What is software?

Why this topic

The Paper

NO SILVER BULLET –
ESSENCE AND ACCIDENTS OF SOFTWARE ENGINEERING

Frederick P. Brooks Jr.
Kenan Professor of Computer Science
University of North Carolina at Chapel Hill
New West Hall 035A
Chapel Hill, North Carolina 27514 USA

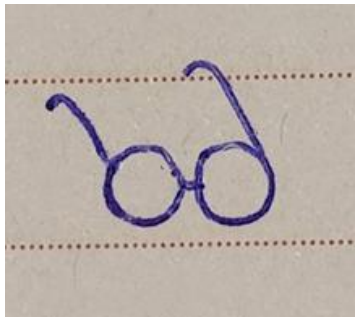
September 1986

The Paper

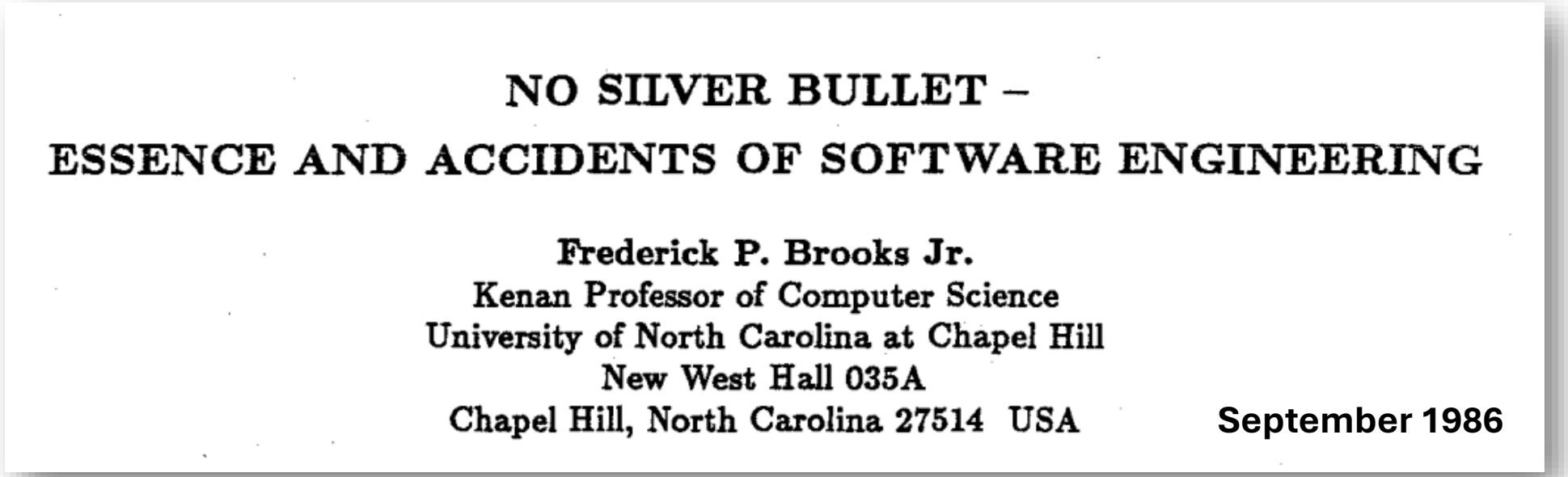
NO SILVER BULLET –
ESSENCE AND ACCIDENTS OF SOFTWARE ENGINEERING

Frederick P. Brooks Jr.
Kenan Professor of Computer Science
University of North Carolina at Chapel Hill
New West Hall 035A
Chapel Hill, North Carolina 27514 USA

September 1986



The Paper



NO SILVER BULLET –

ESSENCE AND ACCIDENTS OF SOFTWARE ENGINEERING

Frederick P. Brooks Jr.
Kenan Professor of Computer Science
University of North Carolina at Chapel Hill
New West Hall 035A
Chapel Hill, North Carolina 27514 USA

September 1986

Very practical question: can we get an increase of efficiency in software development similar to hardware?

„to see what **rate of progress** one can expect in software technology, let us examine the **difficulties** of that technology. Following Aristotle, I divide them into **essence**, the difficulties **inherent in the nature** of software, and **accidents**, those difficulties that today attend ist production but are not inherent.“

Essence and Accidents

Essence

Gives the thing its identity

Example: being unmarried for a bachelor

Therefore also: cannot be removed from the thing, without it becoming *something else* (will lose identity)

Accident

Everything else that still belongs to the thing

Example: hair color for a bachelor

Therefore also: can be removed from the thing

This is not like math!!

Essence and Accidents

Essence

Complexity

Conformity

Changeability

Invisibility

Accident

Representation

Slow turnaround

Integration

Essence and Accidents

„to the extent that the **high-level language** embodies the constructs one wants in the abstract program and avoids all lower ones, it eliminates a whole level of complexity **that was never inherent in the program at all**“

Accident

Representation

Slow turnaround

Integration

What is software?

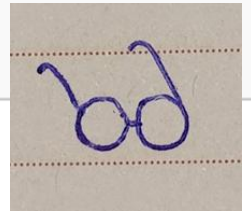
„The **essence** of a software entity is a construct of interlocking concepts: **data sets, relationships among data items, algorithms, and invocations of functions**. This essence is abstract in that such a conceptual construct is **the same under many different representations**.“

What would stay of this if you translated it to pseudocode?

```
template<class T>
typename std::enable_if<std::is_object<T>::value, T*>::type addressof(T& arg) noexcept
{
    return reinterpret_cast<T*>(
                &const_cast<char&>(
                    reinterpret_cast<const volatile char&>(arg)));
}

template<class T>
typename std::enable_if<!std::is_object<T>::value, T*>::type addressof(T& arg) noexcept
{
    return &arg;
}
```

