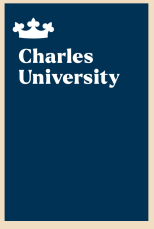
Hackers, engineers, managers, mathematicians and humanists

Cultures of Programming — Part 2

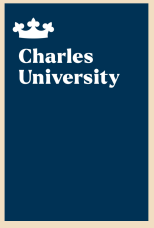
Tomas Petricek, Charles University, Prague

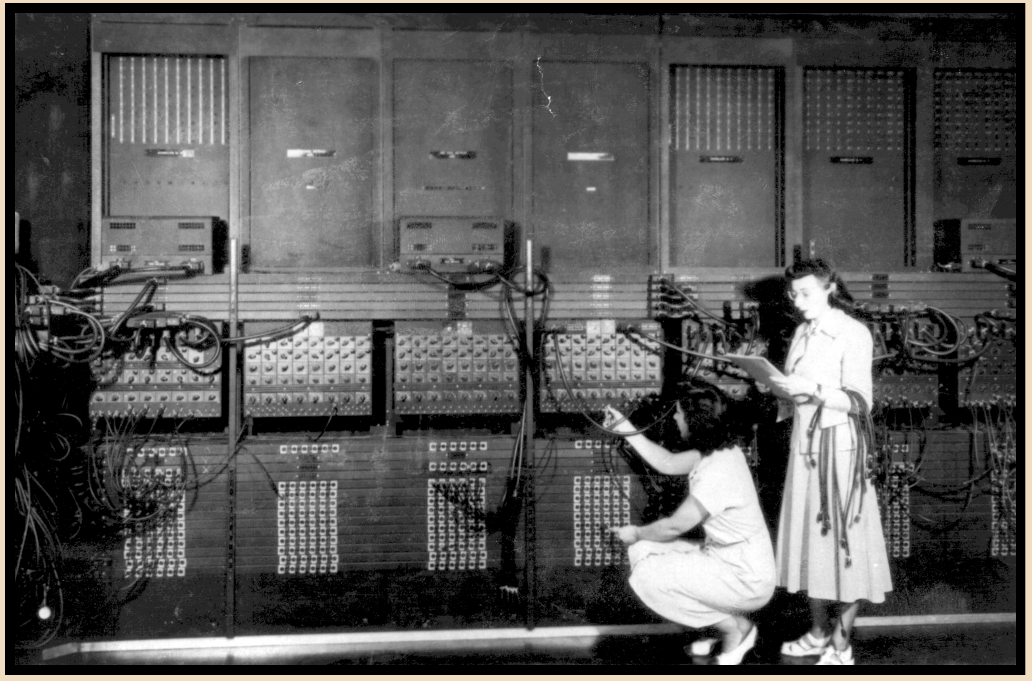
tomas@tomasp.net
https://tomasp.net
@tomasp.net



Mathematical

Program as a mathematical entity





Programming in the 1940s

Plugging cables to connect ENIAC units

Mathematics?

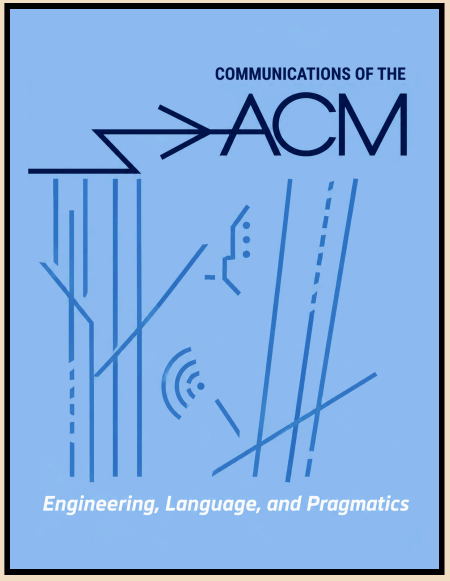
Mathematical problems
Numerical analysis

Charles University

Computing in academia

Letter to the editor (1966)

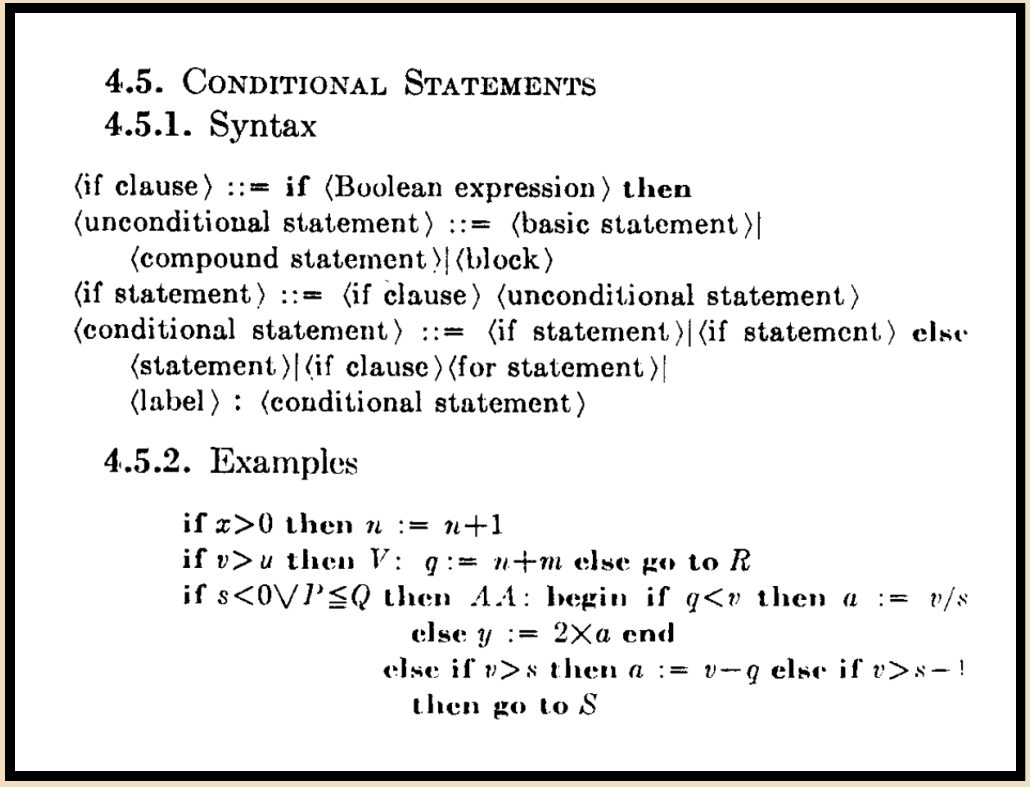
The creation of computer science departments is analogous to creating new departments for the railroad, automobile, radio, airplane or television technologies.



