

NPRG075

History and philosophy of programming

Tomáš Petříček, 309 (3rd floor)

✉ petricek@d3s.mff.cuni.cz

➔ <https://tomasp.net> | [@tomaspetricek](#)

Lectures: Monday 12:20, S7

➔ <https://d3s.mff.cuni.cz/teaching/nprg075>







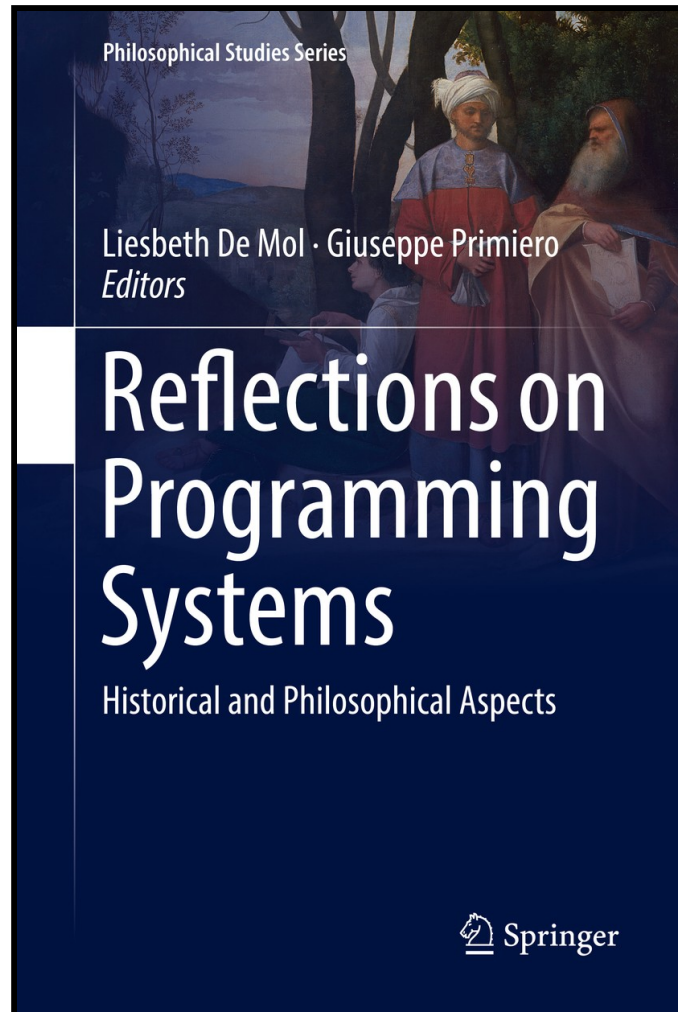
Philosophy of science

Why does it matter?

Philosophy of science

What can we learn about programming?

-  What designers assume and never question
-  How to understand odd designs of the past
-  What is the nature of programming concepts
-  What social forces shape programming



What do philosophers do?

Origins, languages, systems, correctness

How could it have gone differently?

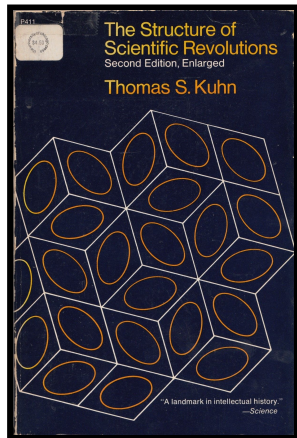
Reflections on ethics, politics, development

What if we took one aspect as primary?

Doing philosophy of programming

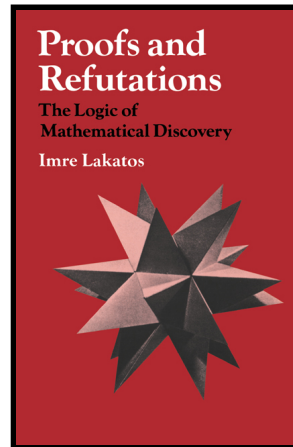
Methods

Try to explain how scientists think and work



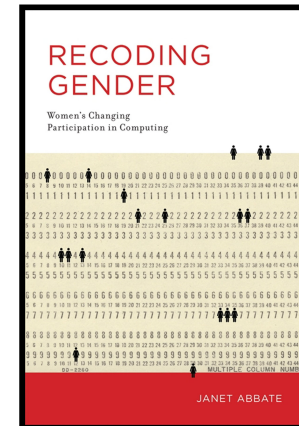
Entities

How concepts evolve & what are they?