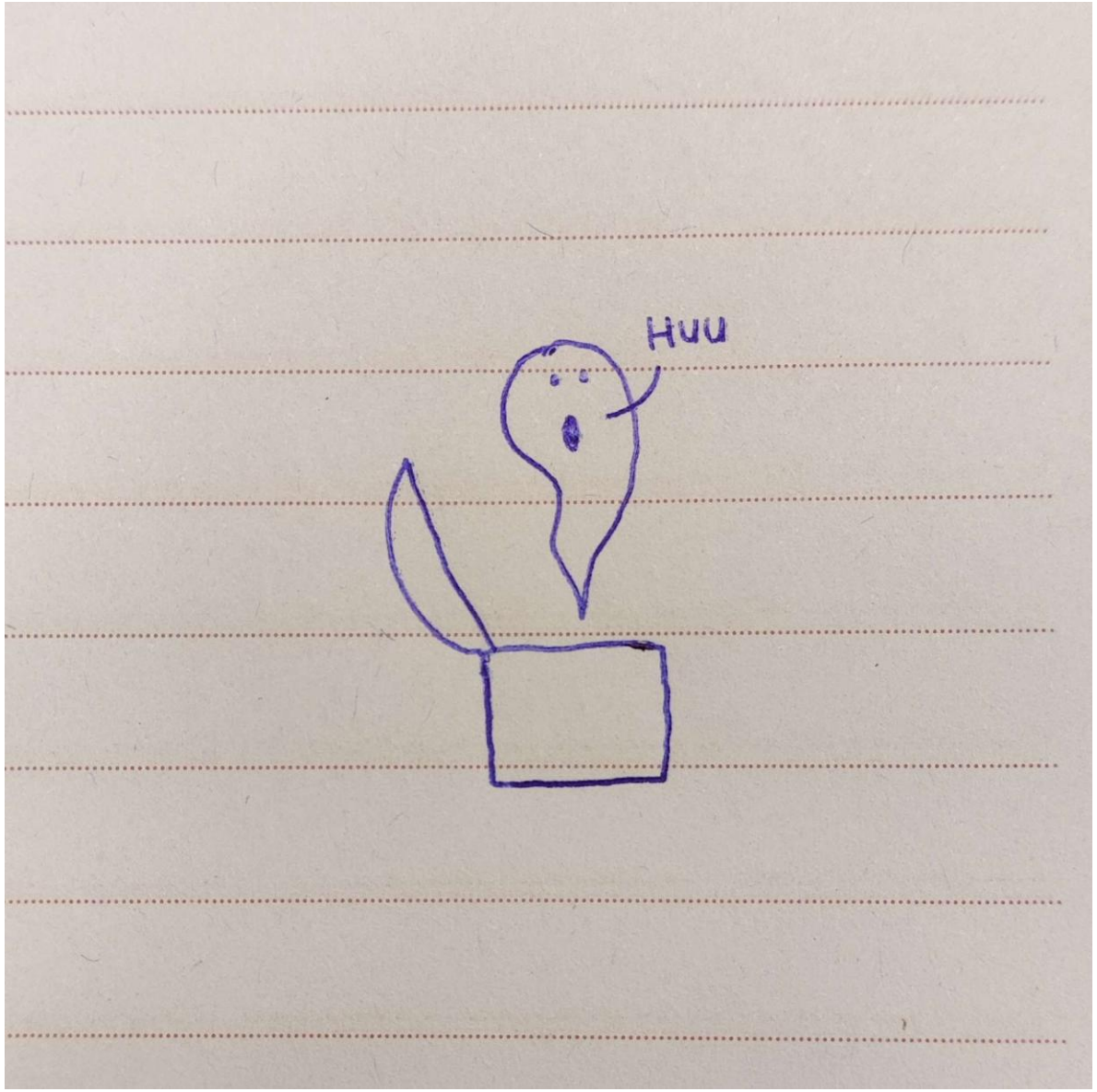
What would stay of this if you translated it to pseudocode?

```
template<class T>
typename std::enable_if<std::is_object<T>::value, T*>::type addressof(T& arg) noexcept
{
    return reinterpret_cast<T*>(
                &const_cast<char&>(
                    reinterpret_cast<const volatile char&>(arg)));
}

template<class T>
typename std::enable_if<!std::is_object<T>::value, T*>::type addressof(T& arg) noexcept
{
    return &arg;
}
```

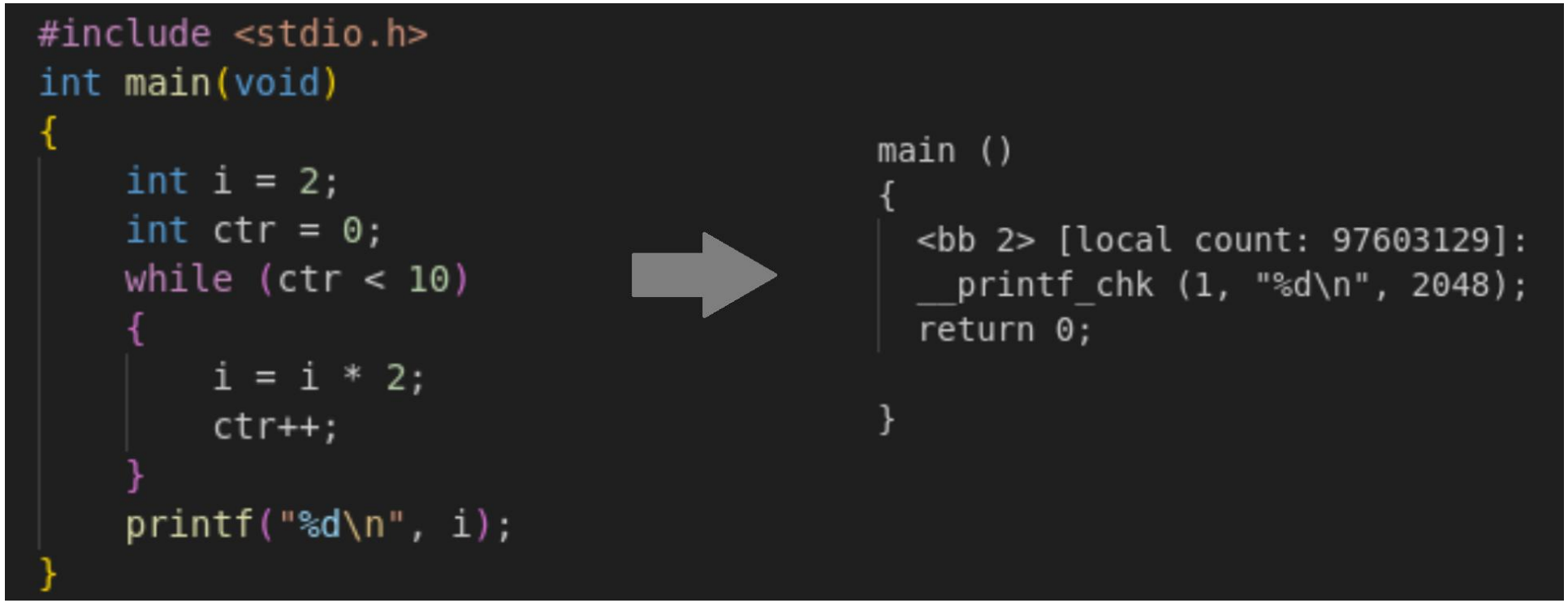


This was also not what I was used to (at all)

Because i spent 3 years in a completely different paradigm


Huu

is this really a good idea?


#include <stdio.h>
int main(void)
{
int i = 2;
int ctr = 0;
while (ctr < 10)
{
i = i * 2;
ctr++;
}
printf("%d\n", i);
}
main ()
{
<bb 2> [local count: 97603129]:
__printf_chk (1, "%d\n", 2048);
return 0;
}

everything that can be removed or changed cannot (by definition) be essential → we haven‘t found the essence yet! there must be something *more substantial* to the software than the logic

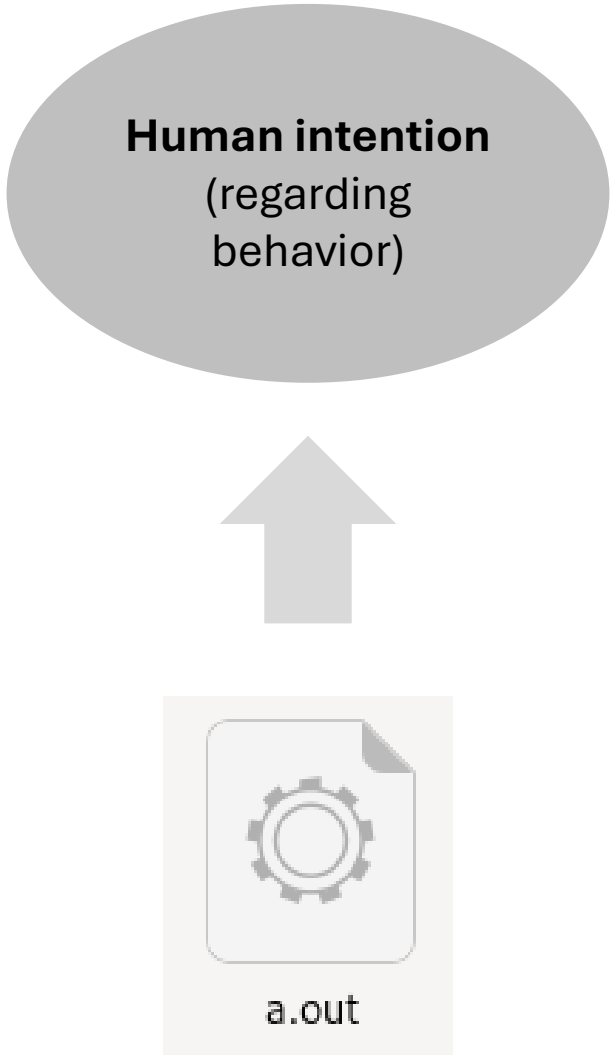
6 The least requirements on a conforming implementation are:

— Accesses to volatile objects are evaluated strictly according to the rules of the abstract machine.

