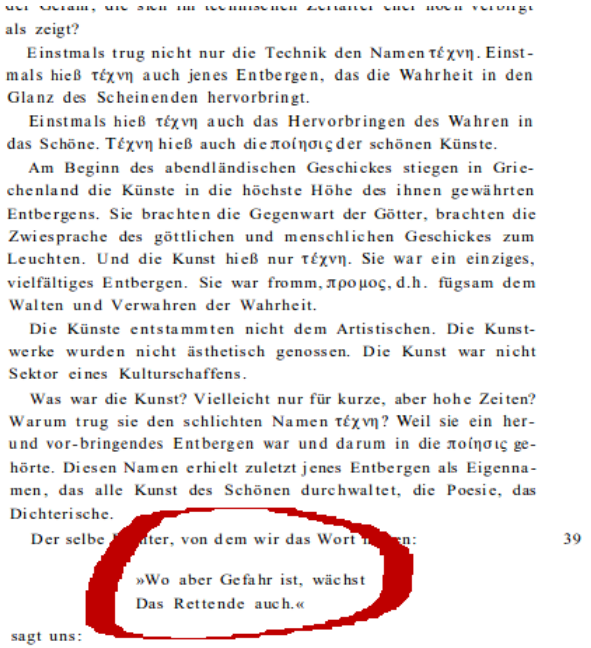
What is a computer program? Historical and philosophical reflections.

Liesbeth De Mol
CNRS, UMR 8163 Savoirs, Textes, Langage - Université de Lille
ldm@program-me.org

"*history has a habit of embarrassing our treasured beliefs*"
Nick Cave 2019

Background (motivations)...



als zeigt?

Einstmals trug nicht nur die Technik den Namen τέχνη. Einstmals hieß τέχνη auch jenes Entbergen, das die Wahrheit in den Glanz des Scheinenden hervorbringt.

Einstmals hieß τέχνη auch das Hervorbringen des Wahren in das Schöne. Τέχνη hieß auch die ποίησις der schönen Künste.

Am Beginn des abendländischen Geschickes stiegen in Griechenland die Künste in die höchste Höhe des ihnen gewährten Entbergens. Sie brachten die Gegenwart der Götter, brachten die Zwiesprache des göttlichen und menschlichen Geschickes zum Leuchten. Und die Kunst hieß nur τέχνη. Sie war ein einziges, vielfältiges Entbergen. Sie war fromm, προμος, d.h. fügsam dem Walten und Verwahren der Wahrheit.

Die Künste entstammten nicht dem Artistischen. Die Kunstwerke wurden nicht ästhetisch genossen. Die Kunst war nicht Sektor eines Kulturschaffens.

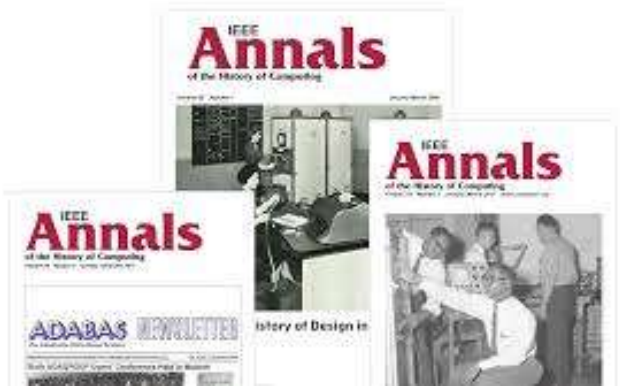
Was war die Kunst? Vielleicht nur für kurze, aber hohe Zeiten? Warum trug sie den schlichten Namen τέχνη? Weil sie ein her- und vor-bringendes Entbergen war und darum in die ποίησις gehörte. Diesen Namen erhielt zuletzt jenes Entbergen als Eigennamen, das alle Kunst des Schönen durchwaltet, die Poesie, das Dichterische.

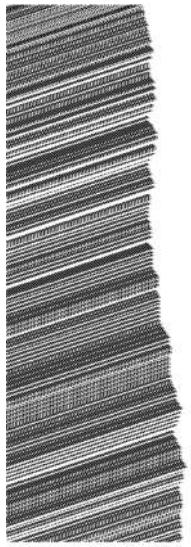
Der selbe [illegible]ter, von dem wir das Wort [illegible]n:

39

»Wo aber Gefahr ist, wächst
Das Rettende auch.«

sagt uns:





My research:

- "Wo aber Gefahr ist, wächst das Rettende auch" (Hölderlin as quoted by Heidegger)
- (some computer-assisted "logic" and so a bit of programming (Scheme and Basic))
- analyses of the histories of logic, computation, mathematics, programming and engineering – historical transparancy

Background (motivations)...

- My collaborations within **PROGRAMme** and especially

with **Tomas Petricek**

See: `https://programme.hypotheses.org/`

"What is a computer program?" – why should you care?

Clear definition? Eg "A computer program is a collection of instructions that performs a specific task when executed by a computer. Most computer devices require programs to function properly. A computer program is usually written by a computer programmer in a programming language."