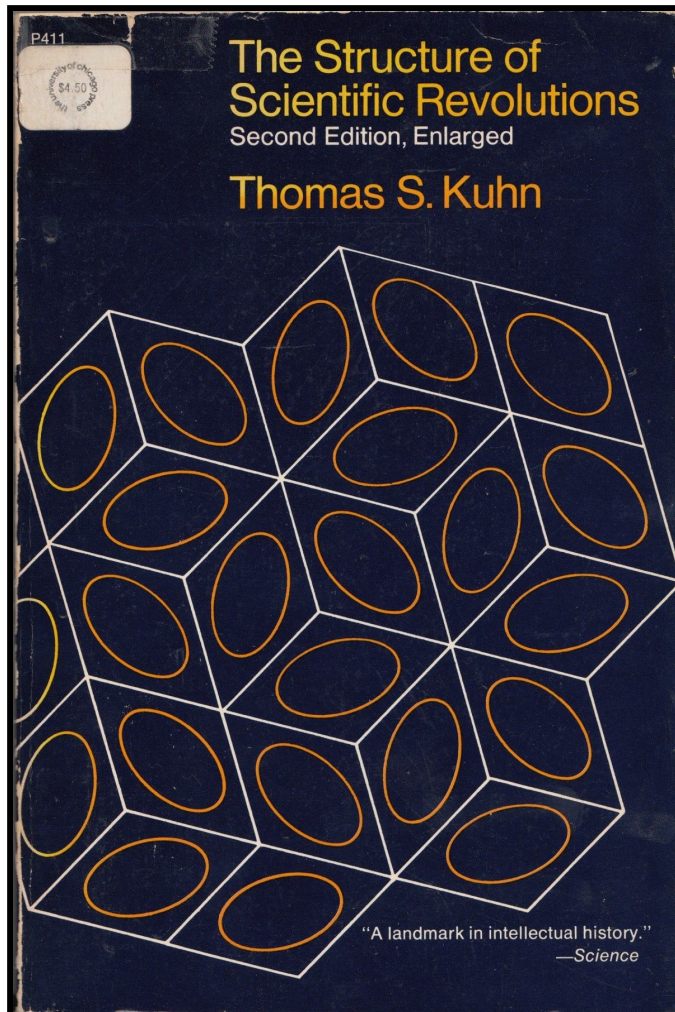
Social forces

How social aspects shape technology



Paradigm shifts

Classic philosophy of science



Scientific revolutions

Periods of normal science
disrupted by revolutions

New era with new
assumptions when the old
ways stop working

New incommensurable
with the old thinking

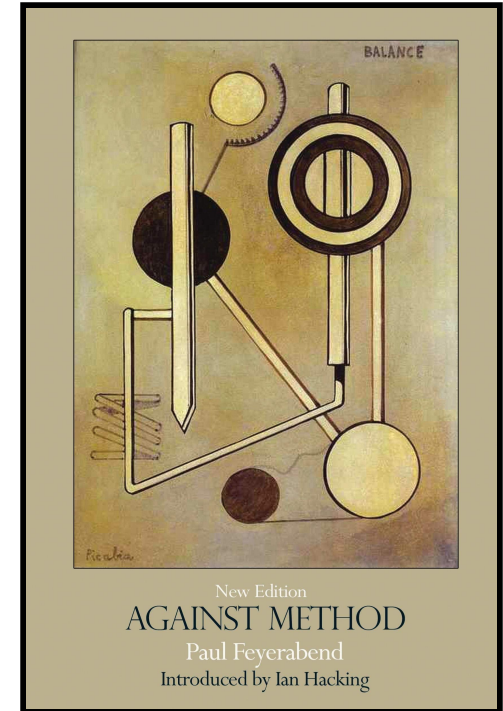
Philosophy of science

Research programmes (Lakatos)

- Groups of scientists share assumptions
- Explain failures by blaming secondary auxiliary assumptions

Against method (Feyerabend)

- No single rule explains science
- Hard to say what is reasonable!



Programming language revolution

(Gabriel, 2012)

From thinking about programming systems

Running, with evolving state, modified interactively

To thinking about programming languages

Relationships in static code

The Structure of a Programming Language Revolution

Richard P. Gabriel
IBM Research
Redwood City, California USA
rpg@us.ibm.com
dreamsongs.com

"I don't want to die in a language I can't understand."
— Jorge Luis Borges

Abstract

Engineering often precedes science. Incommensurability is real. **Categories and Subject Descriptors** A.0 [General] **General Terms** Design **Keywords** Engineering, science, paradigms, incommensurability

In 1990, two young and very smart computer scientists—Gilad Bracha and William Cook—wrote a pivotal paper called "Mixin-based Inheritance" [1], which immediately laid claim to being the first scientific paper on mixins. In that paper they described looking at Beta, Smalltalk, Flavors, and CLOS, and discovering a mechanism that could account for the three different sorts of inheritance found in these languages—including mixins from Flavors and CLOS. They named their new mechanism "mixins."

My attention was directed to this paper by Gilad Bracha himself when he told me in Brazil at AOSD in the spring of 2011 that most Lisp people who read the paper had strong objections to what he and William Cook had written about Lisp and CLOS.

That night I pulled the paper down from the ACM server and read it while outside enormous puffed clouds dwelled overhead, lit from beneath by the town of Porto de Galinhas on the Brazilian coast; the smells of burning sugarcane and bitter ocean pushed into my room.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. *Onward!* 2012, October 19–26, 2012, Tucson, Arizona, USA. Copyright © 2012 ACM 978-1-4503-1562-3/12/10...\$15.00.



Engineering_o A Path To Science

Engineers build things; scientists describe reality; philosophers get lost in broad daylight.