Mathematical science shift

- Program as a mathematical entity
- Analysis of program correctness
- Theoretical computer science

Charles University



4.5. CONDITIONAL STATEMENTS

4.5.1. Syntax

⟨if clause⟩ ::= **if** ⟨Boolean expression⟩ **then**
⟨unconditional statement⟩ ::= ⟨basic statement⟩|
⟨compound statement⟩|⟨block⟩
⟨if statement⟩ ::= ⟨if clause⟩ ⟨unconditional statement⟩
⟨conditional statement⟩ ::= ⟨if statement⟩|⟨if statement⟩ **else**
⟨statement⟩|⟨if clause⟩⟨for statement⟩|
⟨label⟩ : ⟨conditional statement⟩

4.5.2. Examples

if $x>0$ **then** $n := n+1$
if $v>u$ **then** V: $q := n+m$ **else go to** R
if $s<0 \vee P \leqq Q$ **then** AA: **begin if** $q<v$ **then** $a := v/s$
else $y := 2 \times a$ **end**
else if $v>s$ **then** $a := v-q$ **else if** $v>s-1$
then go to S

Algol (1958)

Language definition using formal grammar

Recognized as an object of beauty but implementation lagged

Reference language, publication language, & implementation language

Charles University

Algorithms

Established as a central object of computer science

Talked about, published

CERTIFICATION OF ALGORITHM 58
MATRIX INVERSION (Donald Cohen, *Comm. ACM 4*, May 1961)
RICHARD A. CONGER
Yalem Computer Center, St. Louis University, St. Louis, Mo.

Invert was hand-coded in FORTRAN for the IBM 1620. The following corrections were found necessary:

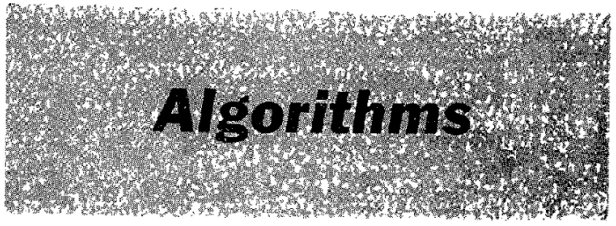
The statement $a_{k,j} := a_{k,i} - b_j \times c_k$ *should be*

$$a_{k,j} := a_{k,j} - b_j \times c_k$$

The statement **go to** back *should be changed to*

$$i := z_k ; \quad z_k := z_j ; \quad z_j := i; \quad \textbf{go to back}$$

After these corrections were made, the program was checked by inverting a 6×6 matrix and then inverting the result. The second result was equal to the original matrix within round-off.



G. E. FORSYTHE, Editor

ALGORITHM 223
PRIME TWINS
M. SHIMRAT (Recd 7 June 1963; in final form 2 Jan. 1964)
University of Alberta, Calgary, Alberta, Canada

```
procedure Prime Twins (t, Twin1, Twin2, Storage, Act);
  value Storage; integer t, Twin1, Twin2, Storage;
  procedure Act;
comment This procedure will generate successive "prime
  twins," i.e. pairs of primes Twin1, Twin2 which differ by 2.
  Storage is the maximum number of primes that can be stored.
  Act is any procedure for recording, examining, or utilizing each
  pair of twins as it is generated. t is a serial number for the
  twins. P[Storage] ↑ 2 is the last number examined;
begin integer array P[1: Storage]; integer j, m, previous,
    current;
  comment P[j] is the jth prime;
  P[1] := 2; P[2] := 3; j := 2; previous := 3; t := 0;
  for current := 5 step 2 until P[j] × P[j] do
  begin m := 1; for m := m + 1 while P[m] × P[m] ≦ cur-
    rent do
    if current = (current ÷ P[m]) × P[m] then go to NoPrime;
    comment If this point is reached, current is not divisible by
      any prime up to sqrt(current) and so is a prime. We now
      record the new prime, if storage permits, then check if it
      is the second of twins;
    if j < Storage then
      begin j := j + 1; P[j] := current
      end;
    if current = previous + 2 then
      begin t := t + 1; Twin1 := previous; Twin2 := current;
        Act (t, Twin1, Twin2)
      end;
    previous := current;
    NoPrime:
  end;
end procedure Prime Twins
```