— At program termination, all data written into files shall be identical to the result that execution of the program according to the abstract semantics would have produced.

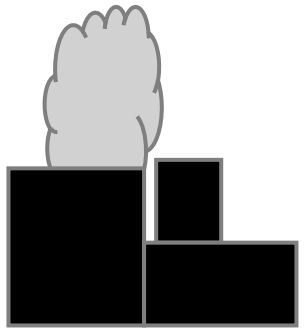
— The input and output dynamics of interactive devices shall take place as specified in 7.21.3. The intent of these requirements is that unbuffered or line-buffered output appear as soon as possible, to ensure that prompting messages actually appear prior to a program waiting for input.

This is the *observable behavior* of the program.

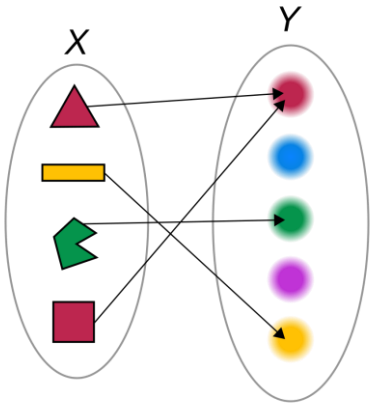

Human intention
(regarding
behavior)
a.out

this is the only thing that matters

Human intention (regarding behavior)

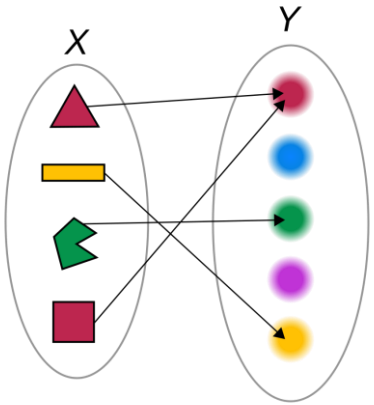


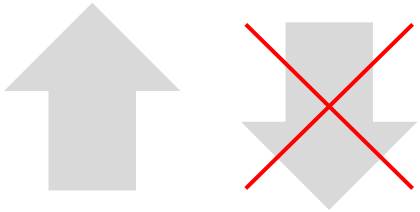
everything else (incl. logic & implementation) is in the way



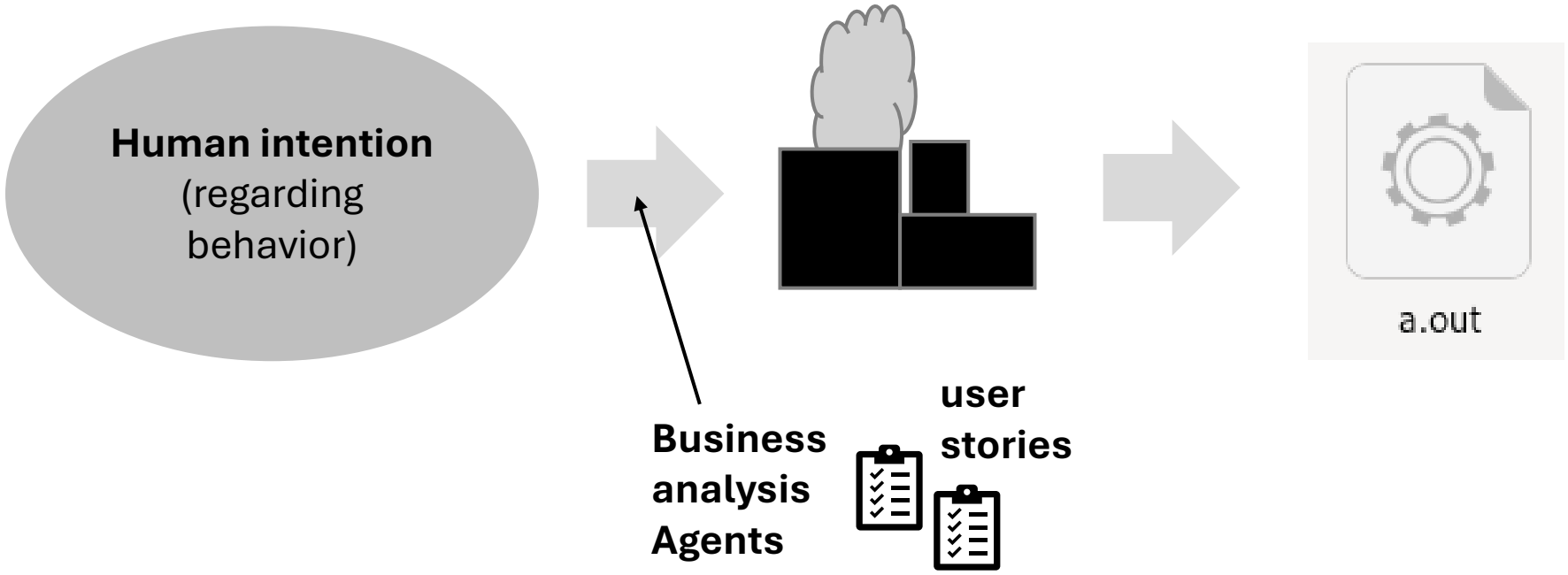
Human intention, but already super formalized