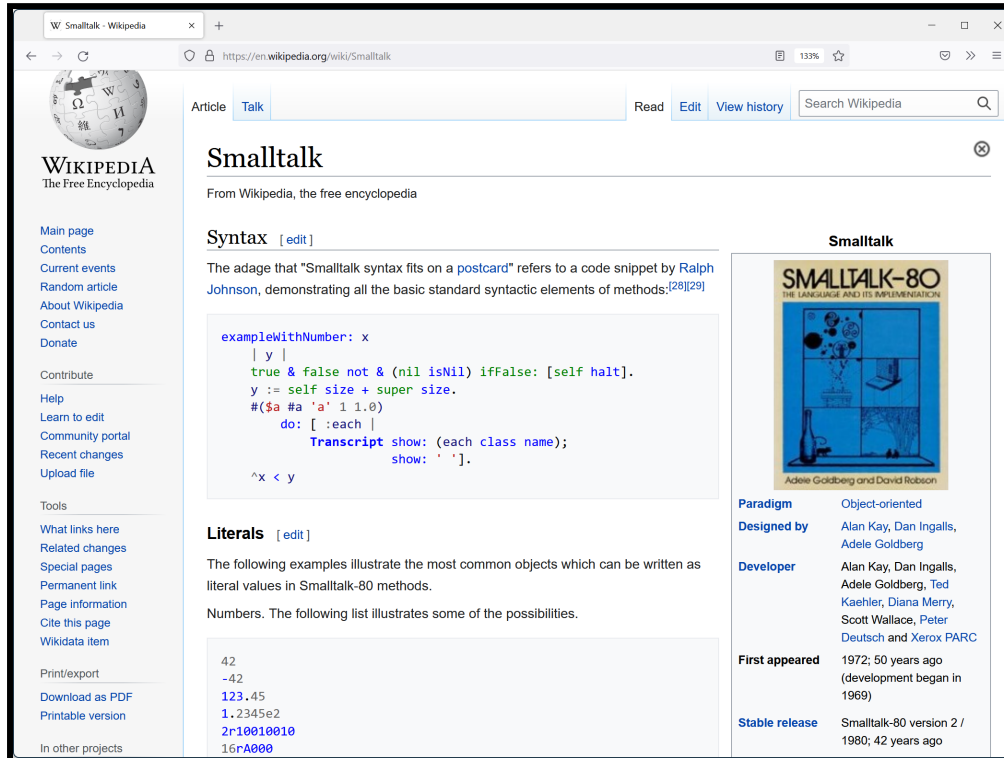
What I read in Brazil reminded me of my quest to demonstrate that in the pursuit of knowledge, at least in software and programming languages, engineering typically precedes science—that is, even if science ultimately produces the most reliable facts, the process often begins with engineering.

I believe it's a common belief that engineers only follow paths laid down by scientists, adding creativity and practical problem solving. Philip Kitcher, a philosopher of science at Columbia University, in an essay for the *New York Times*

Smalltalk language

"Smalltalk is an object-oriented, dynamically typed reflective programming language"

What makes it interesting?



The screenshot shows the Wikipedia article for Smalltalk. The page title is "Smalltalk" and the URL is "https://en.wikipedia.org/wiki/Smalltalk". The article content includes a section on "Syntax" with a code snippet, a section on "Literals" with a list of values, and a table of metadata.

Syntax [edit]

The adage that "Smalltalk syntax fits on a postcard" refers to a code snippet by Ralph Johnson, demonstrating all the basic standard syntactic elements of methods:^{[28][29]}

```
exampleWithNumber: x
| y |
true & false not & (nil isNil) iffFalse: [self halt].
y := self size + super size.
#($a #a 'a' 1 1.0)
do: [ :each |
    Transcript show: (each class name);
    show: ' ' ].
^x < y
```

Literals [edit]

The following examples illustrate the most common objects which can be written as literal values in Smalltalk-80 methods.

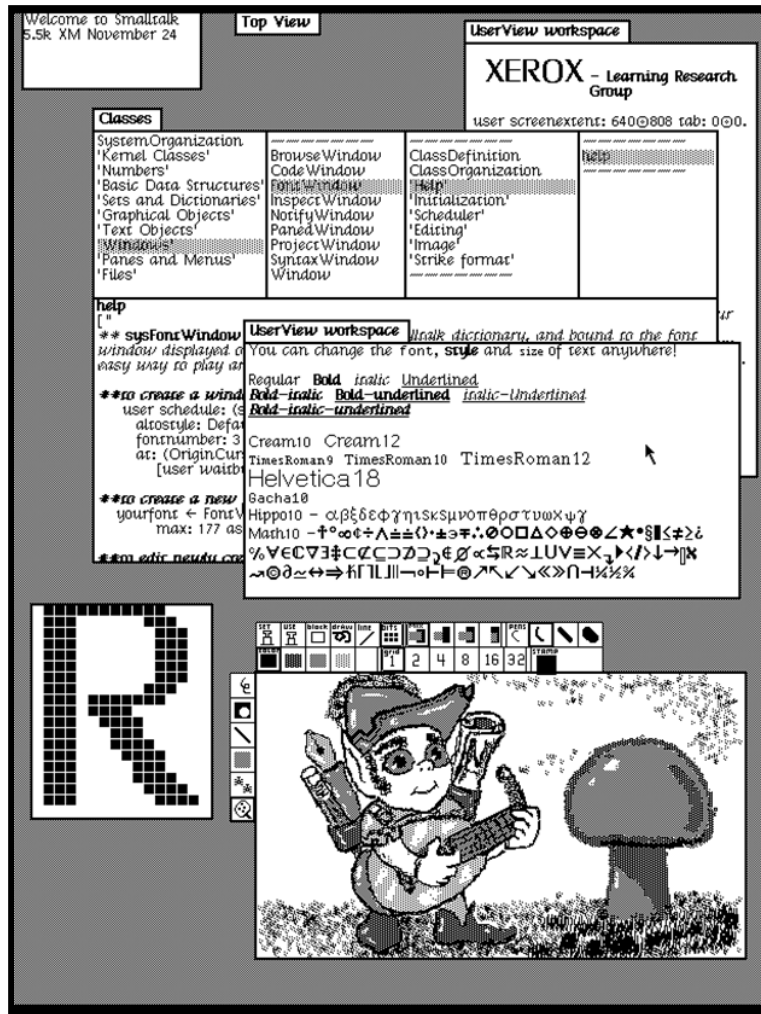
Numbers. The following list illustrates some of the possibilities.

```
42
-42
123.45
1.2345e2
2r10010010
16rA000
```

Paradigm	Object-oriented
Designed by	Alan Kay, Dan Ingalls, Adele Goldberg
Developer	Alan Kay, Dan Ingalls, Adele Goldberg, Ted Kaehler, Diana Merry, Scott Wallace, Peter Deutsch and Xerox PARC
First appeared	1972; 50 years ago (development began in 1969)
Stable release	Smalltalk-80 version 2 / 1980; 42 years ago

Smalltalk as a programming system

Think not about source code, but about evolving system state!