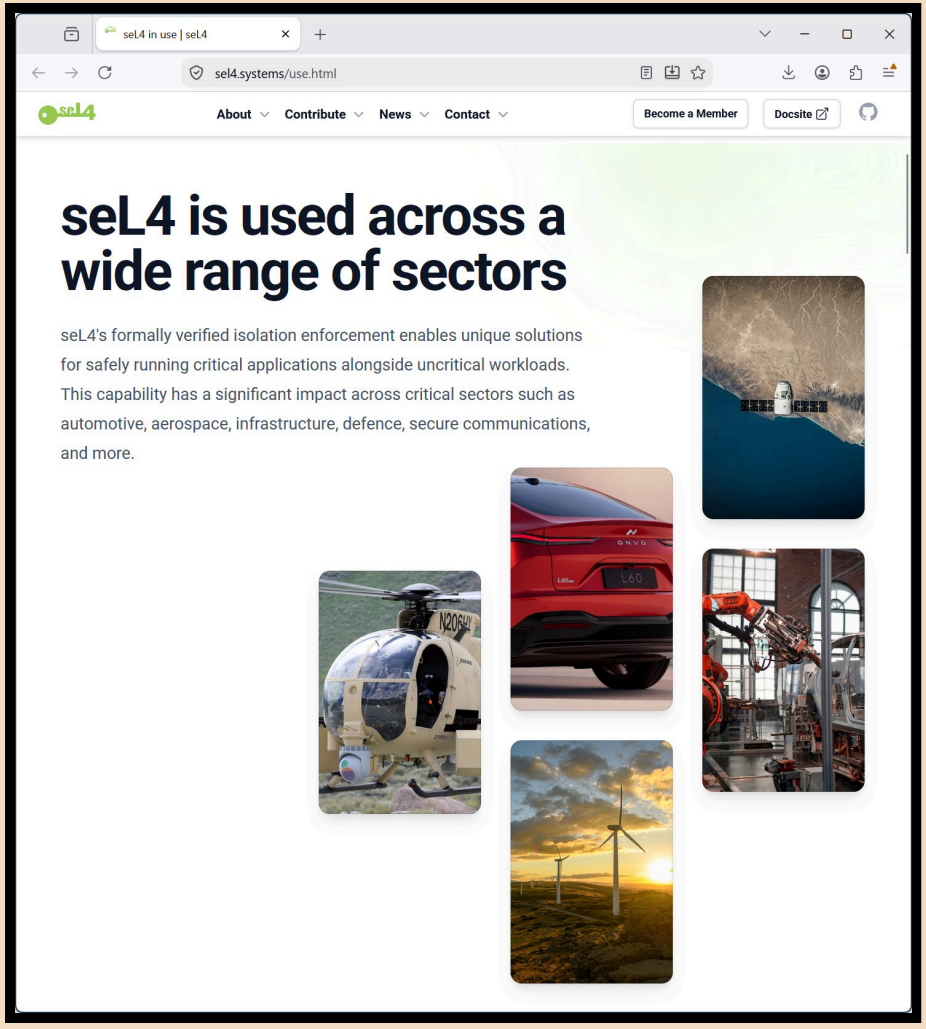
Charles University

Achievements

Mathematical culture today

- Formal language specifications
- Theory of programming languages
- Complexity theory of algorithms
- Type systems in Rust or TypeScript
- Real systems like CompCert and sel4

Charles University



Real-world verified software systems

No longer just a mathematical problem

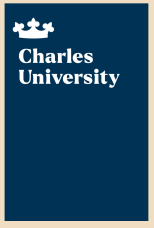
Requires large-scale engineering efforts

One culture is rarely enough!