Looks a lot more complicated than this



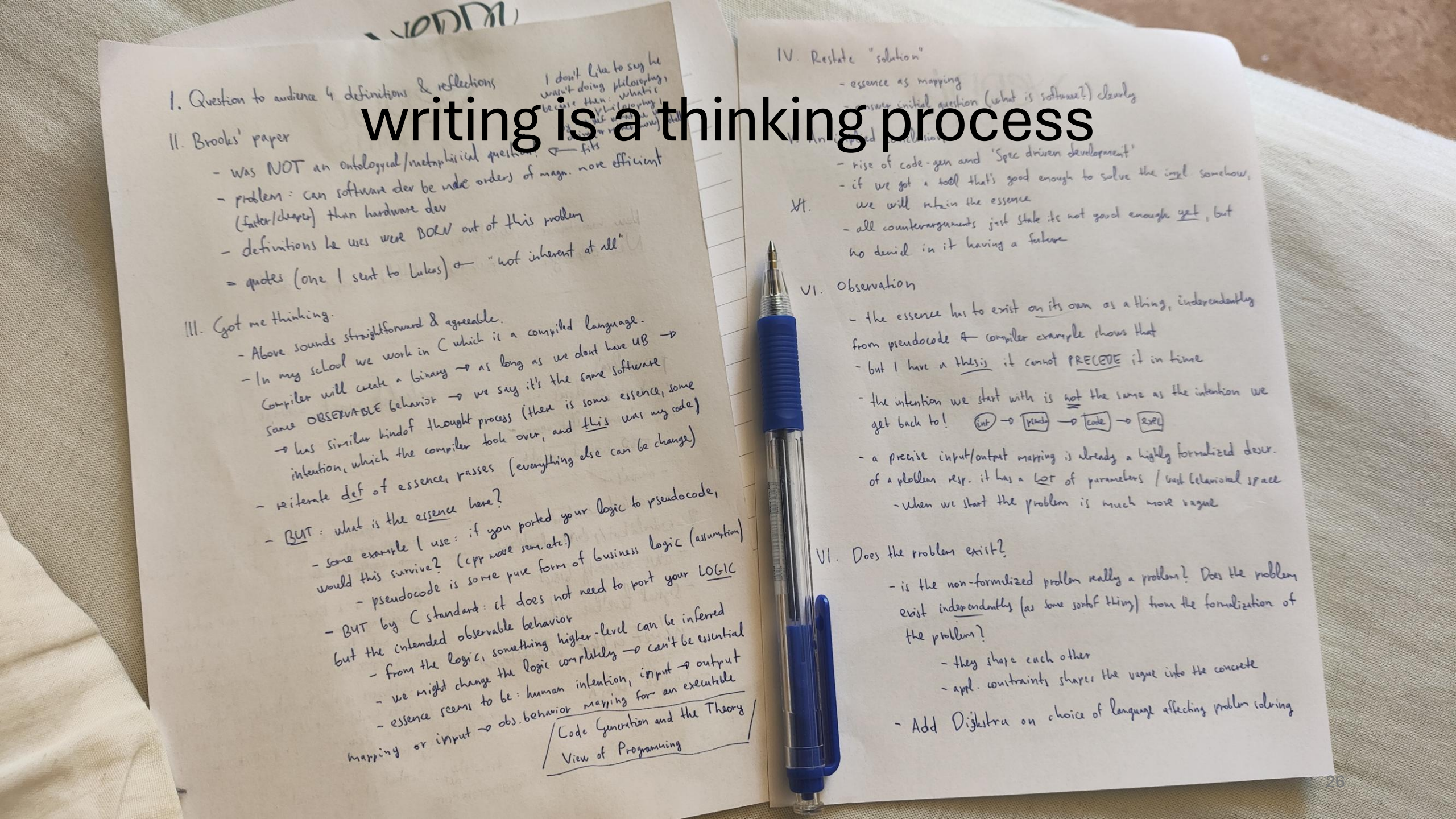


Can be **extracted** from the software, but cannot **precede** it


Human intention
(regarding
behavior)
a.out
Business
analysis
Agents
user
stories

the act of writing software is itself the process of shaping vague intention into a strictly formalized one

writing is a thinking process



so we arrived here

Essence and Accidents

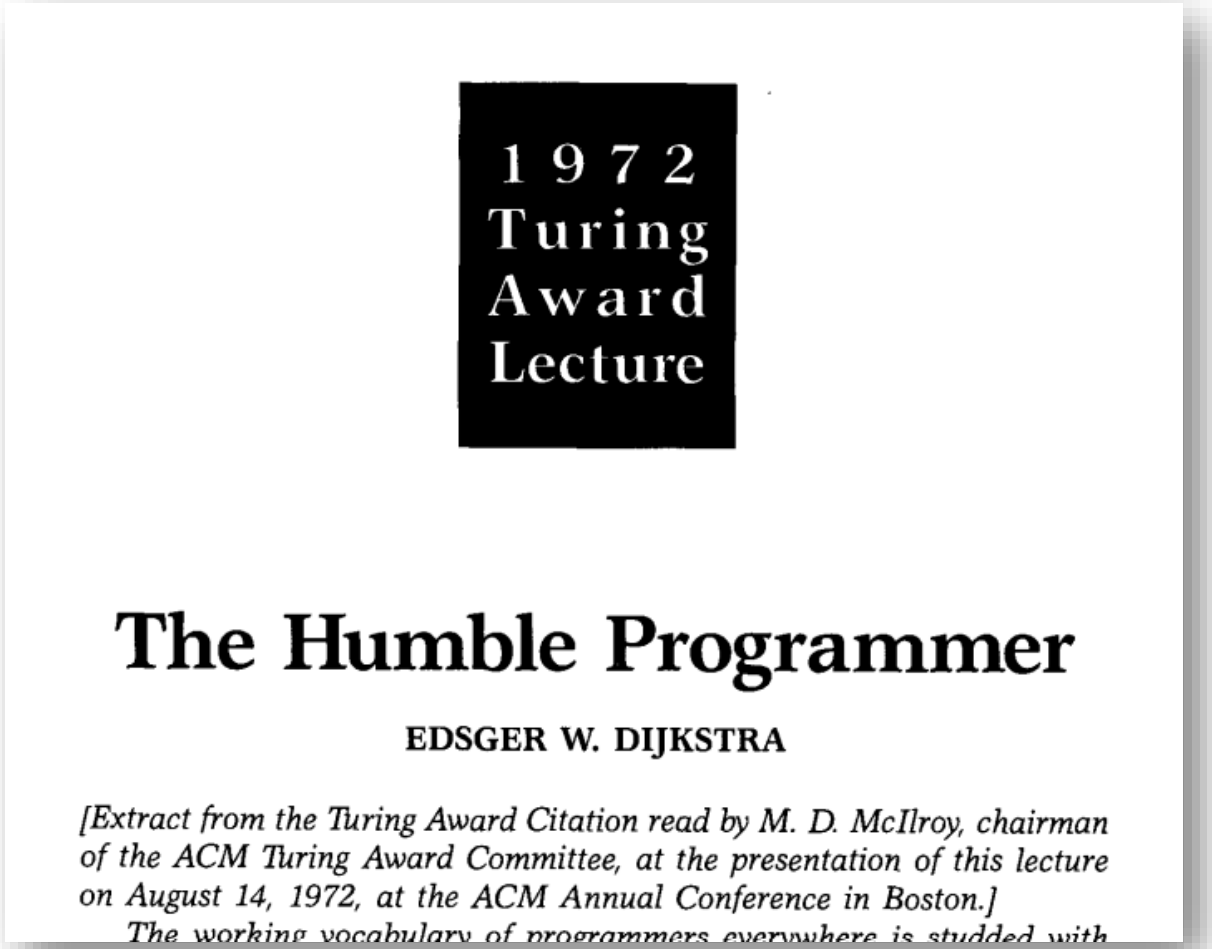
„to the extent that the **high-level language** embodies the constructs one wants in the abstract program and avoids all lower ones, it eliminates a whole level of complexity **that was never inherent in the program at all**“

Accident

Representation

Slow turnaround

Integration



he was also kind of in a crisis

„it has **assisted** a number of our most gifted fellow humans in thinking previously impossible thoughts“

Edsger W. Dijkstra on LISP

Programming experiment

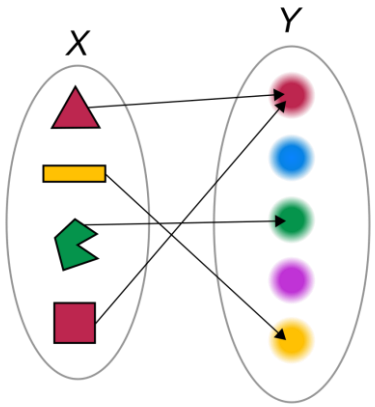
„their **notion of repetition** was so tightly **connected to an idea of an associated controlled variable** to be stepped up, that they were mentally blocked from seeing the obvious“

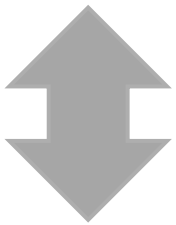
the flow of your logic was supposed to be independent of your language

can thinking be independent of language?

„the tools we are trying to use and the language or notation we are using to express or record our thoughts are the major factors determining what we can think or express *at all*“