Demo

Smalltalk 72 and 78

```
Welcome to SMALLTALK [May 30]
␣ to square length
(␣ length ← :.
do 4 (␣ go length turn 90))!
square
␣ do 72 (␣ turn 5 square 100)!
```

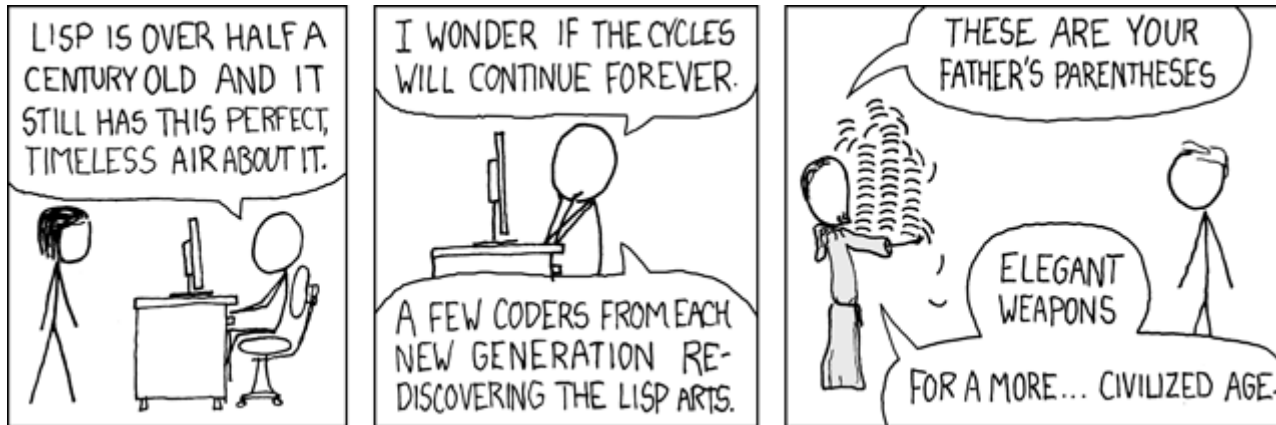
Smalltalk

Programming system view

-  Image-based persistence rather than source
-  Application ships with developer tools
-  Class browser allows inspecting & editing
-  Reflection lets the system change itself

LISP language

Functional programming language
derived from the lambda calculus?



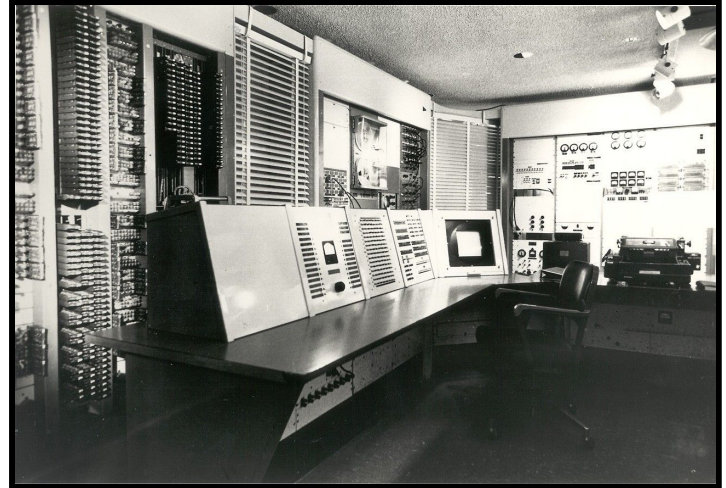
LISP environment

Time-sharing

- Batch processing in the 1950s
- TX-0 ('58) allowed interactive use
- Multi-user machines via teletype

AI research requirements

- Programming with symbolic data
- Interactive experimentation
- Programs that improve themselves



```
*EDITF(APPEND)
EDIT
*(P 0 100)
(LAMBDA (X) Y (COND ((NUL X) Z) (T (CONS (CAR) (APPEND (CDR X Y))))))
*(3)
*(2 (X Y))
*P
(LAMBDA (X Y) (COND & &))
```

LISP editor

(Deutsch, 1967)

Interactive program
editing on the terminal

Teletype, not a screen!