Charles University

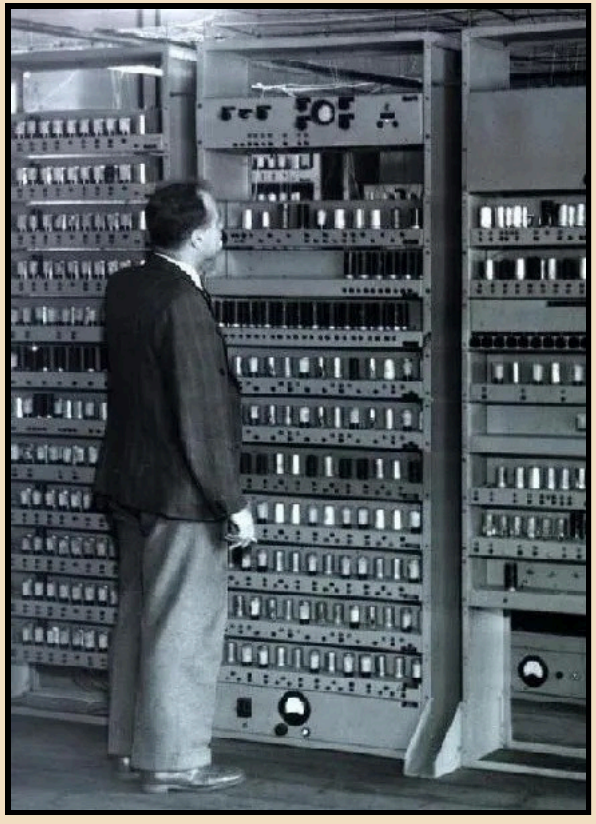
Hacker

The hands-on imperative



Early history of programming

Hands-on in the 1940s



Batch processing in the 1950s

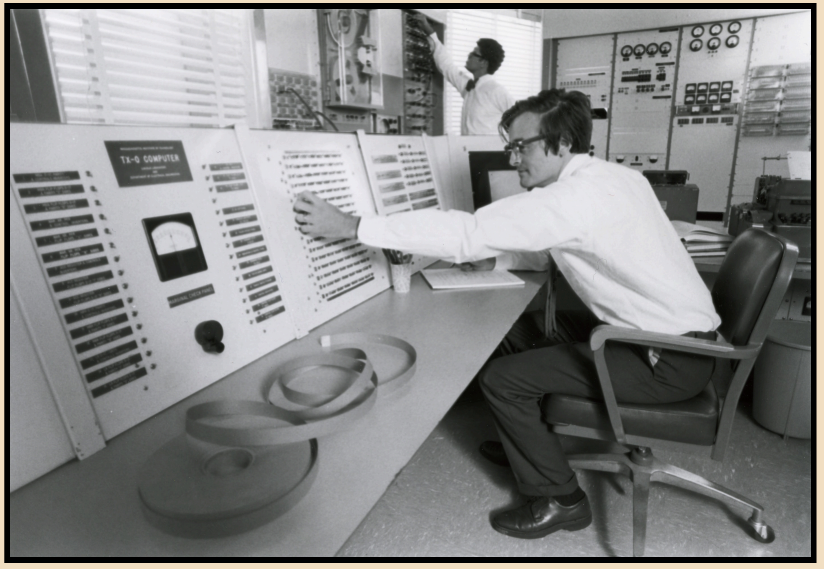


Charles University

MIT Hacker culture (late 1950s)

TX-0 and PDP-1 at MIT

- Test computer with transistors
- Donated to MIT RLE/AI lab
- Hands-on access possible



Hacker culture

- Playing around to
  test what it can do
- Tech Model Railway Club (TMRC) hackers

Charles University

Hacker culture

Typical projects

- **ARPANET** origins and used for information sharing
- **Spacewar!** game to test PDP-1 capabilities
- Interactive debuggers, interpreters and tools

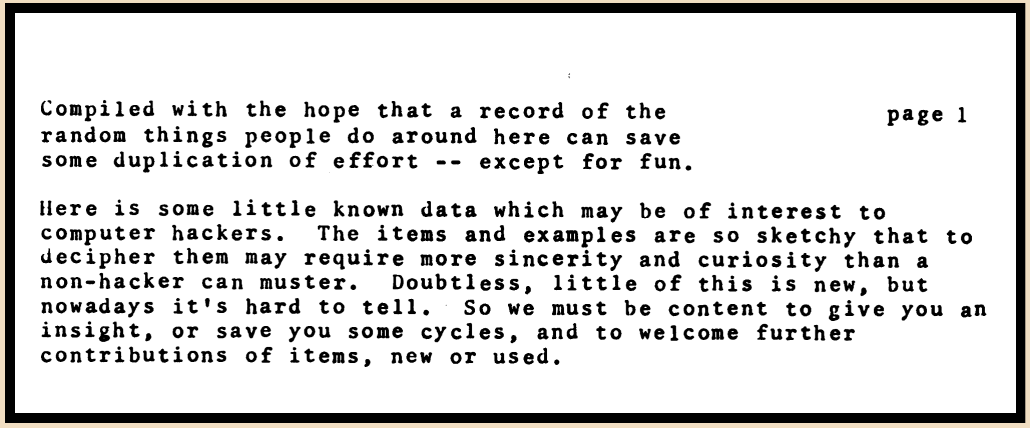


What the culture says

Only skill matters not "bogus criteria such as degrees, age, race, sex, position."

Charles University

HAKMEM



Compiled with the hope that a record of the random things people do around here can save some duplication of effort -- except for fun.

page 1

Here is some little known data which may be of interest to computer hackers. The items and examples are so sketchy that to decipher them may require more sincerity and curiosity than a non-hacker can muster. Doubtless, little of this is new, but nowadays it's hard to tell. So we must be content to give you an insight, or save you some cycles, and to welcome further contributions of items, new or used.

Typical hacker product

Information should be free! Idiosyncratic style, Informal tech report

"Submitted without further explanation or cautions"

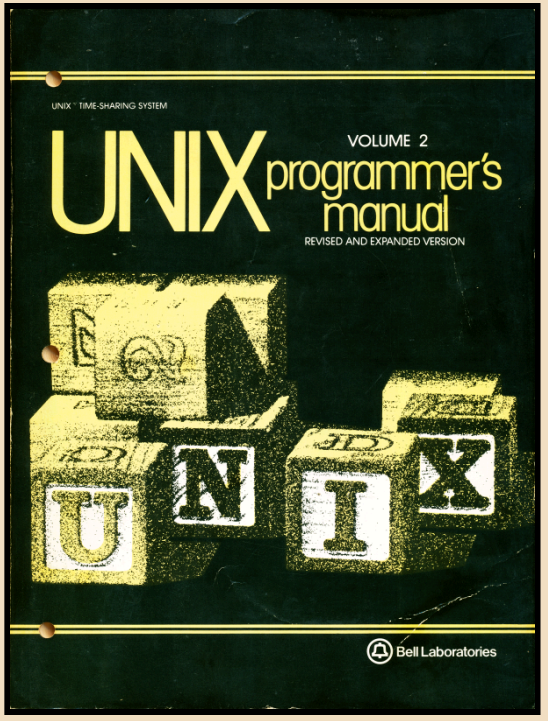
Topics from pseudo-Gaussian distributed random variable to an electrical circuit for an amplifier

Charles University

Technical and social achievements

UNIX and C

- Historical community links
- Lets can exploit machine understanding
- Values clever tricks and skills



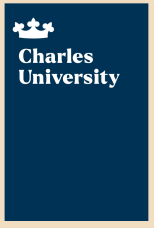
Free software

- Response to proprietary nature of UNIX
- Embodies hacker culture values
- More interesting and difficult history!