BUT how does that help us with:

- Social problems (eg automating poverty)
- Judicial problems (eg the patent vs copyright debate)
- Political problems (eg "war" against encrypted messaging)
- Theoretical problems (eg what is the meaning of a program?)
- Engineering problems (e.g. legacy problems)
- etc

⇒ Assume a specific understanding of what programs are, what they are supposed to be capable of and how they *should* be ... used, made, shared, etc

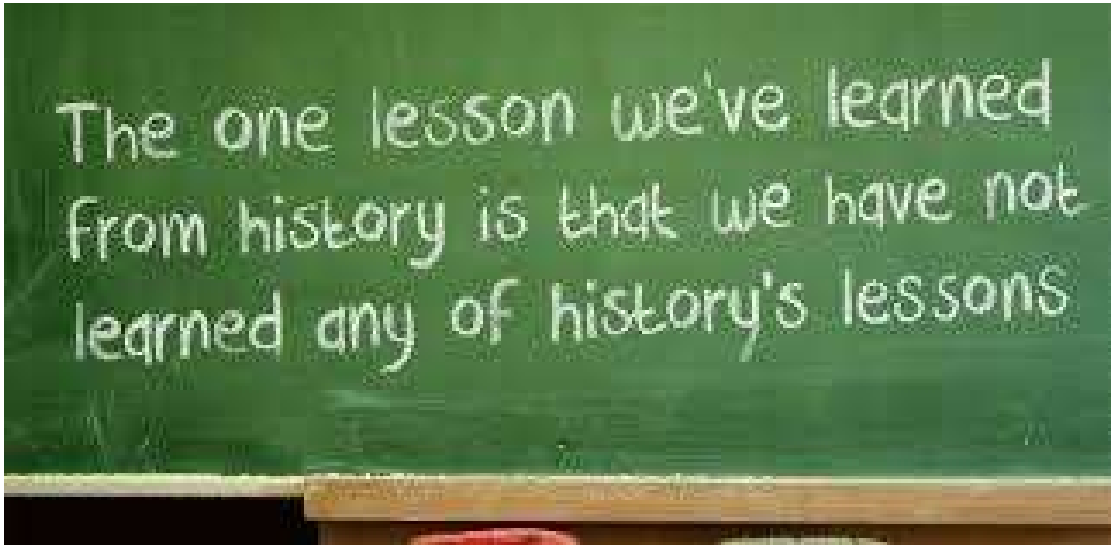
⇒ The indirect and historicized attack...

Structure of this talk

"*History of science without philosophy of science is blind. Philosophy of science without history of science is empty*" (Lakatos 1961)

- The Flesh (or, from History)
  - Words, words, words – two cases
  - Retro-fitting our past
  - In search of an identity
- The Bones (or, from Philosophy)
  - On obstacles
  - Obstacle I: History
  - Obstacle II: Opacity and alienation
  - Obstacle III: dichotomies
- a Proposal and a Discussion

from History



I. Words Words Words

Polonius What do you read, my lord?

Hamlet. Words, words, words.

Polonius. What is the matter, my lord?

Hamlet. Between who?

Polonius. I mean, the matter that you read, my lord.

Shakespeare 1603

A language that doesn't affect the way you think about programming, is not worth knowing.

Perlis 1982

Words Words Words

What's in a word?

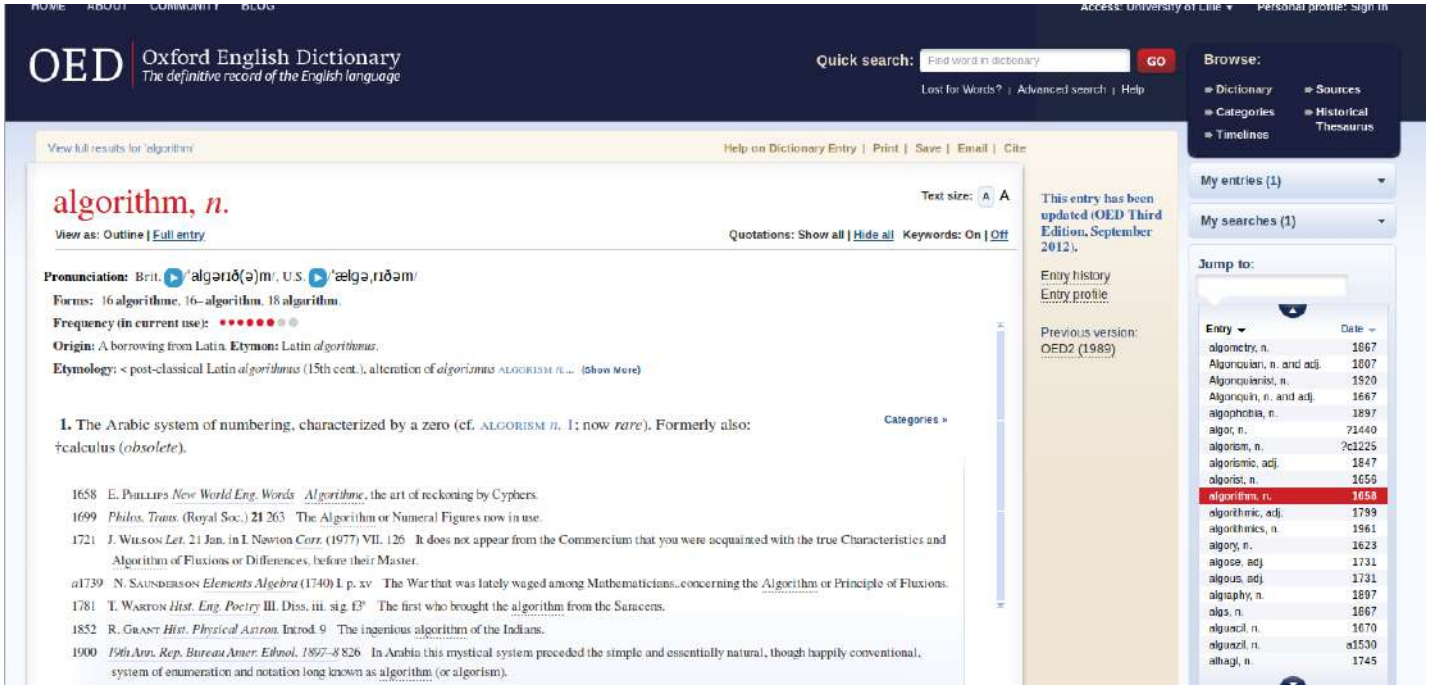
"*Historians often worked under the assumption that the main components of scientific texts problems, algorithms and so on* **are essentially ahistorical objects**, *which can be approached as some present-day counterparts.*" (Chemla 2009)