Print using: **P**

Delete child: **(3)**

Replace child: **(2 ..)**

Interlisp: Interactive Lisp

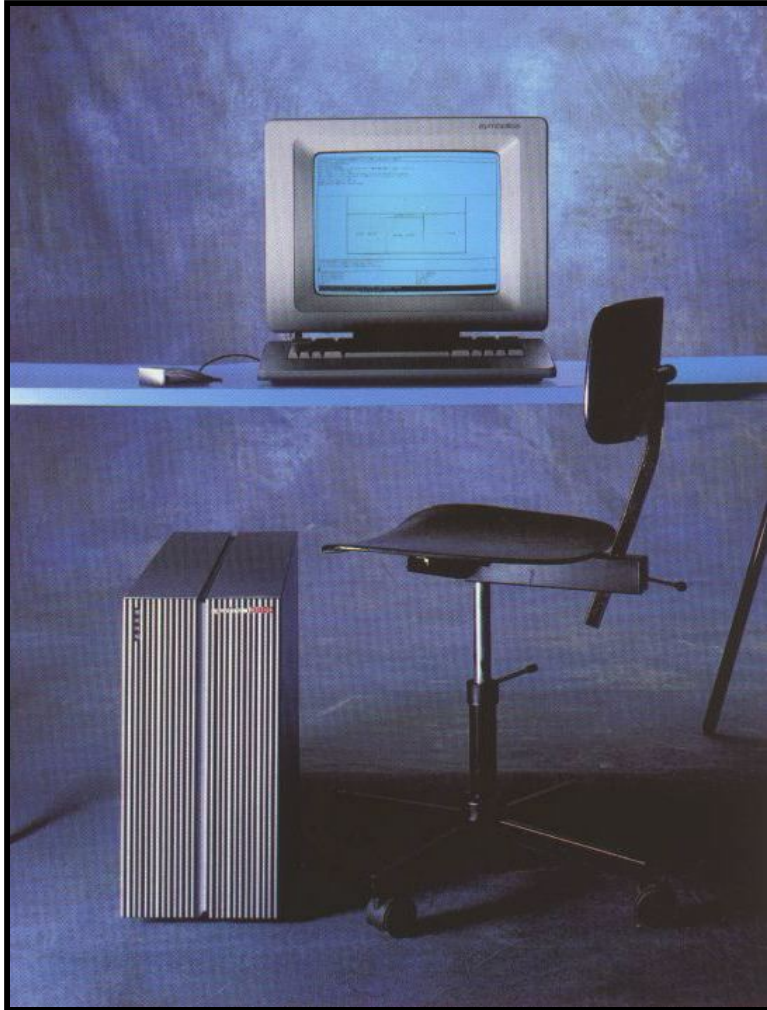
PILOT (1966)

- Edit code via list transformations
- Advising to enhance procedures
- Modifying state of a running system

DWIM (1974)

- Interactive program correction
- Suggests automatic fixes when error occurs
- Do What I Mean / Damn Warren's Infernal Machine





Symbolics Lisp Machines (1980s)





Machines optimized
for LISP with LISP-
based environment

Persistent memory
with just cons-cells

Response to new
hardware architecture

Scientific revolutions

Paradigm shifts in programming

-  Understand what people really thought!
-  The invention of a programming language
-  The shift from systems to languages
-  Functional programming "research programme"

Entities

Evolution of programming concepts

Proofs and Refutations

The Logic of
Mathematical Discovery

Imre Lakatos



How mathematical concepts evolve?

Polyhedra, space, graph, function, convergence, measurable set

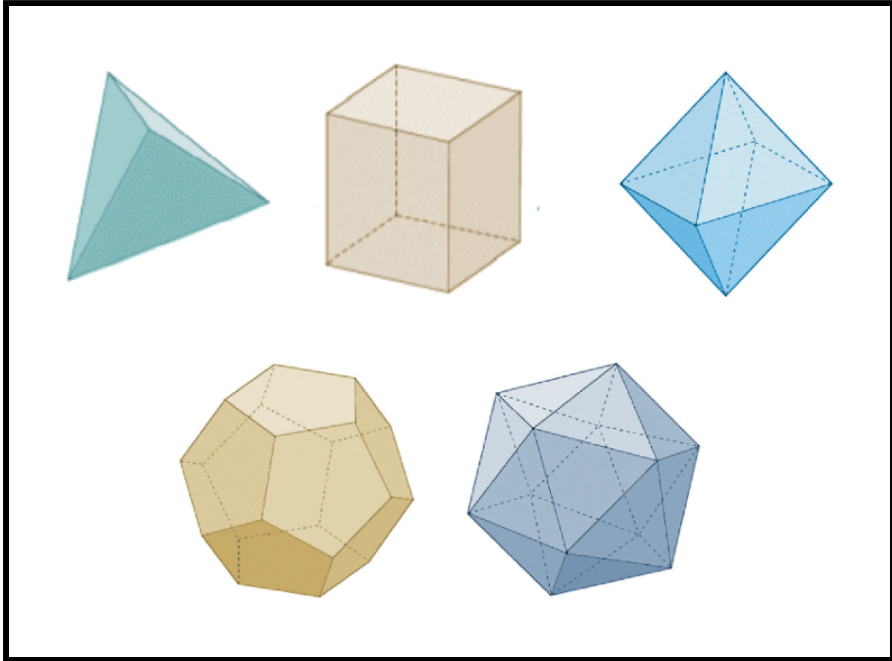
How does the definition change and why?

Polyhedra

Euler's formula

$$V - E + F = 2$$

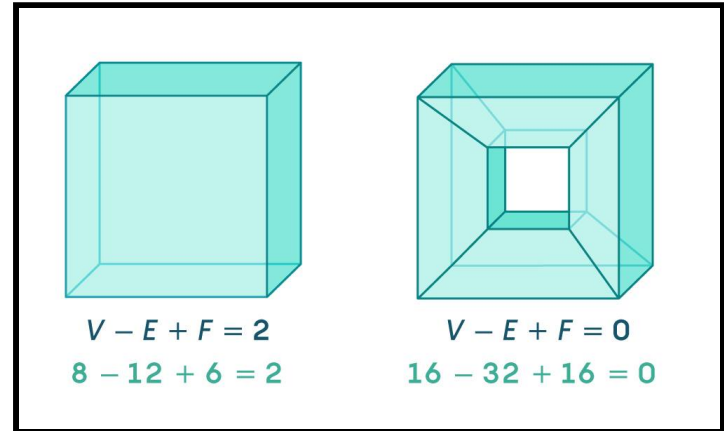
A polyhedron is a solid whose surface consists of polygonal faces?



Counter example?

Convex polygons!

Through any point in space there will be at least one plane whose cross-section with the polyhedron will consist of one single polygon.







Monster-barring

I turn aside with a shudder of horror from this lamentable plague of functions which have no derivatives.