Charles University

Managerial

Software development lifecycles





Semi-Automatic Ground Environment

Started operation in 1955

Radar detection of soviet bomber nuclear strike

Largest software project at the time by far

Charles University

The problem

Managing teams not code

 Estimated 27 programmers grew to 2,000

 SDC spin-off trained 7,000 programmers

 Large-scale military and business projects

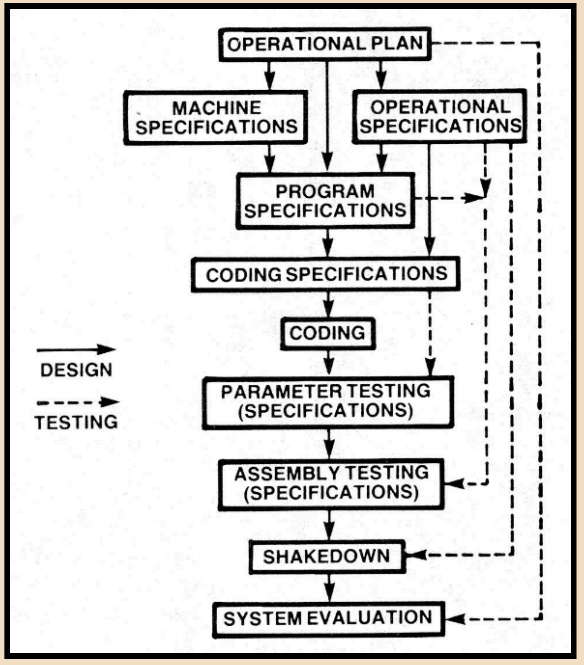
 Software development as a production line

Charles University

How to build software

Development process

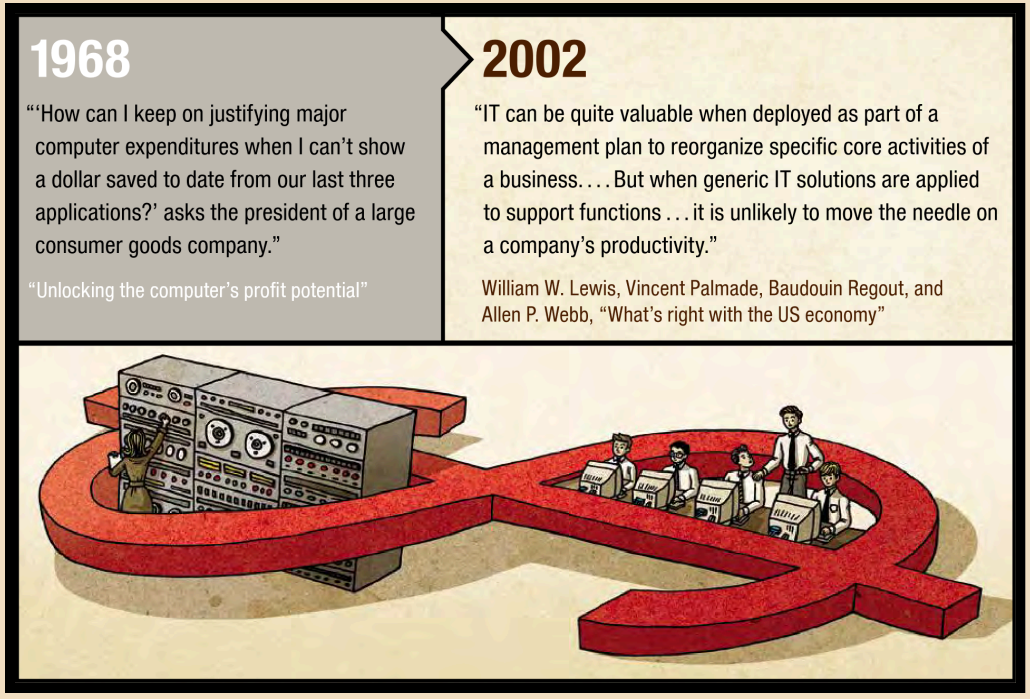
- Focus on specifications and process
- Aim for predictable costs and time
- Don't rely on individual skill



Avoiding risks

- Apollo mission guidance computer
- Bugs despite careful planning
- Risk of fixing higher than risk of mitigation!

Charles University



Unlocking the computer profit potential (1968)

Now that paper processes have been digitized, how can computers bring profit?

Mangers, not programmers, should lead!

Charles University

Engineering

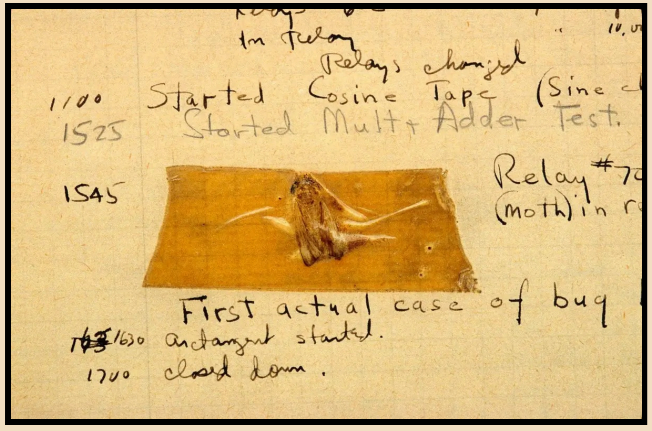
A Proper engineering discipline

Charles University

Software bugs

First actual bug (1947)

"Bugs" are small and easy to correct, not failures.