⇒ Words carry a semantical history

⇒ Words *are* historical

Words Words Words

What's in a word? The case of "algorithm"



Origins of the word – Al-Kwarizmi's *Hindu Art of Reckoning* – *not* mostly about procedures but **a notation** (decimal system)

Words Words Words

What's in a word? The case of "algorithm"

1960s-70s (CACM, ALGOL and Knuth) – disciplinary shaping of CS with reference to math (Tedre 2015; Bullynck and De Mol 2019):
"*One of the ways to help* **make computer science respectable** *is to show that it is deeply rooted in history, not just a short-lived phenomenon [...]*" (Knuth 1968)

"*This department was established to publish algorithms consisting of "procedures" and programs in the ALGOL language*" (Perlis 1960)

Algorithm as step-by-step **procedure; a recipe**:
"*The modern meaning for algorithm is quite similar to that of recipe, process, method, technique, procedure, routine, rigmarole*" (Knuth 1997)

Words Words Words

What's in a word? The case of "algorithm"

Today? Facebook "algorithm"; Algorithmic bias; "How algorithms punish the poor"; etc

Algorithm as **software; system; program**

Steered by industry?

⇒ Process of semantic extension (Fossa 2019);not stripped of its past – it bears its "mathematical" connotation

⇒ Dangerous (and, to some, useful) illusion of a world governed not by human-made and complex software but by mathematics

Words Words Words

'Tis but thy name that is my enemy. The art-to-science narrative in programming

"*programming in the early 1950s was* **a black art**, *a private arcane matter.*" (Backus 1980)

"*programming has arisen not as a science but* **as a craft** *[...] [A]fter some time they felt free to include all kinds of curious facilities of doubtful usability, reassuring themselves by their experience that, no matter how crazy a facility they provided,* **always a more crazy programmer would emerge** *that would manage to turn it into something profitable as if this were sufficient justification for its inclusion.*" (Dijkstra, EWD32, 1962-64)

⇒ Reductive and negative semantics attached to "art" in contrast to "science"

Words Words Words

" 'Tis but thy name that is my enemy."

Fits into Western tradition of: Epistémé vs Têchné? Theory vs. Practice? Creativity vs. Rationality?

BUT:

- being artistic is not necessarily a bad thing – cfr Backus' melancholy:

  "*Programming in the America of the 1950s had a vital frontier enthusiasm virtually untainted by either the scholarship or the* **stuffiness of academia**. *The* **programmer-inventors** *of the early 1950s were too impatient*" (Backus 1980)

- "art" cannot and should not be reduced to crafts
- use of "art" and "Art" also in the so-called scientific discourse not just as a pleasantry but as a method for building (better) software

Words Words Words

"'Tis but thy name that is my enemy".

Cfr Knuth: TeX, literate programming;

"*Given a sequence of points in the plane, what is the most pleasing curve that connects them?*" (Knuth 1979)

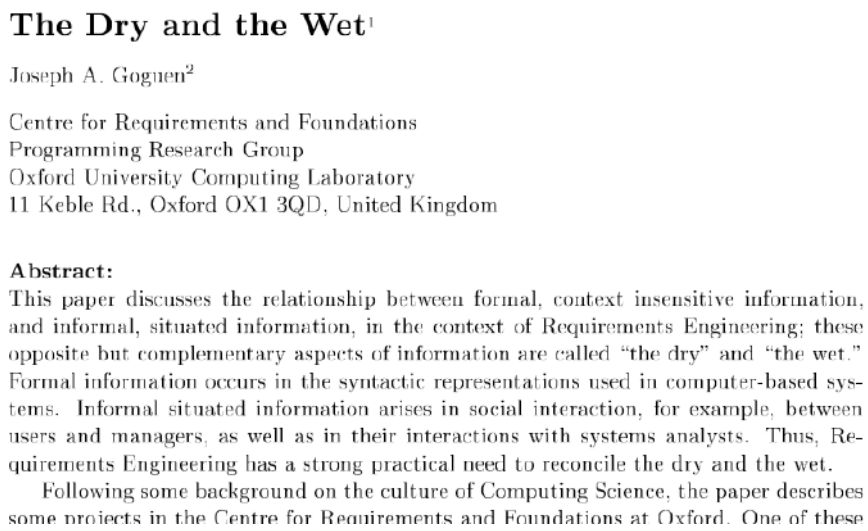
"*better documentation of programs [...] can best [be] achieve[d] [...] by considering programs to be works of literature*" (Knuth 1984)

cfr Dijkstra's discourse:

"*The tool should be charming, it should be elegant, it should be worthy of our love. [T]he programmer does not differ from any other* **craftsman***: unless he loves his tools it is highly improbable that he will ever create something of superior quality. At the same time these considerations tell us the greatest virtues a program can show:* **Elegance and Beauty.**"

Words Words Words

"'Tis but thy name that is my enemy".



The Dry and the Wet[1]

Joseph A. Goguen[2]

Centre for Requirements and Foundations
Programming Research Group
Oxford University Computing Laboratory
11 Keble Rd., Oxford OX1 3QD, United Kingdom

Abstract:
This paper discusses the relationship between formal, context insensitive information, and informal, situated information, in the context of Requirements Engineering; these opposite but complementary aspects of information are called "the dry" and "the wet." Formal information occurs in the syntactic representations used in computer-based systems. Informal situated information arises in social interaction, for example, between users and managers, as well as in their interactions with systems analysts. Thus, Requirements Engineering has a strong practical need to reconcile the dry and the wet.

Following some background on the culture of Computing Science, the paper describes some projects in the Centre for Requirements and Foundations at Oxford. One of these

⇒ Narratives reduce words and "fit" history to the purpose (⇔ history without telos)

⇒ Risk of "blind" spots and misinterpretation (for the historian)

⇒ Contributed to a (communication) gap running through the field, cfr "software crisis"; Goguen's **the dry and the wet**

II. Retro-fitting our pasts

Retro-fitting our pasts...