(Charles Hermite, 1893)

Concepts

Proofs and refutations

-  Concept definitions are not constant but change
-  Arising from proofs, counter-examples, lemmas
-  Monster-barring and exception-barring
-  Concept stretching when understanding evolves

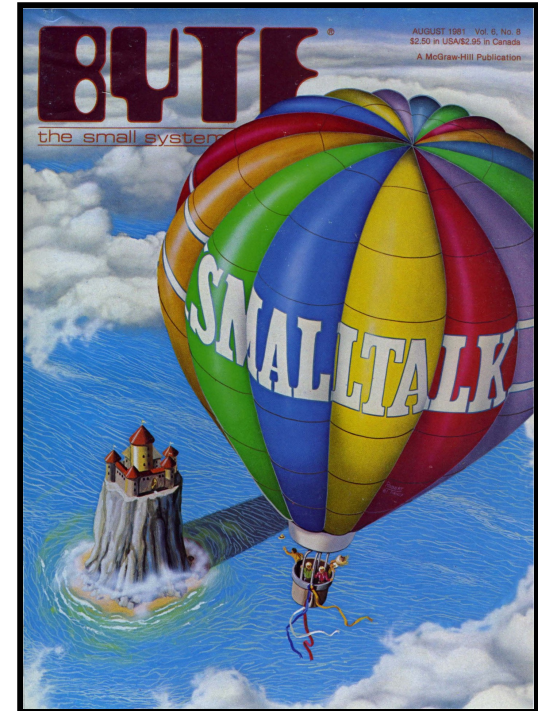
Concepts in programming

Change over time!

- Data types, logical types
- Monads and "railway" metaphor
- Processes become abstract

Multiple forces for change

- New implementation of the concept
- Different metaphor for thinking
- New formalization in a proof



Evolution of types

Implementation & formal modality

Data types like records, modelled as sets

Implementation modality evolves

Abstract data types for modularity

Type checking ala lambda calculus

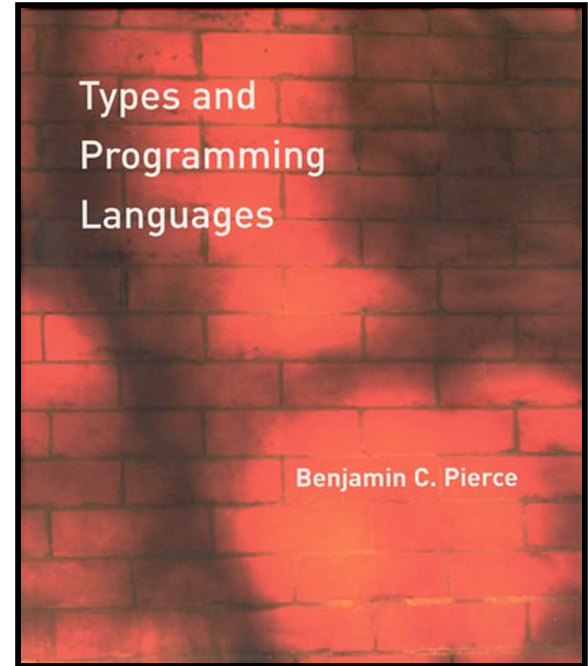
Intuitive modality evolves

Well-typed programs do not go wrong

New type systems based on this

Implementation modality evolves

Types for documentation and editor tooling



Understanding Monads

What are monads

- Origins in category theory
- Abstraction in functional programming
- Used for stateful computations



Writing about monads

- Compare how mathematicians and programmers talk about monads!
- tinyurl.com/nprg075-mcat
- tinyurl.com/nprg075-mprog



Evolution of monads

Formal and intuitive modality

Standard construction in algebraic topology

Monad as a "box" intuition

Implementation modality appears

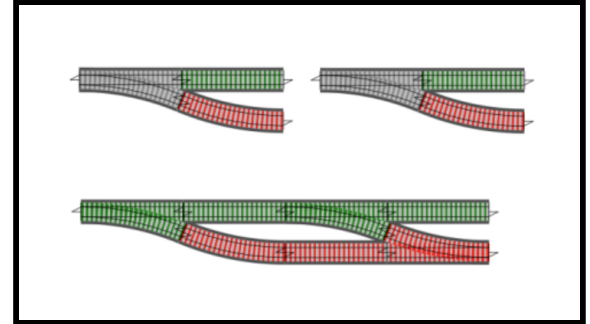
Used for sequencing effectful computations

Definition in terms of *bind* and *return*

Implementation & intuition evolves





Monads in Haskell and the **do** notation

Monad as a "sequencing" intuition



Concepts

Programming language design

-  There is more to concepts than just a name
-  Ideas come from logic, linguistics, biology!
-  Beware of concept stretching as with types?
-  Capture a new intuition in the design?

Social forces

What shapes programming?

Social history of computing

How commercial interests or gender bias shape computing

Redefinition of programming as more masculine software engineering in the 1960s

The image shows a screenshot of a book catalog page with a navigation bar at the top containing: Books, Journals, Open Access, Resources, Give, About, Contact Us. The page displays six book covers in a 2x3 grid, each with a title, author, ISBN, publisher, and publication date. Below each cover is a short description of the book's content.