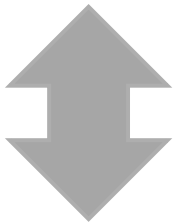
this was also Dijkstra





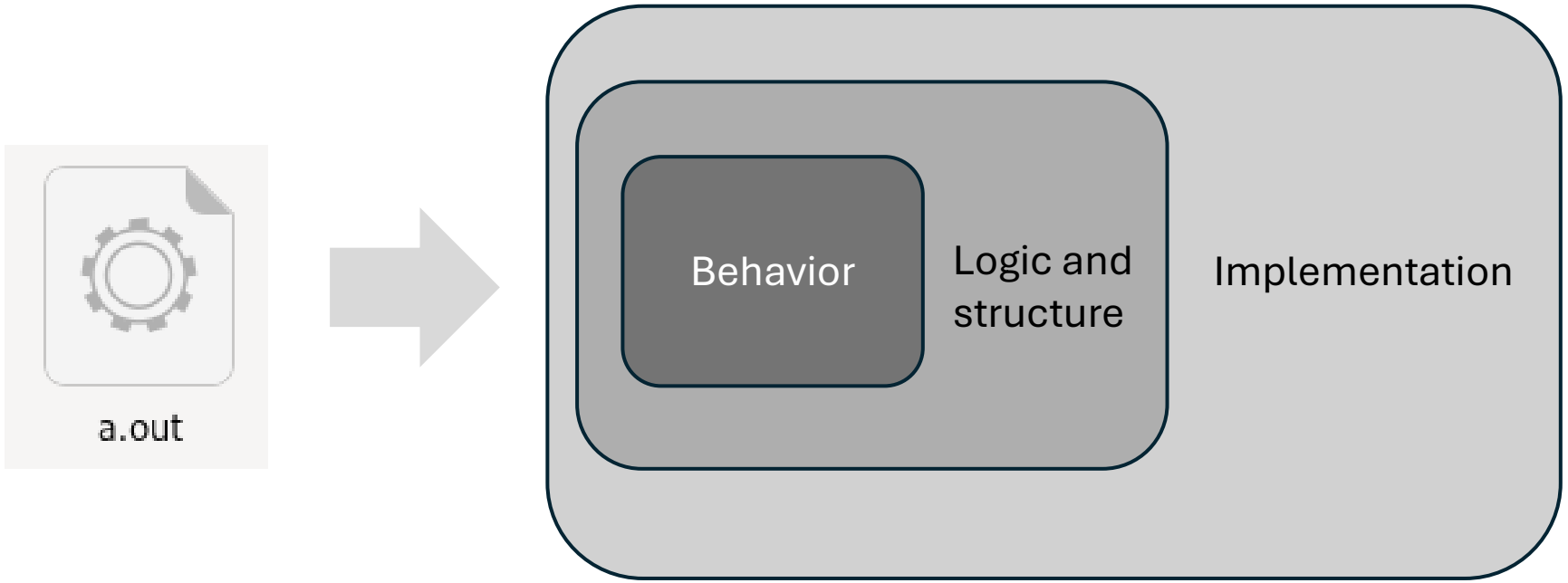
```
1 set rentals to [A, B, C]
2 set accepted to []
3 for rental in rentals
4   if not conflict (rental, accepted)
5     append rental to accepted
```


```
1 set rentals to [A, B, C]
2 set accepted to []
3 for rental in rentals
4   if not conflict (rental, accepted)
5     append rental to accepted
```

everything is shaped by everything –
seems like we have a web of very complex interactions

Checkpoint



Yet Another Paper

1985

PETER NAUR, PROGRAMMING AS THEORY BUILDING

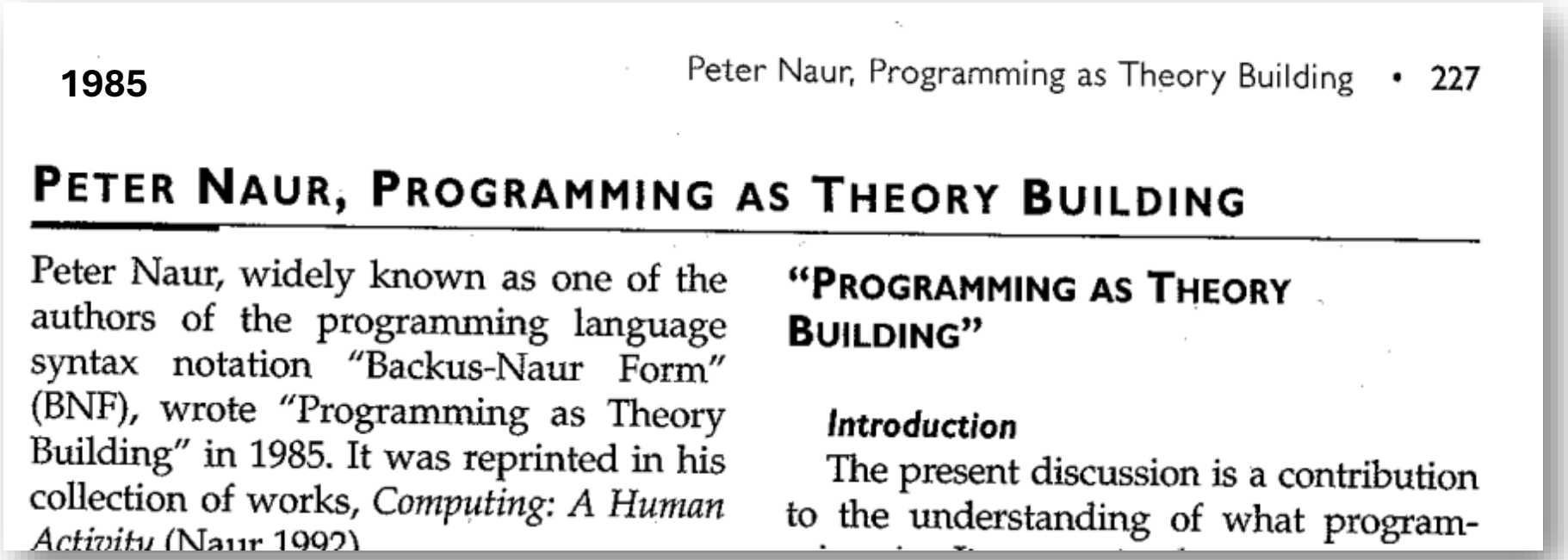
Peter Naur, widely known as one of the authors of the programming language syntax notation "Backus-Naur Form" (BNF), wrote "Programming as Theory Building" in 1985. It was reprinted in his collection of works, *Computing: A Human Activity* (Naur 1992)

"PROGRAMMING AS THEORY BUILDING"

Introduction

The present discussion is a contribution to the understanding of what program-

Yet Another Paper



1985

PETER NAUR, PROGRAMMING AS THEORY BUILDING

Peter Naur, widely known as one of the authors of the programming language syntax notation "Backus-Naur Form" (BNF), wrote "Programming as Theory Building" in 1985. It was reprinted in his collection of works, *Computing: A Human Activity* (Naur 1992)

"PROGRAMMING AS THEORY BUILDING"