Y2K bug (2000)

Estimated $300bn spent fixing the expected "bug".



Charles University

NATO Conference on Software Engineering

Provocative title

"Black art of programming has to make way for the science of software engineering!"



New way of thinking

- Hackers do not have reliable methods
- Mathematics is too complicated (Algol 68)
- Follow-up 1969 conference considered a failure

Charles University

Engineering

Practices and values


Careful development planning


Rigorous testing and measurements


Documentation and use of tools


Overengineering and error handling

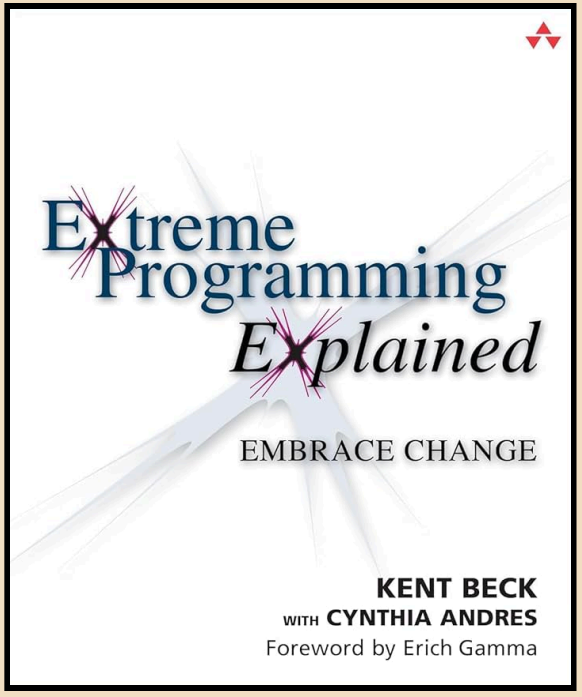

Professionalism and code quality

Charles University

Engineering methodology

Extreme programming

Reduce the cost of changes, shorter product cycles, direct contact with customer and their needs



Focused on code

- Customer part of the team
- Writing tests before programming
- Tests become product specification