Book Title	Author	ISBN	Publisher	Pub Date
PROGRAMMED IN EQUALITY: How Britain Discarded Women Technologists and Lost Its Edge in Computing	Mar Hicks	9780262535182	The MIT Press	February 23, 2018
The Rise and Fall and Reinvention of a Global Icon	James W. Cortada	9780262039444	The MIT Press	March 5, 2019
RECODING GENDER: Women's Changing Participation in Computing	Janet Abbate	9780262534536	The MIT Press	September 8, 2017
ENIAC in Action: Making and Remaking the Modern Computer	Thomas Haigh, Mark Priestley, and Crispin Rope	9780262535175	The MIT Press	January 26, 2018
Making IT Work: A History of the Computer Services Industry	Jeffrey R. Yost	9780262036726	The MIT Press	October 6, 2017
FOR FUN AND PROFIT: A History of the Free and Open Source Software Revolution	Christopher Tozzi	9780262036474	The MIT Press	August 11, 2017

PROGRAMMED IN EQUALITY
How Britain Discarded Women Technologists and Lost Its Edge in Computing
by Mar Hicks
ISBN: 9780262535182
Publisher: The MIT Press
Pub Date: February 23, 2018
This "sobering tale of the real consequences of gender bias" explores how Britain lost its early dominance in computing by systematically discriminating against its most qualified workers: women. (Harvard Magazine)

The Rise and Fall and Reinvention of a Global Icon
by James W. Cortada
ISBN: 9780262039444
Publisher: The MIT Press
Pub Date: March 5, 2019
A history of one of the most influential American companies of the last century.

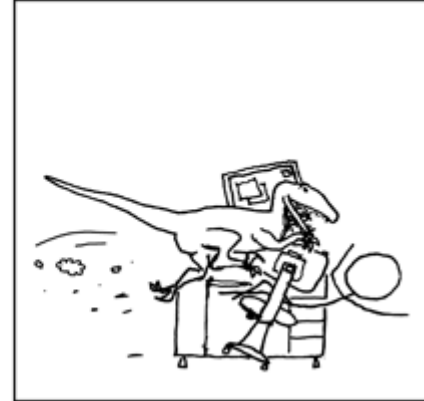
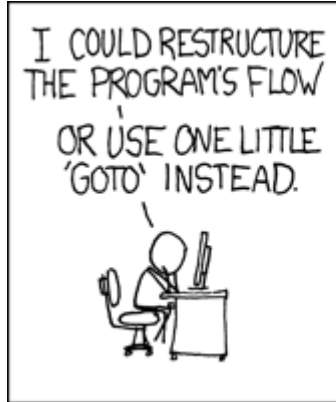
RECODING GENDER
Women's Changing Participation in Computing
by Janet Abbate
ISBN: 9780262534536
Publisher: The MIT Press
Pub Date: September 8, 2017
The untold history of women and computing: how pioneering women succeeded in a field shaped by gender biases.

ENIAC in Action
Making and Remaking the Modern Computer
by Thomas Haigh, Mark Priestley, and Crispin Rope
ISBN: 9780262535175
Publisher: The MIT Press
Pub Date: January 26, 2018
The history of the first programmable electronic computer, from its conception, construction, and use to its afterlife as a part of computing folklore.

Making IT Work
A History of the Computer Services Industry
by Jeffrey R. Yost
ISBN: 9780262036726
Publisher: The MIT Press
Pub Date: October 6, 2017
The evolution of the multi-billion-dollar computer services industry, from consulting and programming to data analytics and cloud computing, with case studies of important companies.

FOR FUN AND PROFIT
A History of the Free and Open Source Software Revolution
by Christopher Tozzi
Foreword by Jonathan L. Zittrain
ISBN: 9780262036474
Publisher: The MIT Press
Pub Date: August 11, 2017
The free and open source software movement, from its origins in hacker culture, through the development of GNU and Linux, to its commercial use today.

Structured programming



Goto considered harmful (1968)

The quality of programmers is a decreasing function of the density of go to statements in the programs they produce.

Problems with goto

- Hard to reason about informally
- Hard to reason about formally
- Code structure does not match runtime behaviour



Structured programming

Not obvious at the time!

- Everyone used to assembly!
- Can the compiler optimize code?
- Is it possible to avoid gotos?

<pre>s = 1; i = 1; while i < n do i = i + 1; s = s × i; end print(s);</pre>	<pre>L1 s = 1; i = 1; if i = n then goto L2 i = i + 1; s = s × i; goto L1; L2: print(s);</pre>
--	---

Structured Programming Theorem (1966)

Us converts waved this interesting bit of news under the noses of the unreconstructed assembly-language programmers who kept trotting forth twisty bits of logic and saying, 'I betcha can't structure this.'



DATAMATION⁷³®

DECEMBER, 1973

volume 19 number 12

This issue 137,600 copies

revolution in programming

According to guest editor McCracken, structured programming is a major intellectual invention that will revolutionize the way programs are produced. Our articles on this subject approach the issue in several ways. Before reading them, be sure to read the overview.

50 Revolution in Programming: An Overview
DANIEL D. MC CRACKEN

52 Structured Programming
JAMES R. DONALDSON

55 Structured Programming: Top-down Approach
EDWARD F. MILLER, JR. and GEORGE E. LINDAMOOD

58 Chief Programmer Teams
F. TERRY BAKER and HARLAN D. MILLS

62 A Linguistic Contribution to GOTO-less Programming
R. LAWRENCE CLARK





Datamation (1973)