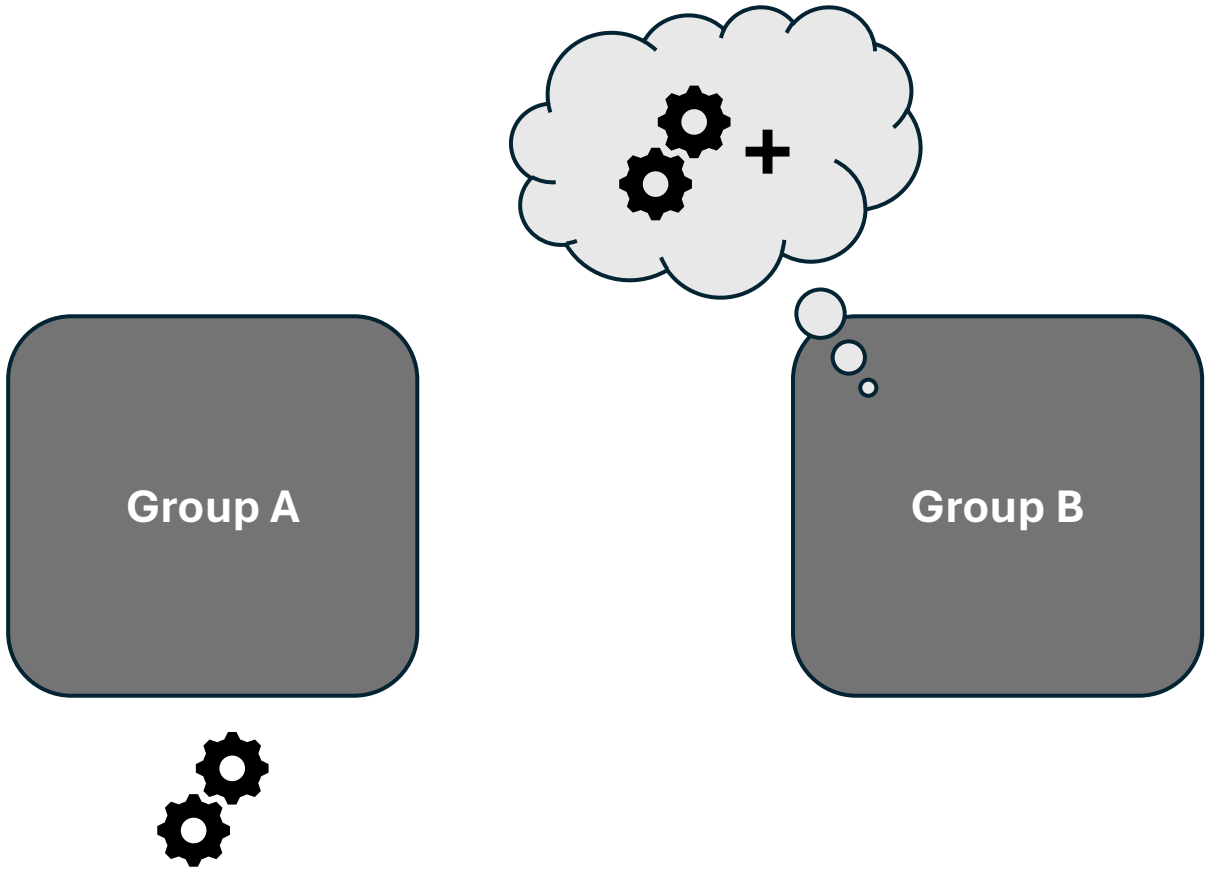
Introduction

The present discussion is a contribution to the understanding of what program-

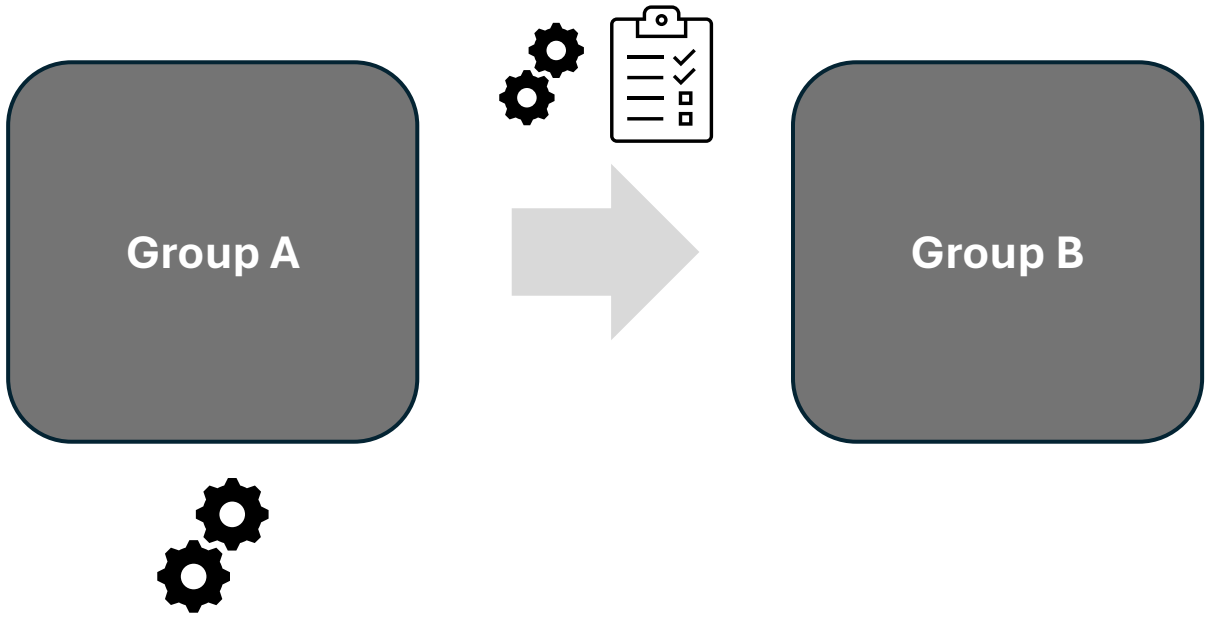
```
relative-ref  = relative-part [ "?" query ] [ "#" fragment ]

relative-part = "//" authority path-abempty
              / path-absolute
              / path-noscheme
              / path-empty
```

← This is augmented Backus-Naur form

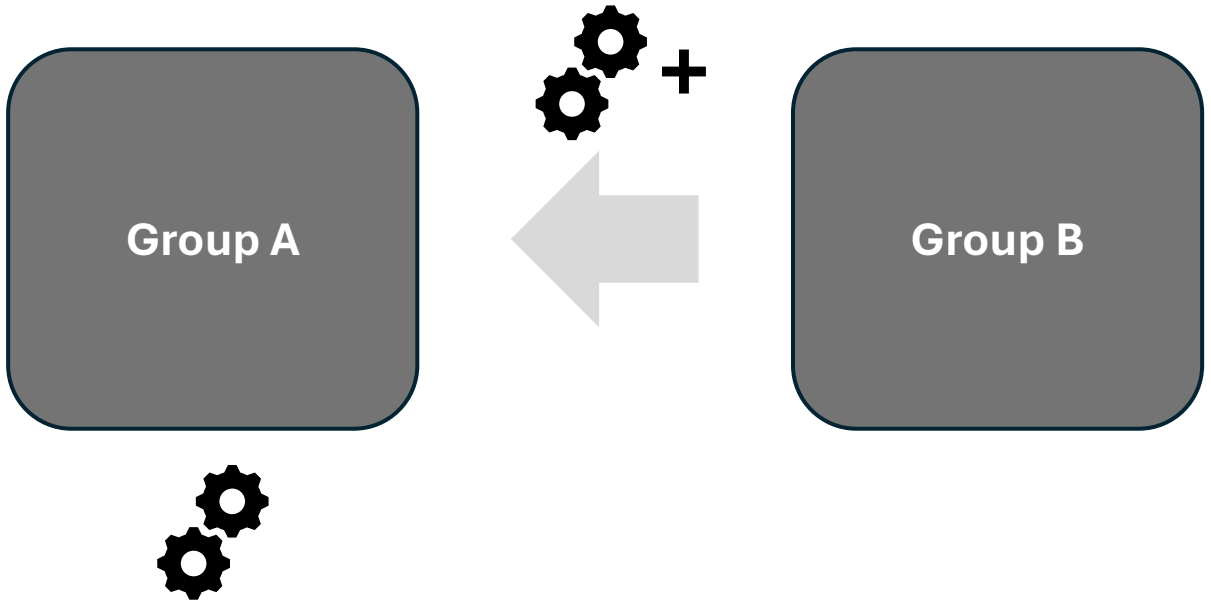
he had some things also that kept him thinking