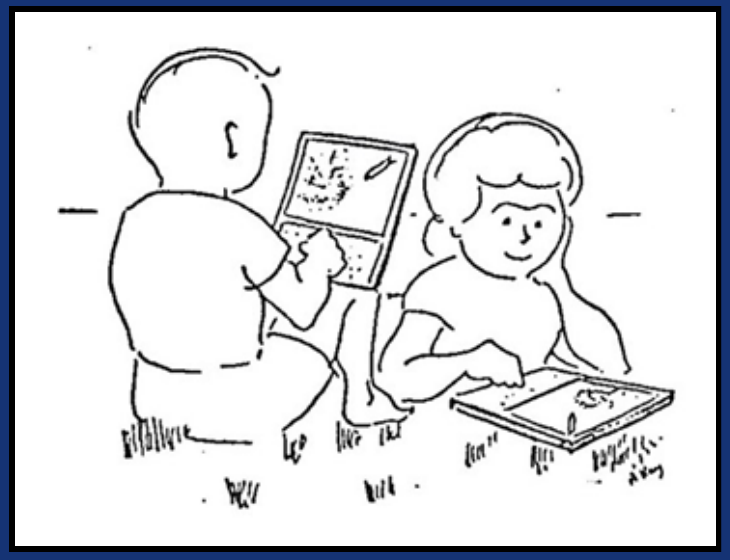
Charles University

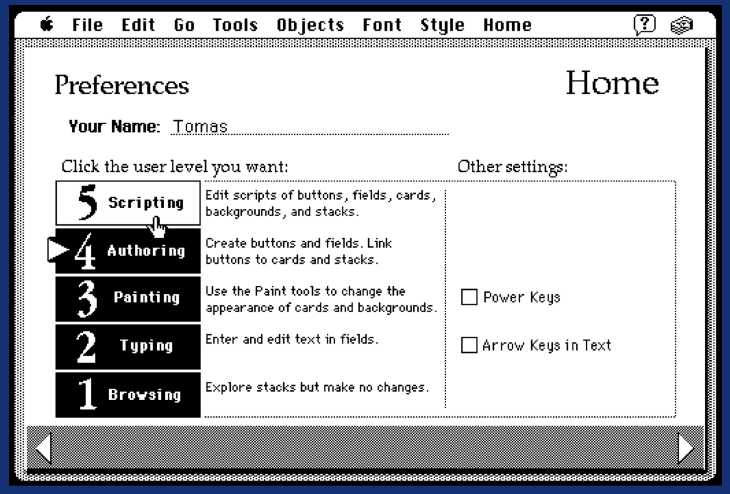
Humanistic

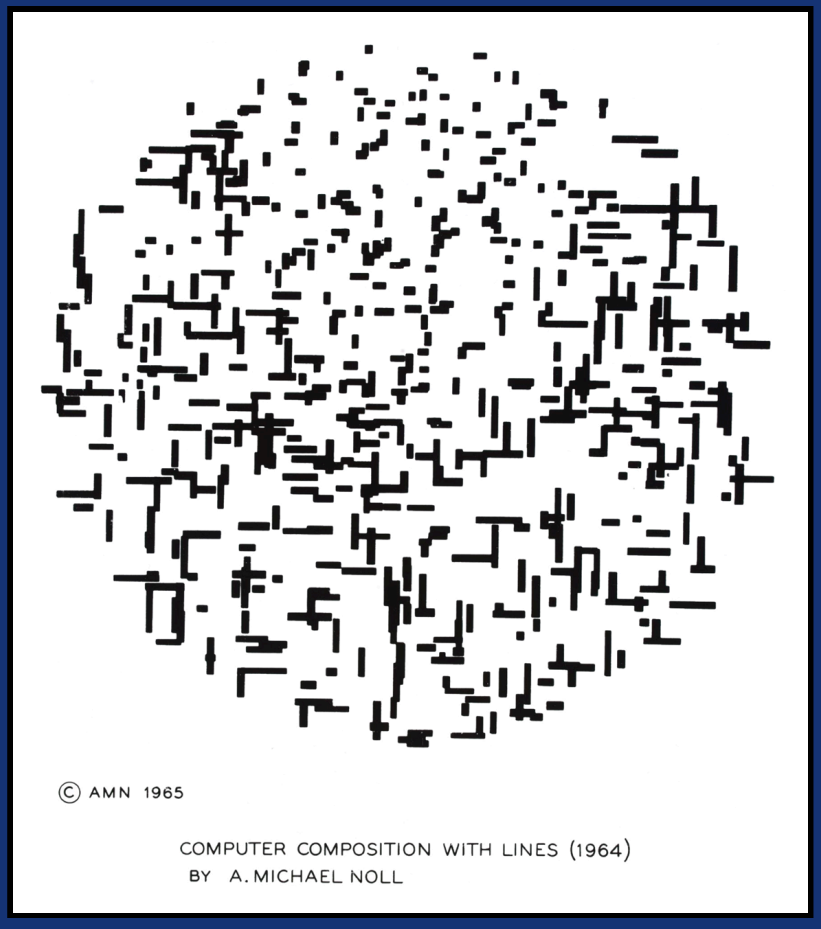
New Media for Thinking

Charles University

Focus on human, learning, thinking



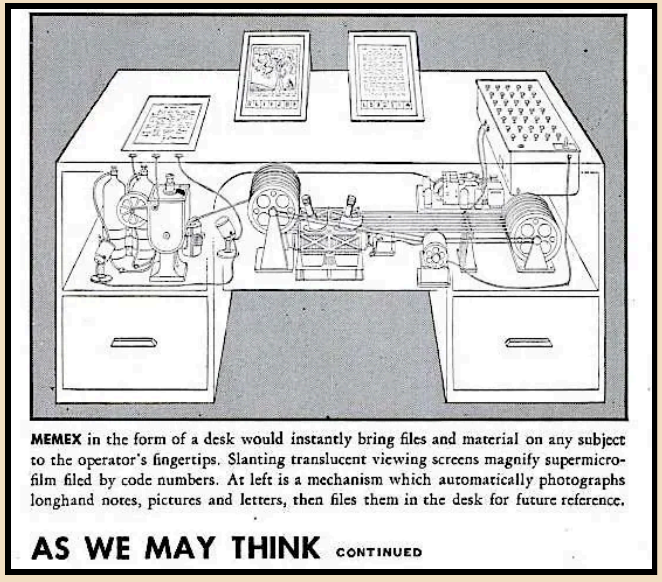




Two visions of computing

As We May Think (1945)

Memory machine making knowledge accessible



MEMEX in the form of a desk would instantly bring files and material on any subject to the operator's fingertips. Slanting translucent viewing screens magnify supermicrofilm filed by code numbers. At left is a mechanism which automatically photographs longhand notes, pictures and letters, then files them in the desk for future reference.

AS WE MAY THINK CONTINUED

Augmenting Human Intellect

Improve intellectual effectiveness of human beings (1962)



Programming as a new literacy

Logo programming language

- Known for turtle graphics
- Rethinking education from scratch
- Whole learning environment



Smalltalk language

- "A personal computer
  for children of all ages"
- Manage all user's informational needs
- Language for "talking" to the system