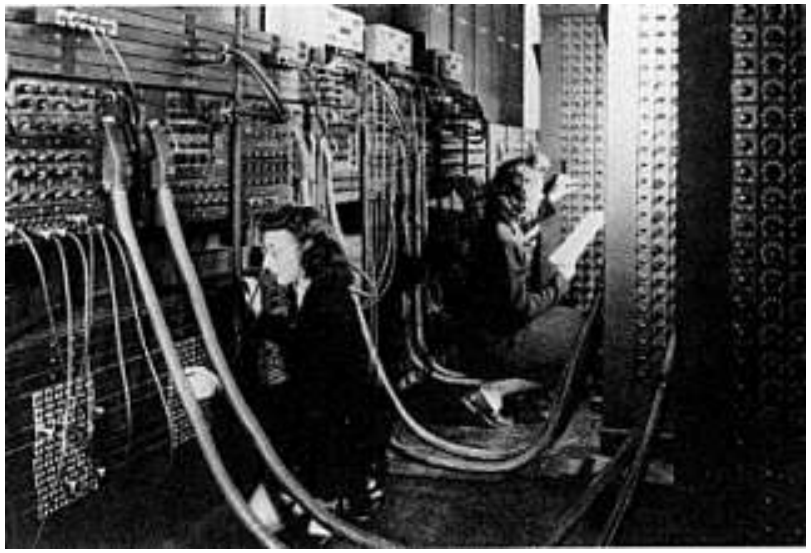
or how basic assumptions are uncovered and deconstructed by going back

$\Rightarrow$ "Theory is the captain, and application the soldier"

$\Rightarrow$ Programs need code

Retro-fitting our pasts...

Programs without code – back to the start



- Publicly presented in 1946
- About 100 differents sorts of computations were ran on ENIAC
- The ENIAC has known two periods of usage:
  1. 1946–1947: Original set-up, a modular and parallel machine with external programming ⇒ **Programs without code**
  2. 1947–1955: Rewiring into a serial stored-program computer with a 'order vocabulary' of some 60 words (Haigh et al 2016)

Retro-fitting our pasts...

Programs without code – historical accumulation of three basic features for precise, correct and fast calculations in applied mathematics

1. Discreteness

→ Higher accuracy cmp to continuous devices

→ Not new, cfr Mark I; Bell lab machines; Stibitz' machines

2. Conditional branching and general-purpose

→ automation of the computational process through conditional

→ allows "general-purpose" – option of internal decisions

→ also used in mechanical machines + embedded in historical development towards general computation (cfr DA)

