

Codework position paper
4.5.2008

I'm going to preface my position paper by responding directly to C's question re. programmability.

Although with W, I believe that the etymology is important, I also believe that changes in a word's value are key.

So, **we can argue that a recipe is a program**—and this brings up all kinds of interesting + important questions of programming and gender—but **it wasn't until the emergence of "software" as we now know it that we would conceive of recipes in this way:**

- With software (what would become software—importantly, the engineers working on computing machines did not foresee the importance or existence of software), program would move from something like this

<slide>

- A descriptive text explaining the events to follow to this:

<show perl program>

- A text filled with commands that allegedly drive the machine
- **Descriptive to something prescriptive**

For me, this change in meaning is linked to **electronic, discrete computation. EDC:**

- Makes it possible to solve difficult problems using numerical methods
- Quickly + error-free (in comparison to human and analog computers)

This means that you could use iterative numerical methods to solve PDEs **b/c noise—which introduced small uncertainties into every analog computer calculation—**

could be bracketed by turning large quantities into discrete signals

<s>

This allowed a certain return, as Turing argues, to a Laplacean universe in which cause and effect are clear and traceable

- **But importantly, what it also made impossible is the kind of elegant integration a differential analyzer could do**

Regardless, once you have this in place, **you can store instructions, as you would data, with the relative confidence that they will “run” as you wish them to**

+ increasingly + bizarrely, SW becomes the machine itself:

- the wonderfully and barely programmable world of the machinic gets erased and **disciplined through the imposition of an inhumanly precise clock that makes it possible for these devices to be read as digital**

Indeed, the question the book I’m now completing, *On Programmability*, is struggling with is:

- **How do we understand the emergence of code—ever increasing layers of SW during the period of 43-63?**
- **Link this emergence to the return to laplace in both Turing and Schroedinger?**
- **That is,** Understand the relations of causality and time within and outside of what would become new media?
- Trilogy: *The Times of New Media*.

OK. so to get specifically at humanly written code for machines and humans, let me address the logos of source code:

<s>

1. logos of source code (2 parts)

- logos of source code is the conflation of word with result, logos with action.

source code, that is, is based on the conflation of word with process, a conflation that makes SC allegedly the source of every action.

To offer you one example that encapsulates this nicely—an example I've used elsewhere—consider Edsger Dijkstra's famous condemnation of go to statements.

In "Go To Statement Considered Harmful," Dijkstra argues, <slide>

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

This is because go to statements and here's an example of one, go against the fundamental tenant of what Dijkstra considered to be good programming, namely:

<slide>

the necessity to "shorten the conceptual gap between static program and dynamic process, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible."

That is, go tos make difficult the conflation of instruction with its product—the reduction of process to command—which grounds the emergence of software as a concrete entity and commodity:

- as something separate from hardware
- as the product of programming
- (so, with the emergence of software, the product of programming moves from a working computer to a working program)

Gotos make it difficult for the source program to act as a legible source, for the program to act as a source when one debugs:

- gotos make the difference between space and time apparent

That is:

<s>

make it clear that **Source code only becomes (can only become) source after the fact.**

<s>

Source code is more accurately a **re-source**, rather than a source.

So, historically, source code came after object code, although not quite true:

- only with introduction of compilers, that you had both source and object code
- early programmers would debug object, rather than the so-called "source"
- getting programmers to treat source code as source not easy

Also, if you think about it:

- Alpha-numeric code only becomes a source after some chunk of corresponding machine code has executed.
- Further, not all "source code" is run. Rather source code is compiled into machine code and then lines of the program read in as necessary
- It can only be a source, after it's run

<s>

So, source code may be written in the present tense, but its temporal reference is simultaneously the past and the future (conflation of the two)

- The present, executability—as Phillippe showed yesterday—complicates this conflation

Importantly, **the notion of source code as source coincides with the introduction of alpha-numeric languages:**

- Source code is arguably symptomatic of human language's tendency to attribute a sovereign source to an action, a subject to a verb.