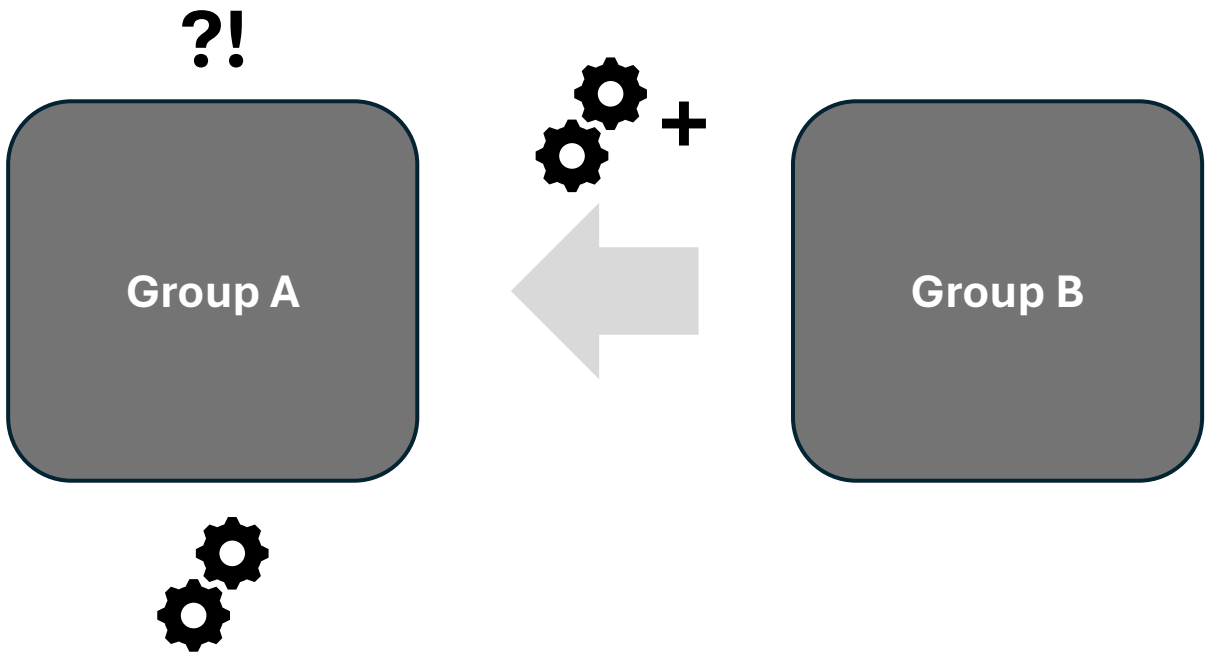
he had some things also that kept him thinking



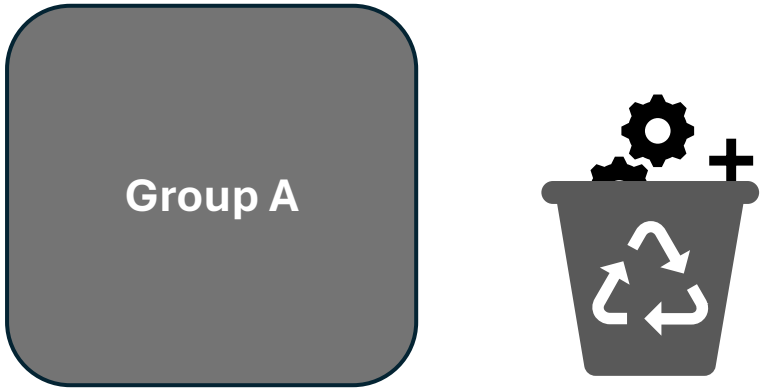
he had some things also that kept him thinking



he had some things also that kept him thinking

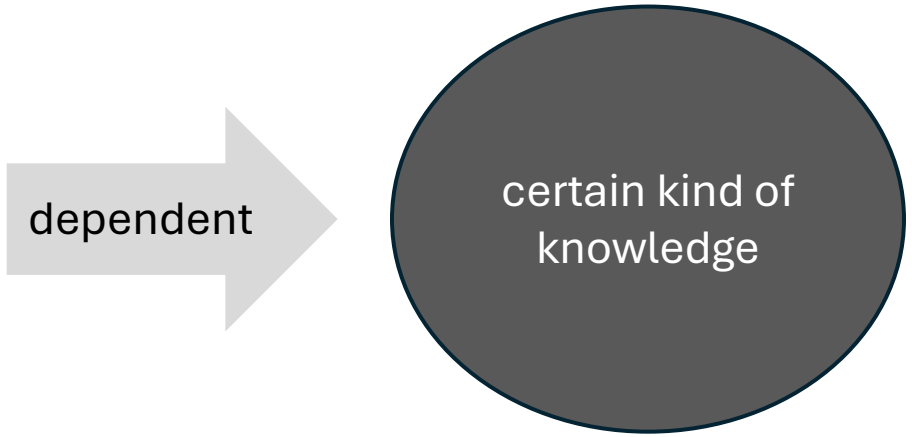


the solutions suggested by group B were „based on **additions** to that structure in the form of patches that effectively destroyed its power and simplicity.“



with large programs:

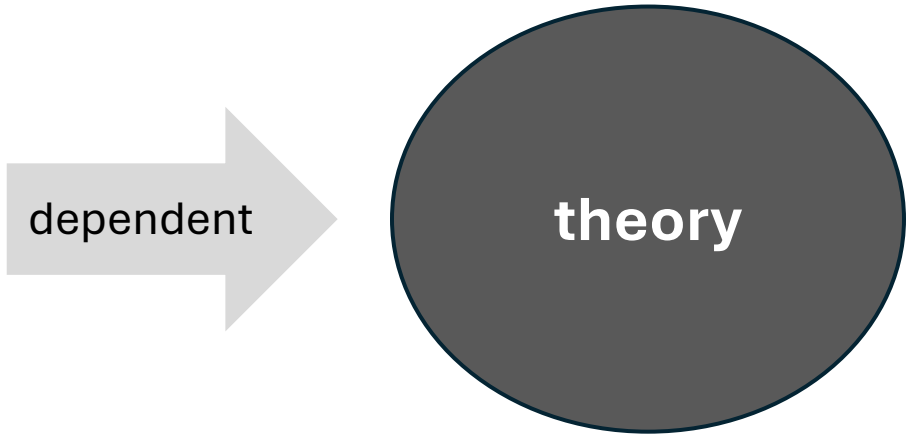
- continued adaptation
- modification
- correction of errors



the **decay** of a program text is a result of **modifications without** this knowledge

with large programs:

- continued adaptation
- modification
- correction of errors



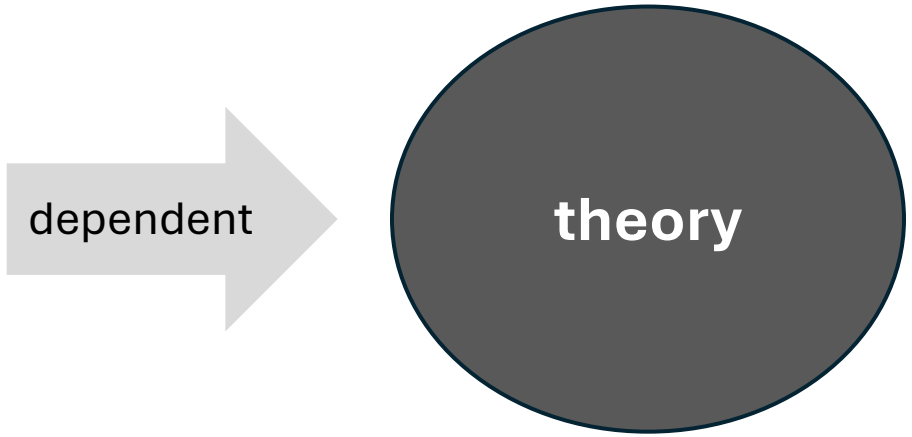
the **decay** of a program text is a result of **modifications without** this knowledge

How to obtain the Theory

- Reading the code?
- Reading the documentation?
- Learning “methods” and good practices?

with large programs:

- continued adaptation
- modification
- correction of errors



possessed by a group of programmers who are **closely and continuously connected** with the software!