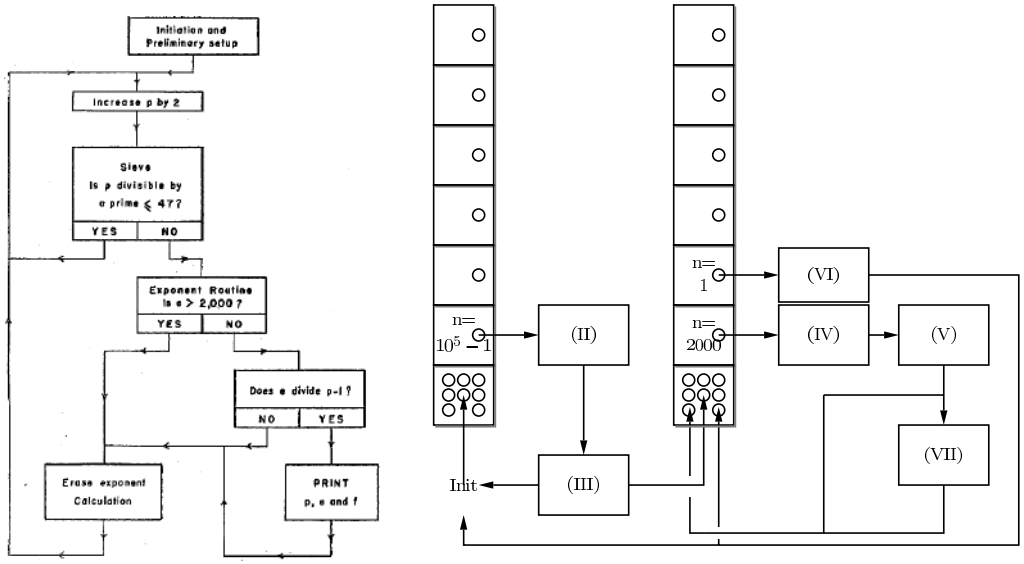
3. High electronic speed

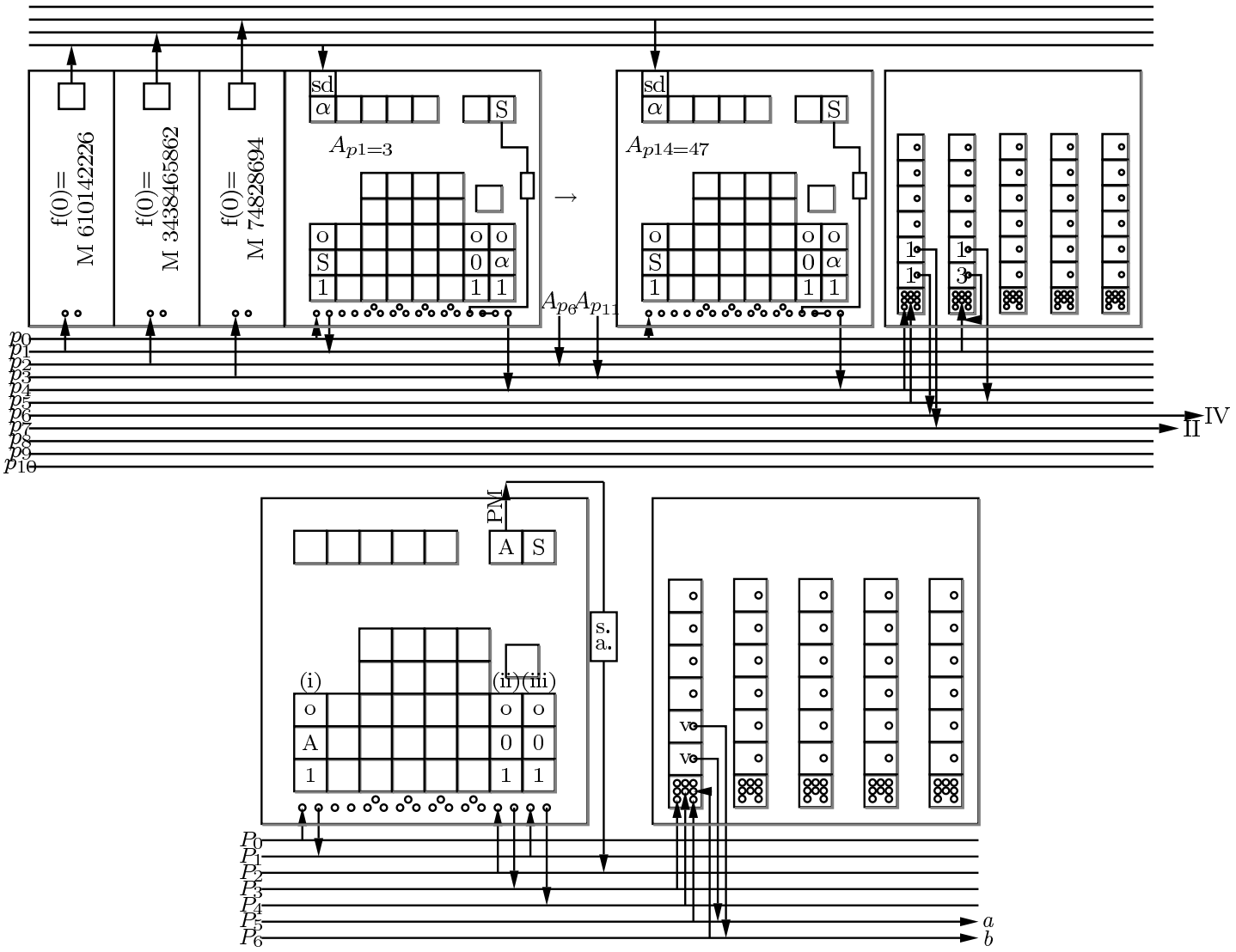
→ absent in the mechanical machines – essential feature which made ENIAC stand out

Retro-fitting our pasts...

Programs without code in original ENIAC

(Cfr De Mol and Bullynck 2012)

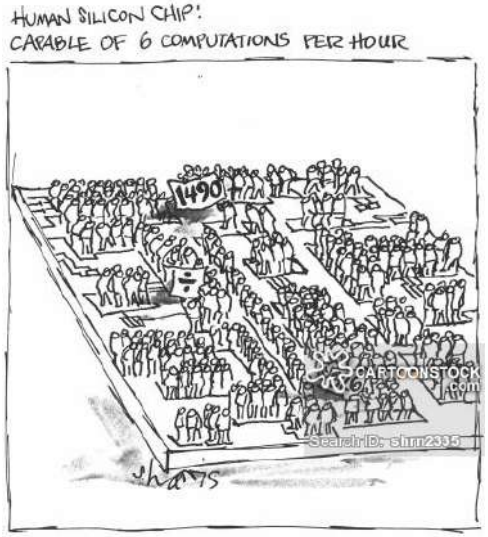



f(0)=
M 6101142226
f(0)=
M 3438465862
f(0)=
M 74828694
sd
α
S
$A_{p1=3}$
$A_{p14=47}$
o
S
1
0
α
1
A_{p_6} $A_{p_{11}}$
p_0
p_1
p_2
p_3
p_4
p_5
p_6
p_7
p_8
p_9
p_{10}
II
IV
PM
A
S
s.
a.
(i)
(ii)(iii)
P_0
P_1
P_2
P_3
P_4
P_5
P_6
a
b

Retro-fitting our pasts...

Programs without code – Consequences (1)

⇒ epistemological impact because of **time gap** between human and mechanical time vs machine time:



Unpredictability and lack of control:

"*we are* **mentally not capable** *of seeing through the tremendous complications, all the consequences of a computer program*" (Alt 1972)

Rethinking of (existing) computational methods, e.g. decrease data increase programs...

Retro-fitting our pasts...

Programs without code – consequences (2)

stored-program considered a corollary: "*It is* **evident** *that in order to be* **efficient**, *you will have to treat the problem of logical instructions on the same level on which you treat other memory problems.*" (von Neumann 1945)

⇒ Rewired "stored-program" ENIAC: but just another rewiring (which had basic impact)

Retro-fitting our pasts...

Programs without code – consequences (3)

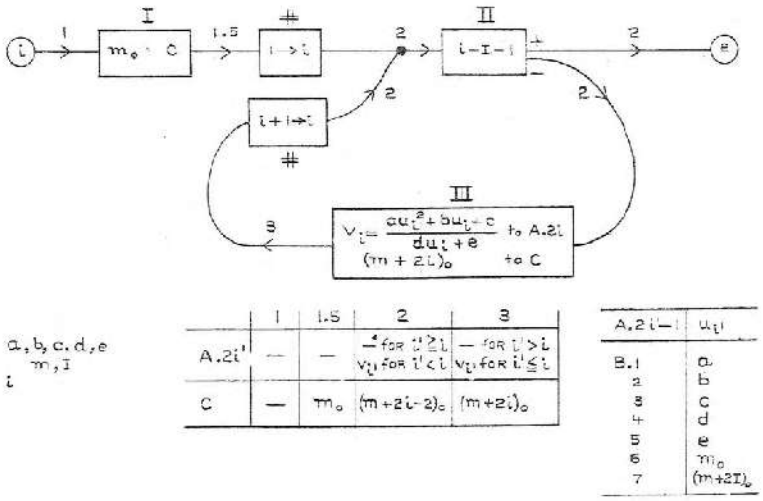
in need of reflections on symbolic/programmed "programming":

"*[C]ontemplate the prospect of locking twenty people for two years during which they would be steadily performing computations. And you must give them such explicit instructions at the time of incarceration that at the end of two years you could return and obtain the correct result for your lengthy problem! This dramatizes the necessity for high planning, foresight, and consideration of the logical nature of computation. This* **integration of logic in the problem is a consequence of the high speed.**"

Retro-fitting our pasts...