Source code is thus a form of

<sourcecery>

that fixes, places digital media by making code the source of our actions and memory a form of storage.

<s>

Source code is a fetish:

An inanimate object worshipped by on account of its supposed inherent magical powers, or as being animated by a spirit.

Our belief in SW is so strong that we sometimes forget that we don't need SW in order for even a discrete electronic computer to run:

Hardware not simply this:

<standard D flip flop>

But also this:

<branching>

You can put in place an if-then statement using only gates

But, to be clear, this is not an argument that hardware is somehow outside a logic that conflates word with action.

this fixing of the source, this substitution of proposition for action, happens when we think of computers schematically

<slide>

2: In the Beginning was the word

<c>

John von Neumann's mythic, controversial and incomplete 1945 "The First Draft Report of the EDVAC" introduced the concept of stored program computing—and memory—to the U.S. military / academic "public."

A hallmark of this report is its abstractness:

- rather than describing actually existing vacuum tubes and mercury delay lines, von Neumann used "ideal elements."

These **ideal elements were**, strangely enough, not drawn from Shannon, **but rather biology**, namely **Warren McCulloch and Walter Pitts' idealized cyberneticized neurons**, which were themselves inspired by what would be called "Turing machines"

These idealized neurons were themselves based on conflation of stimulus with action, word with result:

That is, M + P sought to create

<s>

"A Logical Calculus of the ideas immanent in nervous activity"

through an axiomatic conflation of word with result.

Through their assertion that:

<c>

"the response of any neuron [is] factually equivalent to a proposition which proposed its adequate stimulus."

- Event can be represented by its conditions of possibility
- Without this assumption, their schematization of the nervous system was impossible

SN: perhaps fitting that JvN, dying of a cancer linked to the nuclear weapons he helped produce, spent the last days reciting Goethe's *Faust* from memory

Not just b/c Faust crossed out word for action, but also because you **need some form of storage for there to be stored program computing**

+JvN famously called his storage a memory organ, rather than a store, term used at the time

Importantly, by calling certain parts of the computer a “**memory organ,**” von Neumann was asserting—against biological evidence—that **such an organ (localization) existed and that this memory was digital in form.**

<S>

That is, Von Neumann’s analogy between computer and human memory depends on a leap of faith—it is analogy to something, which he admitted over 10 years after the Draft, unknown but logically necessary.

His memory organ, which I don’t have time to go into now, conflates memory with storage:

- Does so by conflating the different live and dead hierarchies of memory

I bring this up b/c I want to conclude by arguing that memory is not storage—and thus the programmability we assume is natural, always there, something that is (must be) constantly created.

This difference between memory and storage clear in early forms of regenerative memory.

<slide>

The serial mercury delay line took a series of electrical pulses and used a crystal to transform them into sound waves, which would make their way relatively slowly down the mercury tube.

- At the far end, the sound waves would be amplified and reshaped.
- One tube could store about 1000 bits at any given moment.

The MDL could serve as memory b/c its signals could degenerate:

- **Degeneration makes memory possible, while simultaneously threatening it.**
- So the decay, RR talked about yesterday, happens at the level of both what you can and cannot see

Digital media, which is allegedly more permanent, more durable than other media (film stock, paper), depends on the degeneration we so actively deny and repress.

- This degeneration, which engineers would like to divide into useful and harmful (eraseability versus signal decomposition, information versus noise) belies the promise of digital computers as permanent memory machines.

memory stems from the same Sanskrit root for martyr, and the ancient Greek term for baneful, fastidious.

- Memory is an act of commemoration—a process of recollecting or remembering.

To forget this act of commemoration, to forget this effort is to be **haunted by the undead of information:**

<WWW>

- Shells of information that are archived but not remembered
- Undead information that returns, however, imperfectly to remind us of the irreversible nature of programmability