Characteristics of the Theory

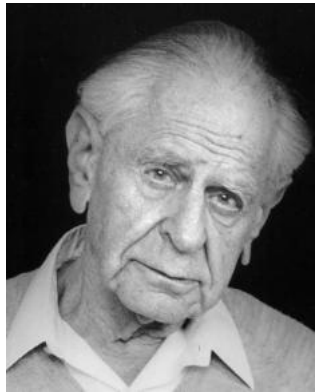
- Something that is *built* alongside the software (see title: Programming as Theory Building)
- Captured by metaphors: „this part works like an assembly line“
- Can be applied to completely new situations based on **similarity**
  - Like a physics theory
  - Cannot be expressed as rules: like similarities between faces and tastes of wine

Characteristics of the Theory

- Only someone who has the theory can support each part of the code with **justification**, why it is the way it is
    - Can this be documented? → intuitive choice

„A main claim of the Theory Building View of programming is that an **essential part of any program**, the **theory** of it, is something that could not conceivably be expressed, but is inextricably **bound to human beings**.“

„A main claim of the Theory Building View of programming is that an **essential part of any program**, the **theory** of it, is something that could not conceivably be expressed, but is inextricably **bound to human beings**.“



Program life, death, and revival

- **Program building**
  - The **theory** is built during this stage
- **Program life**
  - Original team in charge
  - How to acquire theory?
    - learning music instrument
- **Program death**
  - Team is dissolved
- **Program revival**
  - Rebuilding of its theory?

A note on programming methods

„Much current discussion of programming seems to assume that programming is similar to **industrial production**“

A note on programming methods

„Much current discussion of programming seems to assume that programming is similar to **industrial production**“

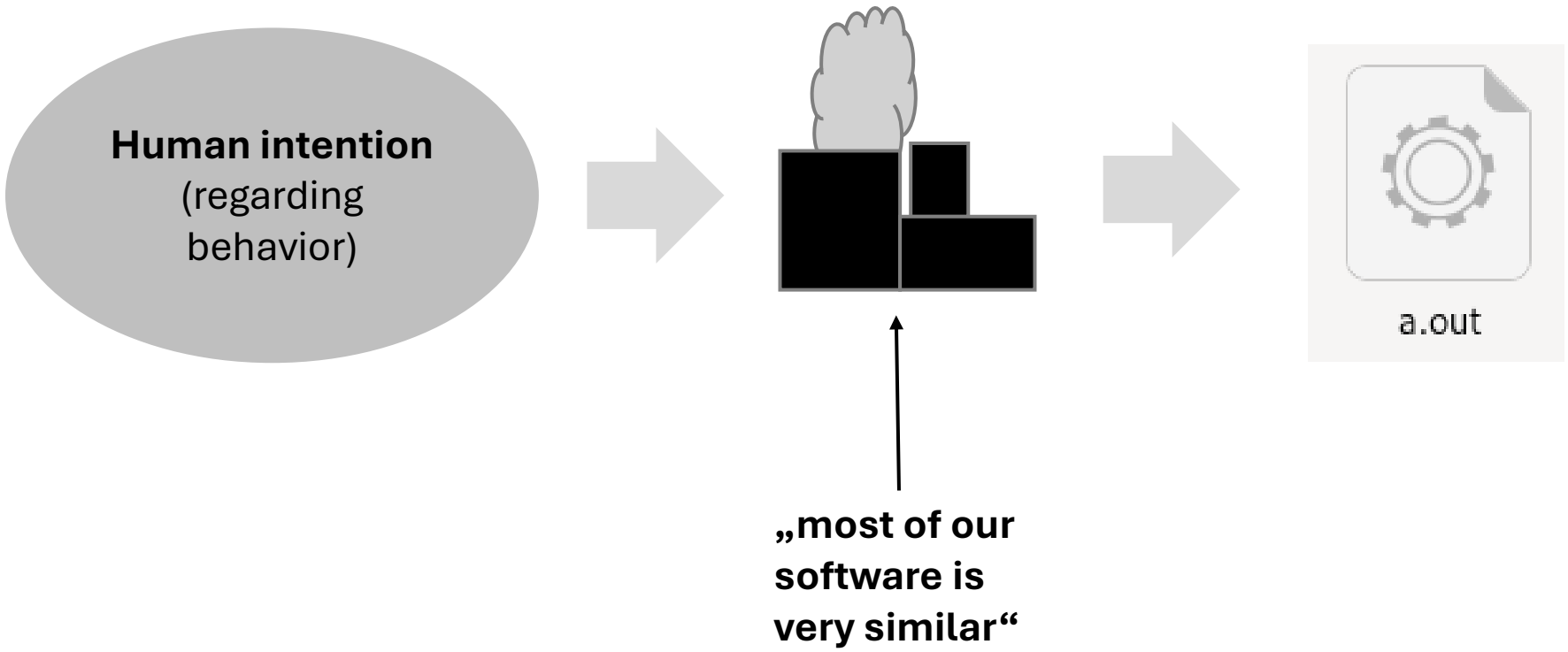
„What does the SDLC look like, when machines do the **mechanical work**, and humans do the judgement?“

A note on programming methods

„Much current discussion of programming seems to assume that programming is similar to **industrial production**“

Programmer → component of production

- should be controlled by **rules of procedure**
- can be **replaced easily**


Human intention
(regarding
behavior)
a.out
„most of our
software is
very similar“

Taylorism plausibility scale?

webshop for the 2147483647th small business selling t-shirt

interpreter for new programming language

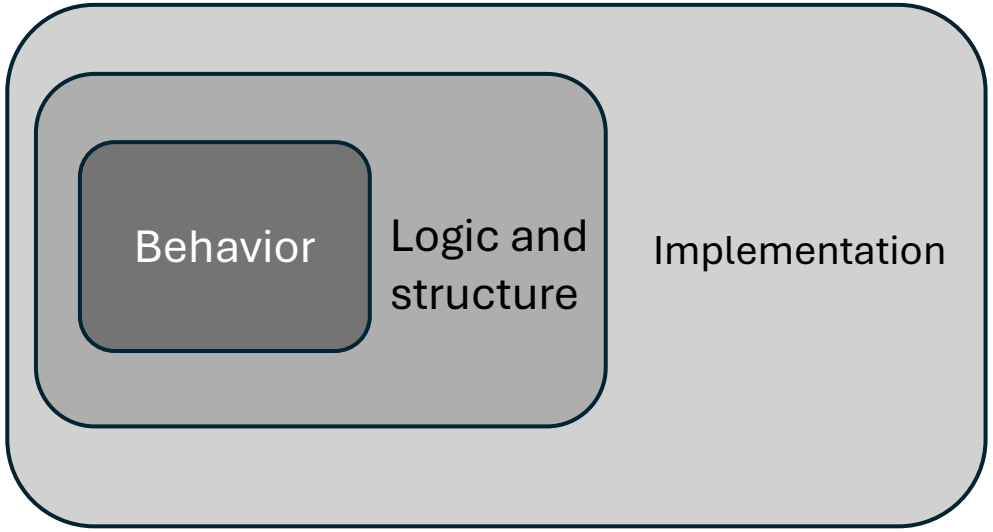

Takeaway

Exercise for home

- „What does the SDLC look like, when machines do the mechanical work, and humans do the judgement?”

Exercise for home

- „What does the SDLC look like, when machines do the mechanical work, and humans do the judgement?”
- „We focus on *what* we are building, not *how*”



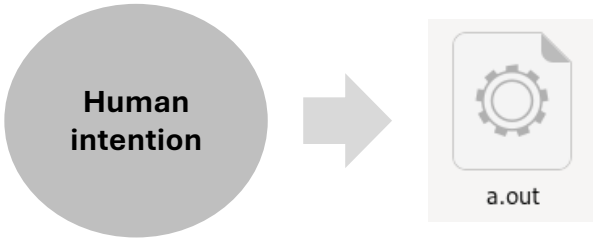
Exercise for home

- „What does the SDLC look like, when machines do the mechanical work, and humans do the judgement?”
- „We focus on *what* we are building, not *how*”
- „Transactions over 10k aren’t appearing in the report. Find it and fix it.”

the **decay** of a program text is a result of **modifications without** having the theory

Exercise for home

- „What does the SDLC look like, when machines do the mechanical work, and humans do the judgement?”
- „We focus on *what* we are building, not *how*”
- „Transactions over 10k aren’t appearing in the report. Find it and fix it.”
- „Did I miss *anything* from *any perspective*?”



Thanks for listening