Programs without code – consequences (3)

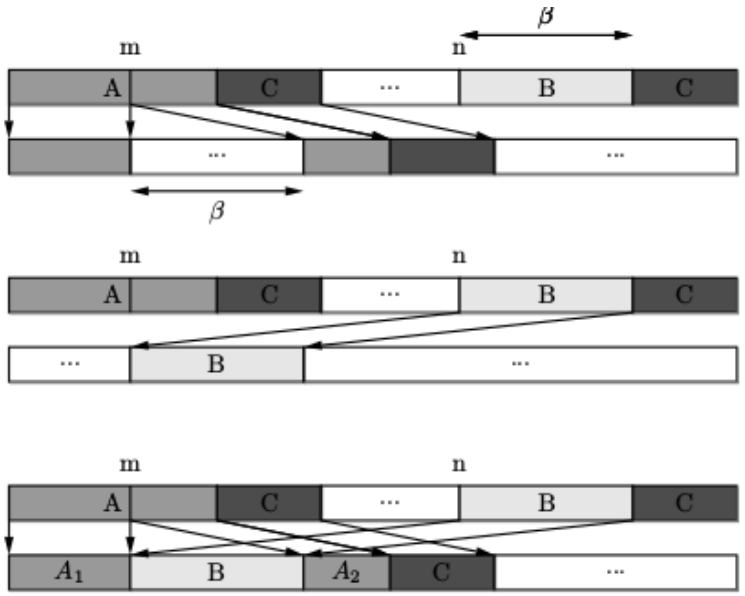
the von Neumann way:



	1	1.5	2	3
A.2i'	—	—	$\frac{}{v_i}$ for $i' \geq i$ v_i for $i' < i$	— for $i' > i$ v_i for $i' \leq i$
C	—	m_0	$(m+2i-2)_0$	$(m+2i)_0$

A.2i−1	u_{i1}
B.1	a
2	b
3	c
4	d
5	e
6	m_0
7	$(m+2I)_0$

the Curry way:



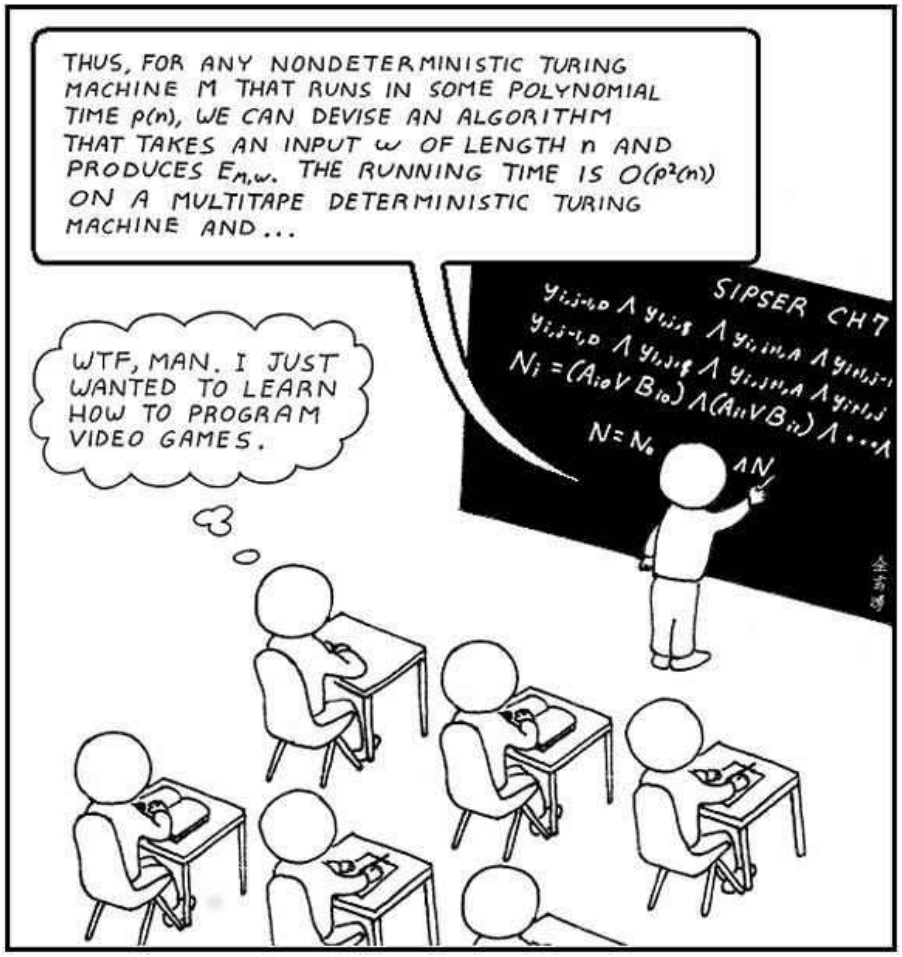
$U_1 \to (U_2 \to (U_4 \to O_1 \& \langle U_1 \rangle) \& (U_5 \to \langle U_3 \rangle \& O_3) \& \langle U_3 \rangle) \& (U_3 \to O_2 \& O_1)$.

Retro-fitting our pasts...

Where is the captain?