Charles University

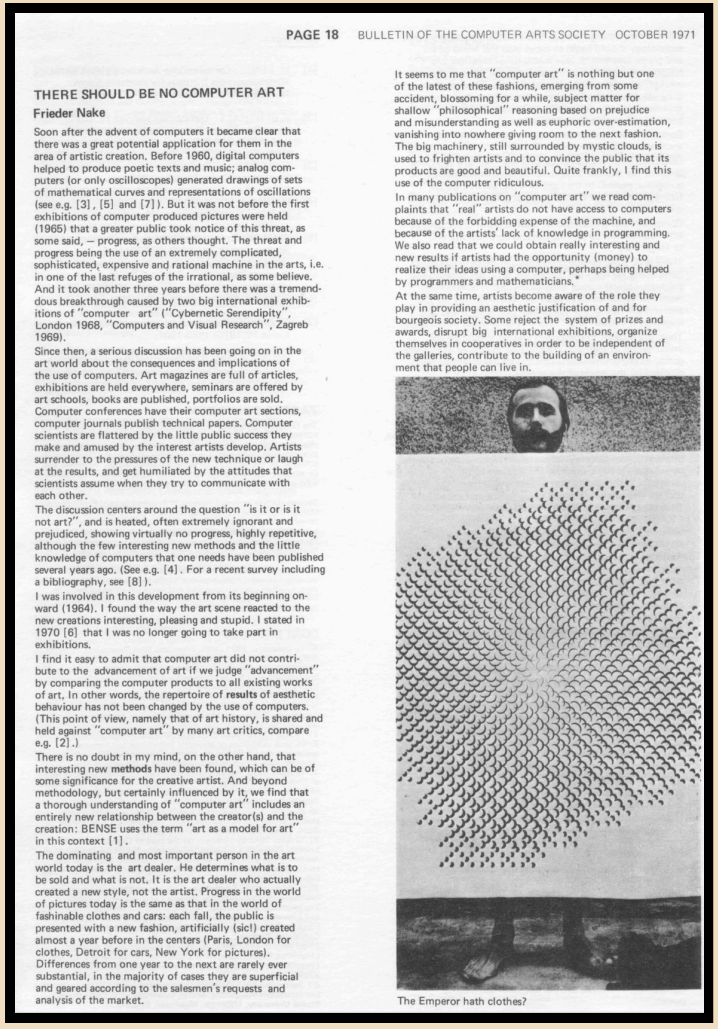


Computers to support creativity

The League of Automatic Music Composers (1970s)

Automatic music production with microcomputers (KIM-1)

Charles University



PAGE 18 BULLETIN OF THE COMPUTER ARTS SOCIETY OCTOBER 1971

THERE SHOULD BE NO COMPUTER ART

Frieder Nake

Soon after the advent of computers it became clear that there was a great potential application for them in the area of artistic creation. Before 1960, digital computers helped to produce poetic texts and music; analog computers (or only oscilloscopes) generated drawings of sets of mathematical curves and representations of oscillations (see e.g. [3], [5] and [7]). But it was not before the first exhibitions of computer produced pictures were held (1965) that a greater public took notice of this threat, as some said, – progress, as others thought. The threat and progress being the use of an extremely complicated, sophisticated, expensive and rational machine in the arts, i.e. in one of the last refuges of the irrational, as some believe. And it took another three years before there was a tremendous breakthrough caused by two big international exhibitions of "computer art" ("Cybernetic Serendipity", London 1968, "Computers and Visual Research", Zagreb 1969).

Since then, a serious discussion has been going on in the art world about the consequences and implications of the use of computers. Art magazines are full of articles, exhibitions are held everywhere, seminars are offered by art schools, books are published, portfolios are sold. Computer conferences have their computer art sections, computer journals publish technical papers. Computer scientists are flattered by the little public success they make and amused by the interest artists develop. Artists surrender to the pressures of the new technique or laugh at the results, and get humiliated by the attitudes that scientists assume when they try to communicate with each other.

The discussion centers around the question "is it or is it not art?", and is heated, often extremely ignorant and prejudiced, showing virtually no progress, highly repetitive, although the few interesting new methods and the little knowledge of computers that one needs have been published several years ago. (See e.g. [4]. For a recent survey including a bibliography, see [8]).

I was involved in this development from its beginning onward (1964). I found the way the art scene reacted to the new creations interesting, pleasing and stupid. I stated in 1970 [6] that I was no longer going to take part in exhibitions.

I find it easy to admit that computer art did not contribute to the advancement of art if we judge "advancement" by comparing the computer products to all existing works of art. In other words, the repertoire of **results** of aesthetic behaviour has not been changed by the use of computers. (This point of view, namely that of art history, is shared and held against "computer art" by many art critics, compare e.g. [2].)

There is no doubt in my mind, on the other hand, that interesting new **methods** have been found, which can be of some significance for the creative artist. And beyond methodology, but certainly influenced by it, we find that a thorough understanding of "computer art" includes an entirely new relationship between the creator(s) and the creation: BENSE uses the term "art as a model for art" in this context [1].

The dominating and most important person in the art world today is the art dealer. He determines what is to be sold and what is not. It is the art dealer who actually created a new style, not the artist. Progress in the world of pictures today is the same as that in the world of fashinable clothes and cars: each fall, the public is presented with a new fashion, artificially (sic!) created almost a year before in the centers (Paris, London for clothes, Detroit for cars, New York for pictures). Differences from one year to the next are rarely ever substantial, in the majority of cases they are superficial and geared according to the salesmen's requests and analysis of the market.

It seems to me that "computer art" is nothing but one of the latest of these fashions, emerging from some accident, blossoming for a while, subject matter for shallow "philosophical" reasoning based on prejudice and misunderstanding as well as euphoric over-estimation, vanishing into nowhere giving room to the next fashion. The big machinery, still surrounded by mystic clouds, is used to frighten artists and to convince the public that its products are good and beautiful. Quite frankly, I find this use of the computer ridiculous.

In many publications on "computer art" we read complaints that "real" artists do not have access to computers because of the forbidding expense of the machine, and because of the artists' lack of knowledge in programming. We also read that we could obtain really interesting and new results if artists had the opportunity (money) to realize their ideas using a computer, perhaps being helped by programmers and mathematicians.*

At the same time, artists become aware of the role they play in providing an aesthetic justification of and for bourgeois society. Some reject the system of prizes and awards, disrupt big international exhibitions, organize themselves in cooperatives in order to be independent of the galleries, contribute to the building of an environment that people can live in.

The Emperor hath clothes?

Frieder Nake (1971)

Computer art is becoming the latest fashion

Instead we should ask

Use computer to analyse aesthetics and its use

How computers alienate artists from their work?

Charles University

Cultures of programming

Hackers, engineers, managers, mathematicians and humanists

 Typical values and ways of thinking

 Applies to the past and the present

 Not intended as strict classification

 Interesting things happen when they meet

Charles University