to insert bugs

Moral

You can't trust code that you did not totally create yourself.

Time Line

* 1965 - Landin / DP
* 1972 - Parnas / IH
* 1978 - Backus / FP
* 1984 - Thompson / Trust
* 1986 - Knight / NVer
* 1989 - Beck / CRC
* 1991 - Gabriel / Lisp
* 1994 - Waldo / Dist
* 1995 - Ungar / Self
* 1997 - Noble / Args

Questions?

Thank You!

git://github.com/jimweirich/presentation_10papers.git

Copyright 2009 by Jim Weirich, Some Rights Reserved

creative commons

COMMONS DEED

Attribution-NonCommercial-ShareAlike 2.0