- Theoretical insights from the 1930s (read: UTM) were *not* needed to develop the modern computer *nor* stored-program idea –
- retro-fitted history for (a) sociological reasons (b) rapprochements between theory and practice (historically, the first)
  See also: (Haigh 2014; Daylight 2014; Daylight, Bullynck and De Mol 2015; De Mol, Bullynck and Daylight 2018)
- Being theorey-grounded is not necessarily the right or only way. Theory need not be the captain.

In search of an identity



Computer science is....????

In search of an identity

(Cfr Tedre 2015, Science of computing. The shaping of a discipline, CRC Press)

What is the discipline (about)?

"Computer science is the study of **information structures**" (Wegner 1968)

"Computing is a **natural science**" (Denning 2015)

"computer science is [...] the study of **algorithm**" (Knuth 1974)

"Computer science is the study of **phenomena related to computers**" (Newell, Perlis and Simon 1967)

"A conscious methodological non-regimentation and a disregard of frequent calls for more methodological rigor render computer science an epistemologically and methodologically **anarchist** discipline." (Tedre 2006)

"Computer science is the study and management of **complexity**" (Dijkstra 1969)

"Computer science is **in part** a scientific discipline [...] **in part** a mathematical discipline [...], and **in part** a technological discipline concerned with the cost-effective design and construction of commercially and socially valuable products" (Wegner 1976)

In search of an identity

What should be its name?

"*It would help our profession to be widely recognized if it had a brief, definitive, and distinctive name. This should be general enough to cover a variety of sub-fields [...] but specific enough to imply that computing applications are involved. Consider the solid professional sound of such terms as "Petroleum Engineer" or "Nuclear Physicist." What can we use that will be equally clear-cut–and* **at least half as impressive?**" (Editors of DATA-LINK 1958) Reply? "*Several names have been suggested [...] Turingineer, Turologist, Flow-Charts man, Applied Meta-Mathematician and Applied Epistomologist*"

Historically fluctuating: see e.g. Dijkstra's shift from engineer to scientist

Today? Informatica, Datalogi, Computer Science, software engineering?

In search of an identity

What should be thaught?

Changes in ACM curriculae: "*Over the course of ten years, the definition of computer science in the ACM curriculum turned* **from a theoretical, mathematically based discipline** *that studies in formation structures* **into a programming and applications-centered discipline**." (Tedre 2006)

⇒ Historically and geographically fluctuating up to today – cfr also the recent debates on CS education for school children.

In search of an identity

Why should we care?

SOFTWARE ENGINEERING TECHNIQUES

Report on a conference sponsored by the

NATO SCIENCE COMMITTEE

Rome, Italy, 27th to 31st October 1969

Chairman: Professor F. Ercoli

Co-chairman: Professor Dr. F. L. Bauer

Editors: J. N. Buxton and B. Randell

April 1970

"*The sense of urgency in the face of common problems was not so apparent as at Garmisch. Instead,* **a lack of communication** *between different sections of the participants became [...] a dominant feature. [...] Just as the realization of the full magnitude of the software crisis was the main outcome of [...] Garmisch, it seems [...] that* **the realization of the significance and extent of the communication gap is the most important outcome of the Rome conference**." (Ercoli, Bauer and Randell 1969)

⇒ Such gaps are plenty and diverse even today – is this the real crisis or just an academic problem?

In search of an identity

What is a discipline?

Modern disciplines are a development of the 19th century

"Discipline" itself is historical (cfr Koseleck's notion of *Begriffsgeschifte – Geschichtliche Grundbegriffe*)

"*only the nineteenth century established real disciplinary communication systems. Since then the discipline has functioned as a unit of structure formation in the social system of science, in systems of higher education, as a subject domain for teaching and learning in schools, and finally as the designation of occupational and professional roles.*" (Stichweh 2001)

In how far should we use/apply a "Begriff" from the 19th century to something that did not exist?

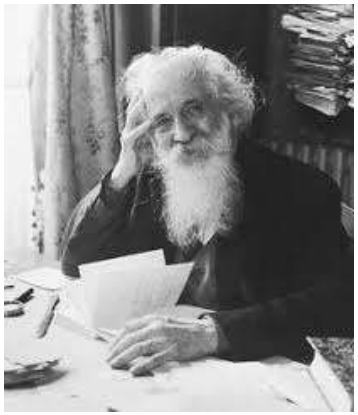
a Recap from History

- Words have a history and bear the semantical connotations with them when used today
- Be careful of retro-fitting history to current assumptions and dare to challenge those assumptions with history
  - programs did not always need code
  - we do not necessarily need to start from theory to achieve something
- The disciplinary debates are perhaps victim of the historicity of "discipline" itself

from Philosophy

On obstacles (Bachelard, 2002/1938)

⇒ Historical insights framed as "**epistemological obstacles**"



"*An epistemological obstacle will encrust any knowledge that is not questioned.*"

"*the task of the philosophy of science is very clear: it is to psychoanalyse interest, to destroy all utilitarianism, however disguised its form and lofty the status it claims*"

"*Historians of science have to take ideas as facts. Epistemologists have to take facts as ideas and place them within a system of thought.* **A fact that a whole era has misunderstood remains a fact in historians' eyes. For epistemologists however, it is an obstacle**"

Obstacle I. History

⇒ ignoring our being shaped in and being part of history is dangerous especially if we see "programs" as a **radical novelty**

Obstacle II. Opacity and alienation

What Makes the History of Software Hard

Michael S. Mahoney
Princeton University

History of commitments constrains choice. Narrow incentives and opportunities motivate choice.
—Rob Kling and Walt Scacchi[1]

We never have a clean slate. Whatever new we do must make it possible for people to make a transition from old tools and ideas to new.
—Bjarne Stroustrup[2]

Creating software for work in the world has meant translating into computational models the knowledge and practices of the people who have been doing that work without computers. What people know and do reflects their particular historical experience, which also shapes decisions about what can be automated and how. Software thus involves many histories and a variety of sources to be read in new ways.