What is structured programming and how to do it in practice

From engineering concept to managerial concept

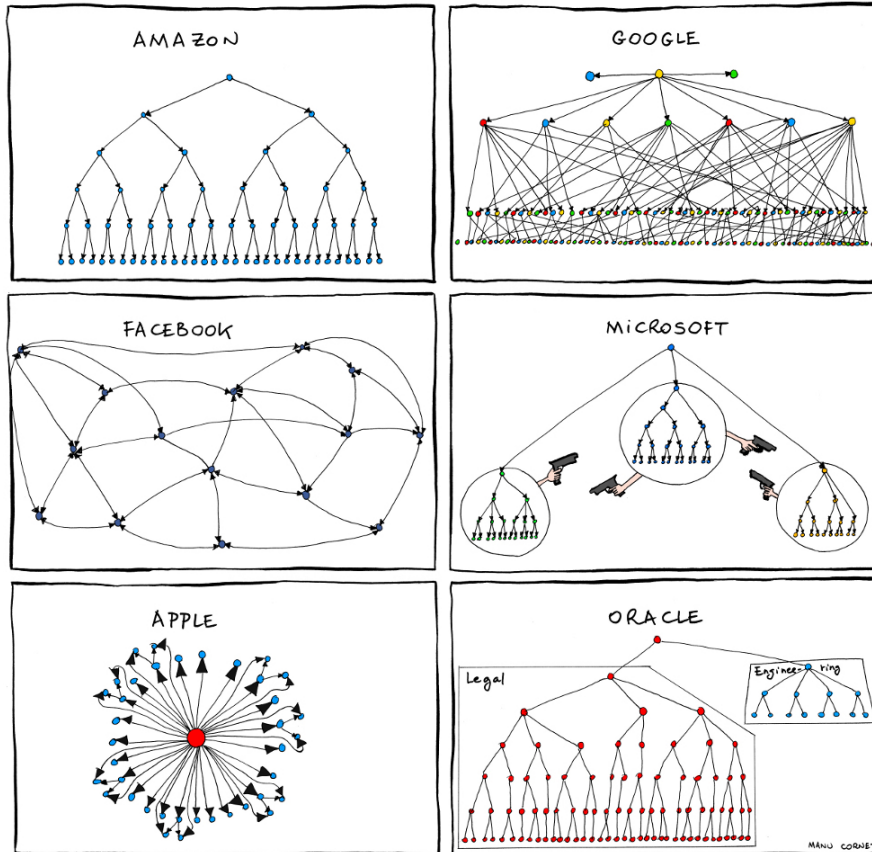
Chief programmer teams

Top-down management technique

-  Structured programming for organizing people
-  Chief-programmer leading & dividing code
-  Supported by programmers, secretary, backup
-  Hostile exchanges between Dijkstra and Mills





Conway's law

Any organization that designs a system will produce a design whose structure is a copy of the organization's communication structure.



Social forces

Programming language design

-  Language features linked to social structures
-  Organizational structure and escape hatches
-  Structured, microservices, information hiding
-  Origins of languages - COBOL, Fortran, Algol

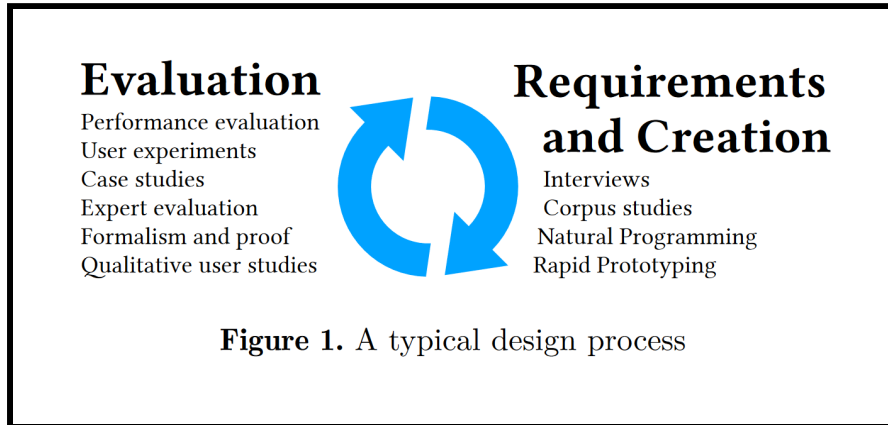
Conclusions

History and philosophy

History and philosophy

Learning from the past

Complex reasons why & how programming ideas work and do not work



Reading

```
10 PRINT CHR$(205.5+RND(1));  
20 GOTO 10
```

- 15: REM Variations in Basic
- <https://10print.org> (look for the PDF)

Why should you read this?

- Fun look at an unexpected bit of programming history
- What can we learn from the past?



Conclusions

History and philosophy of programming

- Scientific paradigms and paradigm shifts
- The history of programming concepts
- How social forces shape programming

Tomáš Petříček, 309 (3rd floor)

✉ petricek@d3s.mff.cuni.cz

➔ <https://tomasp.net> | [@tomaspetricek](https://twitter.com/tomaspetricek)

➔ <https://d3s.mff.cuni.cz/teaching/nprg075>

References (1/2)

Philosophy of science

- Kuhn, T. S., (2012). [The Structure of Scientific Revolutions](#). Chicago
- Feyerabend, P. (1975). [Against Method](#). Verso
- Lakatos, I. (1976). [Proofs and Refutations](#). Cambridge

History & reflections

- De Mol, L., Primiero, G. eds. (2018). [Reflections on Programming Systems: Historical and Philosophical Aspects](#). Springer
- Gabriel, R. (2012). [The Structure of a Programming Language Revolution](#). Onward!
- Petricek, T. (2022). [Cultures of Programming](#). Draft
- Petricek, T. (2018). [What we talk about when we talk about monads](#)

References (2/2)

Historical materials

- Teitelman, W. (1966). [PILOT: A Step Toward Man-Computer Symbiosis](#). MIT
- Teitelman, W. (1974). [Interlisp Reference Manual](#). Xerox PARC
- Deutsch, P. (1967). [Preliminary Guide to the LISP Editor](#). Berkeley
- Dijkstra, E. (1968). [Go To Statement Considered Harmful](#). ACM
- McCracken et al. (1973). [Revolution in Programming](#). Datamation 12