This article originally appeared in L. Böszörmenyi, most current, is in that sense "legacy" soft-

Cybersquare

A Plea for Lean Software

Niklaus Wirth
ETH Zürich

Programs *are* historical – building layer upon layer they too bear history and so, also, a set of cultures, ideologies, methods and practices

⇒ Opacity and complexity → hiding and/or ignoring of that very history: "*the [program] is the stranger inside which something human is locked up, misunderstood, materialized, enslaved*" (Simondon 1958/2017)

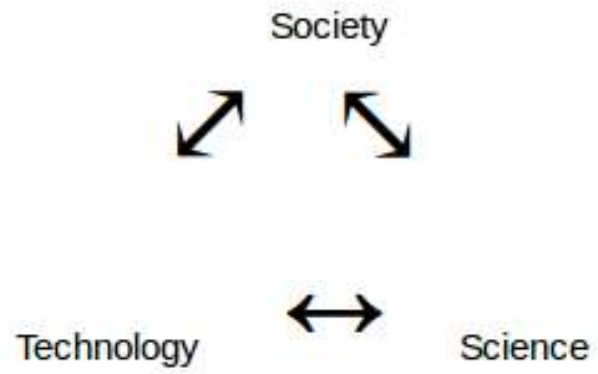
⇒ Alienation: "*objects meant for ignorant users create greatest alienation*" (Simondon 1958/2017)

⇒ In need of more historical transparency and *open* programs

Obstacle III. dichotomies

Western tendency to think in dichotomies – applies also to programming: concrete vs abstract; human vs machine; informal vs. formal; art vs science; bottom-up vs top-down; industry vs academia; modern vs postmodern.

Abstraction in a triangle of oppositions?



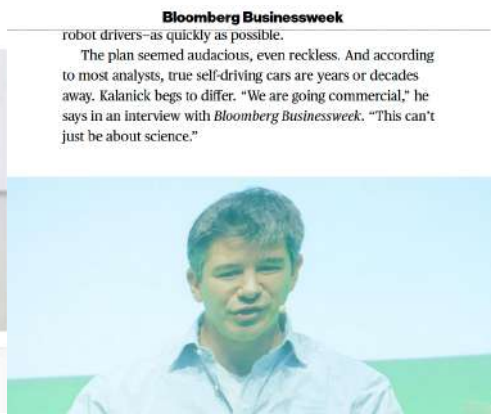
Obstacle III. dichotomies

Not a problem per se but...programming as an undecided:

"*And if one got to thinking that something like [programming] far from being governed by these oppositions, opens up their very possibility without letting itself be comprehended by them [...] if [...] one got to thinking that [programming] cannot be subsumed under concepts whose contour it draws, leaves only its ghost to a logic that can only seek to govern it insofar as logic arises from it one would then have to bend into strange contortions what could no longer even simply be called logic or discourse. [...]* **One must accept the fact that here, for once, to leave a ghost behind will in a sense be to salvage nothing**."(Derrida 1972)

Obstacle III. dichotomies

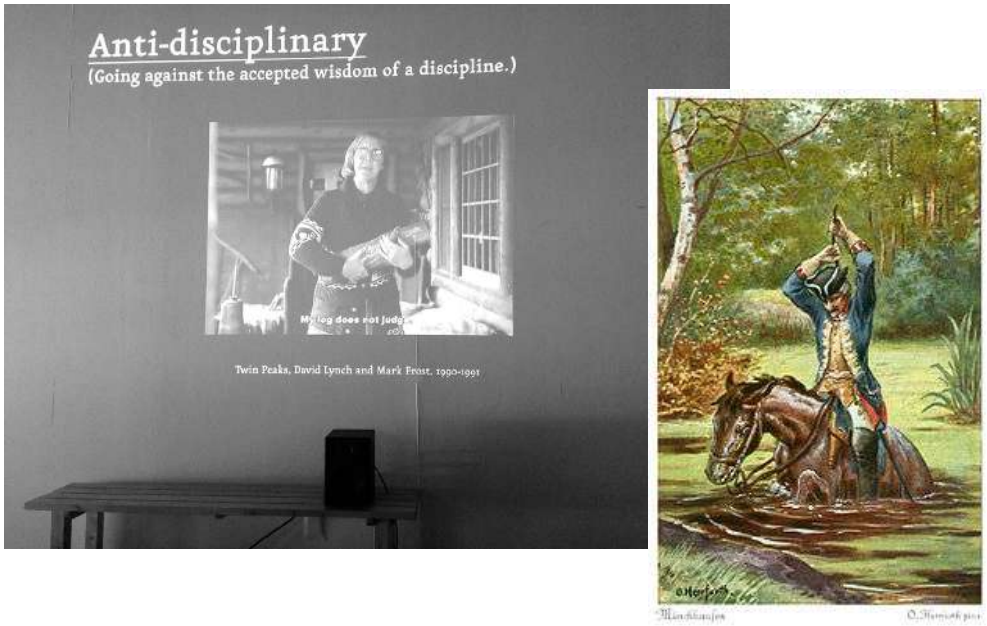
Reductive *and* dichotomal thinking around an undecided $\Longrightarrow$ gaps (communication, ideological, methodological, etc)



- Disconnected insights into what a program is - no shared methods, values or insights
- Separation of concerns

$\Rightarrow$ Reduces the main environments – e.g. Society as User; Technology as Commerce and Science as Technology's servant

a Proposal and a Discussion



... to be *anti-* and un-disciplining about programs

... requires to pull yourself out of your disciplinary confort zone

... history and philosophy understood in general and so undisciplined way *can* help

a Proposal and a Discussion

What is a computer program?

⇒ A machine?

⇒ An art?

⇒ A science?

⇒ a technology?

⇒ etc

"*The closer we come to the danger, the more brightly do the ways into the saving power begin to shine and the more questioning we become. For* **questioning is the piety of thought**." (Heidegger 1954/1977)

Thank you